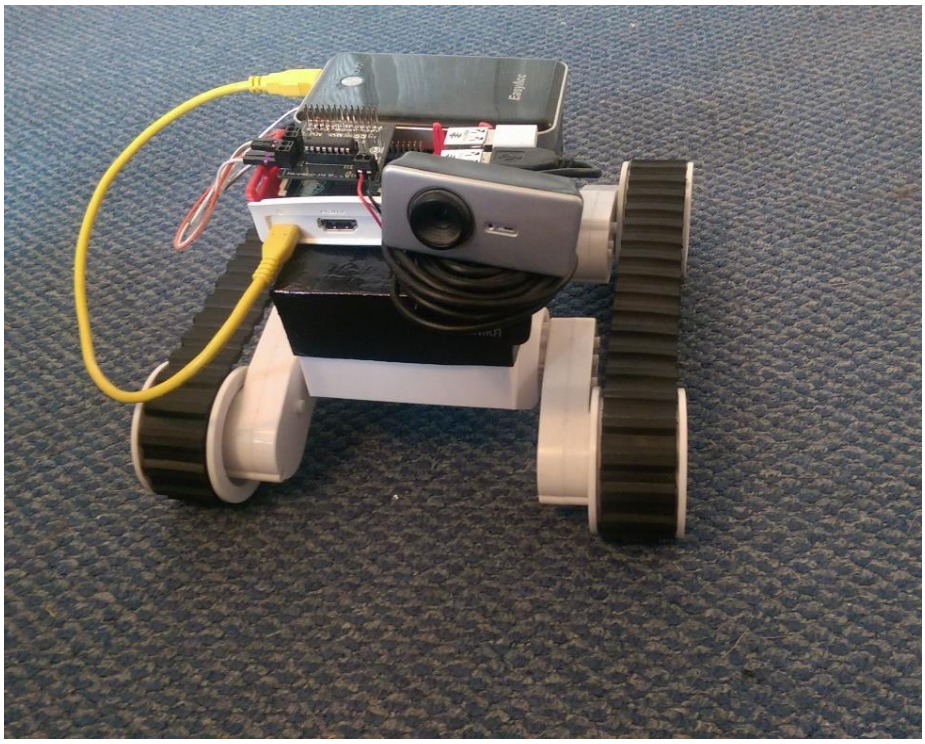
Maximilian Murach-Ward

IP camera based home security system

MSci Hons Computer Science

(With Industrial Experience)

17/03/2016



I certify that the material contained in this dissertation is my own work and does not contain unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation. Regarding the electronically submitted version of this submitted work, I consent to this being stored electronically and copied for assessment purposes, including the School's use of plagiarism detection systems in order to check the integrity of assessed work. My working documents including my consent form, video and more are available at:

http://www.lancaster.ac.uk/ug/murachwa/

I agree to my dissertation being placed in the public domain, with my name explicitly included as the author of the work.

Date:
Signed:

Abstract

Home security has the ability to make people feel safe. There are many different types of home security. However, there is a lack of affordable and varied robotic home security devices. The aim of this project was to create an IP camera based home security system that solves the issue of affordability and availability. Using a Raspberry Pi webcam server written in Python, and an Android mobile application written in Java, the system sends a video stream over WiFi to the user while they control the robot on their smartphone. User testing revealed the system is easy to use and praised for its simplicity, appealing to the everyday person. Participants agreed that they would use the system and that it worked as a successful solution to the problem of home security. Results showed that the system could also be used for different types of domestic monitoring such as pets and children. They also showed that this was a viable concept, and with further testing could be developed as a product. Overall the system was successful in achieving the aim set out and creating a solution to the problem.

Contents

1 Introduction

1.1 The aim of the project

The aim of the project was to build an IP camera based home security system that worked over WiFi, that could also be accessed and controlled from a smartphone and would be comparable to commercial applications. The system would be something that was comparable to available products but something that also had some unique features, such as the ability to move around, as the stationary nature of most household security is something that is often emphasised in the literature. The system should also be at a price that both a renting tenant and a home-owner could afford.

1.2 Overview of the requirements which this project addresses

This project addresses a number of requirements. The main requirement addressed is that people often want to have some home security and be sure that there is no one burgling their home, especially if they do not spend a lot of time at home or have something in particular they want to protect.

A security camera offers peace of mind because it monitors a home when the owner is not there. A problem with standard security cameras is cost; and a large number may be needed to provide coverage of a larger house or building. Traditional security cameras also recorded their feed onto a storage medium such as video tape or (in more recent times) optical or hard discs. It was not possible to view the output of the cameras in real time. The solution allows owners to view their house remotely whenever they like and from almost any location to check everything is safe and allows the owner to move the camera around in order to have coverage of several areas using a single camera. It was also cheap to produce.

Another requirement that is not necessarily security-related but which is addressed by the solution, is for people who are away from home a lot or work late who may want to check on their pets. This project solves this by allowing the owner to move the camera around and find their pets to see what they are doing as the system would be roughly the same height as many pets.

1.3 Why the project is worth doing

The project explores alternatives to home security and also builds on current widely available technologies to build a product which would be well suited to people's needs and would provide extra functionality for people who already use existing home security products. This is because the product is cheap (it would be less than £100), useful, and multipurpose.

1.4 Summary of the report

Chapter 3 looks at some journals and web-articles related to the subject of home security and discusses existing systems that are related to or similar to ours. The implications these have for the project are discussed and the choice of platform, software, and solution are also examined. Chapter 4 gives details about the system's architecture and discusses the different parts of the system. The designs made for the final product and the architecture of the product are also discussed and shown. Chapter 5 covers what the system is like to use and also has a walkthrough of its main features. Chapter 6 looks into the code and hardware behind the system and looks into the implementation of the system in more detail than the other chapters. Chapter

7 looks at tests that were run on the system to make sure it was actually functional and worked the way it was meant to. Also included in this chapter is user feedback. Finally, Chapter 8 revisits the aims and objectives and considers whether they have been met. It also looks at what was learnt by doing this project along with what could have been improved or added if there had been more time.

2 Background

2.1 Literature review

2.1.1 History of surveillance cameras

Thomas Edison and William Dickson gave the first public demonstration of motion pictures in 1893. Miniature movie cameras, which could be held in one hand and used without drawing too much attention, began to be developed from 1939 onwards (Delgado, 2015).

According to (Berthoud, 2013), the world's first security camera was set up in London in 1933 by an allotment holder called Mr. Norbury who was having eggs and chickens stolen from his hut. He used a small, still-taking box camera and rigged it up so the shutter was operated whenever anyone opened his hut door. When the film in his camera was developed, Mr. Norbury found he had a picture of the thief and the police were able to use this to convict the thief in court. The first installation of closed circuit television system (CCTV) was Peenemünde in Germany in 1942. The system was used for observing the launch of V2-rockets (Tudge, 2010). CCTV could supply a live feed, but could not – when first used – record anything directly (though output could be filmed using conventional film cameras). In 1951, however, the first practical videotape recorder was developed by Charles Ginsburg at the Ampex Corporation and the Sony Company began selling video cassette recorders (VCRs) to the mass market in 1971 (Bellis, 2016). Meanwhile, in 1966, Marie Van Brittan Brown and her husband Albert invented the first audio and video home security system which allowed residents to vet visitors over a camera feed before pressing a button to allow them access. (Kelly, 2015).

As video cassette and camera technology became more and more widespread and much cheaper, the use of security and surveillance cameras became increasingly used not just by the police and large companies - as described in (Tudge, 2010) – but by home owners and the general public. Cheap VCR technology that could record for long periods (even if this meant sacrificing quality) and play back at high speed meant that it was no longer necessary to employ anyone to watch CCTV feeds in real time.

This brief history into VCR technology provides information as to how it became a popular method of home security. It led on to the next big revolution, digital technology, something people are using and improving today. This project is an example of how modern technology can improve home security and make people feel safer in their homes.

2.1.2 Improvements in home security

As stated by (Sixsmith, 2000) and (Harris, 2010) people feel safer when they have some form of home security. This is especially the case in the elderly who may be living alone without constant care. However, it is also said that this security should be something that is not visible from the outside. (Schneier, 2003) agrees, as he talks about thieves choosing to ignore alarm systems or, destroying them in order to get to the object that is 'obviously' being protected by home security systems. In a technology-centred world, people are increasingly wanting to upgrade their home security to feature intelligent systems.

Usually, security is focused in one space and has limited mobility for example when attached to walls and doors. There is, therefore, an increasing demand for something that can navigate around and monitor the inside of a home or other premises. There have been a few

developments in robot home security systems in Taiwan and Japan where, for example, the problems of moving around the home, where rooms may be irregularly structured or have obstacles within them, have been considered. One such development is of a 'hopping' robot (Song, et al., 2009) which can extend its 'legs' to move over obstacles at 40cm in height. The robot that has been created for our project is also able to move over or around various household obstacles (discussed in detail later). Other mobile robot designs involve sensors which detect changes in the house, for example a fire, gas or an intruder. These changes are then sent to the user via the internet (Luo, et al., 2005).

2.1.3 Smartphone use in home security

Over time, smartphone use has increased dramatically with over two-thirds of Americans currently using one, an increase of 29% since 2011 (Smith, 2015). The use of smartphones in home security is also on the rise: for example, one study shows how Bluetooth on an Android device can be used to lock, unlock and check the door where a security system has been installed (Potts & Sukittanon, 2012). It can be seen that at the moment, the main uses of Android systems in home security are for automation - used for controlling appliances (Javale, et al., 2013). However, other systems can be used for detection purposes for example using infrared sensors to detect a person, at which point a passcode will have to be entered on the Android device and an SMS is sent to the owner for authentication (Rajadurai, et al., 2015). One example found in the literature review was a study using an Android mobile device as the camera on top of a small car (as opposed to using the Raspberry Pi along with the webcam), as the basis of a security system (Tangtisanon, 2014). They found that, overall, this type of system works well and is easy to use.

2.1.4 Similar systems

Products similar in nature to the system developed here include the 'Spy Snooper Robot' which detects sound instead of a picture, and this is priced at $80 (Fallon, 2007). Other wireless IP cameras range from £50 to £600, but these are stationary and while they may be able to tilt, they cannot move around the house (Maplin, 2015). In comparison to these products, ours not only offers the unique ability to move around a home, but also comes at an affordable price of less than £100.

2.2 My choice of platform, software and solution

Before starting, decisions were made on how to go about producing the system.

2.2.1 The Smartphone

The first decision made was which type of smartphone should be used for the project. The options were: a Windows phone, an iPhone, or an Android phone. This was a fairly easy choice, as the developer already had access to an Android phone so had programming apps for this type of phone and also experience using one. Android is also the most widely used mobile platform (Butler, 2011) and has been used in other home security systems as discussed in Chapter 2.1.

2.2.2 The Programming language

The table below shows the advantages and disadvantages of different languages considered for the system, based on past experience of using them.

Language	Advantages	Disadvantages
Python	• Easy to use – nice syntax • Simple • Existing expertise	• Does not work well on Android
Java	• Works well on Android • Existing expertise • Multi-platform • Garbage collection • Object orientated • Lots of supporting libraries	• Java based UI is quite often clunky and slow • More difficult syntax
C++	• object orientated	• Does not have (built-in) garbage collection • Difficult syntax
C#	• Lots of supporting libraries	• Platform Dependence

The next decision was which programming language to use. The initial thought was that Python would be a good language to use as it is very simple and easy to program in. In addition to this, the developer had experience of programming in Python and there were webcam libraries which could be made use of. The only problem with Python for this project was getting it to run on a smartphone. Through research into the software, it was discovered that it was possible to convert Python applications to Android applications, however, in practice, this was complicated and often simply did not work. This led to consideration of using Java instead. Again the developer had lots of experience with Java and it is one of the main language used to make Android apps. In the end, both languages were used but for different aspects of the project - Python for the webcam server and Java for the client application which would run on the Android phone. This expanded the developer's skills in both programming languages.

2.2.3 The Webcam Server

After deciding on a language, the next decision was what the webcam server would actually be. As was noted in the Introduction and the Literature Review, one goal was for the webcam to be able to move around to overcome the need for multiple cameras. The decision was made to attach a portable computer (powered by a battery) to a robot. The portable computer would control the motors on the robot and the mobile application (running on the smartphone) would send commands to this portable computer over a network. The commands would be converted to motor movements causing the robot to move in the direction required by the user. While this was going on, a video would also stream from a webcam (attached to the portable computer) to the client application.

Given this plan, a decision had to be made as to which portable computer to use. It had to be something very small which could fit on top of a robot and could be powered for a few hours by a rechargeable battery to overcome the problem of it not being possible to have the computer permanently plugged into a mains socket. In addition to this, it had to cost less than £100 as the target market was a range from those who rent a house to those who owned one. This would be very important if the project were ever to be developed as a proper commercial product. The Raspberry Pi was an immediate thought as it is small and cheap, with output pins that could be used to control the motors on the robot. As (Andrews, 2013) suggests in his article about encouraging use of the Pi in schools, the device must be easy to use if school children can use it. He also discusses its "throwaway" nature: if it breaks it does not cost much to replace. The Raspberry Pi comes with many features that were suitable for this project, including the facts that: it can be powered by cheap and light rechargeable batteries adding to its cost effectiveness; it is able to work over WIFI (with the addition of a dongle); it has enough USB ports to plug a webcam in at the same time as a dongle; and its operating system comes with Python pre-installed.

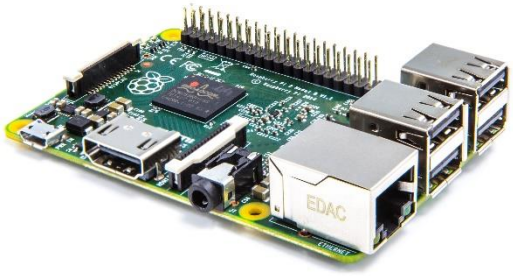


Figure 1 Raspberry Pi (source: raspberrypi.org)

2.2.4 The Motor Control Board

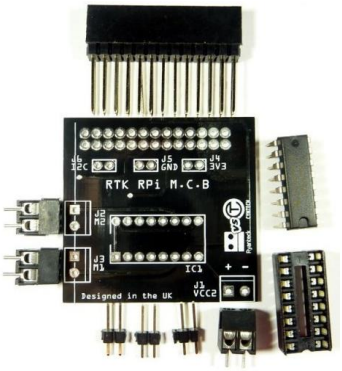


Figure 2 Ryanteck RPi (source: ryanteck.uk)

The only problem with the Pi is that it cannot directly power and control motors. This drawback was however, easily solved by use of a small circuit board which connects to it and sits on top of it and creates an interface between the Pi and the motors on the robot. There were a few chips that do this, however the simplest, smallest and cheapest was the "Ryanteck RPi Motor Controller Board". This board fits on top of the Raspberry Pi and is partly powered by the Pi but also takes another power input (up to 12V) for the motors. The board then has 4 output ports, a positive and negative for each motor, meaning it can drive two motors - which was what was needed for this project. So when the Pi outputs on a certain pin, that signal would be converted to a certain motor movement by the motor controller board. This motor controller board was discovered when reading a project tutorial on the Raspberry Pi website (Pi, n.d.). The robot discussed next, was also used in this tutorial and so it was clear that the motor control board and robot were compatible.

2.2.5 The Battery, webcam and robot

To power the Pi, a 7000mAh battery was used, with a USB output which keeps the Pi running for a few hours. The webcam selected supports up to 640 X 480 resolution, 30fps video. While other webcams may have a better resolution and frame rate, the higher quality images would have been limited by the speed at which pictures they took could have been sent over WiFi. Not only this, but the webcam is cheap, keeping the end price low. The final hardware decision was the robot. This had to be something that could move around even an untidy house well. For example, the robot might have to be able to drive over a heap of clothes or drive onto a carpet without getting stuck, and so a robot with tank treads was used as this would aid the robot on rugged terrain and also meant the robot would even be able to work outside (as long as it was within WiFi range). The robot chosen was the "DAGU ROVER" which has thick rubber tank treads - perfect for a variety of terrains and surfaces - and was completely compatible with the chosen motor controller board.

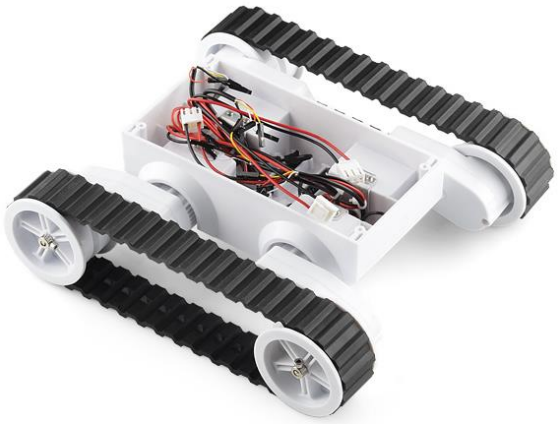


Figure 3 DAGU ROVER (source: proto-pic.co.uk)

2.2.6 Libraries

When starting to programme the system, a decision had to be made on a library to use for accessing the webcam. The library that was originally considered, and which the developer had used before, was "pygame" (pygame, n.d.) - a library which is built for gaming but also has a function to get an image from a webcam. However it soon became apparent that this was going to be a problem because of the format in which pygame gets images. Looking at other libraries, it was decided that the "opencv" library provided the best choice as it has a function to get images from the webcam but also has lots of other functions - one of which allows for the compression of images. This turned out to be very useful when it came to sending images over a network.

2.2.7 Security

Some level of security was also wanted for the system, so it was decided that there should be a username and password which the user would enter into the application and which the webcam server would check before starting the video feed and allowing a user to control the robot. The username and password would be specified when the webcam server was first started and then when a user wished to access the server they would need to enter that same username and password on the client application. Once the user was finished and no longer needed the server, they could log out. However if they wanted to have access again later, they would again need to enter the username and password.

2.2.8 User interfaces

For the client application, a GUI was used, while for the webcam server, a simple text interface was used so that it could be accessed from command line over SSH. Very little user input was required for the server - just the username and password they wish to use. It was planned that the client's application GUI would consist of two pages: one page being the login/connect page and the other being the controls and video. The login page would have

three text boxes, one for the IP Address of the server, one for the username, and one for the password. The page would also contain a button for connecting to the server once everything had been entered. Once this button had been pressed the application would then switch to the second (controls and video) page. This page would contain a big space for the video feed at the top and then, below the video, several buttons (up, down, left and right) for controlling the robot. This second page would also contain a logout button which, as discussed earlier, would log a user out. This would ensure that the application would return to the first (login) page so that if a user wanted to have access again they would need to go through the original process.

3 Design

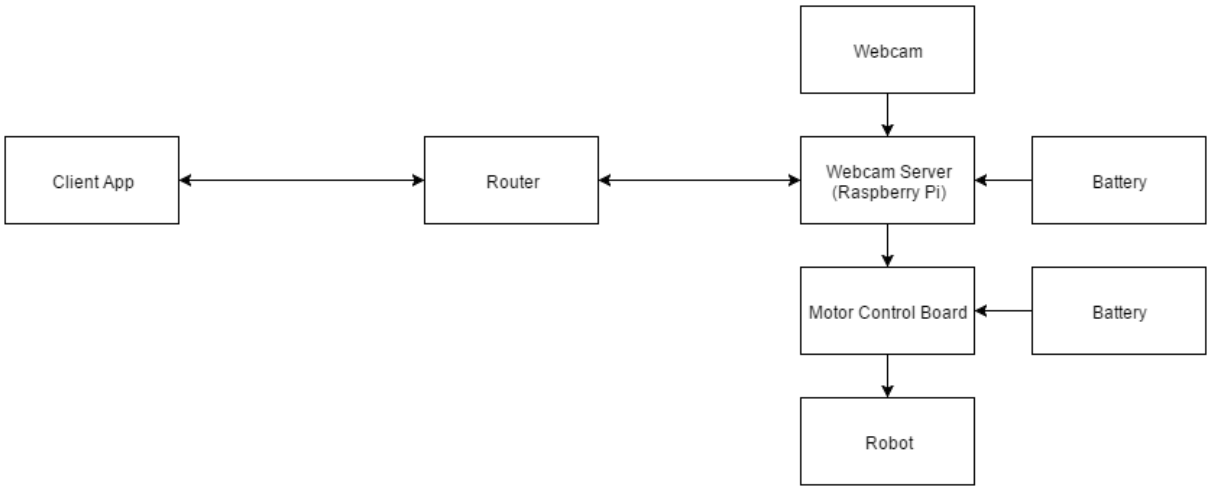


Figure 4 System design

3.1 System Design

The above block diagram for the system was formulated at the start of the design stage. Implementing in accordance with this diagram and making no changes, made the creation of the system more efficient. The Webcam Server (the Raspberry Pi) and the Client application (running on an Android phone) are both connected to the router and that is how both communicate with each other. When a client wants to log in they send the entered username and password via the router to the webcam server which then verifies both the username and password. The webcam is connected to the webcam server. The webcam takes pictures and passes them on to the server. These pictures are then sent on by the server to the client via the router. The motor control board is also connected to the webcam server and the robot itself. When the client sends a command that it wants the robot to perform, it is received at the webcam server after traveling via the router, and the webcam server then passes that command onto the control board which makes the robot perform that command. There are two batteries: one of them is for powering the webcam server and the other is for powering the motor control board which forwards the power to the robot.

Below are listed the different parts of the system, these include parts already talked about in the Background chapter:

Client Application

- Programmed in java.
- Runs on Android.
- Communicates over a wireless network.
- Can connect to the server as long as the Android device it is running on has internet.
- Sends the username and password to server to be checked.
- Sends commands to the webcam server to drive the robot.
- Receives video from the webcam server.

Router

- Passes packets onto the relevant device.

Webcam

- Takes pictures.
- Passes these pictures onto the server as quickly as possible.
- Is powered over USB by the Raspberry Pi

Webcam Server

- Communicated over a wireless network.
- Stores username and password that were entered when the server was started.
- Receives username and password from client and checks them against the currently stored username and password.
- Sends images to the client.
- Passes commands onto the motor control board from the client.
- Is powered by a battery.
- Is written in python.
- Runs on a Raspberry Pi.
- Receives pictures from webcam.

Motor control board

- Powered by the Raspberry Pi and by a battery.
- Redirects power to the robot.
- Controls the robot directly.
- Converts commands from the webcam server into robot movements.
- Works as an interface between the webcam server and the robot.

Robot

- Powered by a battery.
- Has 2 motors to control the 4 wheels.
- Is controlled by the motor control board.
- Tank like treads to get over rough terrain.

Both batteries

- One battery powers the webcam server and is rechargeable.
- One battery powers the motor control board and therefore the robot.

3.2 UI designs

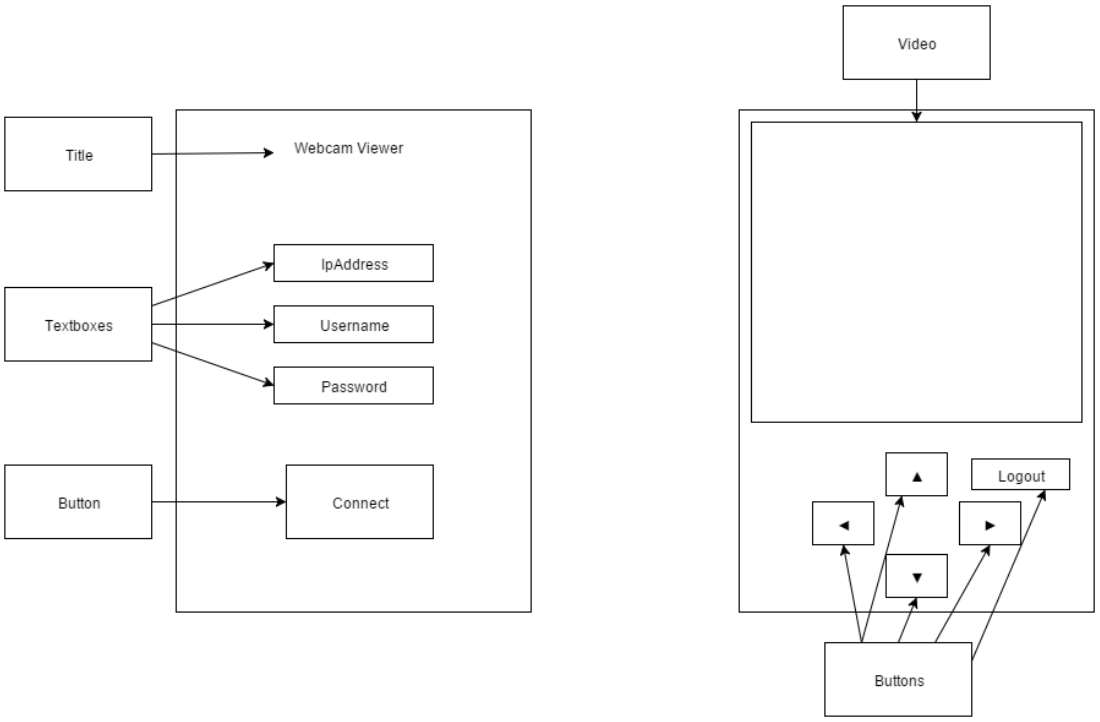


Figure 5 UI design

The user interface design remained consistent throughout the design process. It consists of a front page and a second page. The front page contains 3 textbox fields: two in which a user would enter their username and password (that were entered when the server was started), and one for the actual IP address of the webcam server. Also the front page has a button ("connect") which a user presses after entering all the details in the text boxes which make the connection to the webcam sever.

Once this button has been pressed, the application switches to the second page. The second page contains a big area at the top in which the video is displayed. Also, below the video, there are controls for the robot (including up, down, left, right) as buttons. Finally, there is a "logout" button that is pressed once a user had finished with the system.

3.3 Building the system

Once every part of the system, and how everything would work together, had been decided upon, the next step was building the system - starting with the robot. The version of the robot used was a four motor one (a motor for each wheel) as opposed to the two motor one originally planned. As it did not need 4 separate motors and the motor control board only supported two motors, the 2 wheels on each side were connected in parallel so that the motors worked together as illustrated in Figure 6. Both left side motors (motor 1 and motor 2) are connected together, and both right-side motors (motor 3 and motor 4) are connected together. The wires coming off the motor-pairs could then be plugged into the outputs of the motor control board.

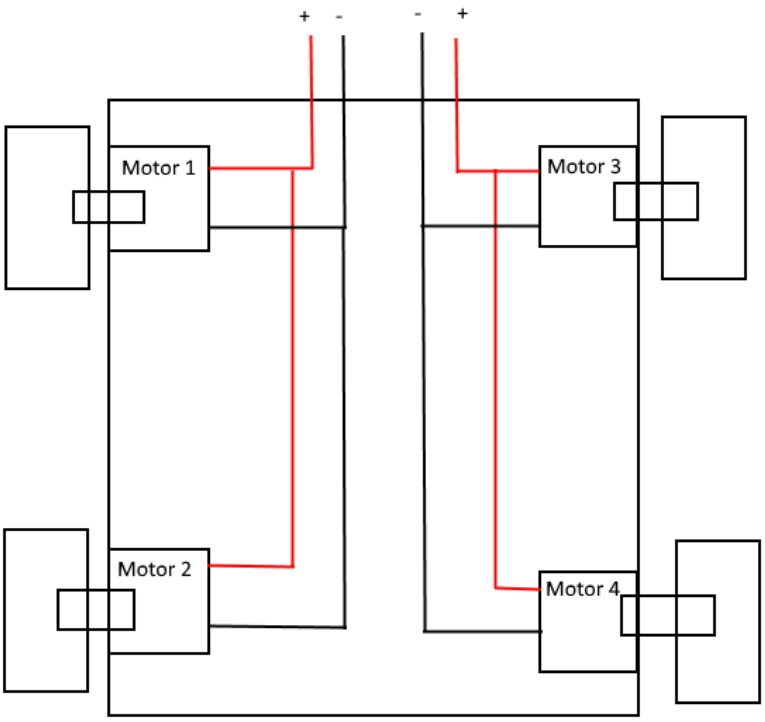


Figure 6 Robot motors

Connecting the motor control board proved more difficult than had originally been expected. The problem was that the board actually came unsoldered so a lot of time had to be spent with a soldering iron putting it together. Once this was done the positive and negative wires for each side of the robot were connected into the two motor outputs on the motor control board, then the positive and negative output of the battery were connected to the power input on the board (as shown in Figure 7). The battery consists of 6 AA cells and this is the battery that powers the motors on the robot.

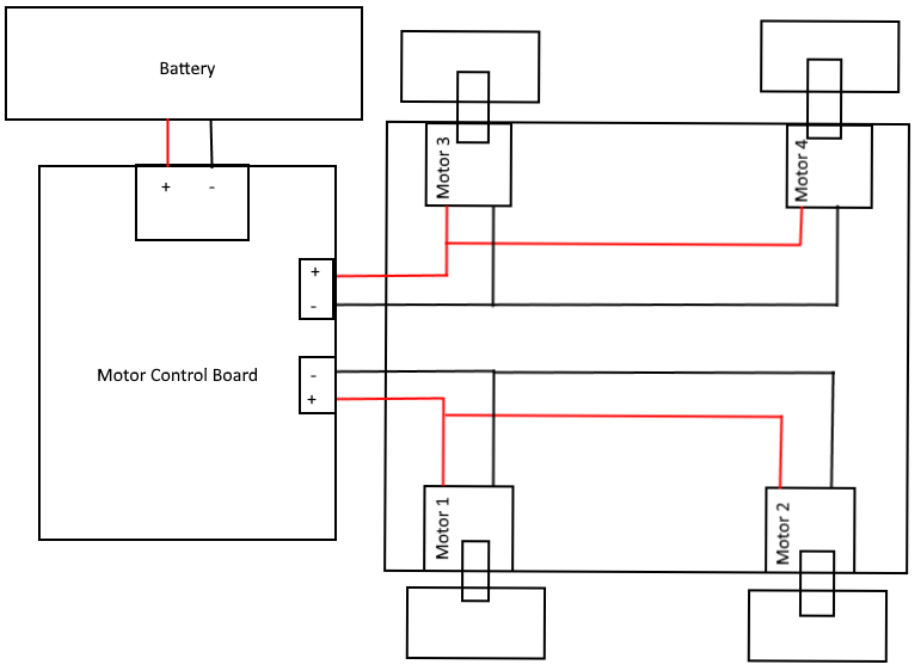


Figure 7 Motor control board

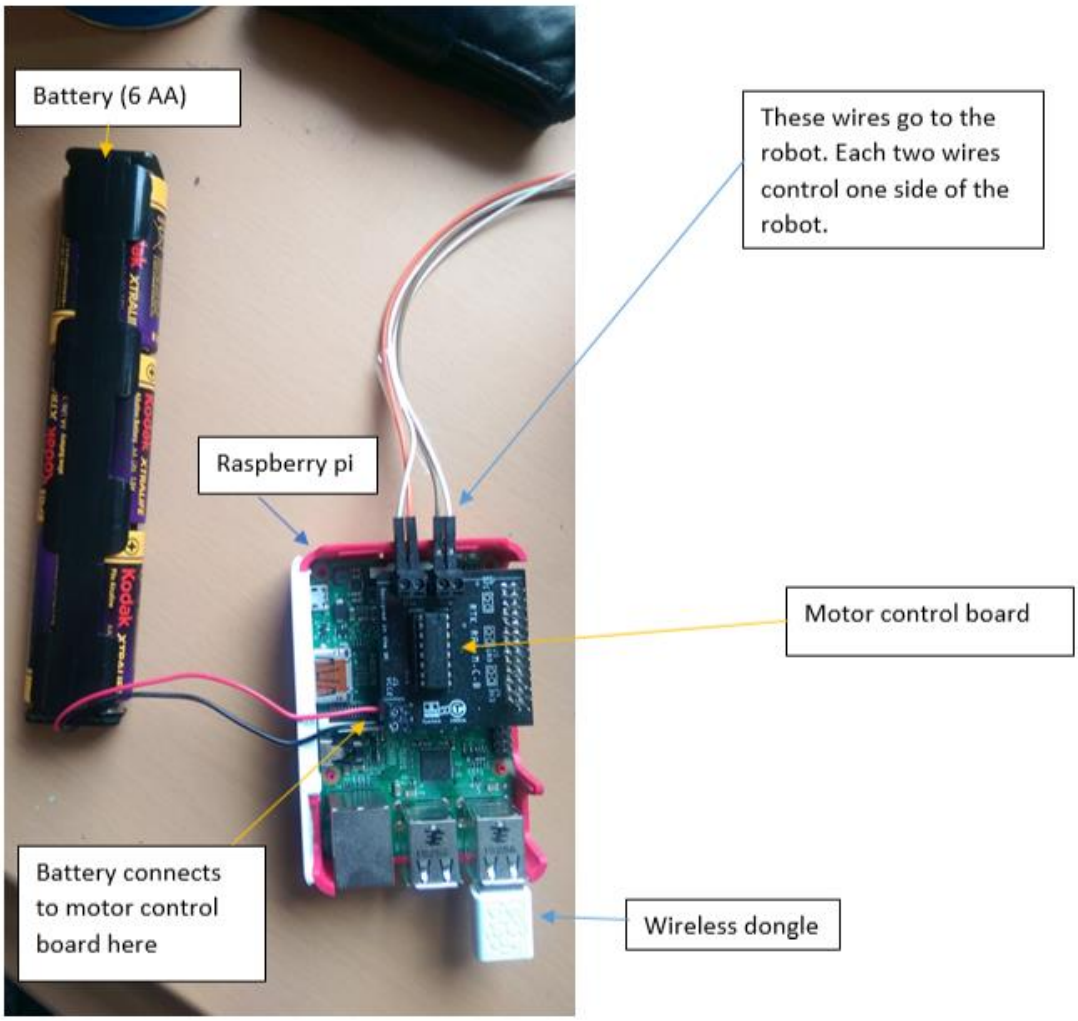


Figure 8 Connecting the Pi

The next step was connecting the Raspberry Pi to the motor control board. This was very easy as the motor control board simply plugs into the GPIO output pins on the Raspberry Pi and so sits neatly on top of the Pi (as shown in Figure 9 below). The Pi had the webcam plugged into it one of its USB ports and a WiFi dongle into its other USB port. It was also connected to a rechargeable battery via its micro USB power port.

Figure 9 GPIO output pins

The final step was mounting everything on top of the robot. To do this a cardboard and Papier-mâché lid was made that would sit on top of the robot and on which everything could be mounted. Velcro was used to attach everything on top, so that those things could be mounted and removed when necessary; the final result is shown below in Figure 10.

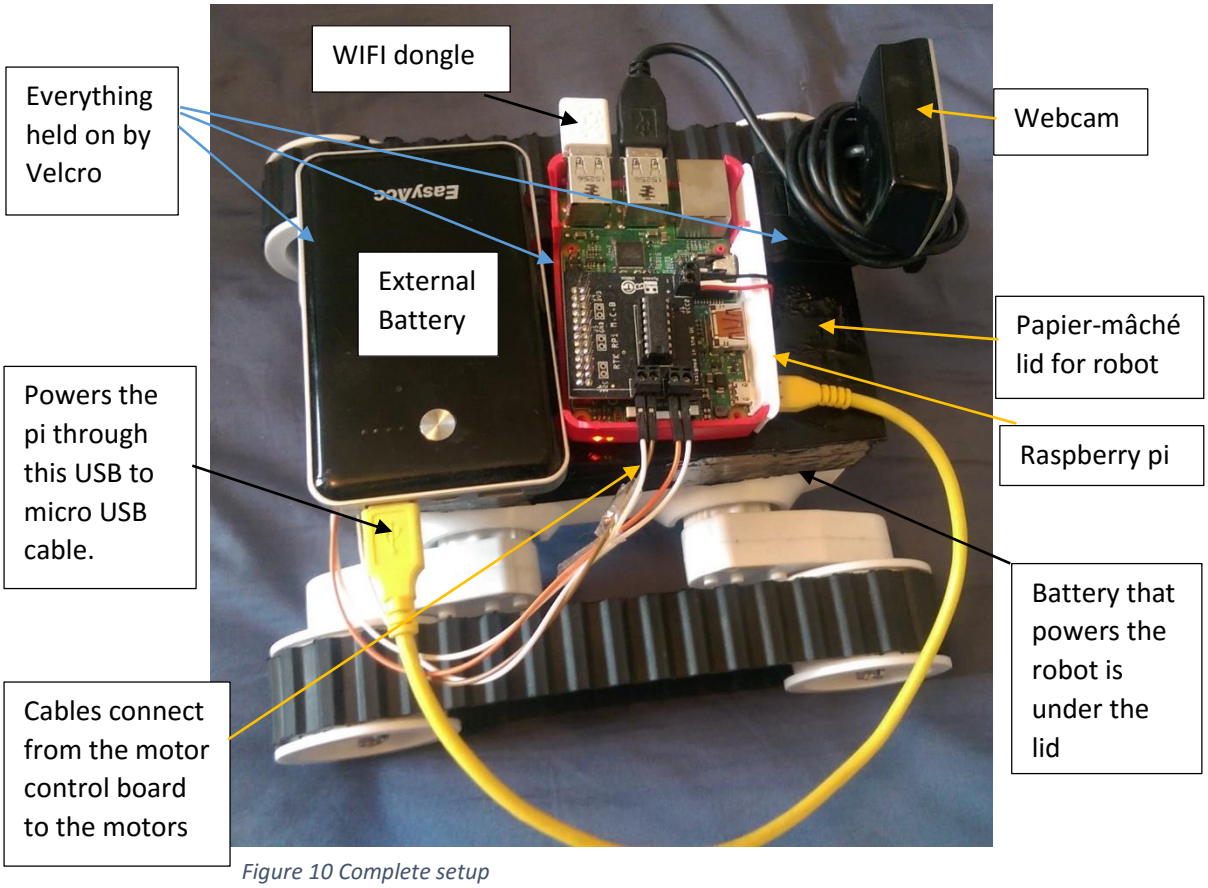


Figure 10 Complete setup

4 The System in Operation

4.1 Walkthrough

4.1.1 Starting the webcam server

There are a few steps the user needs to perform before they can fully use the system - locally and remotely. The first step is to start the webcam server. To do this a user first plugs the external battery into the Raspberry Pi which will make it boot up. Once this has been done, the user plugs in a keyboard and mouse into the Raspberry Pi and starts the server program from there. Alternatively, the user can use a program like Putty to access the Pi using SSH (Secure Shell) from another computer and start the server program from a command line. The SSH method is generally easier because the user does not have to plug the Pi into a monitor, keyboard, and mouse.

One of the problems a user may encounter when using SSH however, is finding the local IP address of the Raspberry Pi. The easiest way to get the IP address is to go into the router and look for the Pi as shown in Figure 11.

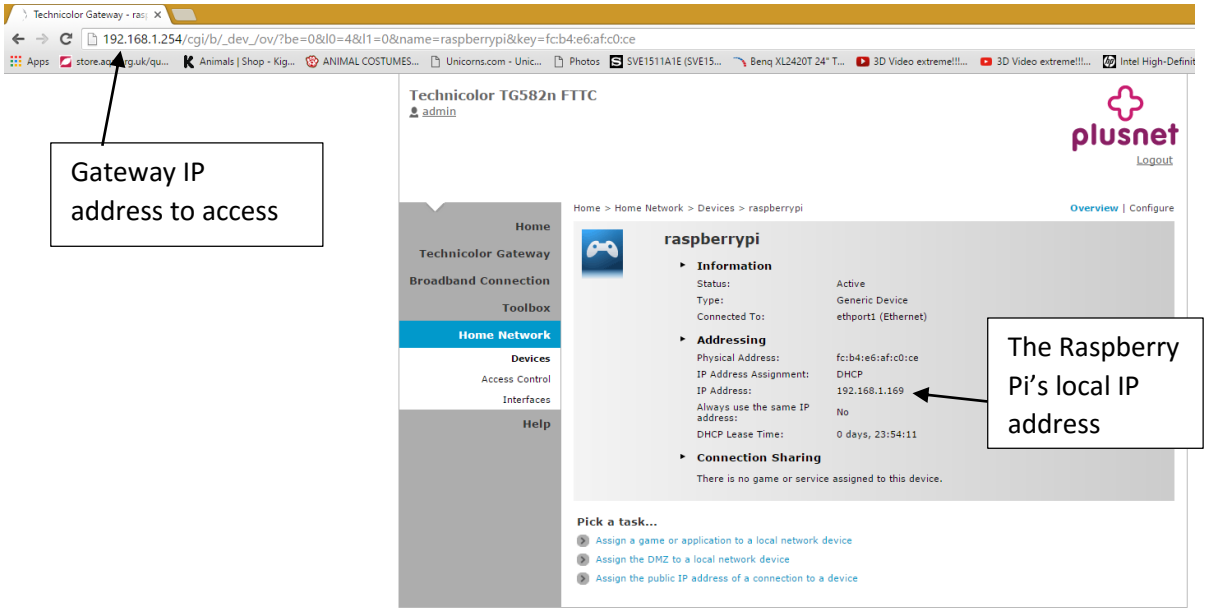


Figure 11 Router interface (specific to this Plusnet router)

It would probably be a good idea to give the Pi a static IP address the first time the system is used so that the Raspberry Pi will always have the same IP address.

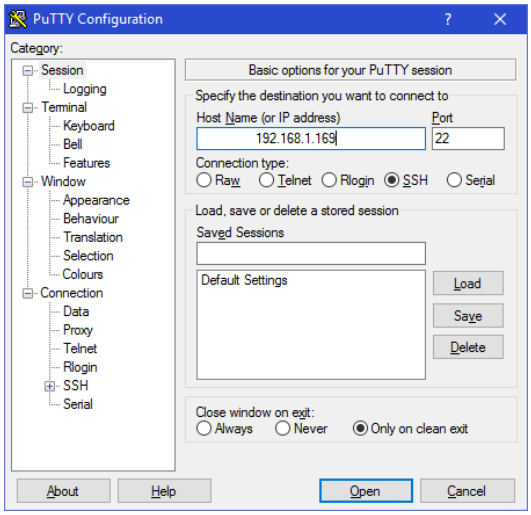


Figure 13 Putty configuration

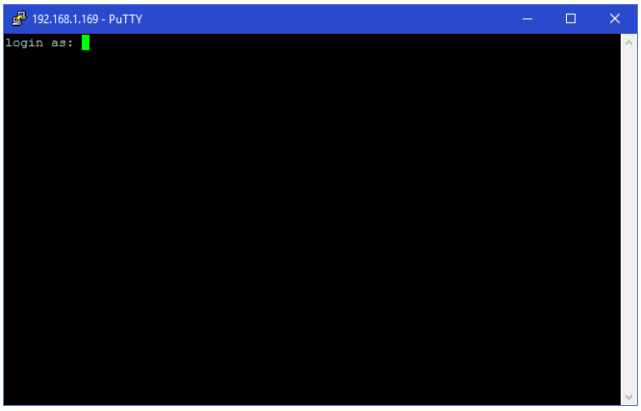


Figure 12 Putty command line

Once the Raspberry Pi's IP address has been entered, the user can connect to the shell of the Pi, as shown above in Figure 13 and Figure 12. The user then needs to enter the login details for the Pi and gain access.

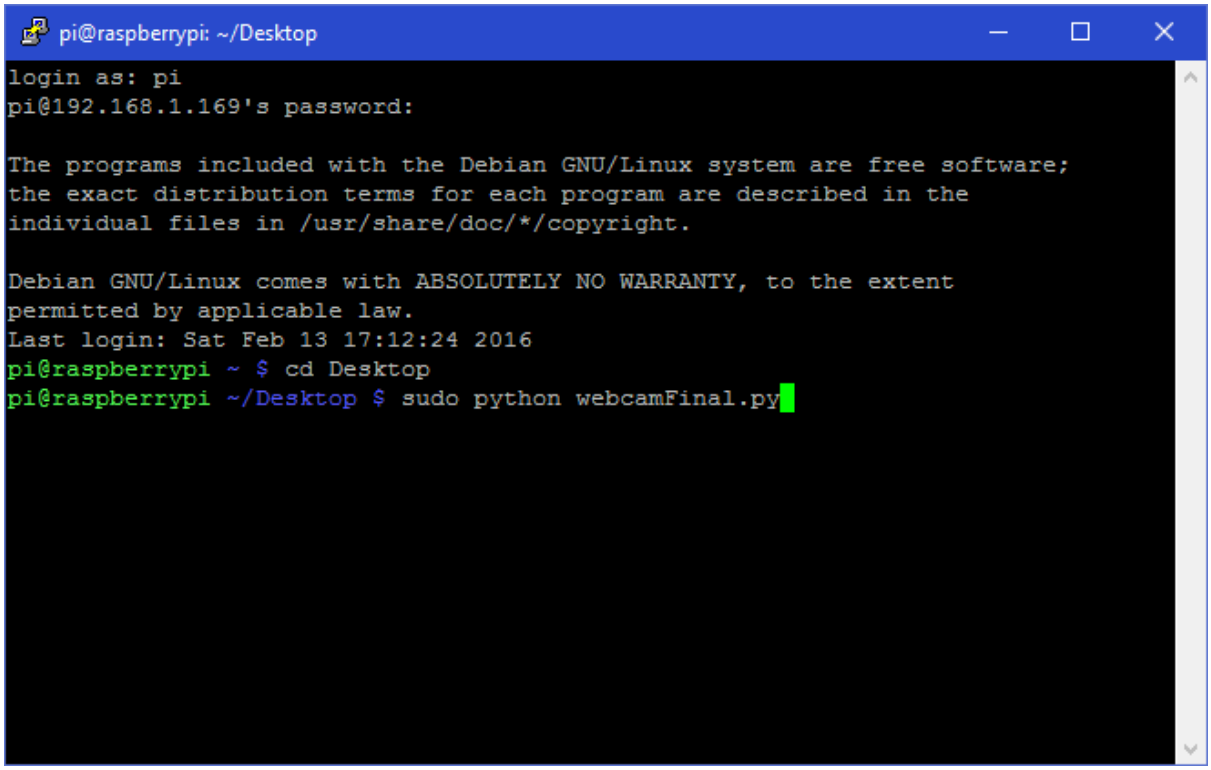


Figure 14 Pi desktop

A user can then simply start the server program by navigating to the folder it is in and running a "sudo python" command followed by the name of the python file to start the server (in this case *sudo python webcamFinal.py)* as seen above Figure 14.

4.1.2 Setting the username and password

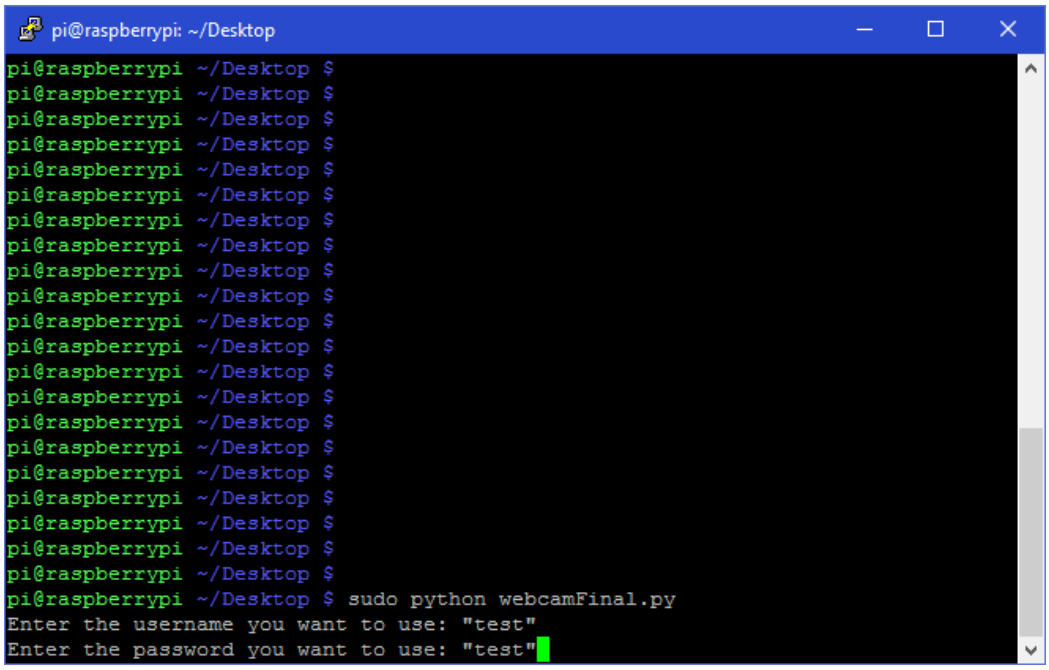


Figure 15 Entering username and password

The program then starts up. The first the thing the program asks for is the username and password – which will determine the details that the user wishing to access the server from their phone application will be asked for later. Both were set to “test” for this example as shown above Figure 15.

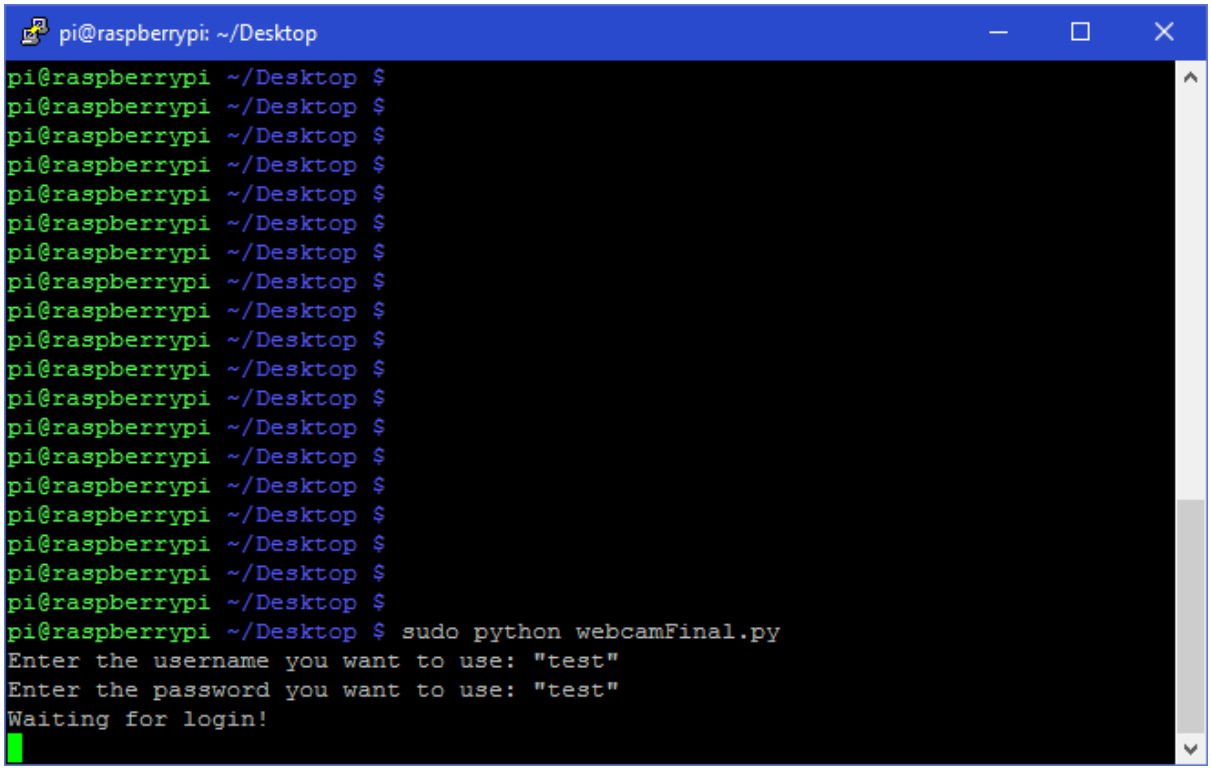


Figure 16 Waiting for login

4.1.3 Logging in from the client

Once the user has entered both the username and password, the system prints “waiting for login!” to the screen. This means that the program is now ready for a user to connect using their phone application, as seen in Figure 16.

The user can then open the webcam client application on a phone. When they open the application, the user is presented with this page (Figure 17). This is where the IP Address of the Raspberry Pi along with the username and password are entered.

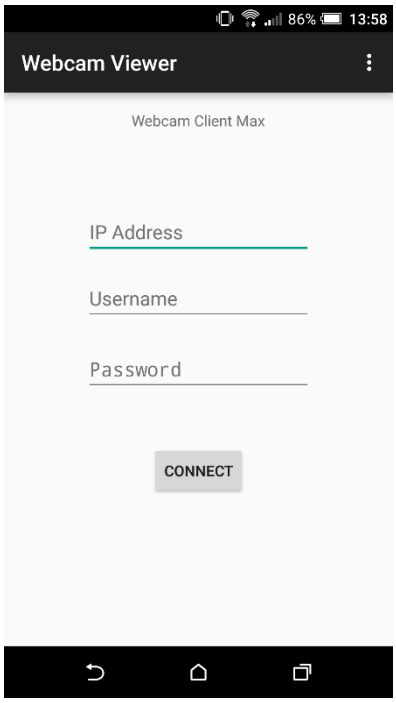


Figure 17 Login page

Once these details have been entered, the user hits “connect”. At this point the application sends the username and password to the webcam server which then checks that the username and password are correct.

If the username and password are correct the server will start up fully and begin sending the video feed and also accept commands sent from the client application to control the robot.

Once the user has pressed “connect” the server displays the username and password used to connect to it and the IP address of the device that has connected to it. A user could later check this to see if an imposter had been trying to gain access.

4.1.4 Using the application

The application switches to the second page as soon as the “connect” button is pressed. From this second page the user can view the webcam video feed and also control the robot using the directional buttons. When a directional button is pressed the robot will perform that action for 1 second. So, for example, if a user presses the forward button, then the robot will drive forward for one second. Similarly for turning; when the left button is pressed, the robot will turn left for one second - which roughly translates to a 45 degree turn.

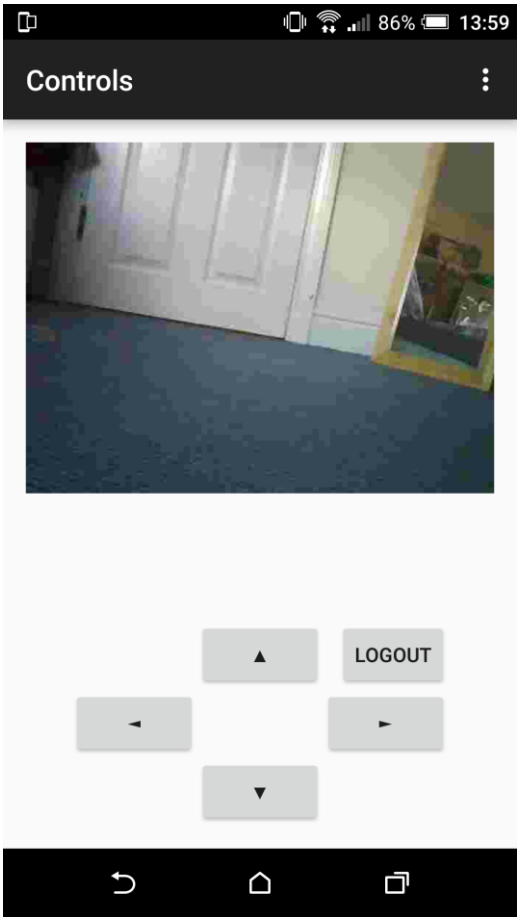


Figure 18 Controls

The user can also log out from here by pressing the “LOGOUT” button. This will bring the user back to the first page ready for them to log in again. The server will then stop sending the video feed and waiting for robot commands and go back to waiting for a login as seen in Figure 18.

If a user closes the application without logging out, the application will wait for the user to reconnect and continue sending the video feed. If a user then wants to reconnect they can just enter the IP address of the server and will be allowed straight in without needing to enter a username or password. This adds some insecurity to the system as anyone who knows the IP address of the server will be able to connect to it. It is, however, a useful feature if the user wants to check the camera every few minutes and does not want to have to enter the username and password every time. It would be possible for a user to accidentally create another connection to the server by going back to the previous page and clicking connect again. To solve this problem the back button is disabled. The user is not be able to get back to the previous page unless they log out or close the application fully. Either of these actions closes the connection to the server.

4.1.5 Remote access

If a user wants to use the system remotely from another internet connection all they need to do is port forward the two ports the program uses, to that Raspberry Pi, from their router. To do this, a user would need to go back into their router’s settings.

They would then need to port forward Ports 5000 and 2500 (the two ports used by the application and the webcam server) to the Raspberry Pi so now when someone uses the mobile application on a different network to the server all they need to enter is the global IP address of the router which the server is connected to. Below (Figure 19) is the port forwarding setup on the router that was used.

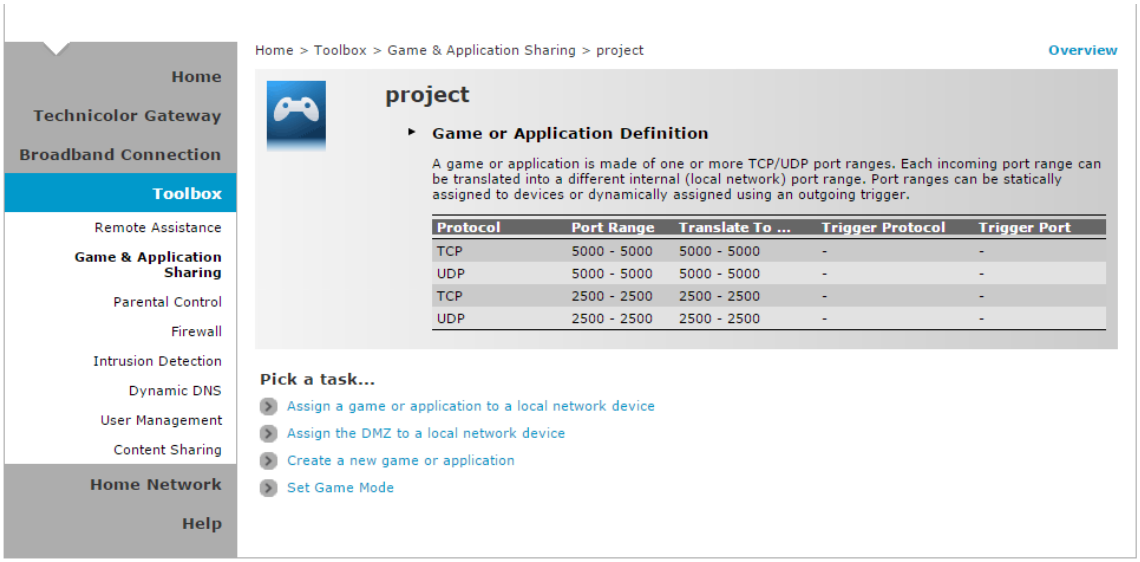


Figure 19 Router new project

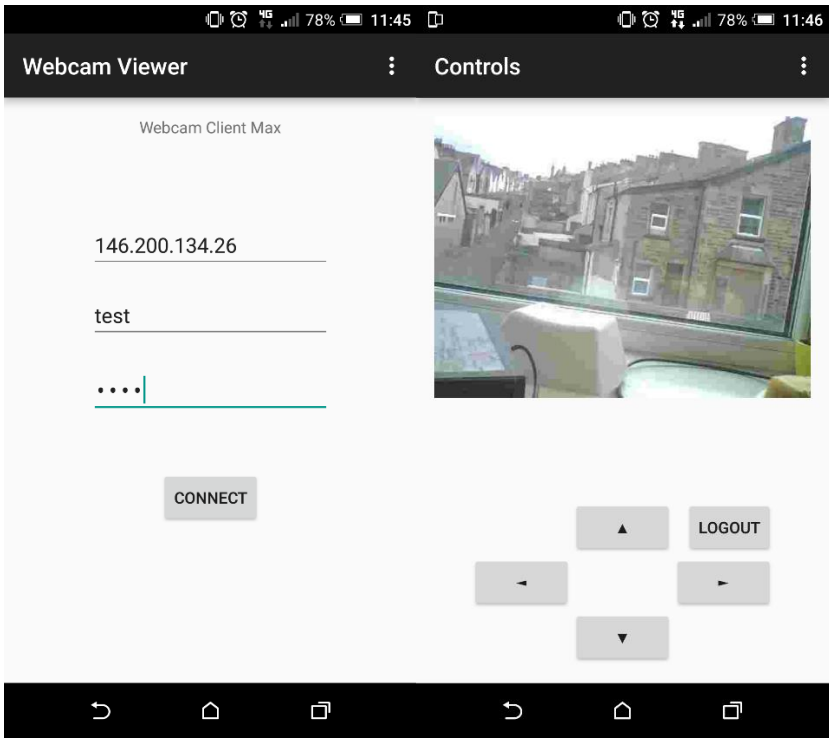


Figure 21 4G 1

Figure 20 4G 2

Here (Figure 21 andFigure 21 4G 1

) is the application connecting (using the ports just forwarded) to the webcam server from a 4G network using the global IP address of the home router.

4.2 Video

A video link showing the system in use appears in a footnote, and also on the final year project working documents webpage.[1]

4.3 Errors and problems

Lots of error reporting was added to the system so that if something goes wrong the system can tell the user what is wrong and how they could possibly fix the problem.

Figure 23 Error when the wrong IP address is entered

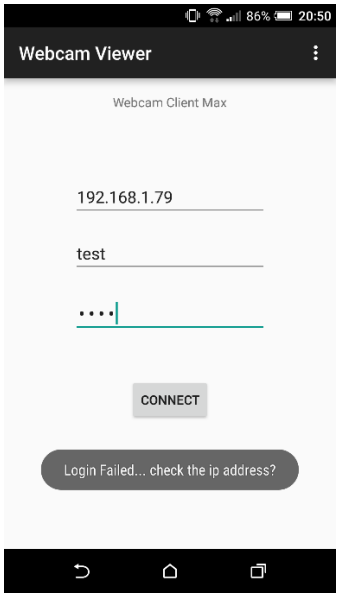


Figure 24 Error when the wrong username and password is entered

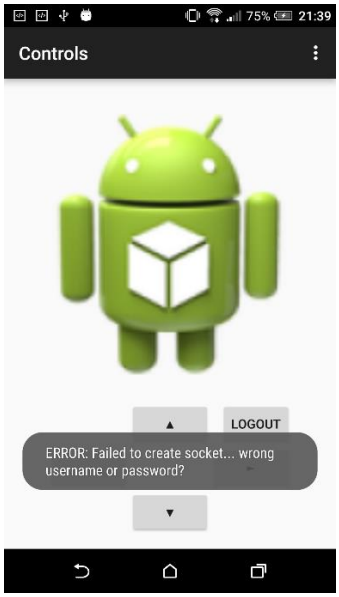


Figure 22 Error when the server goes down

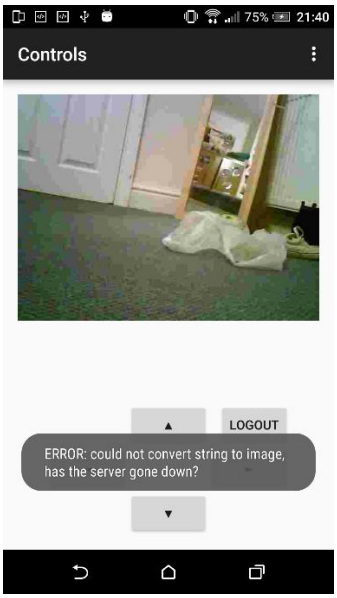


Figure 23 shows what error is displayed when the wrong IP address is entered, Figure 24 Error when the wrong shows what happens when the wrong username or password is entered and Figure 22 shows what happens when the server goes down. This could happen due to a number for reasons: for example if the Raspberry Pi's battery goes flat. The error reports give a technical overview of what has gone wrong for example "Failed to create socket" however they also give a more user-friendly reason for why this might be. This gives the user the information they will normally need to fix the problem but also helps developers by giving more in-depth information about the problem.

[1] https://www.youtube.com/watch?v=uqVCJJgCmuE

5 Process Description

In this chapter, the code behind the system will be discussed. First the client application written in Java will be covered and then the server written in Python. In each section of this chapter snippets of code are used. The full source code is available at the website (listed in the declaration) under working documents.

5.1 The client application

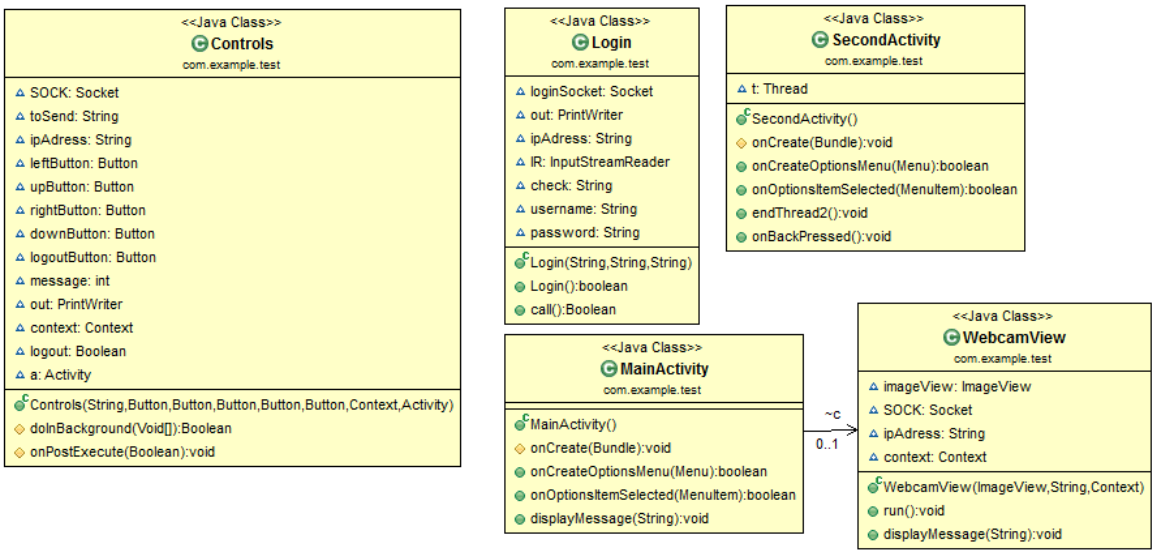


Figure 25-Class diagram

5.1.1 Permissions

```
<uses-permission android:name="android.permission.INTERNET" >
</uses-permission>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" >
</uses-permission>
```

The permissions for Android are set allowing the application to make use of the devices internet and network state.

5.1.2 MainActivity class

The first thing the client program does is start the “MainActivity”. The MainActivity described so far is the first page the user sees. MainActivity has a method called onCreate(), this is called when the activity is first started (created). Here, all the buttons and text boxes that are displayed on the first page are retrieved (as shown below).

```
//get all the buttons and text boxes from the first page
Button connectButton = (Button)this.findViewById(R.id.button1);
final EditText ipText   = (EditText)findViewById(R.id.editText1);
final EditText unText   = (EditText)findViewById(R.id.editText2);
final EditText pwTtext   = (EditText)findViewById(R.id.editText3);
```

The next step was to set up a button click listener which is connected to the "connect" button so that the program can tell when the user has pressed the button. When a user does press the connect button the "onCLick()" function is called.

```
connectButton.setOnClickListener(new OnClickListener()
{

    @Override
    public void onClick(View arg0) //when the connect button is pressed
    {


        //get all the text from the text boxes
        String ipAdress = ipText.getText().toString();
        String uN = unText.getText().toString();
        String pW = pwTtext.getText().toString();

        final ExecutorService service;
        final Future<Boolean>  loginTask;
        service = Executors.newFixedThreadPool(1);
        loginTask = service.submit(new Login(uN,pW,ipAdress));//start the login thread
```

When onClick() is called, the program gets the IP address, the username, and the password from the text boxes and stores them as strings in variables "ipAdress","uN" and "pW". The program then moves onto to starting the Login thread. The login thread returns "true" if it makes a successful connection to the server or returns "false" if it cannot make a successful connection. The Login thread is discussed later in more detail.

```
Boolean check = null;
try {
    check = loginTask.get();
} catch (InterruptedException e) {
    e.printStackTrace();
} catch (ExecutionException e) {
    e.printStackTrace();
}

if(check)//if a connection was made to the server using the ip address continue, if not show error
{
    try {
        Thread.sleep(1000);//wait for the server to get going
    } catch (InterruptedException e) {
    }
    Intent myIntent = new Intent(MainActivity.this, SecondActivity.class);
    myIntent.putExtra("ipAdress", ipAdress);
    MainActivity.this.startActivity(myIntent);//start the second page
}
else
{
    System.out.println("error");
    displayMessage("Login Failed... check the ip address?");
}
```

The main program then gets the return Boolean in the "check" variable and checks to see whether the connection was successful or not. If the connection was successful then the main program goes on and starts the "SecondActivity" which, up until now, has been described as

the “second page” the user sees when they press the connect button. The IP address is also passed onto the SecondActivity. If the connection fails, the program displays an error message to the user.

```
public void displayMessage(String text)
{
    Toast.makeText(this, text, Toast.LENGTH_LONG).show();
}
```

To display errors, a method named “displayMessage()” was used, which can be passed a string and which uses Android’s “Toast” feature to display the message to the user.

5.1.3 Login Class

```
Socket loginSocket = null;
PrintWriter out = null;
String ipAdress = null;
InputStreamReader IR = null;
String check = null;
String username = null;
String password = null;

public Login(String username, String password, String ipAdress)
{

    this.ipAdress = ipAdress;
    this.username = username;
    this.password = password;

}
```

The Login constructor takes 3 parameters: username, password, and IP Address. These are passed through by the MainActivity.

```
@Override
public Boolean call() throws Exception {
    Boolean returnedBool = Login();
    return returnedBool;
}
```

```
public boolean Login()
{

    Socket loginSocket= new Socket();

    try {
        loginSocket.connect(new InetSocketAddress(ipAdress, 2500), 5000);
    } catch (IOException e2) {
        return false;
    }



    try {
        out = new PrintWriter(loginSocket.getOutputStream(), true);
    } catch (IOException e1) {
        System.out.println("error2");
        return false;
    }

    out.println(username+" "+password);
    out.close();
    try {
        loginSocket.close();
    } catch (IOException e) {
        return false;
    }
    return true;

}
```

The “call()” function in Login executes the second “Login()” function. The Login function creates a socket and attempts to connect to the server using the given IP address on port 2500. If the connection is successful then a PrintWriter is created and the output of the PrintWriter is set to the socket. The PrintWriter is then used to send the username and password together with a space between them as a string down the socket to the webcam server. Once this is done, “True” is returned to let the MainActivity know that everything was successful. If something goes wrong, “False” is returned and the MainActivity will know something has failed.

5.1.4 SecondActivity class

```
public class SecondActivity extends ActionBarActivity  {

    Thread t = null;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_robot_control);
        final ImageView image = (ImageView) findViewById(R.id.imageView1);
        Intent intent = getIntent();
        String ipAdress = intent.getStringExtra("ipAdress");
        Context context = getApplicationContext();



        t = new Thread(new WebcamView(image, ipAdress,context));
        t.start();

        Button leftButton = (Button)this.findViewById(R.id.button1);
        Button upButton = (Button)this.findViewById(R.id.button2);
        Button rightButton = (Button)this.findViewById(R.id.button3);
        Button downButton = (Button)this.findViewById(R.id.button4);
        Button logoutButton = (Button)this.findViewById(R.id.button5);

        Controls c =new Controls(ipAdress,leftButton,upButton,rightButton,downButton,logoutButton,this,this);

        c.execute();
    }
```

When the SecondActivity is started again, the onCreate() function is called – just as it was in the first activity (MainActivity). The first thing that gets done is creating the ImageView. This is like a screen in which pictures can be displayed and which is used for showing the video feed sent by the server. The next thing is getting the IP address of the server from the intent which was passed on from MainActivty. Once this has been done the webcam thread is started, this is the thread in charge of receiving and displaying the images from the server. The IP address of the server, the ImageView just created, and the SecondActivity's context are passed on to this webcam thread. An explanation for why this is done will be given in my discussion of the WebcamView class.

After all this, the directional buttons that control the robot are created. The Controls thread is then started – this is the thread in charge of taking user input and converting these inputs to messages which are sent to the webcam server to control the robot. The IP address is then passed to this thread along with all the directional buttons created and the SecondActivity itself is also passed on. More explanation is provided below under the "Controls class" discussion.

```
public void endThread2()
{
    System.out.println("Attempting to end thread 2");
    t.interrupt();
}
@Override
public void onBackPressed() {
}
```

There are also two more methods in the SecondActivity class which the project makes use of. The first method is endThread2(). This method sends an interrupt to the second thread (the webcam thread) causing the thread to finish cleanly. The other method is the

onBackPressed() which overrides the normal onBackPressed() method. This method contains nothing and is used to disable the back button for this activity.

5.1.5 WebcamView class

```
public class WebcamView extends Activity implements Runnable
{
    ImageView imageView = null;
    Socket SOCK = null;
    String ipAdress = null;
    Context context = null;

    public WebcamView(ImageView img, String ipAdress,Context context)
    {
        this.imageView = img;
        this.ipAdress = ipAdress;
        this.context = context;
    }
```

The constructor of this class takes the ImageView, the IP address of the server, and the SecondActivity's context as parameters. It then stores these itself. The class itself also implements "Runnable" so methods within the class can be run on a separate thread.

```
@Override
    public void run()
    {


        String imageDataString = null;
        byte[] imageByteArray = null;
        InputStreamReader IR = null;

        BufferedReader BR = null;


        while(!Thread.currentThread().isInterrupted())
        {

            try
            {
                SOCK = new Socket(ipAdress,5000);
            }
            catch (UnknownHostException e)
            {
                System.out.println("Wrong ip?");
                displayMessage("Wrong ip?");;
                return;
            }
            catch (IOException e)
            {
                System.out.println("Failed to create socket");
                displayMessage("ERROR: Failed to create socket... wrong username or password?");
                return;
            }
```

When this class is executed as a thread, the run() function is called. The first thing that happens in the run function is that all the variables are initialised to null before they are used later on. There is then a while loop which is only broken when an interrupt is called on the thread. Within the while loop, a socket is created from which the thread will receive the pictures sent from the server. If there is an IOException it is very likely that the server has not

yet created a socket for port 5000 and so has not started sending pictures because the user entered the wrong username or password earlier.

```
try
{
    IR = new InputStreamReader(SOCK.getInputStream());
}
catch (IOException e)
{
    displayMessage("ERROR: Problem creating input stream");
    return;
}


BR = new BufferedReader(IR);


try
{
    imageDataString = BR.readLine();
}
catch (IOException e)
{
    displayMessage("ERROR: Problem getting image");
    return;
}

try
{
    imageByteArray = android.util.Base64.decode(imageDataString, android.util.Base64.DEFAULT);
}
catch (Exception e)
{
    displayMessage("ERROR: could not convert string to image, has the server gone down?");
    return;
}
final Bitmap bMap = BitmapFactory.decodeByteArray(imageByteArray, 0, imageByteArray.length);
```

Next, an InputStreamReader is created to read the socket's input stream. Then a buffer reader is created to read this input stream. Doing this allows the program to read in the base 64 string sent by the server and store it in a variable ready to be decoded. The base 64 string is then decoded into a byte array and then this byte array is decoded and stored as a bitmap.

```
    runOnUiThread(new Runnable() {
            @Override
            public void run() {
                imageView.setImageBitmap(bMap);
            }
        });
}
```

The next step is to actually display this bitmap on the ImagView from SecondActivity. To do this, the runOnUiThread() is used as the actual Ui can only be changed from the Ui thread. Then the setImageBitmap() function is used to display the bitmap just created from the base 64 string.

```
System.out.println("Thread 2 done !!!");
return;
```

This thread will keep looping and updating the image when it receives a new one until it receives an interrupt; at which point it will exit.

```
public void displayMessage(final String text)
{

    runOnUiThread(new Runnable() {
        public void run()
        {
            Toast.makeText(context, text, Toast.LENGTH_LONG).show();
        }
    });
    }
```

This class also contains a displayMessage() function, to display any problems or errors that happen during the running of this thread. It uses the exact same code as the displayMessage() function described earlier except the runOnUiThread() function is used so that the message is displayed on the active Ui.

5.1.6 Controls class

```
public class Controls extends AsyncTask<Void, Void, Boolean>
{
    Socket SOCK = null;
    String toSend = null;
    String ipAdress = null;
    Button leftButton = null;
    Button upButton = null;
    Button rightButton = null;
    Button downButton = null;
    Button logoutButton = null;
    int message = 0;
    PrintWriter out = null;
    Context context = null;
    Boolean logout = false;
    Activity a = null;

    public Controls(String ipAdress, Button leftButton, Button upButton,Button rightButton, Button downButton,Button logoutButton,Context context,Activity a)
    {
        this.ipAdress = ipAdress;
        this.leftButton = leftButton;
        this.upButton = upButton;
        this.rightButton = rightButton;
        this.downButton = downButton;
        this.context = context;
        this.logoutButton = logoutButton;
        this.a = a;
    }
```

As the controls class also needs to run on a different thread, it extends AsyncTask. “AsyncTask enables proper and easy use of the UI thread. This class allows for background operations to be performed and results to be published on the UI thread without having to manipulate threads and/or handlers.” (Android, 2016).

The constructor for this class also takes the IP address of the server along with all the directional buttons from SecondActivity, the SecondActivity’s context, and the SecondActivity itself as an activity. These variables are then all stored.

```
@Override
protected Boolean doInBackground(Void... arg0) {
    try {
        SOCK = new Socket(ipAdress, 2500);
    } catch (UnknownHostException e1) {

    } catch (IOException e1) {

    }


    try {
        out = new PrintWriter(SOCK.getOutputStream(), true);
    } catch (IOException e1) {

    }
```

When the class is executed, the doInBackground() function is called. In this function a socket is created for port 2500 and this is the port that will be used for sending commands to the server to control the robot. A PrintWriter is also created to send these commands as strings to the server.

```
leftButton.setOnClickListener(new OnClickListener()
{
    @Override
    public void onClick(View arg0)
    {
        out.println("l");
    }
  });
upButton.setOnClickListener(new OnClickListener()
{
    @Override
    public void onClick(View arg0)
    {
        out.println("u");
    }
  });
rightButton.setOnClickListener(new OnClickListener()
{
    @Override
    public void onClick(View arg0)
    {
        out.println("r");
    }
  });
downButton.setOnClickListener(new OnClickListener()
{
    @Override
    public void onClick(View arg0)
    {
        out.println("d");
    }
  });
```

An OnClickListener is attached to each of the directional buttons so that when one them is clicked, a command can be sent to the server to tell the robot how to react. For example, if the up button is pressed, “u” is written to the output of the socket by the PrintWriter. This “u” is then sent down the port 2500 to the server. This is the same for every button.

```
        logoutButton.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                System.out.println("logout");
                out.println("finished");
                if(context instanceof Activity){
                    ((Activity)context).finish(); }
                logout = true;


            }
          });

        while(logout == false)
        {
        }
        return true;

    }
```

An OnClickListerner is also attached to the “LOGOUT” button so when it is pressed the string “finished” is sent to the server. This lets the server know that it should go back to login mode and wait for the user to reconnect and re-login. The application then also finishes SecondActivity which means the application returns to MainActivity and waits for the user to click “connect” again. The variable “logout” is also set to “true”. The reason for the logout variable is to keep the thread alive until the logout button is pressed, as the thread will be kept in a spin lock until logout is set to true and the while loop exits allowing the function to return true.

```
@Override
protected void onPostExecute(Boolean b)
{
    System.out.println("Thread one done");
    SecondActivity mActivity = (SecondActivity) a;
    mActivity.endThread2();
}
```

When the doInBackground() function returns true then the onPostExecute() function is invoked on the UI thread. The only thing that happens in this function is the endThread2() method from SecondActivity being called which ends the webcamViewer thread. This means the application will be in the exact same state it was when it was first opened.

5.2 The webcam server

5.2.1 Main program

```
import cv2
import socket
import time
import base64
import RPi.GPIO as gpio
import threading

thread1Stop = threading.Event()
port = 5000
```

The webcam server program uses several libraries, the first and most important being “cv2”. This is opencv and it is what is used for getting the image from the webcam and compressing the images ready to send.

Several of Python’s inbuilt libraries are also used. These include “socket”, used for sending and receiving data over the network; “time”, which is used for pausing threads; “base64”, which is used for converting the images to strings; “GPIO”, used for accessing the output pins on the Raspberry PI; and finally “threading”, which is used to execute functions on threads.

5.2.1.1 Login and Authentication

The variable thread1Stop which is used for stopping one of threads later on is also initialised and the port initialised to 5000. This is used later as the port pictures are sent down.

```
username = input('Enter the username you want to use: ')
password = input('Enter the password you want to use: ')
while True:
    thread1Alive = True

    server = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server.bind(("",2500))

    server.listen(5)
    print ("Waiting for login!")


    while True:
        connection, address = server.accept()
        data = connection.recv(1024)
        if len(data)> 0:
            print(data)
            unPass = data.split()
            print(unPass)
            try:
                if unPass[0] == username and unPass[1] == password:
                    break
            except IndexError:
                print("Caught index error")
            print("Login Incorrect")


    connection.close()
    server.close()



    t1 = threading.Thread(target=MyThread1, args=[thread1Stop])
    t2 = threading.Thread(target=MyThread2, args=[])
    t1.start()
    print ("READY!")
    print ("Waiting for connection!")
    t2.start()
    t2.join()
    print ("Thread 2 finsihed")
    thread1Stop.set()
    t1.join()
    print ("Thread 1 finsihed")
    thread1Stop.clear()
```

The next thing the program does is take in and store the username and password the user wants to use (these are used later on). After this a socket is created (which uses port 2500) and SO_REUSEADDR is used to make the socket resuable later on without problems.

The program will then print out “waiting for login!” and move onto server.accept(). The program will hang here until the client connects. When the client connects, the connection and IP address of the client are stored. After this, the program receives data from the connection and stores it in a variable called “data”.

If there is any data received at this point, the data should contain the username and password sent by the client. The username and pasword will be in the format (“username”SPACE”Password”) so, to spit the two up, the received string stored in data on space was split, giving an array of 2 strings. One of the strings is the password and the other

is the username. These received usernames and passwords are then checked against the username and password entered when the server was started. This could result in an index error if the client sends only one string or no strings so there is a try and except to catch the error if it happens. If they do not match, or if the index error happens due to nothing or only one string being sent, the program prints out “Login Incorrect” and loops to wait for the next attempt.

If the username and password do match, the program moves on to close the connection it had opened and closes the socket made earlier. Once this has been done, thread one and thread two are both created and started. “thread1Stop” is also passed onto thread one so that it can use the variable to know when to stop. Thread one being the thread in charge of sending the webcam pictures and thread two being in charge on controlling the robot. After both are started, the main program hangs and waits here until thread two has finished and then proceeds to tell thread one to end by setting “thread1Stop”. The main program then waits for thread one to clean up and end fully before moving on to “unset” i.e. clear() the thread1Stop variable ready for the next time. The whole main program is in a big while loop so the program will simply loop back to the start and wait for another login.

5.2.2 Thread one – Sending images

```
def MyThread1(stopEvent):
    serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    serversocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    serversocket.bind(("",port))
    serversocket.listen(5)
    capture = cv2.VideoCapture(0)
    serversocket.settimeout(10)
    time.sleep(0.5)


    while (not stopEvent.is_set()):
            try:
                connection2, address = serversocket.accept()
            except:
                print("lost connection to client attempting to reconnect")
            frame = capture.read()[1]
            cnt = cv2.imencode('.jpg',frame,[int(cv2.IMWRITE_JPEG_QUALITY), 10])[1]
            data = base64.b64encode(cnt)
            try:
                    connection2.sendall(data)
            except:
                    serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                    serversocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
                    serversocket.bind(("",port))
                    serversocket.listen(1)
                    print("Lost connection")

            connection2.close()

    serversocket.close()
    return
```

Thread one starts by creating a new socket that uses port 5000. This port is also reusable as it may be reused later on again. The function then sets the capture device to the first webcam it finds plugged in. The socket’s timeout is then also set. The reason for this is in case the stopEvent is set by the main program in-between the while loop and the serversocket.accept(), if this happens the function will simply wait forever at serversocket.accept() because at this point the client will have logged off and so will no

longer be connected and the thread will never end, meaning the main program will hang because it is waiting for the thread to end. Giving the socket a timeout means that the thread will hang for a maximum of 10 seconds before ending if there has been a request for the thread to finish (stopEvent being set).

The function then waits for half a second as this gives the client time to catch up and be ready to start receiving the pictures. After this, the function checks if stopEvent has been set. If it has not been set, the function carries on and creates a connection with the client application. Next the opencv library is used to take a picture on the webcam and store it. This picture is then encoded as a jpeg and compressed so that it is a smaller size and quicker to send over the network. Afterwards, the jpeg is encoded as a base 64 string and sent down the port over the connection to the client.

If sending the data is successful, the function closes the connection and loops from the start. This means as many pictures as possible are sent down the connection to the client which is displaying the pictures as soon as it receives them. Depending on the internet connection this can supply up to 30fps video at the client side, as 30 pictures per second are sent to the client. If sending data down the connection fails, then the socket is recreated and re-bound to the 2500 port. Once stopEvent is set, the while loop will exit and the socket will be closed.

5.2.3 Thread two – controlling the robot

```
def MyThread2():
    data = "nothing"

    server2 = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    server2.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server2.bind(("",2500))
    server2.listen(5)
    server2.setblocking(True)


    gpio.setmode(gpio.BCM)
    pins = [17, 18, 22, 23]
    for pin in pins:
       gpio.setup(pin, gpio.OUT)
    client,address = server2.accept()
    print ("Connection from: ",address[0])
    while 1:
                data = client.recv(1024)

                if data:

                    if(data[0]=='u'):
                        gpio.output(22, 1)
                        gpio.output(18, 1)
                        time.sleep(1)
                        gpio.output(22, 0)
                        gpio.output(18, 0)

                    elif(data[0]=='l'):
                        gpio.output(18, 1)
                        gpio.output(23, 1)
                        time.sleep(1)
                        gpio.output(18, 0)
                        gpio.output(23, 0)

                    if(data[0]=='r'):
                        gpio.output(22, 1)
                        gpio.output(17, 1)
                        time.sleep(1)
                        gpio.output(22, 0)
                        gpio.output(17, 0)

                    if(data[0]=='d'):
                        gpio.output(17, 1)
                        gpio.output(23, 1)
                        time.sleep(1)
                        gpio.output(17, 0)
                        gpio.output(23, 0)
                    if(data[0]=='f'):
                        print("logout")
                        client.close()
                        server2.close()
                        return
                else:
                    print ("Lost connection to: ",address[0])
                    client,address = server2.accept()
                    print ("Connection from: ",address[0])
```

Thread 2 creates a new socket on port 2500. It makes this port reusable as it may be used again. The port is also set to blocking mode. After this the function sets the GPIO mode to BCM, which allows for the pins on the Raspberry Pi to be referred to by the “Broadcom SOC channel” number, then the GPIO pins are set to output mode. Once all this is done, a connection between the server and client is created.

The next thing the function does is receive the data it is being sent by the client and, if there is any data received, the function will check whether the string it has been sent is either “u” (up), “l”(left), “r”(right), “d”(down) or “finished”(logout). If the received string is one of the

letters the function will output on the correct pins to turn on the correct motor on the robot by supplying power to that motor for one second and then will stop output on the same pins which will stop the motor/motors being powered. For example, if a “u” is received this means the robot should move forward, so pins 22 and 18 are set to output, the thread then sleeps for one second, before the same two pins are set to no longer output. The function will continue to loop detecting each time what data has been received and performing the relevant action depending on the command sent in the data. If “f” is received then the function knows the client no longer needs control and so begins to clean up by closing the connection and port and then returning from the function. This will trigger the main program to continue running and to go and shutdown thread one (the webcam thread) and then go on to waiting for the next login.

If the function fails to receive any data then it knows that the client has disconnected without logging out so it prints that the user has lost connection and tries to make the connection again.

6 Testing and Evaluation

6.1 User testing

As there were only ten participants, further testing would need to take place in order for these results and the analysis to be representable. To find out what other people thought of the system a questionnaire was designed which they could fill in after trying it out. The participants followed verbal instructions on what to do. If they got stuck on a certain part they were shown how to go about the process and any questions the participants had about the system were answered throughout the testing. Below is a plan of how this interaction took place. For the questionnaires (Survey Monkey, 2016) was used. A copy of the questionnaire can be found with working documents on the website (listed in the declaration).

6.1.1 Methodology

1. The participant is given a mobile phone with the application open and is also given access to the started webcam server program running on the Raspberry Pi from an SSH terminal running on a laptop which is opened by the observer.
2. The participant is then asked to start the server.
3. Once the participant has done this, they are asked to connect using the mobile application. (The IP address of the webcam server is supplied).
4. Then the participant is asked to attempt to drive the system around.
5. They are then allowed to have free rein using the system and no instruction is provided at this stage.
6. The questionnaire is then sent for them to fill in.

6.2 User testing results

6.2.1 Question 1

How much do you agree with this statement: 'The system is easy to use'?

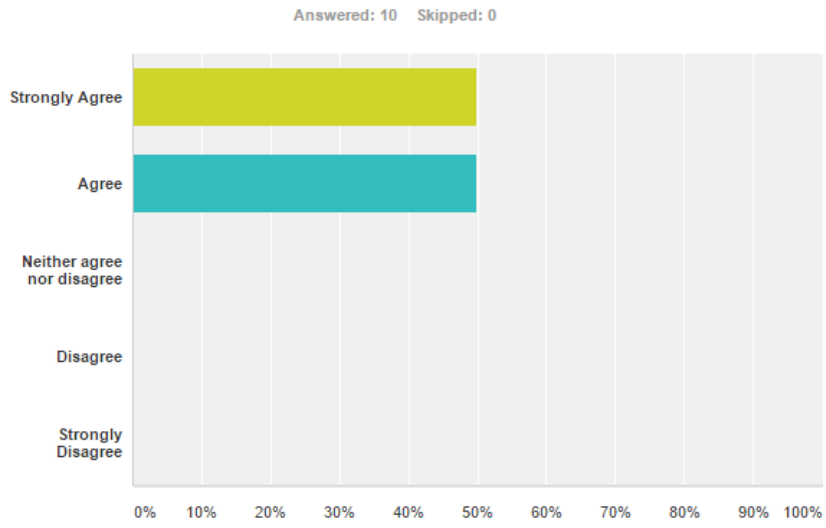


Figure 26

6.2.2 Question 2

Are you aware of any products similar to this one?

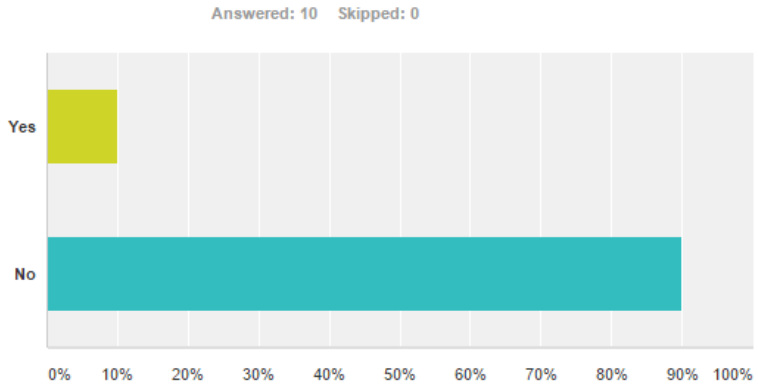


Figure 27

6.2.3 Question 3

Do you have any home security? If yes, please give details.

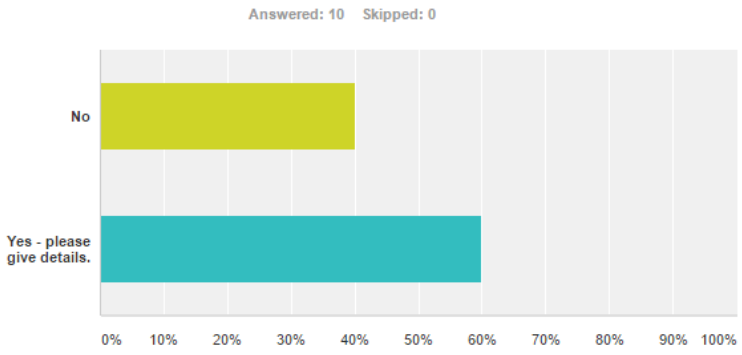


Figure 28

If participants said yes they listed systems they had. These included house alarms, CCTV cameras, infrared alarm systems and anti-burglar lighting.

6.2.4 Question 4

What were the positive aspects of the system? Please list them below.

For this question 40% of participants mentioned that the system was easy to use, even with little guidance. Participants also mentioned that the robot was fun to use, that they liked how you can control where the camera goes and how the robot could drive over obstacles on the floor.

6.2.5 Question 5

What improvements would you recommend for the system? Please list them below.

For this question participants mentioned limitations of the system, such as how it cannot get upstairs without being carried, that the robot was slow and how it may be difficult to use for computer-illiterate people. The improvements mentioned included making the camera height adjustable, making the webcam server's user interface nicer, making the robot more aesthetically pleasing and making a manual on how to use the robot.

6.2.6 Question 6

Would you use this system in your home? And why?

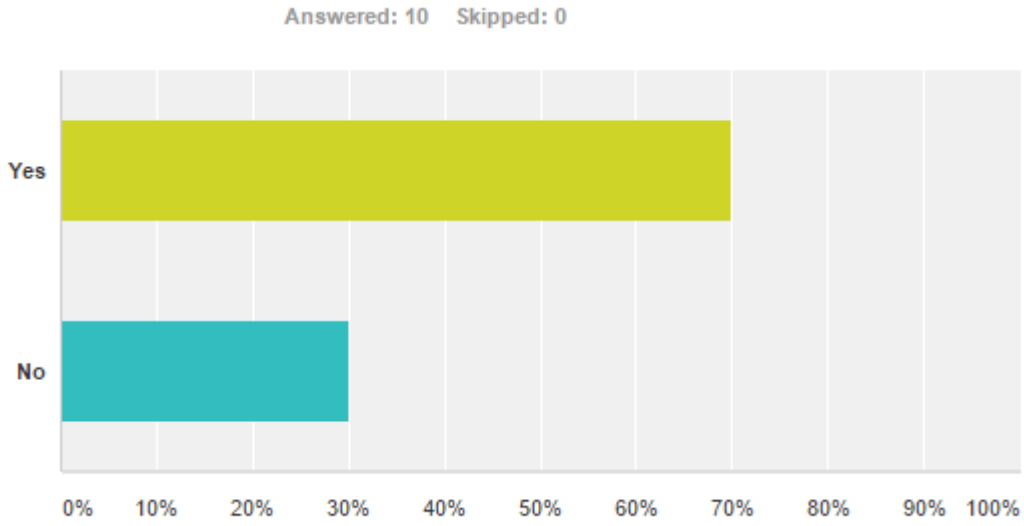


Figure 29

Participants who answered "yes" for this question stated: they would use the system at home because they could watch their pets and children, they would use it to watch for burglars and to check if they had left appliances in their house switched on.

Participants who answered “no” said they would not use this system because its movement is limited by stairs and closed doors or that it was too technical for them.

6.2.7 Question 7

Was the mobile application’s user interface intuitive? And why?

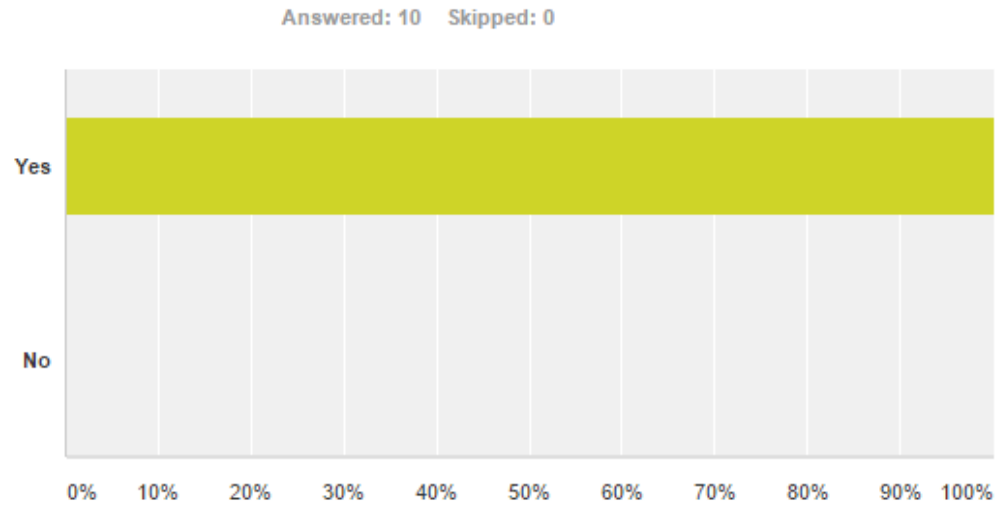


Figure 30

For this question everyone who took part in the study thought the interface was intuitive and liked it for reasons such as the interface being very simple and straightforward, that they found it very easy to use and that they would not need instructions to know how to use it.

6.2.8 Question 8

Was the webcam server’s user interface intuitive? And why?

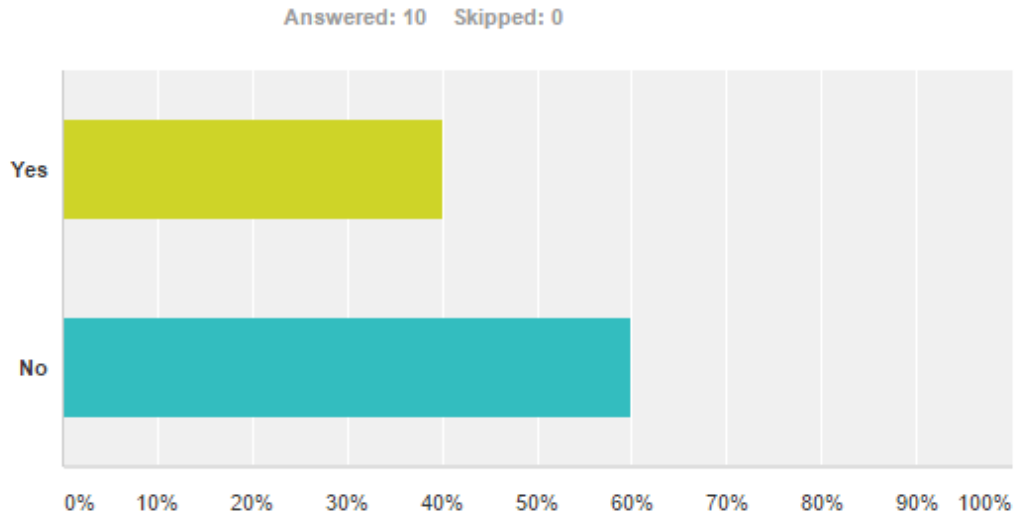


Figure 31

For this question the vote was split with 40% finding the server's user interface intuitive while 60% did not. Participants who answered "no" found the interface complicated, hard to use and not very user-friendly. Participants said it could be improved by using an interface which had buttons and instructions to go with it. Participants who answered "yes" said they found the interface easy to use.

6.2.9 Question 9

Was the webcam server's user interface aesthetically pleasing? And why?

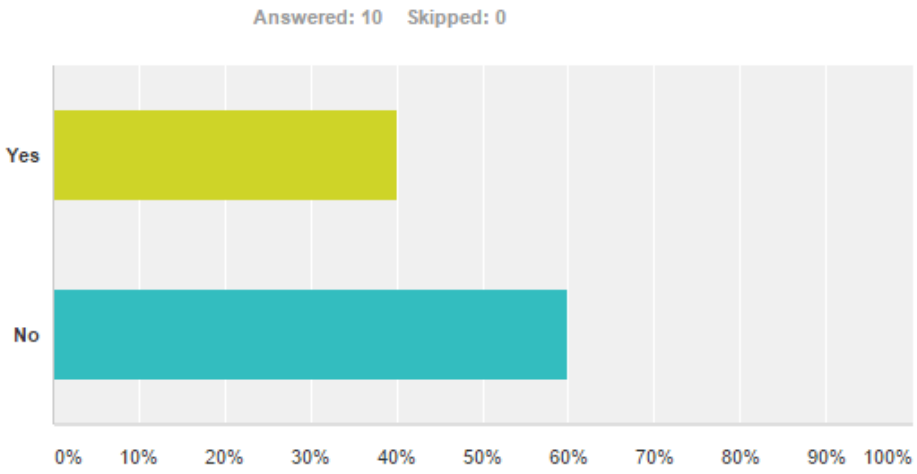


Figure 32

For this question again the vote was split with 40% finding the server's user interface aesthetically pleasing while 60% did not. Participants who answered "no" said it looked complicated and did not look as nice as the mobile interface. Participants who answered "yes" said it looked basic however the system only needed a basic webcam server interface.

6.2.10 Question 10

Was the mobile application's user interface aesthetically pleasing? And why?

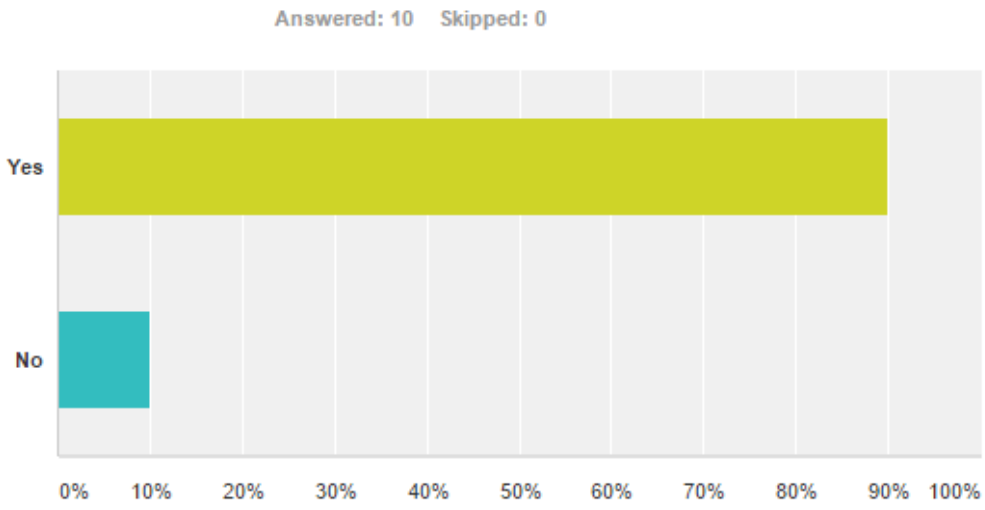


Figure 33

For this question nearly everyone agreed that the mobile application's user interface was aesthetically pleasing; people said that it was simple, clean and had a good colour scheme.

6.3 User testing analysis

Generally, feedback on the system was very good with participants saying that it was simple and easy to use and that they enjoyed using it. Participants particularly liked the client mobile application because of the easy-to-understand buttons. Many also said that they would use the system in their home and not only for security but also for keeping watch over their pets. This very important as it suggests that the system will target a larger group of people and not just those wanting more home security.

The only area where was felt that people were unhappy was the webcam server's interface. Participants did not like the way it looked and found it complicated to use. From observations, it was also realised that people would not have known how to use the webcam server's interface without instruction. The main problem with this interface is that it is a command line interface and not a graphical one and most people are not used to using command line interfaces. One improvement participants suggested was an instruction booklet for those users who are less computer-literate, especially for the webcam server interface. Another problem discovered mainly from observations (though some people also mentioned this in the questionnaire comments) is that when entering the username and password of the webcam server interface, the username and password need to have quotation marks around them. This is because the webcam server was run in the older Python 2 and could be fixed by running the program in Python 3.

Observations made during the testing generally matched what people answered on the questionnaire for example, it was found that everyone found the client application very easy to use while people had a lot more trouble using the webcam server.

6.4 Technical testing

A series of brief (not extensive) tests were thought of to put the system to the limit and try and make the system crash. There were also some tests that were done in order to test the general usability of the system. If the system were to be further developed for commercial use, much more extensive testing would be required.

Test Number	**The Test Being Performed**	**Area of the System being Tested**
1	Entering the wrong IP Address of the server.	Client
2	Entering the wrong username.	Client
3	Entering the wrong password.	Client
4	Using numbers, special characters and spaces when making a username.	Server
5	Using numbers, special characters and spaces when making a password.	Server
6	Shutting the Server while the client is connected	Client
7	Pressing the directional keys very fast and lots of times	Server
8	Battery life	Server
9	Closing the application on the client and not logging off	Server
10	Does the robot respond to directional buttons being pressed on the app?	Client
11	Does logout work?	Both the client and the server
12	Does the video display?	Client

6.5 Technical testing results

Test Number	**Result**	**Solution (if needed)**
1	An error message is saying the IP address could be wrong.	No solution required – the user is made aware of what is wrong

2	An error message is shown saying the username or password could be wrong.	No solution required – the user is made aware of what is wrong
3	An error message is shown saying the username or password could be wrong.	No solution required – the user is made aware of what is wrong
4	Using numbers and special characters work fine however using spaces does not work as the server splits the username and password sent from the client on a space so the server will recognise the part of the username after the space as the password	Rather than using a space to split the username and password, send them separately rather than together
5	Using numbers and special characters work fine however using spaces does not work as the server splits the username and password sent from the client on a space so the server will only read the first part of the password (before the space) and so think the password is incorrect when it may be correct.	Rather than using a space to split the username and password, send them separately rather than together
6	An error message is shown saying “problem getting image”	No solution required – the user is made aware of what is wrong
7	When pressed three or more time in quick succession the robot will only perform the first two commands	No solution required – a user shouldn’t need to queue commands. It also provides a failsafe if the user has pressed the button too many times by mistake
8	7-9 hours depending on usage	A bigger battery could be used to make the system last longer.
9	The server says “Lost connection to” followed by the IP address of the client	No solution required – the user is made aware of what is wrong
10	The robot performs the requested action for one second depending on which direction button was pressed	No solution required – works as expected
11	The app goes back to the first screen and the server	No solution required – works as expected

	goes back to waiting for the next login.	
12	The video displays on the client	No solution required – works as expected

The technical testing was generally very successful. Error messages were always shown when something went wrong. The only unsuccessful tests were tests 4 and 5, whereby a user could not have spaces in their username or password as this would not allow them to log in from the app due to the way the system was programmed.

7 Conclusions

7.1 Review of aims

The aim of the project was to build an IP camera based home security system that worked over WiFi, that could also be accessed and controlled from a smartphone and would be comparable to commercial applications. The system would be something that was comparable to available products but something that also had some unique features, such as the ability to move around, as the stationary nature of most household security is something that is often emphasised in the literature. The system should also be at a price that both a renting tenant and a home-owner could afford.

It was felt that the project met all of these aims. Firstly the system does work over WiFi, the client and server both connect to a wireless router and communicate with each other wirelessly through it. Next, the webcam server can be accessed and controlled from the smartphone's client application; the webcam feed can be viewed; and the webcam can be driven around directly from the application. It was also felt that the system can be compared favourably to commercial products seen in the literature. It may not have some features which other products do, for example motion detection. While motion detection could potentially be implemented into this system, the robot would have to be stationary in order for the motion detection to work, which contradicts the aims as the product is unique in its ability to move around and this, in turn, is what it is offering to the market.

7.2 Improvements and Further work

If the project were to be extended, some additions and improvements would be looked into. One improvement – prompted by user feedback - would be to create a graphical user interface for the webcam server since, as explained above, people did not like the command line interface. Another improvement would be running the webcam server in Python 3 to solve the quotation marks problem when entering the username and password (again explained above).

Security is also an area which could be improved. Currently the video being streamed to the application and the commands sent to the robot are not encrypted, if encryption was added it would make the system less vulnerable to a hacker trying to intercept messages between the client and server.

If the product were to be made commercially, it would be prudent to improve the appearance of the robot for example making a casing for the wiring and mounting the camera within the housing of the case.

There are also more features that could be added to the system. One particular feature which was requested in the questionnaire is the ability for the robot to move up and down stairs. This would not be an easy feature to add as the robot would need to be changed completely - perhaps to one with legs, and as was cited in Chapter 3.1, there has already been such a design and the study still found stairs a problem.

A feature which would not be too difficult to implement would be two-way sound, where the system would record sounds picked up and stream the audio to the application along with the video. Speakers would then also be attached to the robot so that a person using the

application could talk out of the speakers on the robot. This would allow users either to alert trespassers or listen to what trespassers were saying. If users were using the system with their pets, such a feature would add a personal touch as the user would be able to speak to, and hear, their pets. Another feature which would not be difficult to add to the system would be recording of the video stream, this would allow videos of burglars to be used as police evidence.

A feature which would overcome the problem of needing to charge and change the batteries would be to add a solar panel to the robot and then powering everything using solar power. This would work well if the robot was in an area which was very light, however could cause problems if it were meant to work in dark conditions and people normally turn lights off when they leave the house.

7.3 Lessons Learned

To do this project I had to learn about many things. One of the things I had to learn was how to program for Android. I had programmed in Java many times before so this was not too difficult, however I needed to come to terms with how to program in the xml user interface as the only interface library I had used before was the Java swing library. I also needed to learn how to make a good interface by reading about certain aspects of interfaces which are simple and easy to use along with thinking about my experience of using interfaces and what features I had found which made an interface good or poor.

Another thing I had to learn about was how permissions in Android work and which permissions I would need in order for my application to work; and also how to go about defining the permissions I needed, in the manifest file. Something else that I had to learn was how to go about soldering different parts of a microchip together. This was due to the motor control chip arriving unsoldered. I also had to look into how the Raspberry Pi's GPIO output pins work so that I could pick a suitable motor control chip.

Along with needing to learn many things to get started on my project, I also learnt many things from making and programming it. One example of this is using ports to communicate between two systems running programs on different languages (one system running Python and one system running Java). I also expanded my knowledge on threads in Java and Python, and how to shut threads down cleanly - when you have a looping thread - by using interrupts.

From user feedback, I also learned that it is very easy as a developer to design and build a system which the developer personally thinks will be very easy to use for everybody, but which turns out not to be as clear as originally thought. For example, some of the participants were not as computer-literate as I am, and did not understand what some of the terms I used meant. This meant that I had to explain these terms. This might indicate that a simple instruction booklet would be needed.

In terms of doing things differently to improve my time management I could have made a Gantt chart at the start of the project.

I really enjoyed designing and making this system. The system could be used to genuinely help people by improving security in their homes to give them peace of mind. I also feel my system could have an impact on people who are less interested in security, but rather in using the system as a way to observe their pets when they are away from home or to use it simply

for children as a toy to play with - which they could perhaps build or take apart to learn more about electronics and computing. Overall the project went very well and I had very few problems. User feedback on the system was generally very good and it did what it was designed to do.

8 Appendix

8.1 Questionnaire

Final Year Project

System Questionnaire

*** 1. How much do you agree with this statement: The system is easy to use.**

- ◯ Strongly Agree
- ◯ Agree
- ◯ Neither agree nor disagree
- ◯ Disagree
- ◯ Strongly Disagree

*** 2. Are you aware of any products similar to this one?**

- ◯ Yes
- ◯ No

*** 3. Do you have any home security? If yes please give details.**

- ◯ No
- ◯ Yes - please give details.

*** 4. What were the positive aspects of the system? Please list them below.**

*** 5. What improvements would you recommend for the system? Please list them below.**

* 6. Would you use this system in your home? And why?

- ( ) Yes
- ( ) No

Why?

* 7. Was the mobile applications user interface intuitive? And why?

- ( ) Yes
- ( ) No

Why?

* 8. Was the webcam servers user interface intuitive? And why?

- ( ) Yes
- ( ) No

Why?

* 9. Was the webcam servers user interface aesthetically pleasing? And why?

- ( ) Yes
- ( ) No

Why?

* 10. Was the mobile applications user interface aesthetically pleasing? And why?

- ( ) Yes
- ( ) No

Why?

Done

8.2 Consent form

Consent Form

Study Title: IP camera based home security system questionnaire

I am asking if you would like to take part in a research project whereby I have created a home security camera and would like you to test the product and answer a questionnaire based on your experience.
Before you consent to participating in the study I ask that you read the participant information sheet and mark each box below with your initials if you agree. If you have any questions or queries before signing the consent form please speak to the principal investigator, Maximilian Murach-Ward

	Please initial each statement
1. I confirm that I have read the information sheet and fully understand what is expected of me within this study	☐
2. I confirm that I have had the opportunity to ask any questions and to have them answered.	☐
3. I understand that the questionnaire will be kept until the research project has been examined.	☐
4. I understand that my participation is voluntary and that I am free to withdraw at any time without giving any reason, without my medical care or legal rights being affected.	☐
5. I understand that once my data have been anonymised and incorporated into themes it might not be possible for it to be withdrawn, though every attempt will be made to extract my data, up to the point of publication.	☐
6. I understand that the information from my questionnaire will be pooled with other participants' responses, anonymised and may be published	☐
7. I consent to information from my questionnaire being used in reports, conferences and training events.	☐
8. I understand that any information I give will remain strictly confidential and anonymous unless it is thought that there is a risk of harm to myself or others, in which case the principal investigator will/may need to share this information with his/her research supervisor.	☐
9. I give consent to be observed and for that observation to be documented in the study.	☐
10. I consent to take part in the above study.	☐

Name of Participant_____ **Signature**_____ **Date** _____

Name of Researcher _____**Signature** _____**Date** _____

9 References

Andrews, C., 2013. Easy as Pi. *Engineering and Technology,* pp. 34-37.

Android, 2016. *AsyncTask.* [Online]
Available at: http://developer.android.com/reference/android/os/AsyncTask.html
[Accessed 18 February 2016].

Anon., n.d. *http://pygame.org/download.shtml.* [Online].

Barker, R., 2009. *Robotic Video Security Systems. TECH-STYLING.* [Online]
Available at: <http://www.tech-styling.com/robotic-video-security-systems/>
[Accessed 25 January 2016].

Bellis, M., 2016. *The History of Video and Related Innovations.* [Online]
Available at: <http://inventors.about.com/library/inventors/blvideo.htm>
[Accessed 02 February 2016].

Berthoud, P., 2013. *A Forgotten Invention – World's First Security Camera.* [Online]
Available at: <http://www.peterberthoud.co.uk/2013/12/worlds-first-security-camera-london/>
[Accessed 20 January 2016].

Boonma, N. S. A. M. S. a. V. C., 2011. Image Recorder Server with IP Camera and Pocket PC. *Procedia Engineering,* Volume 8, pp. 182-185.

Butler, M., 2011. Smart phones: android: changing the mobile landscape. *Pervasive Computing,* pp. 4-7.

Calin, D. G., 2014. *3 Possible Ways for Real-Time Video Streaming Between Camera and Android Device. intorobotics.* [Online]
Available at: <http://www.intorobotics.com/3-possible-ways-real-time-video-streaming-camera-android-device/>
[Accessed 23 January 2016].

Delgado, R., 2015. *From Edison to Internet: A History of Video Surveillance.* [Online]
Available at: <https://www.aabacosmallbusiness.com/advisor/edison-internet-history-video-surveillance-105000047.html>
[Accessed 22 January 2016].

Fallon, S., 2007. *Gizmodo.* [Online]
Available at: http://gizmodo.com/339128/rc-spy-snooper-robot-is-undetectablelike-a-fat-clumsy-ninja
[Accessed 13 March 2016].

Harris, M., 2010. Safe as Houses. *theiet.org,* pp. 23-27.

Javale, D., Mohsin, M., Nandanwar, S. & Shingate, M., 2013. Home Automation and Security System Using Android ADK. *International Journal of Electronics Communication and Computer Technology,* 3(2), pp. 382-385.

Kelly, K., 2015. *Marie Van Brittan Brown: Inventor of the Home Security System.* [Online]
Available at: <https://originalwoman13.wordpress.com/2015/10/18/marie-van-brittan-brown-inventor-of-the-home-security-system/>
[Accessed 01 February 2016].

Luo, R., Hsu, T., Lin, T. & Kuo, S., 2005. The Development of Intelligent Home Security Robot. *Proceedings of the 2005 IEEE International Conference on Mechatronics,* pp. 422-427.

Maplin, 2015. *I.P. CCTV Cameras.* [Online]
Available at: http://www.maplin.co.uk/c/cctv-and-security/ip-cctv/ip-cctv-cameras?sort=%3DMaplinProduct.price%7C1#
[Accessed 13 March 2016].

Pi, R., n.d. *Raspberry Pi learning resources: robot butler.* [Online]
Available at: https://www.raspberrypi.org/learning/robo-butler/requirements/
[Accessed 11 March 2016].

Potts, J. & Sukittanon, S., 2012. Exploiting Bluetooth on Android Mobile Devices. *Proceedings of IEEE Southeastcon,* pp. 1-4.

pygame, n.d. *Downloads.* [Online]
Available at: http://pygame.org/download.shtml
[Accessed December 2015].

Rajadurai, S., Nehru, P. & Selvarasu, R., 2015. Android Mobile Based Home Security and Device. *International Conference on Innovations in Information, Embedded and Communication Systems,* pp. 1-5.

Schneier, B., 2003. *Beyond Fear: Thinking Sensibly About Security in an Uncertain World.* New York: Copernicus Books.

Sixsmith, A., 2000. An evaluation of an intelligent home monitoring system. *Journal of Telemedicine and Telecare,* 6(2), pp. 63-72.

Smith, A., 2015. *U.S. Smartphone use in 2015.* [Online]
Available at: http://www.pewinternet.org/2015/04/01/us-smartphone-use-in-2015/
[Accessed 13 March 2016].

Song, G., Tin, K., Zhou, Y. & Cheng, X., 2009. A Surveillance Robot with Hopping Capabilities for Home Security. *IEEE Transactions on Consumer Electronics,* 55(4), pp. 2034-2039.

Survey Monkey, 2016. *Survey Monkey Survey Platform.* [Online]
Available at: https://www.surveymonkey.co.uk/
[Accessed 10 January 2016].

Tangtisanon, P., 2014. Android-based Surveillance Car. *TENCON,* pp. 1-4.

Tudge, R., 2010. *The No-Nonsense Guide to Global Surveillance.* 1st ed. Oxford: New Internationalist.

Zeman, E., 2015. *iPhone Vs. Android: Apple's Success Only Goes So Far.* [Online]
Available at: http://www.informationweek.com/mobile/mobile-devices/iphone-vs-android-apples-success-only-goes-so-far/a/d-id/1321437
[Accessed 10 February 2016].

10 Acknowledgments

I would like to thank Dr Keivan Navaie, my tutor for his help and support throughout the project, along with the Computer Science department. I would also like to thank my parents, friends and Hannah for proof-reading and support. Finally, I would like to extend my gratitude to the participants in the testing stage.