Final Year Project Report

Evaluation of Drivers Ability based on Image Processing

Student - Keith Reilly

Supervisor - Ciaran Cawley

Second Reader – Denis Manley

Student Number – C12410532

DT211/4 – Computer Science Infrastructure

Date Due – 11/04/2016

Abstract

In this day and age, when google cars and tesla's autopilot features are becoming more mainstream, combined with the huge advances in machine learning and computer vision. It has become more important for us to understand and learn the fundamental and basic steps of computer vision.

The purpose of this project is to identify a method of image processing that can then be used to develop a software tool. The tool itself will be used to work out the distance that certain objects are away from it using stereo vision.

This tool will also allow the user to calibrate the cameras and take both pictures and videos. The hardware used in this project will be low-end but the steps and methods used should be easily transferable and usable with most equipment.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

_____

<Student Name>

09/04/2016

_____

<Date>

Acknowledgements

I would like to first acknowledge my supervisor, Ciaran Cawley. I am grateful to Ciaran for given me this opportunity to work on this assignment under his guidance and support.

I would also like to thank my family for putting up with me during this time, and also supporting me and motivating me till the very end.

Table of Contents

Table of Figures

1) Project Statement

This project is a software tool that can analysis video footage frame by frame. This footage will be of the user's drive, similar to that of a dash cam. This video footage will be attained from two cameras. Using functions from OpenCV I will extrapolate data that will allow me to judge the distance from other objects. I will also be tracking these objects i.e. cars on the road. This software tool will then use the information it gathers to judge the driver's performance based on rules I create. For example, the distance they keep between themselves and other vehicles. This tool will then display the results to the driver along with a grade. If the driver has used this tool before then they will be able to compare their new score to their last.

1.1. Document Guide

Chapter 2 Research:

This chapter will go into detail about the research done into existing solutions along with technologies that could be utilized to help facilitate the building of this project. It will also deal with the hardware that will be required to achieve the goals and requirements set out for this project.

Chapter 3 Design:

This chapter will go into detail about different methodologies. It will then give a explanation for the methodology that was chosen to complete this project. It will focus on the design decisions for this project related to both software and hardware. It includes case-study's and flowcharts to give an idea of the way the code will flow.

Chapter 4 Early Prototyping and Development:

This chapter will go over the early development of the project, it is split into different iterations. Each iteration will deal with one or two specific problems.

Chapter 5 Implementation and development:

This Chapter will explain and give examples of the different features that were being created. It is split into different sections and each feature gets its own section. This chapter also goes in detail about the different issues and risks that were encountered while implementing code, and setting up hardware.

Chapter 6 System Validation:

This chapter will deal with testing the software and hardware. There are examples of test cases along with links to videos to early prototypes.

Chapter 7 Project Evaluation and Conclusion:

This chapter will go into detail about possible future work, it will focus on what this project can become with more work. It will also give explanations on features that were not added in the end product along with a reason. This chapter will give the authors general view of the project. It will touch on the things he has learned along with a brief explanation of their experience creating this project.

1.2. Project Requirements

At the end of this project I hope to have implemented these feature mentioned below.

- Determine depth of object in image
- Setup low end cameras and hardware to record and take pictures
- Setup network to allow user to start cameras recordings
- Learn and developer a better understanding of computer vision

1.3. Objectives

The objectives below are going to be researched and completed in order to complete the above requirements.

- I want to start and develop new skills in C++
- Calibrate cameras for more accurate results
- Create a GUI in C++ to allow user interaction
- Give user feedback on driving ability (Score / Grade)

1.4. Project Challenges

These are the challenges that I will face while trying to complete the above requirements.

- The scope of this project is quite large
- There are a lot of complex challenges that need to be overcome
- I personally have very limited experience with computer vision
- Using hardware, if anything breaks it will cause large time delays

1.5. Motivation

I came up with the idea to this project while I was practicing driving with my dad. I had just finished a lesion and I was going over the different techniques I was shown. I was arguing with my dad about the distance I was from the car in front of me. Then when it came time for me to come up with a project for final year. I decided that I would try to create a device that I would be able to place on the dashboard of a car. I would then be able to go home and analysis the video of the drive and see if I ever came to close to another vehicle or cyclist.

2. Research

2.1. Introduction

This chapter will deal with the multiple technologies out there that are already in use which met some of the requirements that are found in this project. There are many motion tracking and distance detection technologies used in sports at the moment, and also in other projects like Googles driverless car. They aren't however many examples of these technologies being used to give a user feedback on their own driving ability. This is one of the main reasons why this project will stand out among these other technologies even though it is similar from a function standpoint. The main problems that will have to be overcome are distance detection and motion tracking. The results that this project produce must be semi-accurate when estimating the distance that the user's car is from other vehicles. It also has to be able to track different vehicles on the road. One problem while tracking these objects is whether or not they are vehicles, cyclists or just pedestrians.

2.2. Alternative existing solutions

Autonomous cars, or self-driving cars as there are more well known, have started to become more mainstream during the last five years. These cars have to be able to sense their surroundings and also avoid obstacles such as vehicles, cyclists and pedestrians. Google has been paving the way for a fully self-driving car, while other companies like Tesla have been aiming for a car that allows the user to switch to auto-pilot. In either scenario these companies will need to overcome a multitude of problems. The main one being, how are they going to get the car to recognize its surroundings and then make rational decisions based on its results.

2.2.1. Googles self-driving car

There are a number of ways to do this. Googles driverless cars uses a LiDAR system. LiDAR is an acronym short for light detection and ranging. LiDAR is similar to RADAR, but where RADAR sends out radio waves, LiDAR sends out light waves. "It's a powerful data collection system that provides 3-D information for an area of interest or a project area "[1] Using these results, the engineers at google can create a good visualization of all the objects within a few meters to the car. They are then able to program the car to respond to the different scenarios that it will more than likely face. All this equipment to keep googles driverless car on the road is very expensive, and the LiDAR alone costs $75000.

Figure 1- Googles Driverless car [19]

2.2.2. Tesla's Autopilot

Tesla's autopilot works using four different systems. The first is RADAR that is coming out the front of the car. This can scan the cars ahead of the vehicle and is able to scan up to long distances. Using RADAR means that it can see through rain, snow, fog and many other things that would be hindrance on an average drivers journey. The second system in place is a camera that is also facing out towards the front of the car. This camera is used to read traffic lights, read road signs and also see the lanes on the road. It can also tell the difference between other vehicles, cyclists and pedestrians. The third system is a 360-degree ultra-sonic sonar. This has a long range and can detect objects beside the car all the way from other vehicles to animals like dogs or cats. The fourth and last thing that the car uses for autopilot is a GPS system. This will allow the car to know the area it is in, which will allow it to adjust to roadworks and other issues that may arise. [13]

2.2.3. Hawkeye

Computer vision is heavily implemented in sports today. Years ago we did not have the technology to return reliable and accurate details. Over the last number of years' computer vision has made its way into a number of sports like tennis, football, rugby and Gaelic football. One of the main technologies used today is Hawk-Eye. For example, in a Gaelic match where the Hawk-eye system is in place there are eight high speed cameras set up to watch the pitch. The ball position is triangulated using 4 different cameras covering each end of the pitch. If a player shoots for a point during the game and his shot is ruled a miss. He is able to challenge this and then a computer generated replay will be show. This is done using the results gathered from the cameras tracking the ball.

2.2.4. Kinect

Back in 2010 Microsoft released the Kinect. The Kinect is an external piece of hardware that can be used with the Xbox 360. Its goal was to track the players' movements and allow them to control games without using a controller. Players also had the option to use certain voice commands to control some games. The Xbox uses a few different systems to accomplish this but the main one this project will focus on is its sensing hardware. "The depth sensor consists of the IR projector combined with the IR camera which is a monochrome complementary metal-oxide semiconductor" [2] So how does it work? Let's say you set up a Kinect in your room, the IR projector will shoot out tiny dots onto the room that are invisible to the human eye. The Kinect then uses these dots that are hitting objects in the room to triangulate distances. Using this information, the Kinect is then able to get a good understanding of the room it is in and will be able to detect any movement.

2.3. Technologies researched

2.3.1. Open CV

OpenCV is an open source computer vision library. It was written in C and C++, these are not the only two languages that are available though. There is an active development on interfaces for Python, Matlab and many other languages. The OpenCV library contains a lot of functions. There is "over 500 functions that span many areas in vision, including factory product inspection, medical imaging, security, user interface, camera calibration, stereo vision and robotics". [3] The main function from that list, will be used heavily in this project is stereo vision. Stereo vision is a way of inferring depth from two or more cameras. So using two cameras which are facing the same way on the dash of a car, one could get the distance of objects in the view by using triangulation. "Stereoscopic pictures allow us to calculate the distance between the camera(s) and the chosen object within the picture" [4]

OpenCV also provides a lot of other algorithms that allows users to track objects such as cars, bikes or people walking on the road. Functions like cv2.calcOpticalFlowPyrLK() which allows you to track feature points in a video or cv2.absdiff() which gives the user the option to subtract two images. There are some drawbacks to using OpenCV as it limits what video type can be used to capture the users drive. On the OpenCV site is shows that it supports a number of different image types, such as .bmp, .jpeg, .tiff, .sr and. pbm. However, it only gives one example for video format and that is of type .avi. This does not mean it is impossible to create the tool, it does however limit my options when it comes to getting video footage which may be problematic in the future. [9]

2.3.2. MATLAB

MATLAB stands for matrix laboratory, and it is a high level language that is used for technical computing. MATLAB is not free, right now on their website there is a student deal that would allow one to purchase it for sixty-nine euro [14], that also includes 10 packages. The main package that would be needed for this project is the Image Processing and Computer Vision package. MATLAB has a few resources online, things like sample code and videos are easy to come by, most of which can be found on their website. MATLAB requires at least 1GB of free space, which isn't taken into account the packages that might be needed to be installed later on.

Because MATLAB is a programming environment users can type anything they want in the editor and only execute a section of it. Users can also enter in command line arguments to test smaller lines of code. MATLAB is built on Java so this means that when you enter these command line arguments or run your script you do not have to worry about memory management. This means that the user does not have to worry about memory leaks while running their code. However, they will suffer a slightly longer waiting period as MATLAB goes through their code and then executes it.

2.3.3. Raspberry Pi

The Raspberry Pi seems very much suited for a project like this as it is small and portable with a decent amount of computing power. The Raspberry Pi Model B+ looked like a good choice. It has four USB ports which means even if you were to connect two webcams to it, it will still have two free ports for a memory stick along with a Wi-Fi adapter. It also has lower power consumption, neater form factor and a micro SD instead of the SD found on older models [6]. Right now on amazon you can pick up the latest model for around thirty-five to forty euro. After reading through some of the Raspberry Pi forums to see what other users thought of it. The overall reaction is good, along with a lot users praising the design which has ironed out a lot of kinks from the last version. The Raspberry Pi that that is available on amazon comes with the operating system NOOBS installed.

2.3.4. Cameras

This project will require two cameras, or a specialized stereo vision camera. There are a few special cameras out that are designed with stereo vision in mind, but these are fairly expensive. A cheap stereo vision camera that a lot of people might know of is the Kinect [15]. Using a Kinect with the Raspberry Pi wouldn't be too difficult with all the guides online. It would however be difficult to power the Kinect along with the Raspberry Pi while driving a car. Apart from the Kinect there are many other cameras that offer stereo vision, but a lot of them were intended for small businesses or large research groups. The extra software that is installed on them could also make it difficult for me to use them with the raspberry pi. There were a few occasions where Logitech webcams popped up in forms where people were using them for their own projects, so after doing some research into the brand. The two Logitech webcams that were found which would be suitable for my project where the c270 and the c310. The c270 can capture photos at 3mp while the c310 can capture photos at 5mp. Both of these are affordable and they don't come with a lot of pre-installed software.

Figure 2- Kinect [15]

2.3.5. Stereo Vision

This is the gathering of 3D information from two or more images. By comparing two images that have been taken together but at slightly different angles, you can work out the depth of all the objects in the image. You would find stereo vision in a lot of automated systems, like robotics.

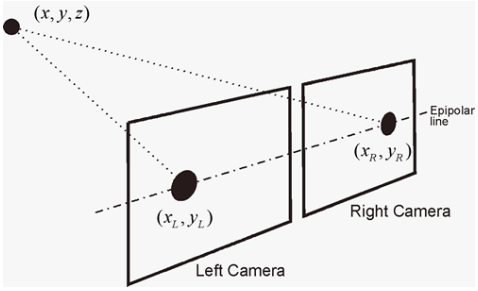


Figure 3- Stereo Vision Example [33]

2.3.6. Ultra-sonic

Apart from stereo vision there are other ways to detect the distance of vehicles on the road, and this is through ultra-sonic. However, within the first few minutes of research that appeared to be many problems with using this method. It only has a range of 500cm and its results are only accurate if objects are straight in front of it. This means it will not get the distance to any object which are at an angle to it, and if it does get a results it will more than likely be inaccurate. That isn't to say that this module has no value. It would be useful if placed on the front and back bumpers of the car. As you would be able to use it to assist the driver while parking. It might also be possible to use it to detect cars that in your blind spot if they come too close to the car. Although it would not be of much use when detecting other vehicles on the road due to the range.

Figure 4- Ultra-sonic module [16]

2.3.7. Apache

This is a cross-platform webserver application. It allows the system it is on to serve webpages. To install it, you will require 50 MB of disk space of which is will occupy only 10 MB. "… need at least 50 MB of temporary free disk space available." [5] Apache is used in more than 50% of all websites. It is also open source software and has a large online community updating it. With this installed on a Raspberry Pi, it would allow the user to display the contents of the Raspberry Pi through a local network. This would be useful in this project as the user will need a way of turning on the cameras while they in their car. In both Googles and Tesla's car this is not an issue as it is done automatically on the engine start.

2.3.8. Programming Languages (Python and C++)

One of most important aspects of this project that will have the biggest effect in the long run is the language it is coded in. This project will need to be written in a language that is easy to use, portable and also one that has access to large array of libraries. The language that is chosen will need to meet these requirements. Is it accessible? Does it support OpenCV, MATLAB or are similar libraries easily available? Can it be used to create a simplistic GUI?

Python is a very accessible programming language. It is easy to learn and there are many guides online for any questions that you would have. It will run on most systems including the Raspberry Pi (Raspbian operating system) and also supports a wide range of different libraries. With OpenCV, python will be a little bit slower doing computations than other languages like C++. "Compared to other languages like C/C++, Python is slower." [10] OpenCV was written in C/C++ so it makes sense that is optimized for those languages. As well as that, Python and libraries like Numpy would take up more space than C++ would on the Raspberry Pi, this would be an issue in this project as it needs as much free memory as possible to store video

recordings. It may also be more difficult to install OpenCV and get all the necessary libraries for python then it would be using C++.

Both languages provide different ways of creating GUIs. This will be important for this project as it will be displaying the results to the user on their driving ability. This interface should be simplistic, intuitive and easy to read. Python provides users with the Tkinter module. "Tkinter provides Python applications with an easy-to-program user interface" [12] This module is easy to use and provides a number for features that allow the user to create simple to complex menus. C++ also has useful libraries for the user, wxWidgets, QT and GTKMM. These provide just as many options as python does. However, they are slightly more difficult to use then Tkinter.

2.3.9. Other relevant research done

I created a questionnaire online. Link to survey [8]. I created this as I wanted to get an idea of what other people thought was important when it came to driving. There are nine questions in the survey and they are structured around finding out what distance drivers would keep from other vehicles and how they would react in certain circumstances. The rules that will be created for the driver's evaluation will be based off the rules in the RSA guidebook, but in the meantime, it would be good to know what the average driver thinks. This way I will be able to tweak my rules to find a good balance between being a great driver and an average driver. Although I will have to be realistic here as most of the people who answer this survey will be my friends or people who know me. This means I am only getting answers from a certain age group and it by no means represents what the actually overall population might think.

I have been looking over the Rules of the Road document published by the Road Safety Authority (RSA). The two sections that I have been looking over are sections 5 Good driving practice and section 6 Traffic signs and road markings. I have found this useful as it has given me actually figures to work with when it comes to distance, speed and the driver's reaction time. On page 116 of the document it shows you speeds from 30 up to 120 km/h and the Minimum Reaction Distance, Minimum Breaking distance and the Total Minimum Stopping Distance in meters beside each one of these speeds. [11] This is very useful to me as it will allow me to set a standard in my own project on which the driver is evaluated.

Speed (km/h)	Minimum Reaction Distance (m)	Minimum Braking Distance (m)	Total Minimum Stopping Distance (m)
30	6	6	12
40	8	10	18
50	10	15	25
60	12	21	33
80	16	36	52
100	20	50	70
120	24	78	102

Figure 5- RSA Guide Lines [11]

I sent an email to Dr. Derek Molly [7] from DCU as I found a video of his on YouTube channel. He was working with OpenCV and gave a brief demo on it for edge detection. I was curious to

get his opinion on my project and also see if he had any experience he could share with me about using OpenCV. In his email back to me, he confirmed that my idea was possible and that OpenCV would be a good choice. “There are good calibration tools for stereo vision available in OpenCV so it is a good choice.” [7] His personal advice was to not process video footage but individual frames. He also told me that if I wanted to do it in real time than I would have to use a simple image features and that it may be a bit too difficult. “My advice is that you process individual image frames, not video and that if you want real-time performance then you will have to use very simple image features. Real-time performance will be difficult to achieve.” [7]

2.4. Resultant findings

2.4.1. OpenCV vs MATLAB

After reviewing the information gathered in the technologies researched portion of the project, it seems that OpenCV will be the most ideal technology to use to complete this project in. This decision was influenced by a few different factors. The first one being the memory issue that would occur if MATLAB was chosen. It requires at least 1GB of space and that is something that is not possible using the Raspberry Pi. There are workarounds to this problem as there are other versions online of MATLAB that will work on the Raspberry Pi but they don’t offer anywhere near the same level of support that MATLAB would have on windows or MAC. Another reason is that OpenCV free and MATLAB is bit too expensive for me. Due to the use of the Raspberry Pi in this project, it will be necessary to monitor the performance of the device to ensure it achieves the best performance possible. MATLAB is a lot harder on the CPU than OpenCV would be, however this does come with drawbacks. The first one being that there is the possibility of memory leaks when using OpenCV. These would be less likely to occur using MATLAB but performance would suffer.

Another thing that was factored in, was the amount of support that was found on the online communities for both technologies. It appears that there is better support out there for OpenCV then there is MATLAB. There are a lot of resources on MATLABS website with instructions on how to implement different packages. But due to the fact that OpenCV is free, it seems to have a larger audience. This means that there are a lot of answers to even some of the smallest problems. Help from online sources will be needed in the development of this project, so this was another reason to go with OpenCV. The nail in the coffin for MATLAB was when the lecturer Derek Molly responded to my email and confirmed that OpenCV would be a good choice when approaching this project. He has past experience with multiple image processing software. His advice should be taken into account continuing on with this project.

2.4.2. Cameras

It has been decided not to use the Xbox 360s Kinect because this would go against the goals for this project. One of my goals was to set up the cameras myself and work out how to calibrate them. If Kinect was used than that would mean skipping over some of the most complex parts of this project. Therefore, it will be necessary to purchase two Logitech cameras which will be connect to the Raspberry Pi. The webcams that were purchased are the c310s, they were bought while they were on sale and have saved around twenty euro. With these two camera It will be possible to use stereo vision to work out the depth of objects that are seen in the images gotten back. “This paper has shown the feasibility of constructing high performance stereo vision system on programmable logic, while maintaining construction simplicity and low cost” [34]

Figure 6- Logitech Camera[17]

2.4.3. Raspberry Pi

When conceiving the idea for this project, use of a Raspberry Pi was already at the forefront of my mind. To learn more & get experience this was selected. It is a piece of technology that those studying computer science should be more familiar with. The model purchased was the latest version of the Raspberry PI (model B+) as it is currently the best one on the market. The four USB ports on the model B+ will be very useful as it gives me the option of plugging in a memory stick and a Wi-Fi dongle while still having my two webcams connected. A big advantage of using the Raspberry Pi is that it is portable. This will make it easier for the user to set it up in their car. They will be able to move it around until they find a place that suits it and the cameras that will be connected to it.

2.4.4. Apache

Being familiar with Apache from my work placement and having used it there and setting it up on multiple machines, it was my first choice starting this project. The reason it is used at work is to display the results that are collected by my scripts. Once the scripts collect the results an email is sent to the team with a link to the machine. This saves time and lets other members of the team collect a large amount of results without having to manually SSH onto the machine and find them themselves.

This software would be useful in this project to make life easier on the end user. One of the problems that occurred was how does the user start recording with the web cameras while sitting in the car. In my opinion the method would work the best would be installing apache on the Raspberry Pi. The next step involves the setup a webpage to display options for the user to choose from. This way the user can use their mobile device to start the cameras from the car.

The only downside to using this is that the Raspberry Pi will need to be connected to the local network while in the car. The router in my house is in a room beside the driveway, so this is not a problem for me, but it could be an issue for other users.

2.4.5. Python vs C++

The two programming languages that have been researched for this project are Python and C++. OpenCV was created using C++ and C so it is optimized for both these languages. It is also important to consider how the results will be displayed to the user of the completed program. Having been familiar with python in the past and have created multiply GUIs with it. Never having created any with C++. One thing that is having a heavy influence on this decision is the amount of help out there for OpenCV that is in C++. There is a lot less help online for OpenCV in python. Learning a new programming language at the minute while useful, would not be feasible giving the current time constraints on the project. Having looked online and found many books to help, C++ would seem to be the better option. If this proves to be false, then changing to python will have to be considered. But for now finishing my project in C++ is the most practice. At the end of the day the decision is to go with C++.

2.4.6. Displaying Results

Having already looked into how to store the user's results once the tool is finished going through the video footage, research into storing it in a data base has already been performed, but this might be a touch overkill as it will only be storing a small amount of information. The current plan is to write all the data to a csv file. The data from this csv file could be used to display a graph to the user. This would also mean you could add an option to allow the user to load different csv files. These csv files would then be stored in a results folder that the tool would have access to. Then pull the results from the csv files and display them through the C++ GUI that has been programmed.

2.5. Requirements

Non-Functional	**Functional**
Raspberry Pi Model B+	User Interface
Wifi dongle (Raspberry Pi)	Motion Tracking
MicroSD (Raspbian pre-installed)	Distance Detection/Estimation
Two Logitech c310 webcams	Storing results in csv file
OpenCV installed on Raspberry Pi	Overall Score based on users results
Familiarize myself with OpenCV library's	List of rules for drivers evaluation
Familiarize myself with C++	Webpage setup using apache which will allow user to run scripts

2.6. Conclusion

After looking though some real world examples and existing technologies. The decision was made to use a raspberry PI along with two Logitech webcams. The reason is that that it is cheaper and will be efficient enough to provide somewhat accurate results. Using OpenCV as it is open source and has a large amount of support from a large community. while use C++ when coding, even though it is a new language for me to learn. There should be more than enough help online and from books to create this software tool.

After looking at different methods used by google car and tesla I have decided to use stereo vision to work out the distance of object. This is the cheapest way and should not be possible with the help of functions that come with OpenCV. There are functions to calibrate the cameras and also many others that could be used for motion tracking.

Having decided to store the results that are gathered from the video inside a csv file. It may be possible in the future to store them in a database. But before that you will have to tackle the distance, motion detection issues. Setup a web-page using apache that will allow the user to start the cameras remotely. This will make it much easier to use the device while in the car.

3. Design

3.1. Introduction

This chapter will discuss the different methodology's that were researched before taking on this project. It will also focus on the design of the project and how each of the main features will work using the hardware and software that was decided upon in the previous chapter.

This chapter will also include some flowcharts and use case diagrams of how the code is meant to flow, this will help later in the implementation stage.

3.2. Methodology's Researched

3.2.1. Waterfall Methodology

The waterfall model was proposed by Winston W. Royce in 1970 to describe a software engineering practice. This model is made up of five steps, Analysis, Design, Implementation, Testing and Maintenance. The advantage of using this approach for a project is how easy it is to follow. Each step is laid out clearly and you know exactly what's going to follow next. The problem with this approach is that there is no deviation from the plan. It is very difficult to go backwards in this model. If you mess up one of the requirements early on and you only realize this once you are in testing. Then you are more than likely too late to go back, as you have already committed to the current plan that is in place. This methodology is a good approach when you are a hundred per cent sure of your idea, and you have must have the kinks ironed out beforehand. But if you are still unsure of system requirements going in then it would be advisable to follow a more agile methodology.

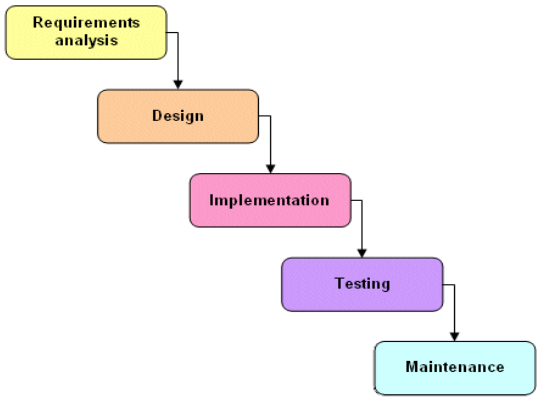


Figure 7- Methodology Waterfall Model [18]

3.2.2. Agile Methodology

This methodology is similar to the waterfall approach when it you look at it on paper. However, the execution is very different. Using this approach, you are not confined to the same restrictions that you are in the waterfall methodology. Its similar in that when you start off you do your analysis, or system requirements. Then you design, implement, test. But the difference here is that you can go back again and again and redo something that you feel was underdeveloped. This is a massive advantage when you are dealing with customers who change their mind frequently, or are working in a market that is changing constantly. The

disadvantages of using this methodology is when you are working with a large team as lead developers are the only ones who can make decisions about moving on. Since there is no set plan, lower level engineers cannot make any big calls on development. But overall this is a good methodology and is widely used in a lot of software companies.

3.2.3. Spiral Methodology

The Spiral methodology is different to the two listed above. This one is broken down into four steps. They are Identification, Design, Build, Evaluation and Risk Analysis. The reason it has the name spiral is the software project that the team or individual is working on goes through these phases again and again. This then gives the appearance of a spiral. There are many advantages of using this methodology. One is that it has a high risk control, as the team or individual is constantly going back to the risk analysis after a complete iteration. It is a good methodology for large teams as there is a lot of documentation and room to breathe. A disadvantage is that it is costly to implement in company's due to the time it takes to run through each iteration. It is not as affective on smaller projects due to the lack of planning and detail that goes into it.

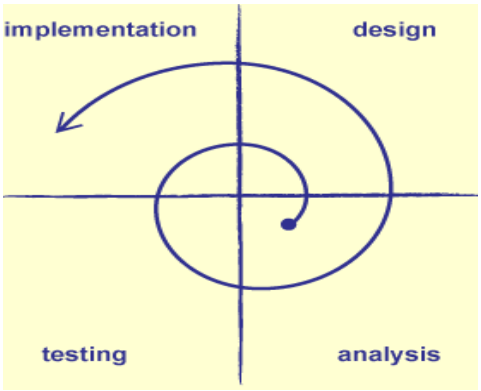


Figure 8- Methodology Spiral Model

3.2.4. Rapid Application Development

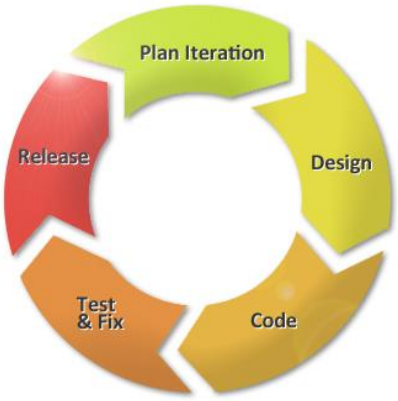


Figure 9 - Methodology Agile RAD Model [20]

The Rapid Application Development (RAD) model is designed to allow a much faster development time than the normal methodologies would. RAD is more user focused and has an incremental approach. It is tested and evaluated by the end users and the developers at the same time. This allows for design decisions to be created on the fly, which are based on prototypes and the user response to the prototypes. One of the main advantages of using this methodology approach is that it is fast and also provides developers the freedom of trying things out first, and throwing the idea away if it does not work. Another great advantage is that you get instant customer feedback and can make changes to the product as you are developing to meet the customer's needs. This approach would not be recommended for large projects, as there is no long term plan. It is only recommended to be used on small projects that are low cost and have time restrictions.

3.2.5. Chosen Methodology

The methodology chosen to use for this project was Rapid Prototyping (RAD). This project was more suited towards this approach for a number of reasons. The main one being that there will only be a short amount of time to complete this project. With this in mind came the realisation that you couldn't just hope that the solutions, when found in books and online would work when tested. You needed to make sure that by testing out each one of these to find the ones that worked best, and to also familiarize yourself with the process of setting up the project. There were many advantages of working through these different solutions and creating different prototypes.

A major one was how it highlighted areas that were more difficult than previously thought. This helped me alter my design and functional requirements to meet a much more realistic result. A good example of this is when trying to find a way to run my program without using a wireless keyboard. This was to emulate the Raspberry Pi that had been placed in a car. The two solutions for this are to either SSH onto the device and run the scripts manually, or the user could find the IP address of the device and look up its site that was set up after installing Apache2. The user could then click a button on this site that would trigger the program to run.

Another example of this methodology highlighting a problem was during the second week when looking at the difference of performance between C++ or Python. After seeing firsthand how the cameras performed much better while using C++ making an informed decision that was acceptable. Another advantage of this approach was being able to throw away ideas that didn't work. While testing to see how the cameras performed in real time when the code was creating a disparity mapThe aouther noticed that the cameras were starting to stutter after prolonged use. This was due to the Raspberry Pi getting slower over time. This is a problem that would not have been found on paper. If we were to just test code on its own, then it would have worked and we would of ran into this problem once we integrated the cameras later on.

This approach helped find multiple library's that would be needed while using C++. Additionally, OpenCV had to be installed on the Raspberry Pi. This took a lot longer than was expected and it was also much more difficult due to the multiple versions available online. In addition to the software that was installed on the Raspberry Pi, an update to the Raspberry Pi was also necessary. This was to set up the two Logitech web cams. When they were first plugged into the Raspberry Pi, they were both checked to see that they were connected using the lsusb command. After running a sample script to take a few picture it was noticed that it was returning errors. It turned out that a firmware update was needed and after running the command rpi-update, the cameras stared working.

3.3. Design Limitations

Since the hardware being used in this project is cheap, certain design choices will have to be made. The overall performance of the cameras will be affected by the Raspberry pi. The structure of the code will have to take that into account. A good example of this is the how the

project will be doing all the image processing after the video footage has been taken. This is due to the fact that the Raspberry Pi would not be able to support two cameras and the processing needed at the same time.

Another example of how the raspberry pi is limiting is the constant need for a power supply. To solve this issue, another piece of hardware will have to be purchased to allow the car cigarette liter to be used as an adapter. However, this will not be an issue for early prototypes, as all the testing will be done using the normal power adapter.

3.4. Users experience

When designing this project, the users experience will have to be at the forefront when deciding on certain aspects of the design. This project is aimed at users who are learning how to drive so that narrows down the type of end user. However, it does not eliminate them all. This project will have to be intuitive so that all users from novice to experienced can use it sufficiently.

There will be users who will not be able to use this software tool. Any user that is blind or suffers some form of disability that has a major impact on their sight will not be able to navigate the GUI.

3.5. Appearance

Since the project is using hardware, it opens itself up a wide range of issues when it comes to design. The hardware components will have to be placed together to avoid them being lost. It also makes it easier for storage when placed in a car. There are however many components, two webcams, a raspberry pi and a USB adapter. This makes it difficult for the project to be aesthetically pleasing.

This hardware will also have to be moved from the car to the user's desk or table. There the user will have to be able to connect the raspberry pi to their television or monitor. This will have a massive impact on the design of the box that will store all this equipment.

3.6. Chosen Software

In the last chapter, it was decided that C++ and OpenCV would be the chosen technologies when approaching this project. This was due to the overall stable performance that can be achieved through C++ while using library's from OpenCV. One of the issues that will affect design while using this language is that it does not deal with memory management well. This will affect the code. While coding this project steps will have to be taken to stop memory leaks, such as releasing images once they have been used, and also refreshing any buffers used.

3.7. Technical architecture diagrams

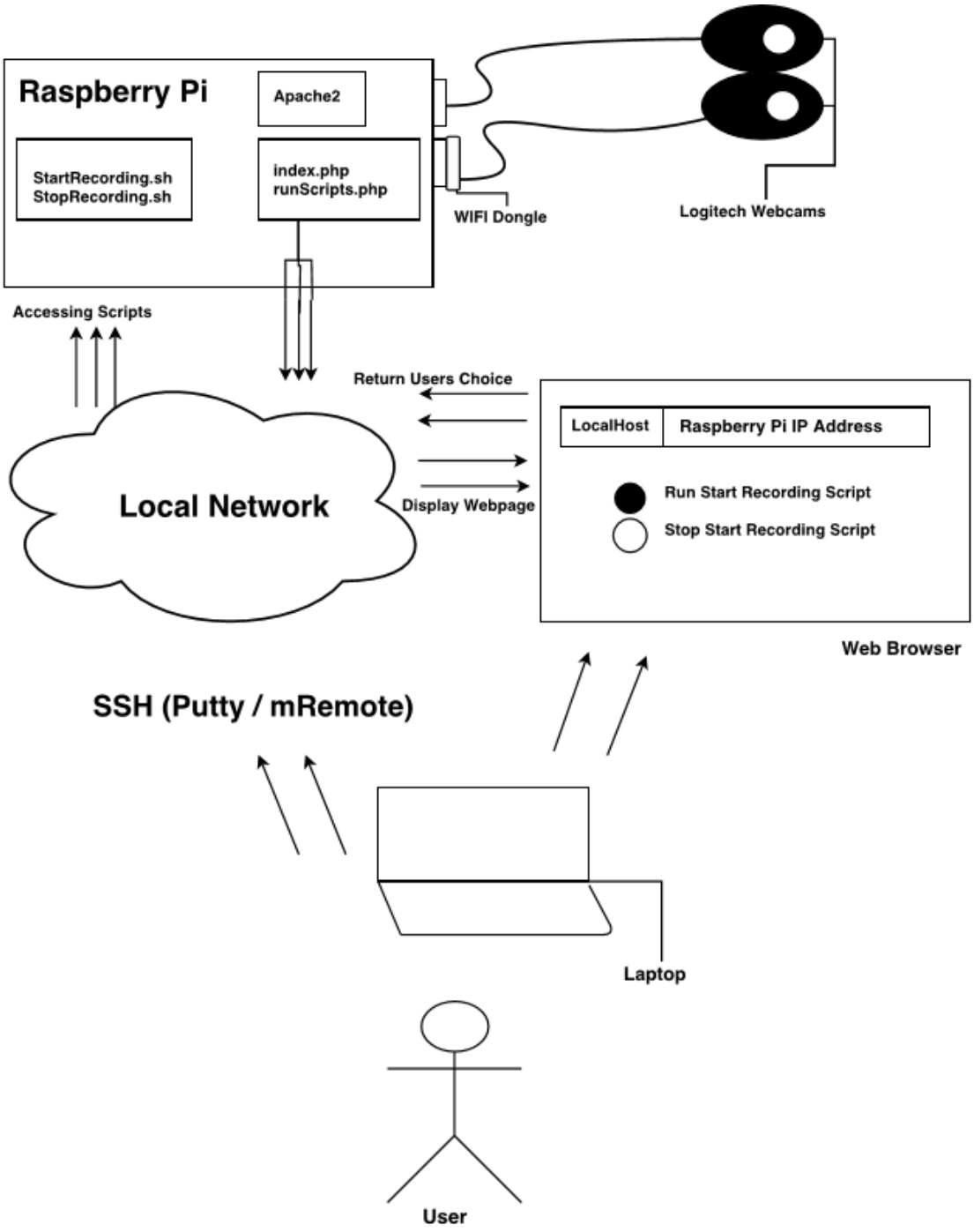


Figure 10- Technical Diagram Starting Cameras

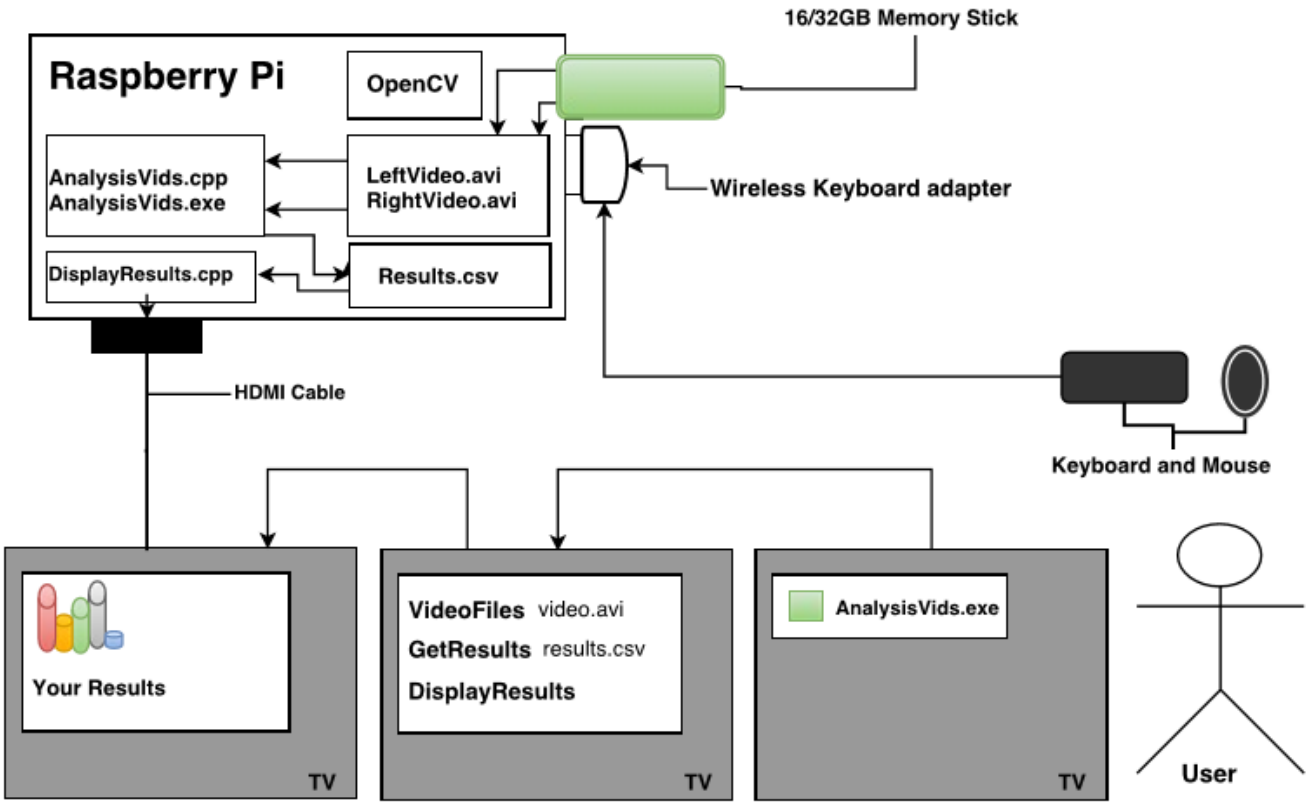


Figure 11- Technical Diagram 2 Getting Results from video files

The two technical diagrams above, figure 1 and figure 2 are both a very simplified view of how the software and hardware will work together. There are also actors there to show how the user will interact with the system.

Figure 1:

In this diagram we see how the user will activate the cameras and get them to start recording. The user has two options, they can either SSH onto the Raspberry Pi through Putty or they can look up the Raspberry Pi's IP address on their browser. If the user decides to use putty or a similar program too SSH onto the Raspberry Pi, they will have to go into the directory holding the scripts that start the cameras and run them themselves. If the user decides to open up their web browser on their laptop or phone and look up the IP address of the Raspberry Pi. They will then be able to select the option to run the script on the Raspberry Pi which starts the camera.

This diagram also shows that Apache is installed on the Raspberry Pi. Therefore, the index.php script can be viewed by the user if they and the Raspberry Pi are both connected to their same home network. This allows the user to access to files that are made available through the index script. In this case both the start and stop recording scripts have been made available to the user.

Figure 2:

This diagram shows both what the user sees and what is going on inside the Raspberry Pi. So when the user runs the program AnalysisVids.exe they are then presented with a GUI. This GUI contains three options the first being "Video Files". This option allows the user to select two video files from either the

Raspberry PI or an external memory source. The next option down is “Get Results”. Once the user has chosen their video files then they can select this option. The program will use these video files for analysis and work out a score for the user. These results are written to a csv file, this file is stored in the same location as the video files that were used. The final option that the user has on this menu is to “Display Results”. When the user selects this option the program will find the CSV file that created last and then display its results through either a bar chart or some sort of graph. You will also notice that OpenCV is show to be installed on the Raspberry Pi, this is to indicate that a lot of functions will be used from this while analyzing the video.

3.8. Hardware architecture diagram

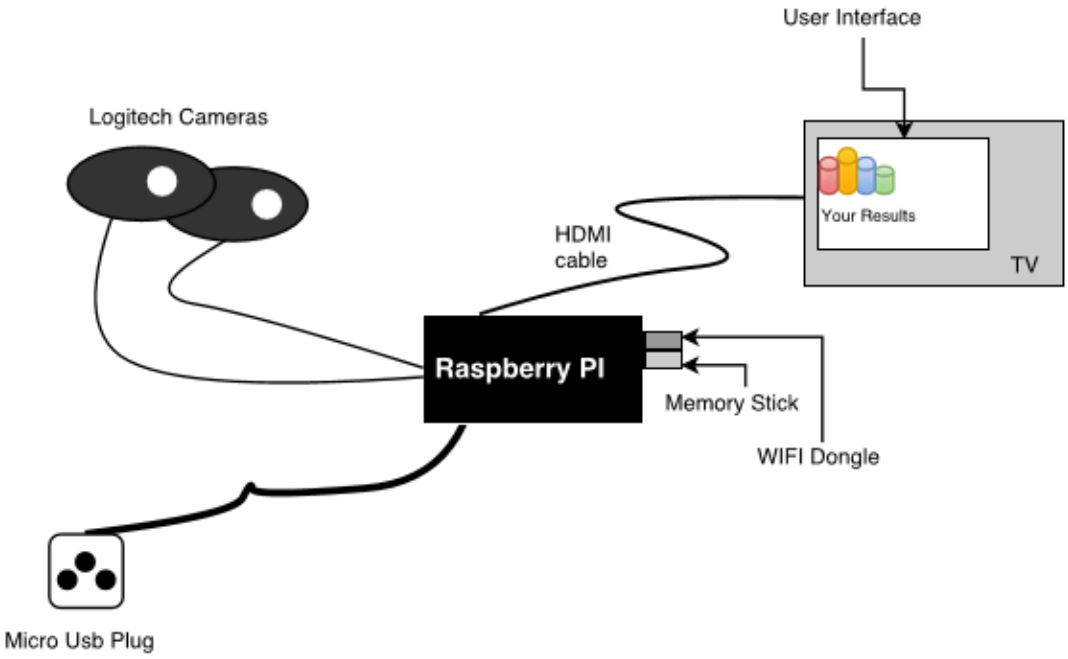


Figure 12- Hardware Diagram - All components

The above diagram figure 3 is an example of all the hardware components connected together. The user should never have the hardware set up like this, it is only an example to show all the components working together. There are two different ways the user will have to set up the hardware. It will depend on what the user is trying to do.

Setup One: Recording video footage

When the user wants to go out and capture some video footage they will need these pieces of hardware. The Raspberry Pi, Two Logitech Cameras, a WIFI dongle, Micro USB lead and USB car adapter. The memory stick is optional as the user might not need the extra storage and as they are only recording a small amount of footage. The user will need to make sure that the WIFI dongle is in range of their

router. The user would also be advised to have the car started before plugging the Micro USB lead into their Raspberry Pi, as turning on the car after its plugged in may damage it.

Setup Two: Analyzing/Viewing results

When the user wants to analysis their video footage or just look at old results. They will need these pieces of equipment the Raspberry Pi, HDMI cable, Micro USB plug and a television that has a HDMI input port. If their video footage or old results are on a memory stick, then they will obviously need that too. The user will have a similar setup to the diagram above with the exception of the two Logitech cameras.

3.9. Hardware Box designs

Here are some of the designs that were consider for the project. The main aim of this design was to allow the user easy access to all the wires, and have the box be portable. The only big differences between designs were how to conceal the hardware inside the box.

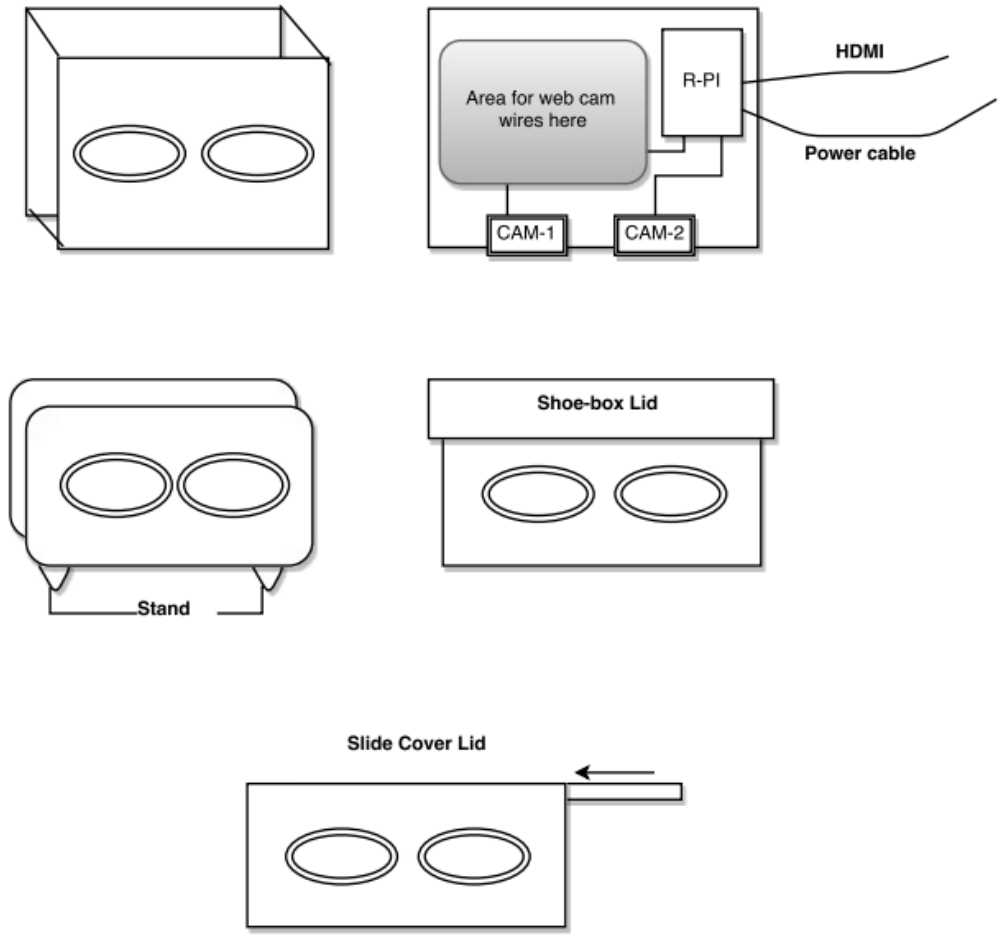


Figure 13- Hardware Diagram Box Designs

3.10. Flowcharts

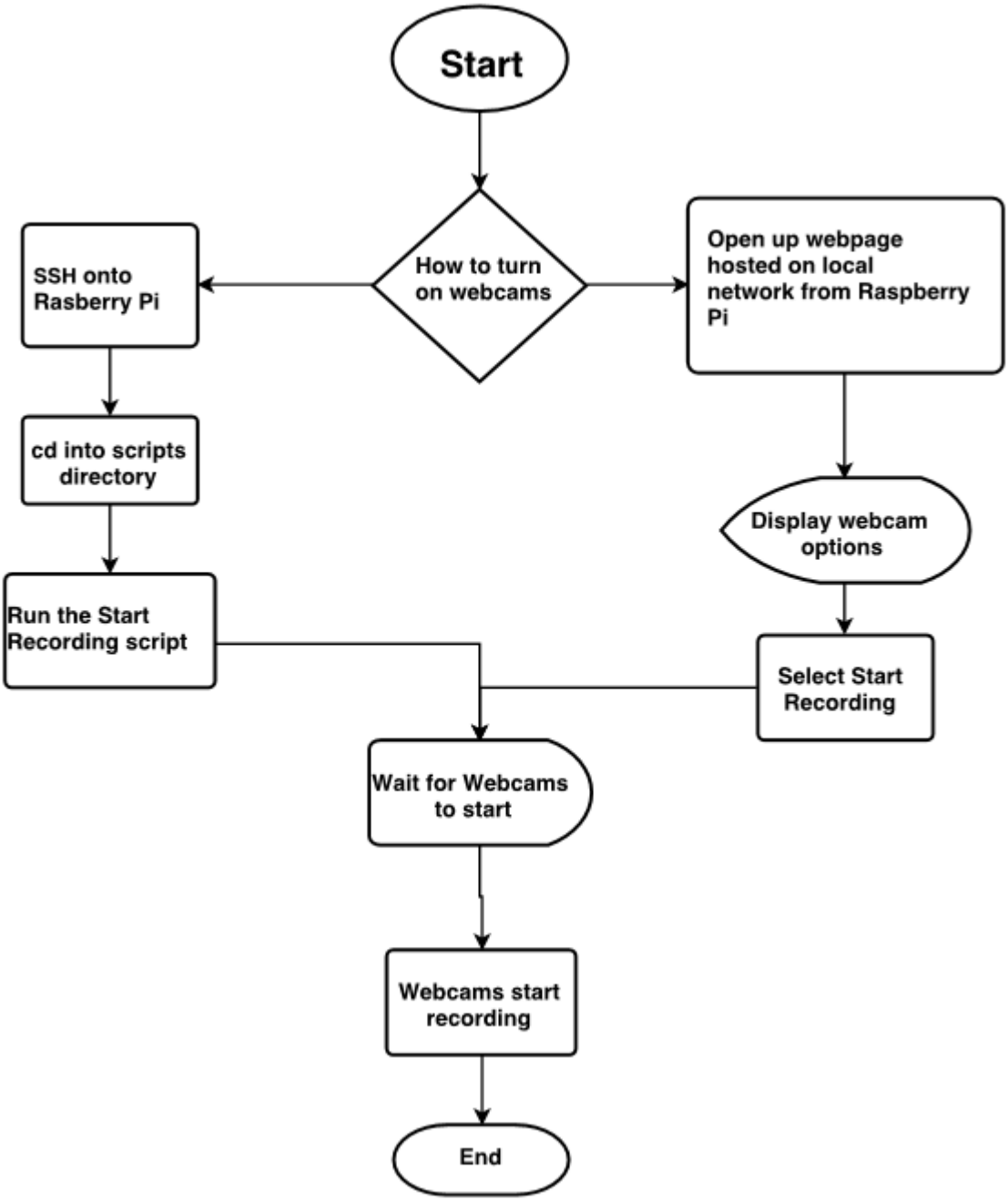


Figure 14 - Flowchart Turning on Cameras

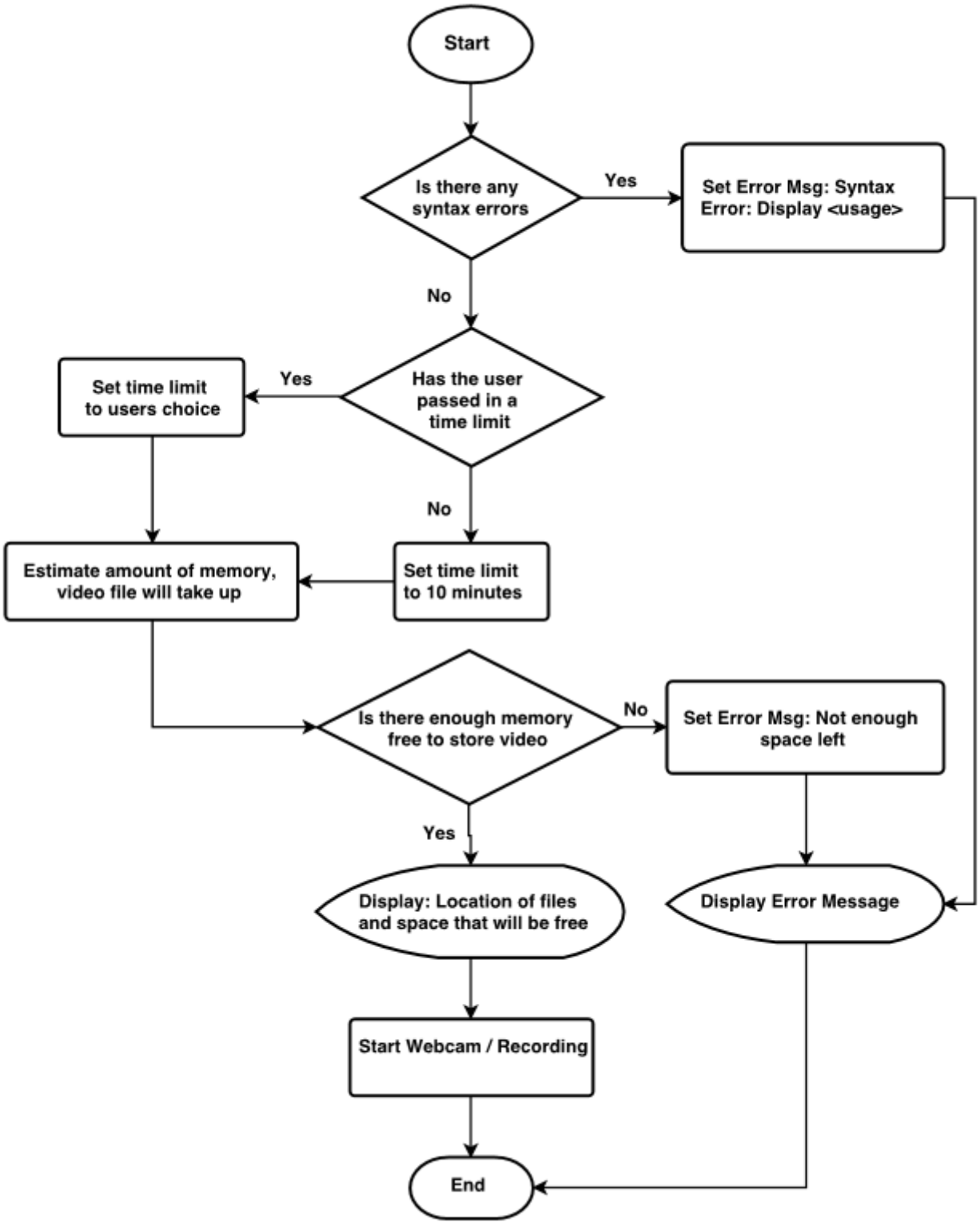


Figure 15- Flowchart Start Stop Recording Script

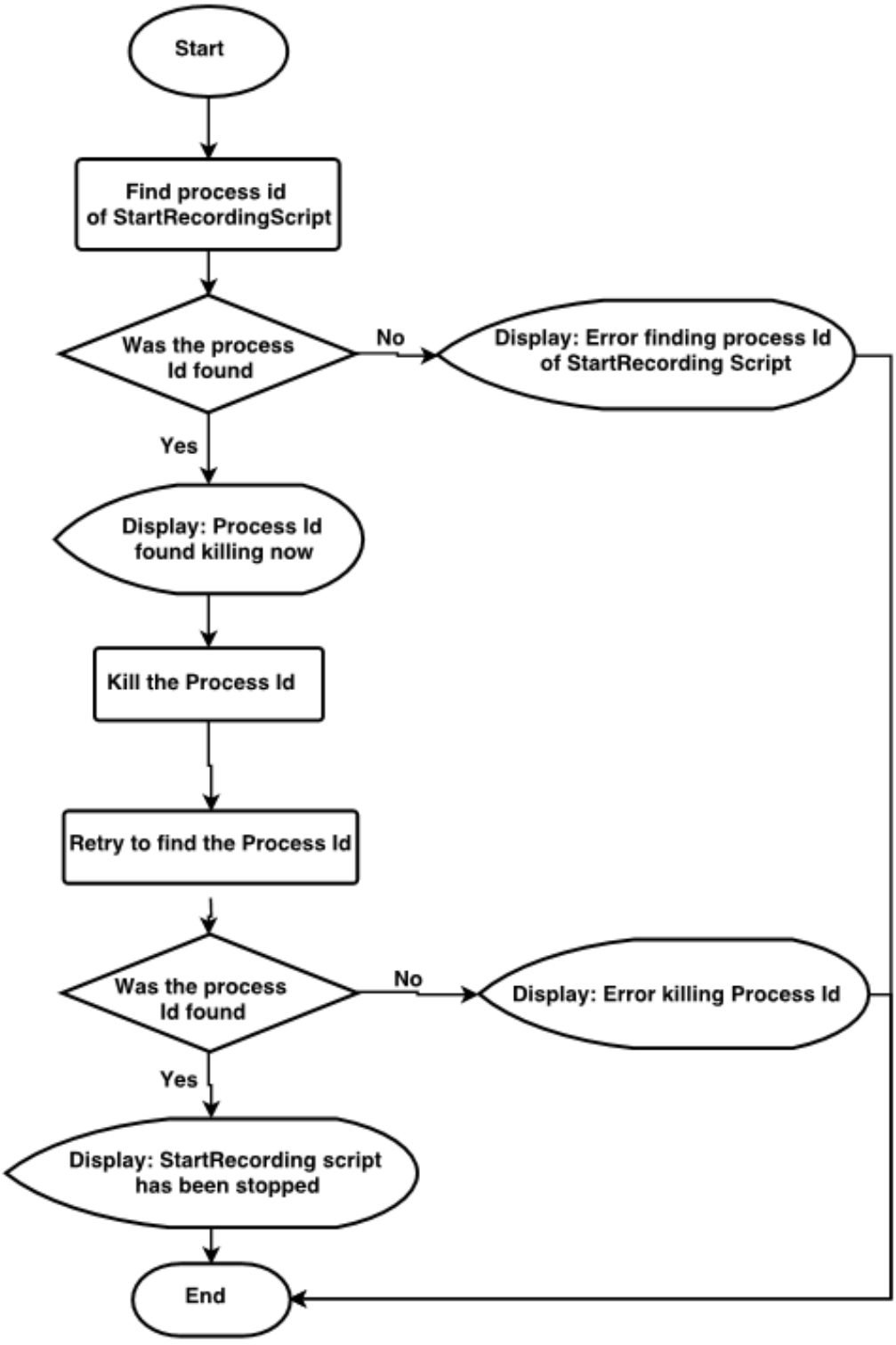


Figure 16- Flowchart Stop Recording Script

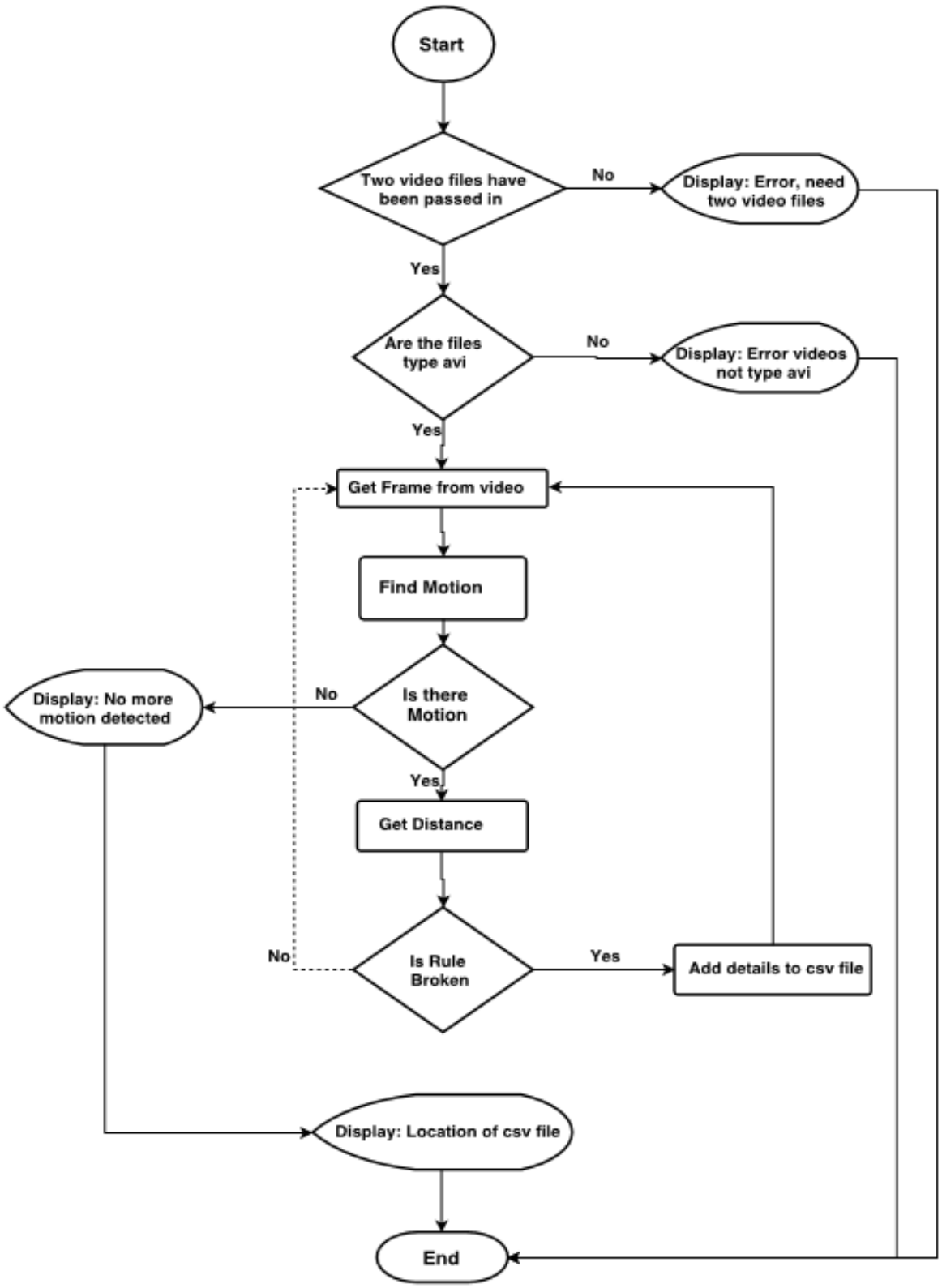


Figure 17- Flowchart Analysis.exe

The flowcharts above give you a very simple view of the process and steps that will happen in each script or program. There is obviously going to be a lot more to some of these programs but these flowcharts should give some indication to how they will work.

3.11. Conclusion

In this chapter we have discussed the methodology that will be used in the implementation of the project. We have also gone over some designs, both hardware and software. Now that this project has a clear goal along with these diagrams, it will be much easier to start implementation based of the guidelines set up in this chapter. If there are any errors of blocking issues that appear later on, it will be easier to work around them due to the methodology chosen.

4. Early Prototyping and Development

4.1. Introduction

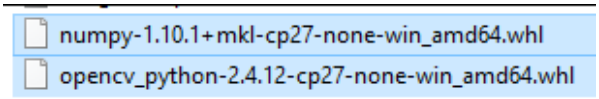
In the last chapter, the methodology was decided upon, alone with some examples of how the hardware and software were going to work together. This chapter will go over the early implementation of those designs to see if it is indeed possible to continue. This chapter is split up into iterations as if problem was faced on its own and was given a certain amount of time until completion or until it had to be scraped for another approach.

4.2. Iterations

4.2.1. Iteration One:

The goal of this iteration was to learn more about OpenCV and how it can be used with the programming language python. It would also be beneficial to have a working example of OpenCV on a laptop by the end of this iteration. It was very difficult when starting off as it was difficult finding the packages needed. A guide online that has instructions on how to setup OpenCV for python, proved to be very useful. [22] Once the instructions were followed the "import cv" command was run in IDLE. This then returned back an error. It turned out that the problem was the packages that were downloaded where for a 32-bit system while the laptop being used was a 64-bit. This meant that unofficial 64-bit version had to be downloaded. There was a site [23] that had a list of different packages and which included both the numpy and OpenCV packages for a 64-bit system.

```
>>> import cv
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    import cv
ImportError: No module named cv
>>>
```



Once the files were downloaded, the pythons package management system pip was then used to install them. They both installed successfully without any problems. Once they were installed the command "import cv" was entered again on IDLE and it worked.

```
>>> import cv
>>>
```

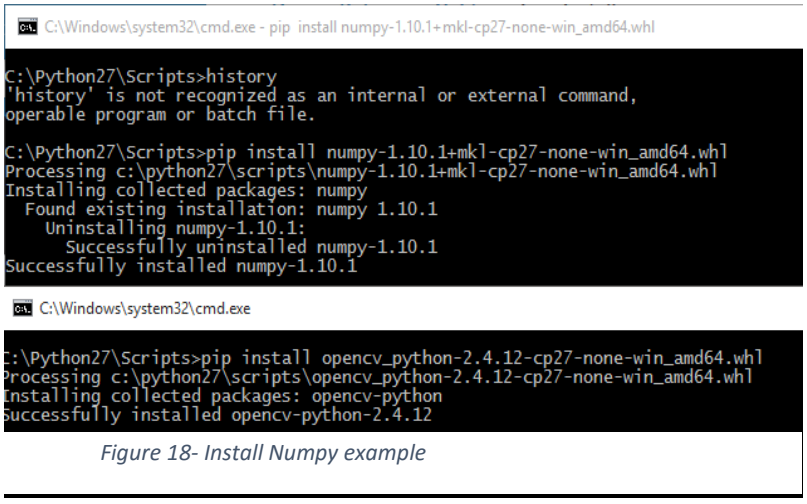


Figure 18- Install Numpy example

Once OpenCV was installed, some examples of code found online that were being used to generate a disparity map were downloaded. [24] Then the two images were downloaded from the website to test.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

imgL = cv2.imread('ImageL.png',0)
imgR = cv2.imread('ImageR.png',0)

stereo = cv2.StereoBM(1, 16, 15)
disparity = stereo.compute(imgL, imgR)

plt.imshow(disparity,'gray')
plt.show()
```

Figure 20- Disparity Map Code Example

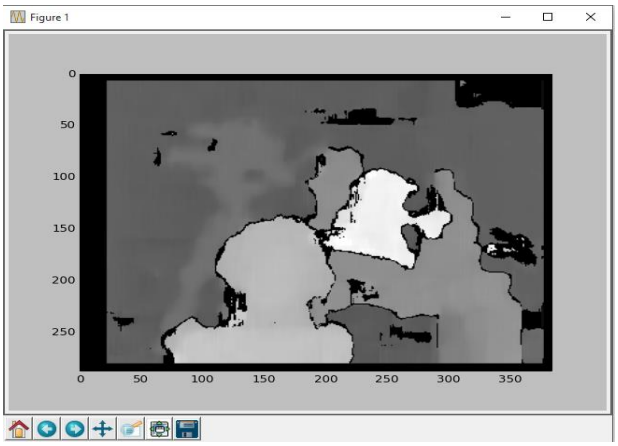


Figure 19- Disparity Map Example

So here we can see that the disparity map generated worked as the lightest colors are the closest while the darker ones are the furthest. Once these results were achieved then it was time to move onto the next iteration.

4.2.2. Iteration Two:

During this iteration the goal was to find more examples of code online, along with a few more pictures that then could download and tested. After searching through the internet for a while two different sites were found. [25] [26] A few images that were found on these sites were downloaded, these were then tried out with the python script.

Figure 21- Test Image one and Result

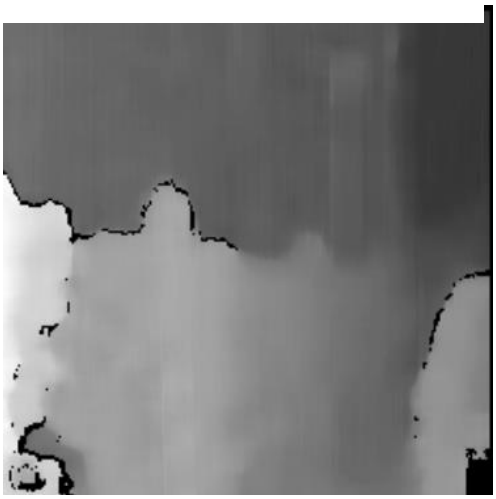


Figure 22- Test Image Two and Result

As you can see above, the results are ok but not that great. There is a lot of detail lacking. However, after going online and finding a piece of sample code that claimed to create a much more detailed disparity map. [27] Another test was done with the parking meter and the result was much sharper than previous one. This was interesting as it did really show that some changes to the code can affect the quality of the image dramatically. This really impacted the decision on the language that was chosen. There were many articles online suggesting that C++ was the better choice. So during this iteration it was decided that C++ would be given a try when moving forward.

Figure 23- Test Image Three and Result

This python code below is what produced the image of the parking meter on the above page that is clearly more detailed then the other results. [27]

```
import numpy as np
import cv2.cv as cv
from matplotlib import pyplot as plt

def cut(disparity, image, threshold):
 for i in range(0, image.height):
  for j in range(0, image.width):
   # keep closer object
   if cv.GetReal2D(disparity,i,j) > threshold:
    cv.Set2D(disparity,i,j,cv.Get2D(image,i,j))

# loading the stereo pair
left  = cv.LoadImage('meter_l.pgm',cv.CV_LOAD_IMAGE_GRAYSCALE)
right = cv.LoadImage('meter_r.pgm',cv.CV_LOAD_IMAGE_GRAYSCALE)

disparity_left  = cv.CreateMat(left.height, left.width, cv.CV_16S)
disparity_right = cv.CreateMat(left.height, left.width, cv.CV_16S)

# data structure initialization
state = cv.CreateStereoGCState(16,2)
# running the graph-cut algorithm
cv.FindStereoCorrespondenceGC(left,right,
                              disparity_left,disparity_right,state)

disp_left_visual = cv.CreateMat(left.height, left.width, cv.CV_8U)
cv.ConvertScale( disparity_left, disp_left_visual, -20 );
cv.Save( "disparity.pgm", disp_left_visual ); # save the map

# cutting the object farthest of a threshold (120)
cut(disp_left_visual,left,120)

cv.NamedWindow('Disparity map', cv.CV_WINDOW_AUTOSIZE)
cv.ShowImage('Disparity map', disp_left_visual)
cv.WaitKey()
```

Figure 24- Code Example python disparity map

4.2.3. Iteration Three:

Figure 25- Hardware Setup Cameras Example

During this iteration the Logitech web cameras arrived from amazon. So the goal was to setup the web cameras on the Raspberry PI and to also verify that it was possible to use both web cameras and that they could take pictures. After the cameras were plugged into the Raspberry Pi, the "lsusb" command was used to check that they had been found. Once they both appeared it was obvious that they were working. The package fswebcam was then installed. This allowed the Raspberry Pi to take pictures with the web cameras. The command that was used to take the picture was "fswebcam test.jpg". It was then found out that it would be possible to control which camera took the picture by adding" fswebcam -d /dev/video0 test.jpg". This was tested out on both cameras a few times and then different commands including one that saved the pictures onto a memory stick were tried out. Once these cameras were setup and commands were working it was time to move onto the next goal.

```
pi@raspberrypi /var/www $ lsusb | grep Logitech
Bus 001 Device 004: ID 046d:081b Logitech, Inc. Webcam C310
Bus 001 Device 006: ID 046d:081b Logitech, Inc. Webcam C310
```

```
fswebcam -d /dev/video0 /home/pi/Desktop/picsTest/today/testOne.jpg;
fswebcam -d /dev/video1 /home/pi/Desktop/picsTest/today/testOne.jpg;
```

4.2.4. Iteration Four:

After iteration two it had been decided to try out some C++ code to see how easy or difficult it was to use. Some code samples were then tested out on the Raspberry Pi using the web cameras that were just setup. There were lots of examples found online but one example that was found online [28] had the skeleton code which different approaches. This code was used as it was a good opportunity for the author to get experience using OpenCV in C++.

```
#include "cv.h"
#include "highgui.h"
#include <iostream>
int main(int,char**) {
    cv::VideoCapture capLeft(0); // open the Left camera
    cv::VideoCapture capRight(1); // open the Right camera
    if(!capLeft.isOpened() || !capRight.isOpened()) // check if we succeeded
    {
        std::cerr << "ERROR: Could not open cameras." << std::endl;
        return -1;
```

```
if (isValid)
{
    try
    {

        cv::Mat img1, img2, g1, g2;
        cv::Mat disp, disp8;

        cv::imwrite("frameLeft.bmp", frameLeft);
        cv::imwrite("frameRight.bmp", frameRight);

        img1 = cv::imread("frameLeft.bmp",1);
        img2 = cv::imread("frameRight.bmp",1);

        cv::cvtColor(img1, g1, CV_BGR2GRAY);
        cv::cvtColor(img2, g2, CV_BGR2GRAY);

        cv::StereoSGBM sgbm;

        sgbm(g1, g2, disp);
        normalize(disp, disp8, 0, 255, CV_MINMAX, CV_8U);

        // Create DistMap
        cv::imwrite("Dist.bmp", disp8);

    }
```

Figure 26- Code Example C++ Disparity Map

OpenCV was not installed, so when the code was run it did not work on the Raspberry Pi. A guide found online that went through each step and was very useful and easy to follow, there were optional packages in the guide but only the necessary ones were downloaded. [29] Once this was installed the code was run again. The book Learning OpenCV [3] was used a lot when editing the C++ script to create custom disparity maps, using the web cameras.

After using some examples from the book, some edits were added to the code and these added commands were to the first if statement. The script was run again, and it produced these results. As you can see the disparity map is not very accurate but that is more than likely due to the fact that the left camera did not grab the correct frame. This means that when the two frames were compared there was too much of a difference to get any meaningful results.

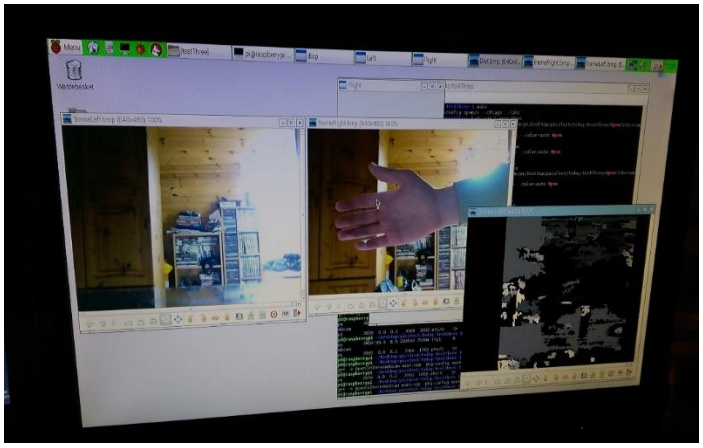


Figure 27- Early Example of Depth Map

After looking into this issue for a while it is still hard to determine why the frames did not match. Taking notes of the problems that were encountered and the ones that still have to be solved, it was decided to move on as motion tracking was still not tested.

4.2.5. Iteration Five:

My goal for this iteration was to get some sort of motion tracking working, the ideal case would be for the user to be able to track an object. A lot of help was found on YouTube as there were many different tutorials. On one of the videos found, the user had included the source code. [30] once the code was downloaded onto the Raspberry Pi. It failed the first time it was ran as the library structure was different to what was there previously, and this required a change to the path so it would work on the system that was there. Once the paths where fixed the code was ran again and it worked straight away. The user was able to highlight objects that were inside the web cameras view and it would track them all over the screen as they were moved around. Two examples of the code working have been added below.

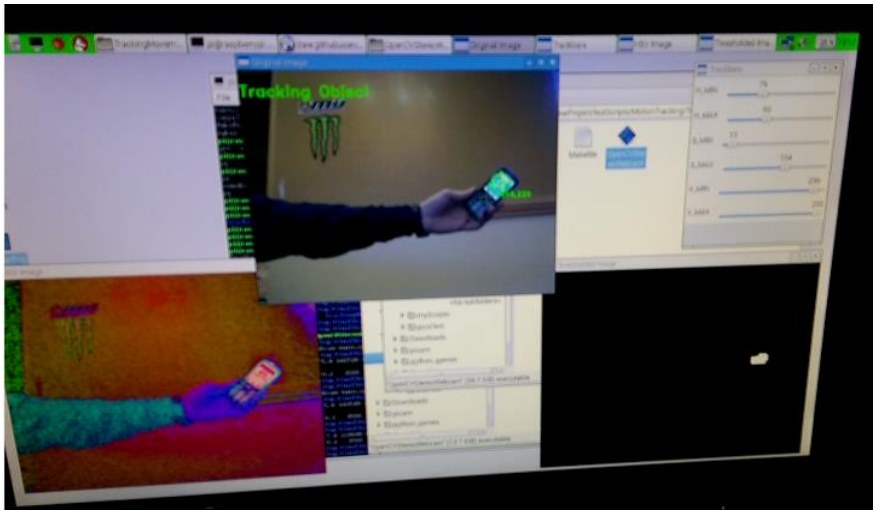
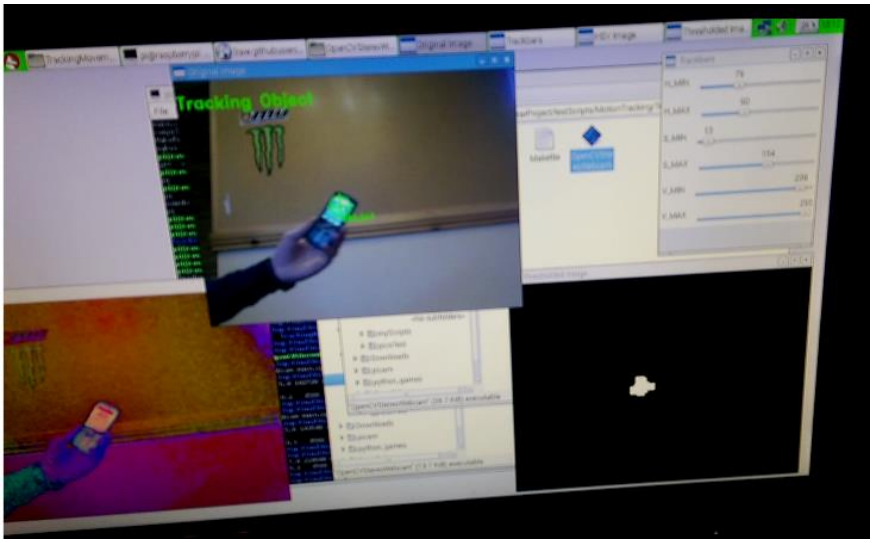


Figure 28- Early Examples of Motion Tracking



There are many other ways to track motion using OpenCV but since the goals were met for this iteration and it was time to move on. The author decided to move onto the next iteration.

4.2.6. Iteration Six:

This iteration the goal was to get apache installed on the Raspberry Pi and setup a small html or php script that would display something. The author had used apache at work so he was familiar with the steps. Once he installed the package and went in the directory /var/www/, he was able to get up the default index that was being displayed to the user. Then a few more packages were installed that allowed php to be used with apache. A new index script was created that would display the time.

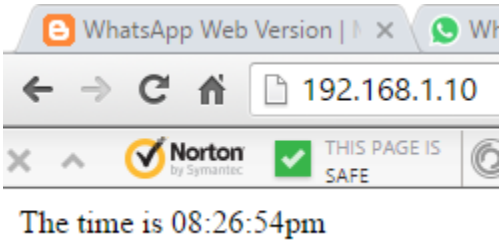
```
pi@raspberrypi /var/www $ ls
index.php
pi@raspberrypi /var/www $ cat index.php
<?php

date_default_timezone_set('Europe/Dublin');
echo "The time is " . date("h:i:sa");
```

Figure 29- Example of Code setting up network / php

The command “ifconfig” was run as to get the Raspberry Pi’s IP address.

```
wlan0     Link encap:Ethernet  HWaddr 48:ee:0c:83:0d:be
          inet addr:192.168.1.10  Bcast:192.168.1.255  Mask:255.255.255.0
```

When this URL was then entered in the URL address, it was clear that the script was working, as the current time was being displayed.



Once this was working the author thought that it would be a good idea to set the default IP address of the Raspberry Pi to 192.168.1.10. This way he would not have to run the “ifconfig” command every time he wanted to find out what the IP address was. After making some changes to the interfaces files in the network directory of the Raspberry PI. It will now start up every time with the default IP address of 192.168.1.10.

```
pi@raspberrypi /var/www $ cat /etc/network/interfaces
auto lo
iface lo inet loopback

auto eth0
allow-hotplug eth0
iface eth0 inet dhcp

auto wlan0
allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp

iface wlan0 inet static
address 192.168.1.10
netmask 255.255.255.0
network 192.168.1.0
broadcast 192.168.1.255
gateway 192.168.1.254
```

Figure 30- Example of network settings

This was tested by turning off and rebooting the Raspberry Pi and connecting to it through putty. This was done three times to verify that the new settings that were added were working.

4.3. Conclusion

This chapter shows that this project is possible. There is work required in a lot of different areas, like camera calibration, a steady setup to hold the cameras, and more robust code for image processing. It also confirms the plan set out in the design chapter is good enough to follow, there will be areas that require some innovation, but luckily the methodology choice has helped with that in the past, and more than likely will in the future.

5. Implementation and development

5.1. Introduction

This chapter will deal with the implementation of the designs that were created in the design chapter. It will also continue on with most of the progress made in the prototyping stage. This chapter will also give more information on how the different functions from the OpenCV libraries are being used, and how they work. This chapter will also go into detail on the steps to installing OpenCV on the Raspberry PI.

5.2. Installing OpenCV

In the last chapter, when openCV was installed it was just a few packages that were needed for simple tasks. This time however, it will be more beneficial to install the complete package. There are a lot of tutorials online that give examples and step by step guides on how to install it. The book, Learning OpenCV recommends going to SourceForge as the most up to date versions are kept there. [3]

Since this project is using the Noobs operating system on the raspberry pi, the unix version of OpenCV is the one that is required. Using this link for SourceForge. [34] They are a clear number of options when it comes to installing OpenCV. Version 2.4.9 was chosen to work with as there is a large community on the web that has had experience with it in the past. Another reason for this choice was due to the fact that most of the literature written about OpenCV, is assuming the user has version 2.4.

5.2.1. Required Packages

There are other dependences that are required before installing OpenCV. They are listed on their webpage. [35]. Here is the list as of 2016.

- GCC 4.4.x or later
- CMake 2.6 or higher
- Git
- GTK+2.x or higher, including headers (libgtk2.0-dev)
- Pkg-config
- Python 2.6 or later and Numpy 1.5 or later with developer packages (python-dev, python-numpy)
- ffmpeg or libav development packages: libavcodec-dev, libavformat-dev, libswscalse-dev

5.2.2. Optional Packages

According to OpenCV webpage, there are optional packages that are available to download. These will not be downloaded for this project as there is no point adding extra packages that will not be used.

- Libdc1394 2.x
- Libjpeg-dev, libpng-dev, libtiff-dev, libjasper-dev, libdc1394-22-dev

Once the required packages above were installed, the openCV zip file was downloaded using a wget command. Then it was possible to follow the steps form a source online. [36] The install time was about an hour, to an hour and a half. There was a progress bar that would indicate where the install was, once it reached 100% the setup was finished and the OpenCV libraries were now open to use on the Raspberry Pi.

```
[100%] Built target example_ocl_squares
Linking CXX executable ../../bin/ocl-example-stereo_match
[100%] Built target example_ocl_stereo_match
Linking CXX executable ../../bin/ocl-example-tvl1_optical_flow
Linking CXX executable ../../bin/ocl-example-surf_matcher
[100%] Built target example_ocl_tvl1_optical_flow
[100%] Built target example_ocl_surf_matcher
user@snf-36237:~/openCV/opencv-2.4.9/build$ sudo make install
```

Figure 31- Installing OpenCV

Once the make install command was run, OpenCV version 2.4.9 would be available to the user.

5.3. Updating Raspberry Pi 2 B+

After OpenCV 2.4.9 was installed, the Raspberry Pi was updated. This was to ensure that the latest software was being used and to also limit the possible configuration errors that could pop up in the future. The commands that were ran on the Raspberry Pi are shown below.

```
pi@raspberrypi ~ $ sudo rpi-update
pi@raspberrypi ~ $ sudo apt-get upgrade
pi@raspberrypi ~ $ sudo apt-get update
```

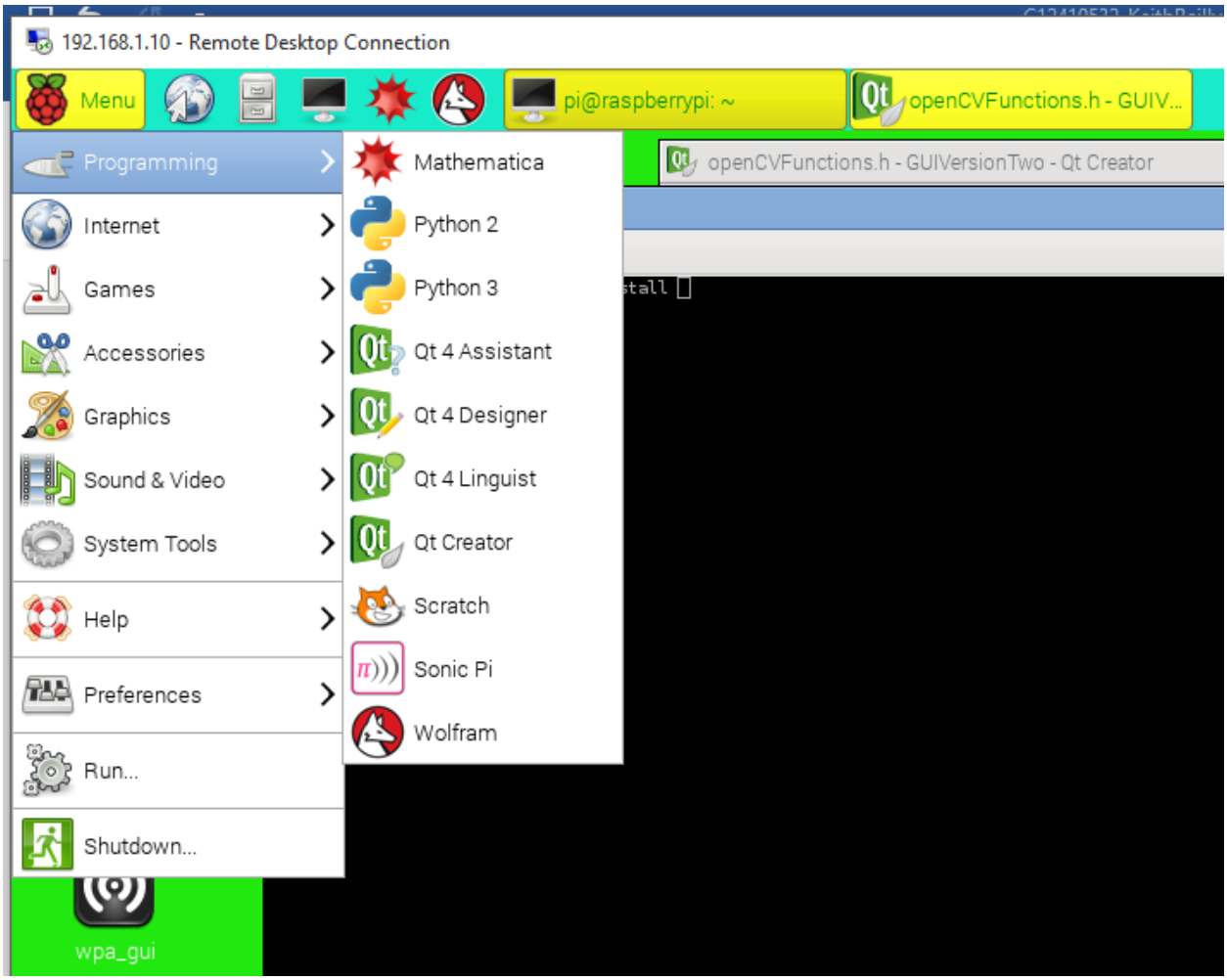
- **rpi-update,** this command performs a firmware update. This allows the web cameras to be used.
- **apt-get upgrade,** this will check all the current packages and install any new updates.
- **apt-get update,** this will update a list with new packages that are available for download.

5.3.1. Setting up QT Creator

To make the development of the GUI easier, QT was installed onto the Raspberry Pi. The QT Creator makes building GUIs very simple. It is as simple as drag and drop. You can drag a button or label onto the main window and it will create the required code for that object. However, there were some configuration settings that needed to be configured to allow it to be used with OpenCV. The commands to install QT are, sudo apt-get install qt4-dev-tools and sudo apt-get install qtcreator.

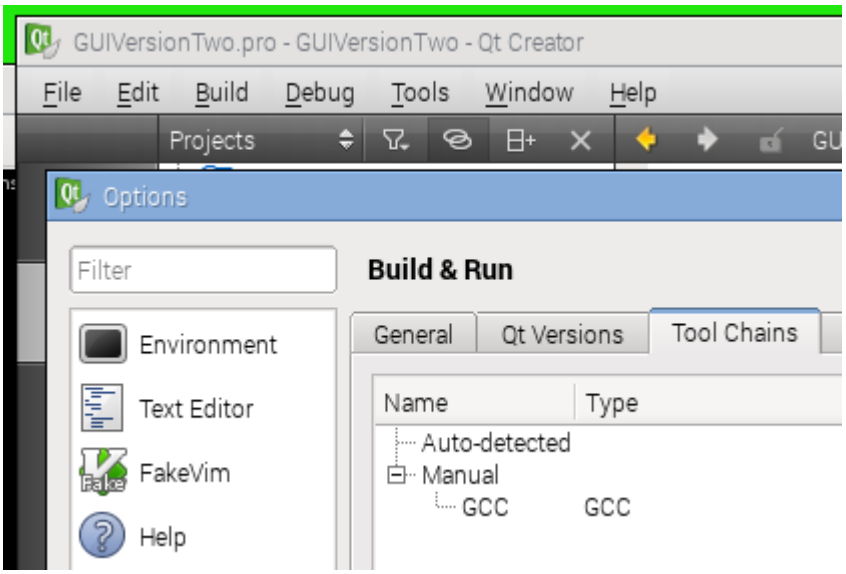
```
pi@raspberrypi ~ $ sudo apt-get install qt4-dev-tools
```

```
pi@raspberrypi ~ $ sudo apt-get install qtcreator
```



Now that QT was installed on the Raspberry Pi, it had to be configured to allow it to compile code, and use libraries that were just installed though OpenCV. The configuration is fairly simple, you go into the tools menu and select options. Then you go to QT Versions tab and manually add the path to the installed QT version on the disk. Then you go to the Tool Chains

tab and manually add the path to the gcc compiler. Once this is done the last step is to edit the pro file that is created when you start a new project. The only thing you have to do to this file is add in the paths to the libraries from OpenCV that were just installed. Once this step was finished, QT was ready to be used for development.



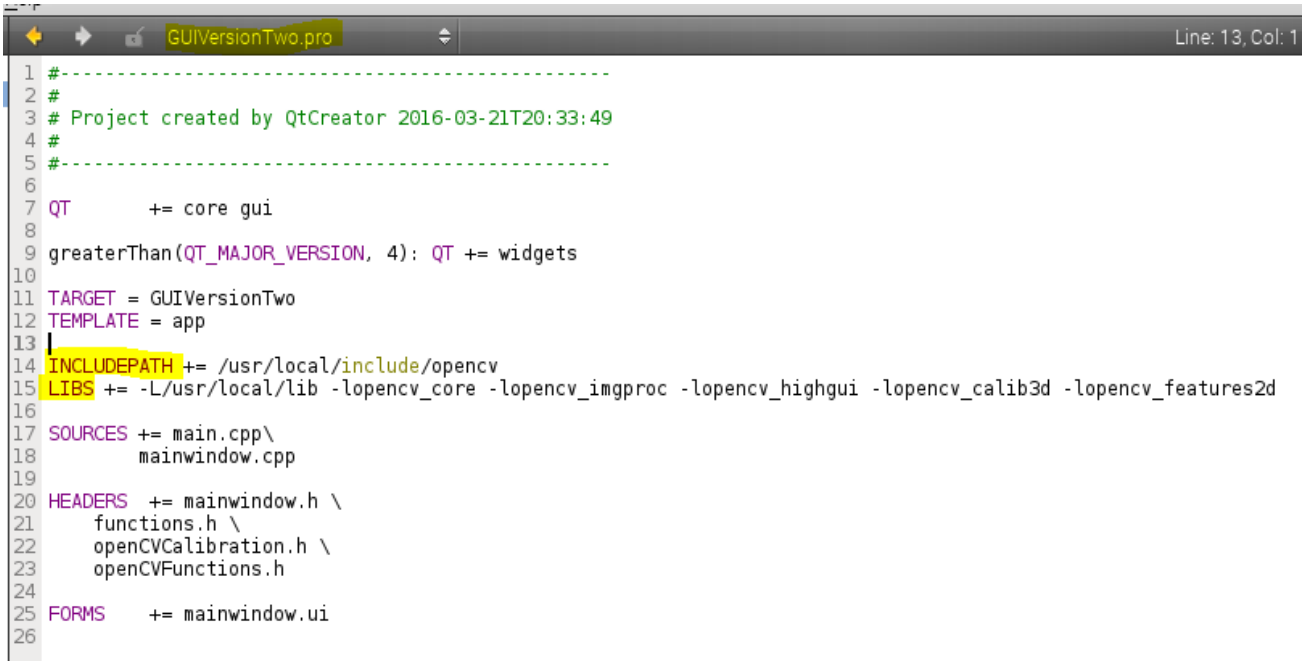


Figure 32- Configuring OpenCV in QT

5.4. OpenCV Modules

Below is a brief description of the openCV modules that are being used in this project, these were added to the .pro file in QT. Theses descriptions can be found on the openCV website. [37]

- **Core** – A compact module defining basic data structures, including the dense multi-dimensional array Mat and basic functions used by all other modules.
- **Imgproc** – An image processor module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversation, histograms, etc.
- **HighGui** – An easy to use interface to video capturing, image and video codecs, as well as simple UI capabilities.
- **Calib3d** – Basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence algorithms, and elements of 3D reconstruction.
- **Features2D** – Salient feature detectors, descriptors, and descriptor matchers.

5.5. Taking Pictures

To take pictures using the two web-cameras. You first need to get the device numbers. The command to do achieve that is `ls -al /sys/class/video4linux/`. The output of this command will show you the devices numbers that you will need. These numbers will be the same each time you turn on the raspberry pi, unless you add more web cameras and connect them through the USB ports.

The image above shows the output of this command when it was run on the Raspberry PI. As you can see we get the device numbers 0 and 1. This will be used in the C++ code when using the constructor VideoCapture(NUM).

Once we know the numbers of the devices we can start coding. The first thing we want to do is set the devices up, and name them appropriately so that we know which one is the left camera and which one is the right camera.

Here is the C++ code that retrieves the video footage from the cameras. As you can see the width and height has been set on the footage. The reason for this is that the default frames sizes where 680x600. This caused performance issues such as stuttering. Lowering the frames size solved that issue. The if statement is checking to see that the devices that were set up can be opened, i.e. are detected. If it fails it will display a message to the user letting them know that there was an error with the cameras, before it closes the program.

```
// Testing OpenCV
// Open up two cameras (Left and Right)
VideoCapture capRight(1); // open the Right camera
VideoCapture capLeft(0); // open the Left camera

// I had to set the height and width
// as any-thing above 640x480 is unstable
capRight.set(CV_CAP_PROP_FRAME_WIDTH,440);
capRight.set(CV_CAP_PROP_FRAME_HEIGHT,360);

capLeft.set(CV_CAP_PROP_FRAME_WIDTH,440);
capLeft.set(CV_CAP_PROP_FRAME_HEIGHT,360);

// Check that they are both running
if(!capRight.isOpened() && !capLeft.isOpened() ){
    printf("Cameras Failed to Open");
    return;
}
```

```
// Create two frames for both cameras
Mat frameRight;
Mat frameLeft;
```

Once the capture devices are set up along with the frame sizes. The next step is to create two matrixes that can store color images. Mat, will allow the user to do just that. Create two different frames, one for each camera and then pass the frame being retrieved for the video capture device into each matrix. Then use the imwrite constructer to save the picture. When using this command, a name and file type must be passed in along with the Mat that is storing the picture you want to save. i.e. cv::imwrite(fileOne.ppm, frame);

In this example, two windows are being created. The idea behind this is that you will need to see the frames that are being saved. To show the picture in the window, you need to pass in the window name that will store it, along with the matrix or path that is storing the picture.

```
capRight >> frameRight; // get a new frame from camera Right
capLeft >> frameLeft; // get a new frame from camera Left

// do any processing
cv::imwrite(nameRight.str(), frameRight);
cv::imwrite(nameLeft.str(), frameLeft);

// Display the images to the user afer
// Create windows to display images
namedWindow( "Left Image", WINDOW_AUTOSIZE );
namedWindow( "Right Image", WINDOW_AUTOSIZE );

// now give windows paths to the images
imshow( "Left Image", frameLeft);
imshow( "Right Image",frameRight);
```

Here are examples of images that were created using the above method.

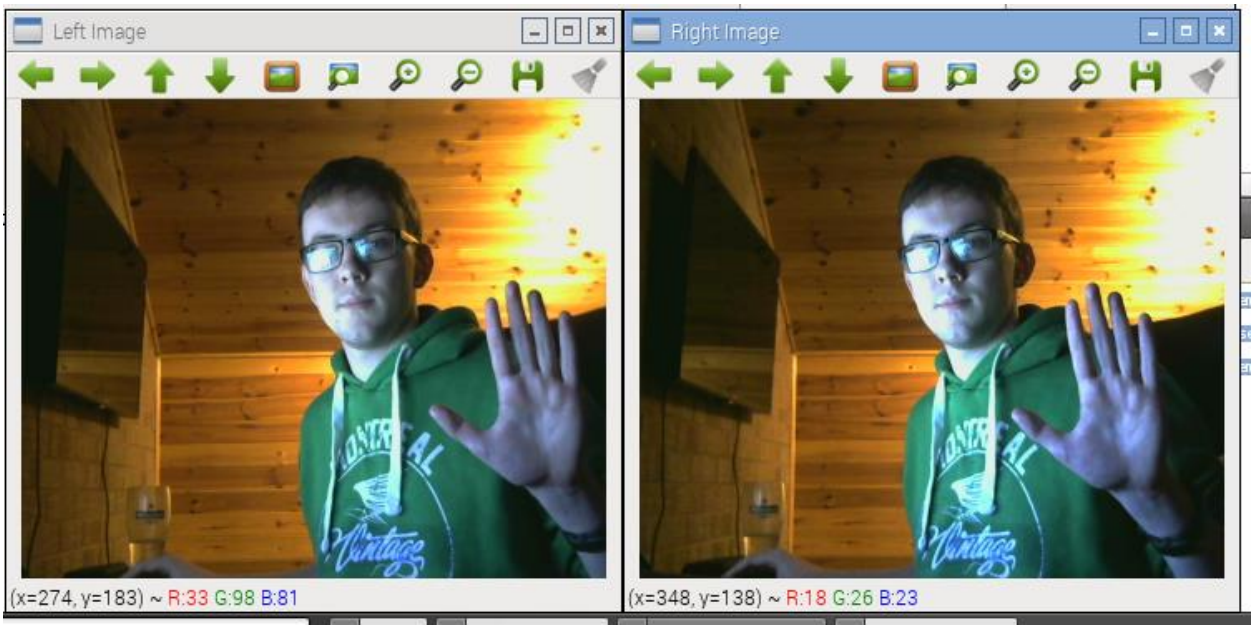


Figure 33- Taking Pictures Example

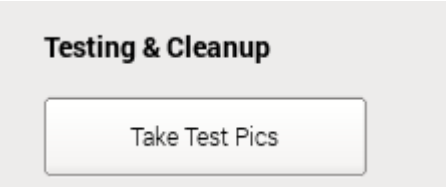
```
// release mem once
// I am finished with it
frameLeft.release();
frameRight.release();
capRight.release();
capLeft.release();
```

Once the code has executed, then it is time for cleanup. “Once we are through with an image, we can free the allocated memory.” [3] The way to free up the allocated memory is to run the command release on the mat, or device that you were using. This is very important as this project will be taken multiple picture while calibrating, if this code was executed in a loop and the memory was never cleared. It would more than likely lead to errors in processing speed and possible a crash.

In this project there are two different ways to take pictures, the user can just test the web cameras, and take one picture per device. The other options are to take a number of pictures that the user specifies. This is to make it easier on the user when calibrating the cameras, this will be explained in more detail in section 5.7 Calibrating Cameras. This section will just explain how the test pictures are taken, with examples of the code.

When the user clicks the Take Test Picture on the GUI. The on_takeTestPics_clicked function is called. This function then calls the takePic() function that is found in the openCVFunctions.h header file. When it calls takePic(“input”), it passes in a string that will be used in the naming of the files. In this case it will be “Test”. Then inside the header file, the input string will be used when the imwrite constructor is called. This will create two files Left_Test_Image.ppm and Right_test_Image.ppm, these images will both be stored inside the path /home/pi/openCV_Project/Images/.

The pictures below are examples of the code, that were being referenced to in the previous paragraph. These two test images are used in another function, but that will be explained in greater detail in section 5.8 Disparity Map.

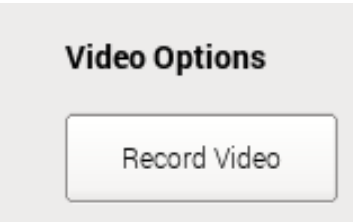



```
void MainWindow::on_takeTestPics_clicked()
{
    // This will allow the user
    // to test the cameras to
    // make sure both are working
    takePic("Test");
}
```

```
// This function will take a pic with
// the left/right camera and display
// the output to the user
void takePic(const std::string& input){

// Set Names to test or date
// also store in the correct directory
nameRight << "/home/pi/OpenCV_Project/Images/Right_"<< input << "_.ppm";
nameLeft << "/home/pi/OpenCV_Project/Images/Left_"<< input << "_.ppm";
```

5.6. Recording Video

To record video from both web-cameras, you first have to set them up in a very similar way to taking pictures. You can look at how the cameras were set up in the last section if you want to refresh your understanding of it. When the user clicks the button on the GUI, it calls the start startRecordingCams() function.




```
void MainWindow::on_RecordVid_clicked()
{
    // This function will
    // record a small clip
    // to test that record
    // feature is working
    // (add option to allow user to pass in time limit)
    startRecordingCams();
}
```

The function startRecordingCams() will set up the cameras in a near identical way to when they are taking pictures. However, there naming of the videos is not left up to the user. The videos are given their name based on the date that the button in the GUI was clicked.

```
std::string dateName = currentDateTime(); // getCurrentDateString
```

A string called dateName is created, this string will be used to store the results from the function currentDateTime(). This function that is being called is located in the functions.h header file.

```
// Get current date format is YYYY_MM_DD.HH:mm:ss
const std::string currentDateTime() {
    time_t     now = time(0);
    struct tm  tstruct;
    char       buf[80];
    tstruct = *localtime(&now);

    // date/time format
    strftime(buf, sizeof(buf), "%Y_%m_%d.%X", &tstruct);

    return buf;
}
```

The above code is what is being used to get the current date. Information on strftime can be found on this site Link To SIte , “Converts the date and time information from a given calendar time to a null-terminated multibyte character string str according to format string format. Up to count bytes are written.” Help from this source (Link to source)was also used when creating this function.

Once the current date is returned, it is then used in the naming of two video files, one for each web-camera. The files are type avi, this is due to OpenCV only supporting that video type, “OpenCV for video containers supports only the *avi* extension”. [VideoWriterInfo]

```
// Set Names to date
// also store in the correct directory
std::string dateName = currentDateTime(); // getCurrentDateString
nameRight << "/home/pi/OpenCV_Project/Videos/CamCapture_"<< dateName << "_Right.avi";
nameLeft << "/home/pi/OpenCV_Project/Videos/CamCapture_"<< dateName << "_Left.avi";
```

The code above gives you an example of the date being retrieved, along with the naming of the avi files. The next step is to call the class VideoWriter. This will allow the input of the cameras to be saved in video format. In the example below, you can see the there are two classes called one for each camera.

```
// Set up video writers
VideoWriter Rightwriter(nameRight.str(),CV_FOURCC('M','J','P','G'),10,
                        Size(capRight.get(CV_CAP_PROP_FRAME_WIDTH),
                             capRight.get(CV_CAP_PROP_FRAME_HEIGHT)),true);
VideoWriter Leftwriter(nameLeft.str(),CV_FOURCC('M','J','P','G'),10,
                       Size(capLeft.get(CV_CAP_PROP_FRAME_WIDTH),
                            capLeft.get(CV_CAP_PROP_FRAME_HEIGHT)),true);
```

The codec chosen to be passed into the video writer was MJPG or better known as Motion JPEG. This is where each individual video frame is compressed into a JPEG image. There are a

wide range of codecs that can be used, here is a link to a list of them List of Codecs , the two reasons behind this chosen codec are.

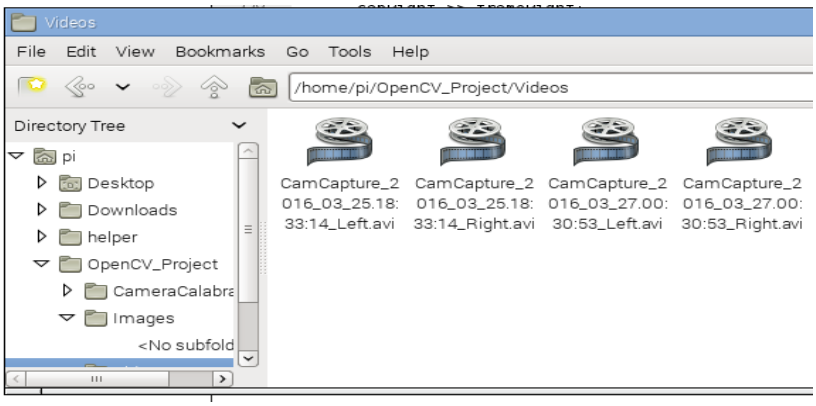
- It is useful when using low end hardware as it is not computationally intensive.
- It does not experience the same quality loss that other compression schemes using interframe compression do.

If you pay attention to the last image you will also see the number 10. This number represents the FPS(Frames Per Seconds) that the video is been written in. The reason it is set to 10 is that the web-cameras being used are both recording at 15 frames per second. If the number 30 was used when setting up the video Writer. The video would appear to be moving faster than in real life once it was played back. The next thing that is passed in is the frame size. These were set up along with the capture devices. So the frames are 440x360. The true at the very end on that line of code is for color. False for no color, true for yes there is color.

```
for( ;; )
{
  capRight >> frameRight;
  capLeft >> frameLeft;

  Rightwriter.write(frameRight);
  Leftwriter.write(frameLeft);

  imshow("LeftFrame", frameLeft);
  imshow("RightFrame", frameRight);

  char c = (char)waitKey(33);

  if( c == 27 ) break;
}
```

The code in the image to the left, is how the video file is created. The frames from both capture devices are moved into two different matrixs. These two frames are then written to the video files. Each frame will go to the correct file, as in all the left frames will go to the left avi file while the right goes into the right avi file. The frames that are being saved are also being displayed to the user. There is also an if statement that is checking whether or not the user has hit the q key. Once the user hits that key, the if statement will break out of the loop and writing frames to the video files.

The video files are stored in the following path /home/pi/OpenCV_Project/Videos. This path will be used later, but that will be cover in more detail in section 5.8.



5.7. Calibrating Cameras

The camera calibration was made easier through the use of the book Leaning OpenCV[3], as there is an entire chapter dedicated to it. This was extremely useful throughout this development stage. Another useful source was Martins Peris's blog[32] which gave examples on how to save and store the calibration results. The reason behind the calibration is that it is very easy to work out the disparity when two images that have epipolar lines going throw each of them in the same space.

Before writing any code, an object was needed to be used for the calibration. The author chose to use a simple picture of a chessboard, as easy to read patterns are recommended. "any appropriately characterized object could be used as a calibration object, yet the practical choice is a regular pattern such as a chessboard" [3] The chessboard image used for this project was taken on the openCV site [39]

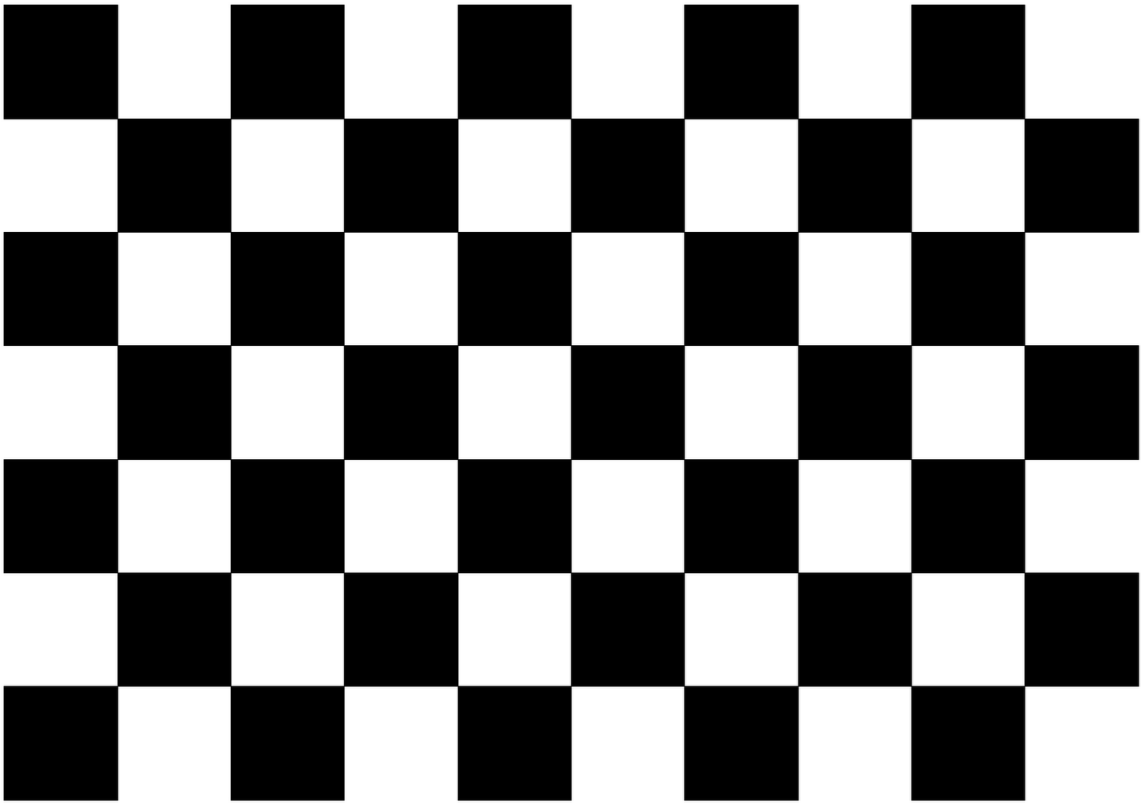
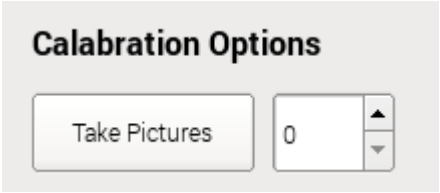


Figure 34- Chessboard

The chessboard is 9x6 and when printed on A4 paper, each square is 2.5 cm in width. This chessboard was printed out and taped onto the front of shoebox lid. This made it much easier to handle, which would be necessary as pictures of it in many different positions were needed.

Below are pictures of the chessboard that was used for this project. There are multiple prints of A4 paper on top of each other to stop the paper looking transparent under bright light.

Once the chessboard was printed, the code had to be updated to allow the user to take multiple pictures. So a spin box was added to the main window on the GUI. This spin box allowed the user to enter in a number. This number would represent the number of pictures that both cameras would take. The more picture of the chessboard, the more accurate the calibration.



The spin box was placed beside the button for obvious reasons. If the user does not increase the spin box and clicks the Take Pictures button, nothing will happen as there is an if statement checking the value.

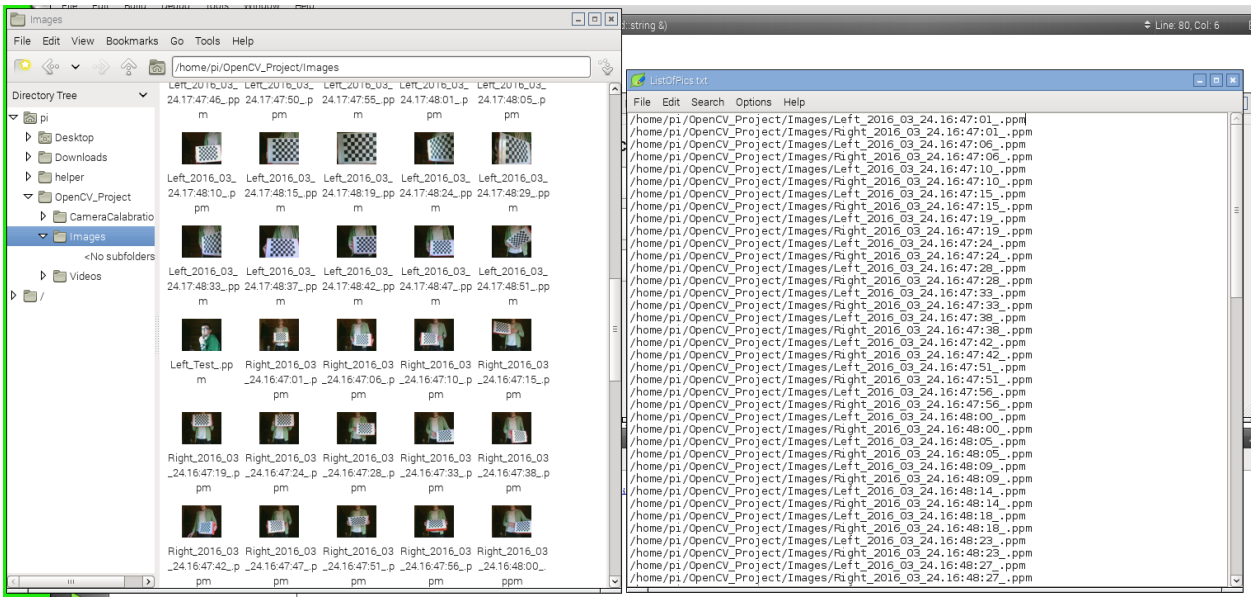
```
void MainWindow::on_buttonTakePictures_clicked()
{
    // Get the number of pics that the user has chosen
    QString spin = ui->spinBoxNumOfPics->text();

    // Only display message if user
    // has increased the spin box
    if ( spin.toInt() != 0 ) {

        //Debug message - checking number of pics being taking
        QMessageBox::information(this,tr("Info"),tr("#Pics: %1").arg(spin));

        // Take number of pics based on users choice
        for( int i = 0 ; i < spin.toInt() ; i = i + 1){
            Sleeper::sleep(3); // Sleep for five seconds
            std::string input = currentDateTime(); // getCurrentDateString
            takePic(input); // pass in dateString
        } // end of for loop

    }

}
```

In the above example of code, we can see that once the Take Pictures button is clicked. The value inside the spin box is stored inside a string. This string is then used to display a message to the user letting them know how many picture they are taking. This string is then converted to an int inside the for loop. This is because it is used with the count i. Once the count is greater than the users' choice, the for loop will end.

At the very beginning of the for loop there is a sleep function being called. This is to give the user time to position themselves with the chessboard in clear sight of both cameras. The number passed into the sleep function is the number of seconds that the loop will pause for. After that we can see there is a string called input being set to the current time. This string will then be passed into the takePics function. This string will be used in the name of these images, it is very similar to the method in which the video files are named. To keep track of all the files that are being created, a text file is created. All the names are then passed into this text file.

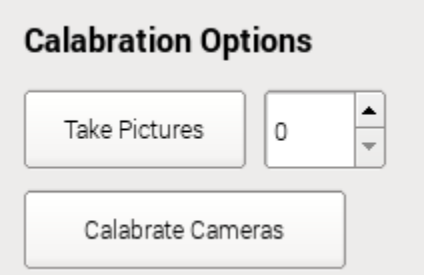
```
// If test is not in input
// then wright names to ListFile
// This will be used later for calabration
if( input != "Test" ){
    // write to TXT file
    // Path to results directory
    std::ofstream toFileTXT("/home/pi/OpenCV_Project/ListOfPics.txt", std::ios_base::app);
    toFileTXT << nameLeft.str();
    toFileTXT << "\n";
    toFileTXT << nameRight.str();
    toFileTXT << "\n";
}
```

As you can see in the image above, there is a check to see if the input passed into the takePic() function is a string containing the word Test. If it is not, then it opens up a text file named ListOfPics.txt. This text file is then used to store all the files names that are being created. This text file will be used later on in the calibration.



Once all the pictures are taken and the list file contains all their names. The web-cameras can be calibrated. A button was added to the main window of the GUI. This button would then call the calibration function, when it was clicked.

```
void MainWindow::on_calibrateCams_clicked()
{
    // This function will use the
    // ListOfPics file to calibrate
    // the cameras
    StereoCalib();
}
```



The calibration code is stored in the header file openCVCalibration.h. Most of this code was taken from the book Learning OpenCV. On page 446 of that book, all the code that's needed is there to calibrate two cameras. There were some small changes made to the code however. Instead of passing in a text file that contained the paths to the images. A string was hardcoded in with the path already set. The size of the chessboard was also hardcoded in. So nx was set to 9 and ny to 6. useUncalibrated was hardcoded to 1 as it was true for this project. The image below shows these changes.

```
// Chessboard settings
int nx = 9;
int ny = 6;
const float squareSize = 2.5; //Chessboard square size in cm
int useUncalibrated = 0;
```

Another slight change that was made to the code was some much need error checking. Rather than just ignoring any issues with images, code was added that would write the image name causing issue, to a text file. This text file could then be used to remove all the bad images that were causing errors. The user would then be able to run the calibration again, once they had removed all the bad images.

```
//FIND CHESSBOARDS AND CORNERS THEREIN:
for( int s = 1; s <= maxScale; s++ )
{
    IplImage* timg = img;
    if( s > 1 )
    {
        timg = cvCreateImage(cvSize(img->width*s,img->height*s),
            img->depth, img->nChannels );
        cvResize( img, timg, CV_INTER_CUBIC );
    }
    result = cvFindChessboardCorners( timg, cvSize(nx, ny),
        &temp[0], &count,
        CV_CALIB_CB_ADAPTIVE_THRESH |
        CV_CALIB_CB_NORMALIZE_IMAGE);
    if( timg != img )
        cvReleaseImage( &timg );
    if( result || s == maxScale )
        for( j = 0; j < count; j++
    {
        temp[j].x /= s;
        temp[j].y /= s;
    }

    // Now if there was an error with the
    // iamges and the chessboard was not
    // found, I will create a list of all
    // the images that are causing issues
    if( result )
        break;
    else {
        // name of ppm file >> errorLog
        errorLog(buf);
        // This will cause the programt to stop before calibration
        errorImgFlag = true;
    }
}
```

int result

If you look at the last image, you will be able to see an else statement added at the end of the last if statement inside the for loop. This is where the name of the file passes into the errorLog function. This function then writes that name to a text file called CalibrationPicsErrors. This file is placed beside the other text file ListOfPics so that it is easy to find. Then an error flag is set to true, this will stop the calibration once it goes throw all the images to give the user time to remove the ones that are causing the errors.

```
// IF BY CALIBRATED (BOUGUET'S METHOD)
if( useUncalibrated == 0 )
{
    CvMat _P1 = cvMat(3, 4, CV_64F, P1);
    CvMat _P2 = cvMat(3, 4, CV_64F, P2);
    cvStereoRectify( &_M1, &_M2, &_D1, &_D2, imageSize,
        &_R, &_T,
        &_R1, &_R2, &_P1, &_P2, &_Q,
        0/*CV_CALIB_ZERO_DISPARITY*/ );
    isVerticalStereo = fabs(P2[1][3]) > fabs(P2[0][3]);

    //Precompute maps for cvRemap()
    cvInitUndistortRectifyMap(&_M1,&_D1,&_R1,&_P1,mx1,my1);
    cvInitUndistortRectifyMap(&_M2,&_D2,&_R2,&_P2,mx2,my2);

    //Save parameters
    cvSave("/home/pi/OpenCV_Project/CameraCalabration/M1.xml",&_M1);
    cvSave("/home/pi/OpenCV_Project/CameraCalabration/D1.xml",&_D1);
    cvSave("/home/pi/OpenCV_Project/CameraCalabration/R1.xml",&_R1);
    cvSave("/home/pi/OpenCV_Project/CameraCalabration/P1.xml",&_P1);
    cvSave("/home/pi/OpenCV_Project/CameraCalabration/M2.xml",&_M2);
    cvSave("/home/pi/OpenCV_Project/CameraCalabration/D2.xml",&_D2);
    cvSave("/home/pi/OpenCV_Project/CameraCalabration/R2.xml",&_R2);
    cvSave("/home/pi/OpenCV_Project/CameraCalabration/P2.xml",&_P2);
    cvSave("/home/pi/OpenCV_Project/CameraCalabration/Q.xml",&_Q);
    cvSave("/home/pi/OpenCV_Project/CameraCalabration/mx1.xml",mx1);
    cvSave("/home/pi/OpenCV_Project/CameraCalabration/my1.xml",my1);
    cvSave("/home/pi/OpenCV_Project/CameraCalabration/mx2.xml",mx2);
    cvSave("/home/pi/OpenCV_Project/CameraCalabration/my2.xml",my2);

}
else
    assert(0);
```

Another change made to the code adding in cvSave. This was to save all the calibration parameters into XML files. This way the files could be loaded into code later on and used again. Here are the definitions of some of the parameters seen above in the code.

- Camera matrix _M
- Distortion Vector _D
- Rotation Matix _R
- Translation Vector _T
- Essential Matrix _E
- Fundamental Matrix _F
- 3-by-4 left and right projection equations _P
- 4-by-4 reprojection matrix _Q

Now that certain values are hardcoded in, along with some extra code to save the calibration results. It was clear to run the stereo calibration. Once the calibration is started it will display each image after it has been converted to greyscale. The reason it converts the image to grey scale is that It makes it much easier to identify patterns. The image should display the lines that were automatically drawn once the chessboard was detected. If there are only red lines on the chessboard it means that it was not found. If that happens then that file name will be added to the error log.

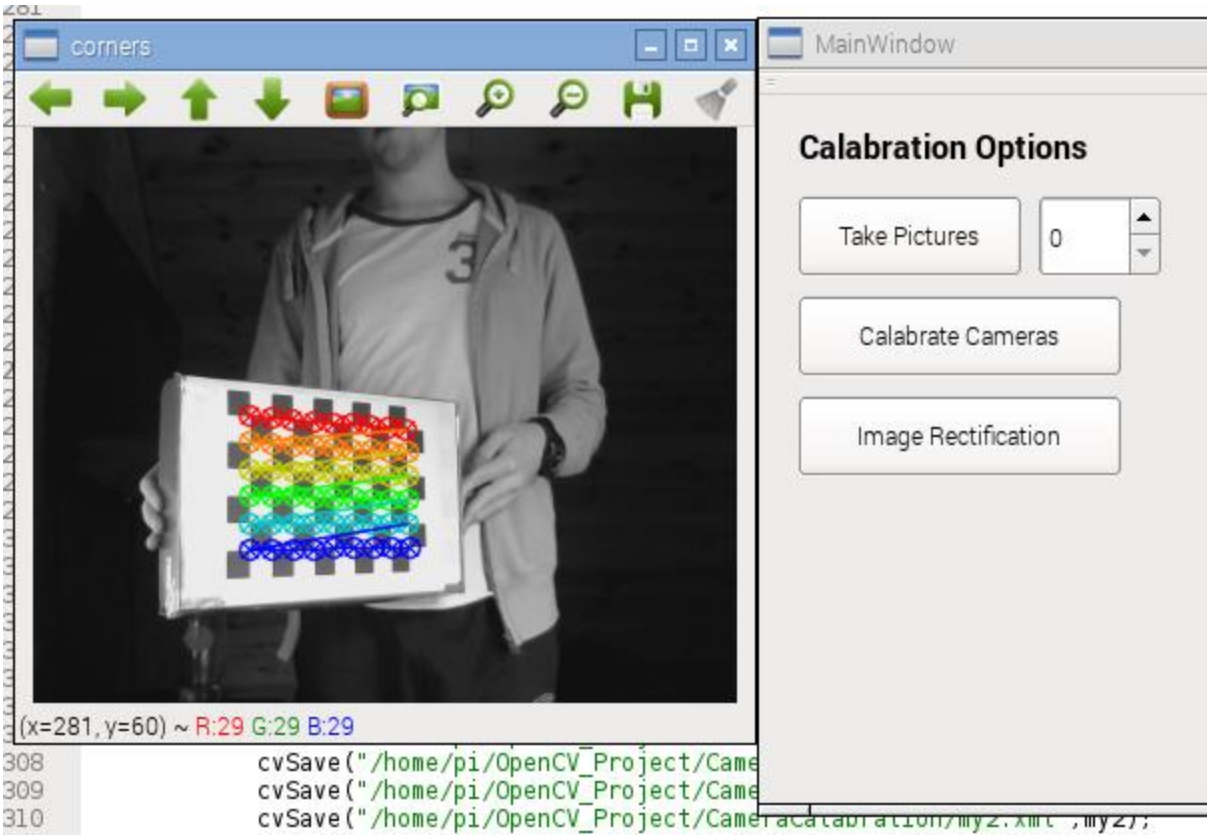
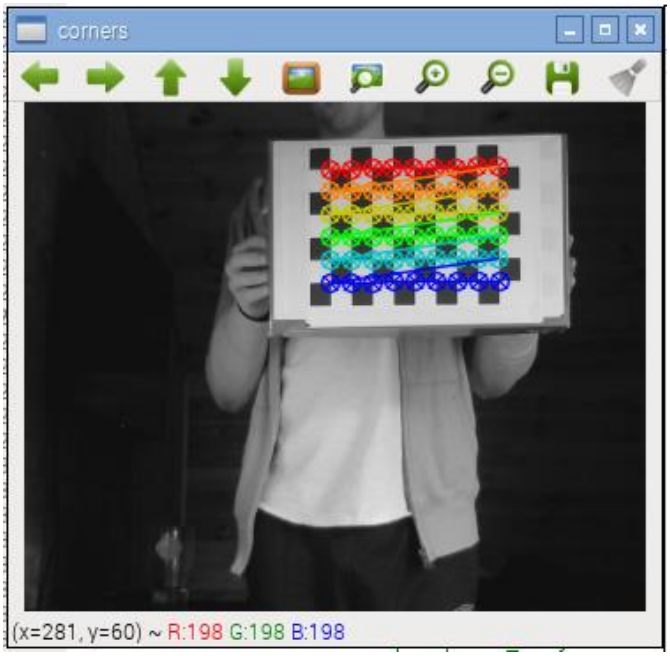
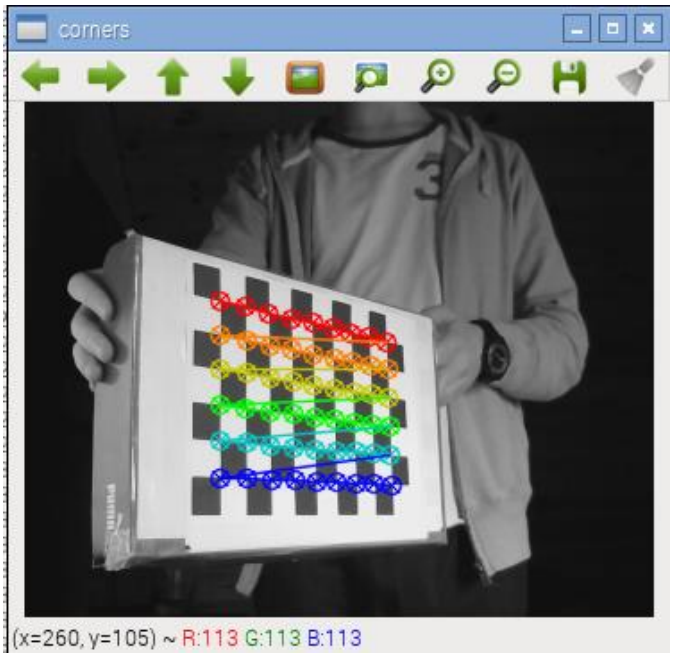


Figure 35- Example of calibration

In these pictures you can see the output of the program and how it finds the chessboard. This process is very time consuming and depending on the amount of images you are calibrating could take an hour to two hours to finish. There is also a picture of the XML files that are saved at the end of this process. These are the files that can be used over and over again, unless of course you move the cameras.





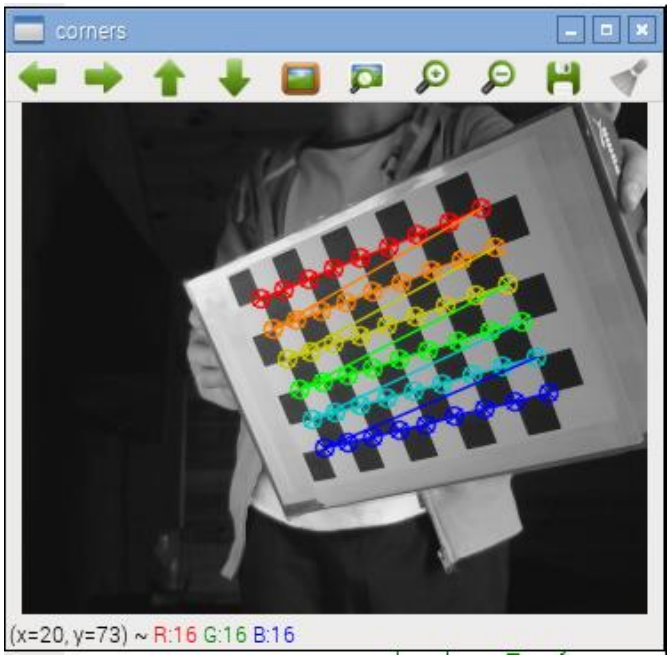


Figure 36- Calibration XML Results

Once you have these XML files you have officially calibrated your cameras. Now it is possible to move on to creating the disparity map. However, to confirm that the XML files are somewhat accurate, it is important to view the rectified image. A rectified image should show both images together with lines going from one side to the other. These lines should be on all the same objects in both images, but they should be in different positions on the x axis.

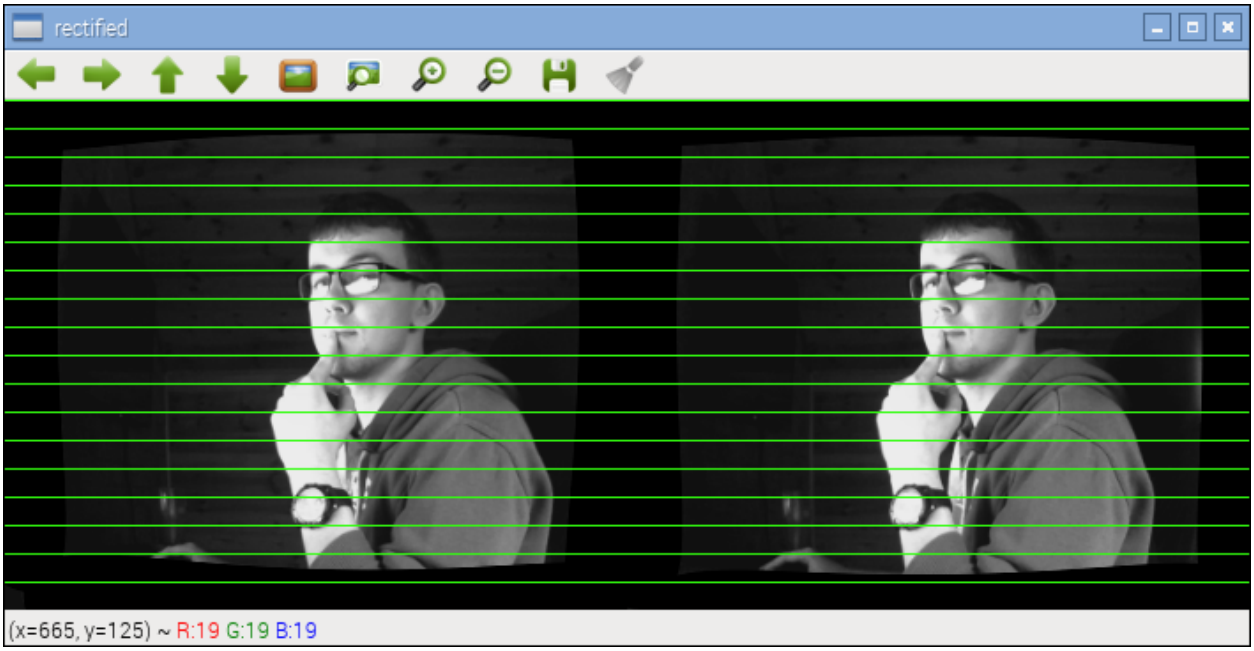


Figure 37- Example of Rectified Image

Here is the output of the ratified image, the two test pictures that it is using are below it. As you can see there is some blackness creeping onto the screen, but overall it is a decent result. The lines are matching up on the left image to the right image. If we look at the bridge of the nose in both images, it is clear the line is hitting the same position. These are the expected results for such low end equipment.

```
// Load the calabration settings
CvMat *mx1 = (CvMat *)cvLoad("/home/pi/OpenCV_Project/CameraCalabration/mx1.xml",NULL,NULL,NULL);
CvMat *my1 = (CvMat *)cvLoad("/home/pi/OpenCV_Project/CameraCalabration/my1.xml",NULL,NULL,NULL);
CvMat *mx2 = (CvMat *)cvLoad("/home/pi/OpenCV_Project/CameraCalabration/mx2.xml",NULL,NULL,NULL);
CvMat *my2 = (CvMat *)cvLoad("/home/pi/OpenCV_Project/CameraCalabration/my2.xml",NULL,NULL,NULL);

// 360/440 image size Height/Width
// CV_8U -unsigned 8bit pixel
// CV_16S - 16 bit
CvMat* img1r = cvCreateMat( imageSize.height,imageSize.width, CV_8U );
CvMat* img2r = cvCreateMat( imageSize.height,imageSize.width, CV_8U );
CvMat* disp = cvCreateMat( imageSize.height,imageSize.width, CV_16S );
CvMat* vdisp = cvCreateMat( imageSize.height,imageSize.width, CV_8U );
CvMat* pair;
CvMat part;

cvNamedWindow( "rectified", 1 );

pair = cvCreateMat( imageSize.height, imageSize.width*2,CV_8UC3 );

IplImage* img1=cvLoadImage("/home/pi/OpenCV_Project/Images/Left_Test_.ppm",0);
IplImage* img2=cvLoadImage("/home/pi/OpenCV_Project/Images/Right_Test_.ppm",0);


cvRemap( img1, img1r, mx1, my1 );
cvRemap( img2, img2r, mx2, my2 );

cvGetCols( pair, &part, 0, imageSize.width );
cvCvtColor( img1r, &part, CV_GRAY2BGR );
cvGetCols( pair, &part, imageSize.width,imageSize.width*2 );
cvCvtColor( img2r, &part, CV_GRAY2BGR );
for( j = 0; j < imageSize.height; j += 16 ) // Draw Lines
    cvLine( pair, cvPoint(0,j),cvPoint(imageSize.width*2,j),CV_RGB(50,255,15));

// Show with lines
cvShowImage( "rectified", pair );
```

Here is the code that produced the rectified image on the last page. At the top you can see how the save configuration files are now loaded into matrix's that can be used in the program. Under that you can see how some other matrixes being initialized, for now ignore disp and vdisip. These will be explained in section 5.8 Disparity Map. Pair is being set to twice the width as it need to store two images. Then we use the two test images that were taken earlier.

cvRemap is then used to transform the images using the saved configurations from earlier.

cvRemap(inputArray src, OutputArray dst, InputArray map1, InputArray map2)

- Src: source image
- Dst: Destination image
- Map1: First map of x,y points
- Map2: Second map of x,y points

Then the two images are spliced together, first the left image and then the right image. After that the only thing left is to draw the lines that go from the left image over to the right image. Then the rectified image is displayed to the user.

5.8. Disparity map

Now that the cameras are calibrated, the results are stored, and we are happy with the output of the rectified image. It is now time to get the disparity image/map. The code is very similar to the code in section 5.7 Calibrating cameras. However, it is missing a few small changes that make all the difference.

```
// 360/440 image size Height/Width
// CV_8U -unsigned 8bit pixel
// CV_16S - 16 bit
CvMat* img1r = cvCreateMat( imageSize.height,imageSize.width, CV_8U );
CvMat* img2r = cvCreateMat( imageSize.height,imageSize.width, CV_8U );
CvMat* disp = cvCreateMat( imageSize.height,imageSize.width, CV_16S );
CvMat* vdisp = cvCreateMat( imageSize.height,imageSize.width, CV_8U );
CvMat* pair;
CvMat part;
```

The matrixes highlighted in yellow are the ones that you will need to pay attention here, while the other ones are not important for creating a disparity map.

```
//Setup for finding stereo corrrespondences
CvStereoBMState *BMState = cvCreateStereoBMState();
assert(BMState != 0);
BMState->preFilterSize=41;
BMState->preFilterCap=31;
BMState->SADWindowSize=41;
BMState->minDisparity=-64;
BMState->numberOfDisparities=128;
BMState->textureThreshold=10;
BMState->uniquenessRatio=15;
```

CVStereoBMState is where block matching parameters and the internal scratch buffers are kept. There is a lot of good documentation fond on this in Learning OpenCV page 443 [3]. I will give brief explanations of each of these numbers.

- preFilterSize – Window size of the prefilter
- preFilterCap – Sets the cap to x, so its 0-x
- SADWindowSize – The sums of the distances are calculated to find corresponding pixels
- minDisparity – Smallest disparity, default is 0, should be set to negative number
- numberOfDisparities – This is the search range of disparities
- textureThreshold – only calculate disparity, where texture is larger than
- uniquenessRatio – accept computed disparity d

```
IplImage* img1=cvLoadImage("/home/pi/OpenCV_Project/Images/Left_Test_.ppm",0);
IplImage* img2=cvLoadImage("/home/pi/OpenCV_Project/Images/Right_Test_.ppm",0);


cvRemap( img1, img1r, mx1, my1 );
cvRemap( img2, img2r, mx2, my2 );

cvFindStereoCorrespondenceBM( img1r, img2r, disp,BMState);
cvNormalize( disp, vdisp, 0, 256, CV_MINMAX );

// Show the disparity image
cvNamedWindow( "disparity" );
cvShowImage( "disparity", vdisp );
```

So in the code above, we can see that the test images are being loaded into img1 and img2. Then we use the cvRemap function to transform the images. These are the same steps that were done in the last sections while getting the rectified image. The next part is where things differ. cvFindStereoCOrrespondenceBM is used to dinf the disperity of the images that were passed into it. It then places the results into disp. Then cvNormalize normalizes the elements in the arrays disp and vdisp.

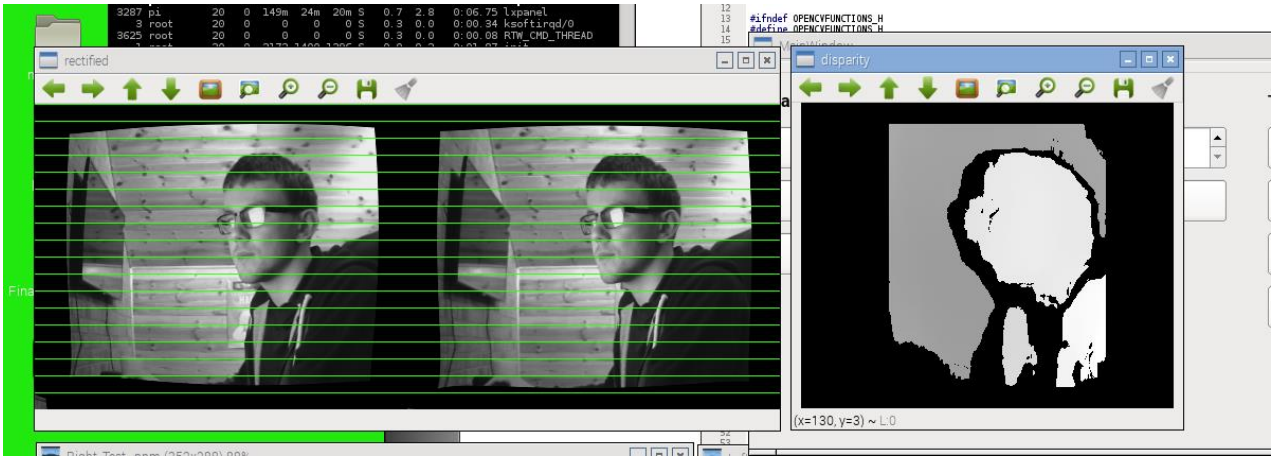


Figure 38- Example of Disparity Map

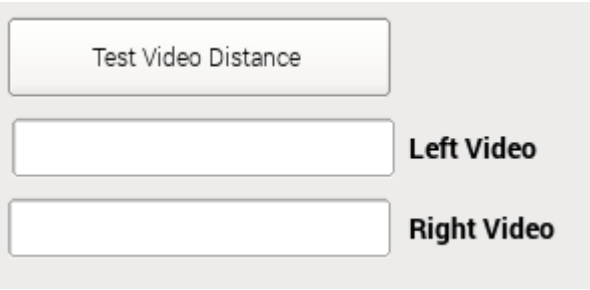
Above is an example of the disparity map beside the rectified image. Both of these have used the same two test images.

5.9. Getting Distance

In the last three sections of this chapter, we went over recording video, calibrating the cameras and also getting the disparity map. In this section we will combine all of those together to make it possible for the distance to be found. The first thing we have to do is load up all the saved configuration results.

```
void testDistanceTwo(const std::string& Left, std::string& Right){
    printf("\nLoading Calabrations...\n");

    Mat Q, mx1,my1,mx2,my2;
    Q = (CvMat *)cvLoad("/home/pi/OpenCV_Project/CameraCalabration/Q.xml",NULL,NULL,NULL);
    mx1 = (CvMat *)cvLoad("/home/pi/OpenCV_Project/CameraCalabration/mx1.xml",NULL,NULL,NULL);
    my1 = (CvMat *)cvLoad("/home/pi/OpenCV_Project/CameraCalabration/my1.xml",NULL,NULL,NULL);
    mx2 = (CvMat *)cvLoad("/home/pi/OpenCV_Project/CameraCalabration/mx2.xml",NULL,NULL,NULL);
    my2 = (CvMat *)cvLoad("/home/pi/OpenCV_Project/CameraCalabration/my2.xml",NULL,NULL,NULL);


    // Get the names of the last two video files created
    std::string FilenameLeft = "/home/pi/OpenCV_Project/Videos/" + Left;
    std::string FilenameRight = "/home/pi/OpenCV_Project/Videos/" + Right;

    //VideoCapture
    VideoCapture leftVidCap(FilenameLeft);
    VideoCapture rightVidCap(FilenameRight);
    cout << "LeftVid: " << FilenameLeft << " <|> RightVid:" << FilenameRight << endl;
```

Once this is done, we now need to find the last two videos that the user has recorded. These will be provided by the user. These names would have been entered into the text boxes provided on the GUI. When the user clicks the button in the image below, two strings will be set to the text entered. These are the strings that are passed into the function that is finding the distance. Unless of course the user enters nothing, then when he clicks the button a message will prompt him to enter in a file name.




```
// Get the name of the left vid the user entered
std::string leftVid = (ui->lineEditLeft->text()).toStdString();

// Get the name of the Right vid the user entered
std::string rightVid = (ui->lineEditRight->text()).toStdString();
```

Once we have the name of the last two files, we have to set up the video capture use these files as its source. This is the exact same approach as setting up the cameras, however this time, instead of passing in a device number. We are passing in the path to the avi file. Then we check that both of these can be opened and there are no errors with the files themselves.

```
VideoCapture LeftVid("/home/pi/OpenCV_Project/Videos/CamCapture_2016_04_04.03:17:25_Left.avi");
VideoCapture RightVid("/home/pi/OpenCV_Project/Videos/CamCapture_2016_04_04.03:17:25_Right.avi");
```

We then set up all the necessary matrixes to store the frames from both the left and right video. We also set up the StereoBMState, in the exact same way we did for both the rectified map and the disparity map.

```
 // Get two frames ready
 Mat frameRight;
 Mat frameLeft;


// get other mats ready for disparty
CvMat* img1r = cvCreateMat( imageSize.height,imageSize.width, CV_8U );
CvMat* img2r = cvCreateMat( imageSize.height,imageSize.width, CV_8U );
CvMat* disp = cvCreateMat( imageSize.height,imageSize.width, CV_16S );
CvMat* vdisp = cvCreateMat( imageSize.height,imageSize.width, CV_8U );



//Setup for finding stereo corrrespondences
CvStereoBMState *BMState = cvCreateStereoBMState();
assert(BMState != 0);
BMState->preFilterSize=41;
BMState->preFilterCap=31;
BMState->SADWindowSize=41;
BMState->minDisparity=-64;
BMState->numberOfDisparities=128;
BMState->textureThreshold=10;
BMState->uniquenessRatio=15;
```

Once all the above is set up we can now grab the frames from the videos and pass them into the matrixes we initialized at the beginning. Then we create two files, one for the left and the one for the right frame.

```
LeftVid >> frameLeft;
RightVid >> frameRight;

// do any processing
cv::imwrite("frameRightVid.ppm", frameRight);
cv::imwrite("frameLeftVid.ppm", frameLeft);

// Write the images to files
IplImage* img1=cvLoadImage("frameLeftVid.ppm",0);
IplImage* img2=cvLoadImage("frameRightVid.ppm",0);
```

Once we have the frames from the video saved, we can now create the disparity map. Once we have it we will then be able to work out the distance to a certain object or area from the cameras.

```
// In here I will compute
// the disparity map
cvRemap( img1, img1r, mx1, my1 );
cvRemap( img2, img2r, mx2, my2 );

cvFindStereoCorrespondenceBM( img1r, img2r, disp,BMState);
cvNormalize( disp, vdisp, 0, 256, CV_MINMAX );
```

The code above is how the disparity map is worked out. This is the stage where we now have to start adding in some new code. The first thing we have to do is create two matrixes. One while will store the pointcloud and another that will store an exact copy of the vdisp(I.e., disparity map).

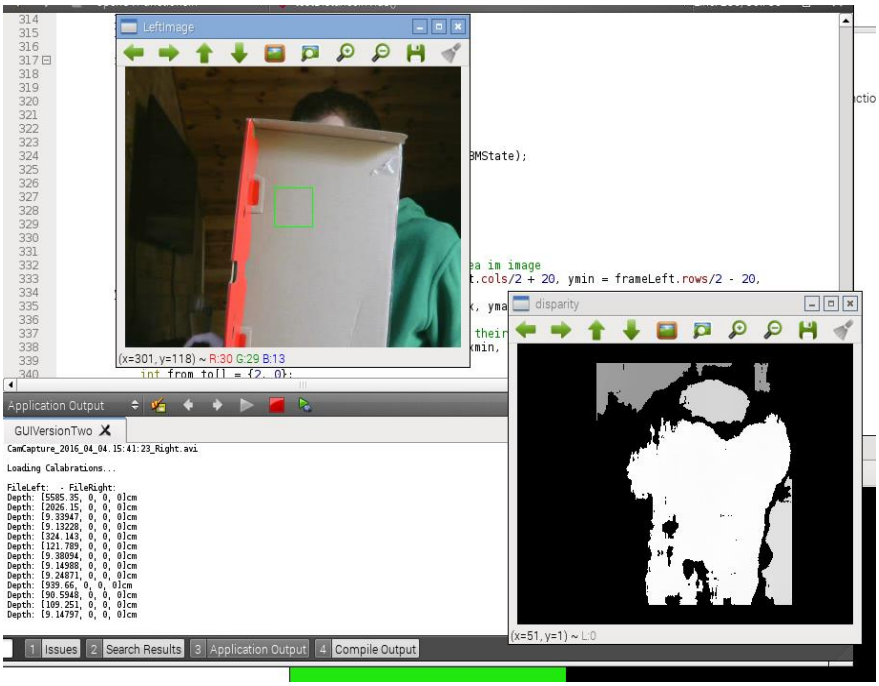
Then we can call the openCV class reprojectImageTo3D (**disparity**, **_3dImage**, **Q**, **handelMissingValues**) to store the xyz results inside the pointcloud. So what you can see it does this by taking both the disparity and the disparity to depth matrix Q results that were stored earlier.

```
Mat pointcloud,zdisp;
zdisp = vdisp;
// Calculate 3D co-ordinates from disparity image
reprojectImageTo3D(zdisp, pointcloud, Q, true);
```

Now that the xyz values are stored inside the pointcloud, we can try to work out with the distance of a certain area of in the image. The method that is being used for this project is quite simple. A green box is drawn in the center of the screen; it is 40 pixels in overall size. The x and y values that were used to create this box are then used again, except this time there are used to extract the depth of the 40-pixel area from the pointcloud.

```
    // Draw red rectangle around 40 px wide square area im image
    int xmin = frameLeft.cols/2 - 20, xmax = frameLeft.cols/2 + 20, ymin = frameLeft.rows/2 - 20,
ymax = frameLeft.rows/2 + 20;
    rectangle(frameLeft, Point(xmin, ymin), Point(xmax, ymax), Scalar(0, 255, 0));

    // Extract depth of 40 px rectangle and print out their mean
    pointcloud = pointcloud(Range(ymin, ymax), Range(xmin, xmax));
    Mat z_roi(pointcloud.size(), CV_32FC1);
    int from_to[] = {2, 0};
    mixChannels(&pointcloud, 1, &z_roi, 1, from_to, 1);

    cout << "Depth: " << mean(z_roi) << "cm" << endl;

    // Show the disparity image
    cvNamedWindow( "disparity" );
    cvShowImage( "disparity", vdisp );
```

In the code above you can see how the pointcloud is set to just the 40-pixel size and then how it gets the mean distance of z for that area. The accuracy is affected by the camera calibration, if you did not take enough pictures with the cameras then the distance results might be off.



Here on the left is an example of the code is action. The user was trying to get the distance of the green box that can be seen in the top picture. However, the results in this test were completely off and they jumped from four digits to the single digits. This was due to the way the code was written.

Figure 39- Distance Detection box example

As the results were extremely inaccurate at first. The author of this work went online and looked for examples of code where the user was retrieving the z axis. There was a book online that dealt with this exact scenario. It is called Practical OpenCV, there is a chapter in this book that gives examples of getting the distance. The code that was currently being used was not too far off, however there one or two key differences. In the book instead of using StereoBM (Block Matching), he was using StereoSGBM(semi block matching). StereoSGBM is newer and some people in the openCV community clam that it creates a better quality disparity map. "SGBM appears to produce the cleanest map" [40]

```
//Setup for finding stereo corrrespondences
StereoSGBM stereo;
stereo.preFilterCap = 63;
stereo.SADWindowSize = 3;
stereo.P1 = 8 * 3 * stereo.SADWindowSize * stereo.SADWindowSize;
stereo.P2 = 32 * 3 * stereo.SADWindowSize * stereo.SADWindowSize;
stereo.uniquenessRatio = 10;
stereo.speckleWindowSize = 100;
stereo.speckleRange = 32;
stereo.disp12MaxDiff = 1;
stereo.minDisparity=-64;
stereo.numberOfDisparities=128;
stereo.fullDP = true;
```

This is one of the changes that was made to the code, the old StereoBM was removed and replaced with this. The values where copied from the book as I was unsure what to set them as. One of the other big changes to the code was how the frames were being pulled from the video. Before they were being passed into a predefined frame, but in the book he uses the grap() call. This grabs a few frames at once. The reasoning behind this according to his code was that they would be less apart in time. Then he used the retrieve() to place the frame grabbed into a matrix. I have added a picture below that shows how it was done.

```
//grab raw frames first
capl.grab();
capr.grab();
//decode later so the grabbed frames are less apart in time
Mat framel, framel_rect, framer, framer_rect;
capl.retrieve(framel);
capr.retrieve(framer);
```

Some changes were made to the way the disparity map was created as SteroSGBM cannot be used the same way as StereoBM. Once these changes were made to the code, it was recompiled and tested again.

```
Mat disp, disp_show, disp_compute, pointcloud;
stereo(framel_rect, framer_rect, disp);
disp.convertTo(disp_show, CV_8U, 255/(stereo.numberOfDisparities * 16.));
disp.convertTo(disp_compute, CV_32F, 1.f/16.f);
```

In the image above we can see the change that was made to the code. You will notice that there are four matrixes created and three of them are just for the disparity map. The reason behind this is very simple, the value disp is used to store the results from stereoSGBM after it rectifies the left and right frame. The value disp_show is used to store the actually map the end user will see. The disp_compute value is used to store the results of the dispirity map after it is

divided by 16. This is due to the fact that StereoSGBM is scaled by 16, so to get the true disparity you must divide by 16.

Once these changes are in place and the program is run again, we see much more accurate results. These results are not the exact number although they are definitely in the correct range. There is more detail on these results in the Testing Chapter. The picture added below shows the new output after these changes.

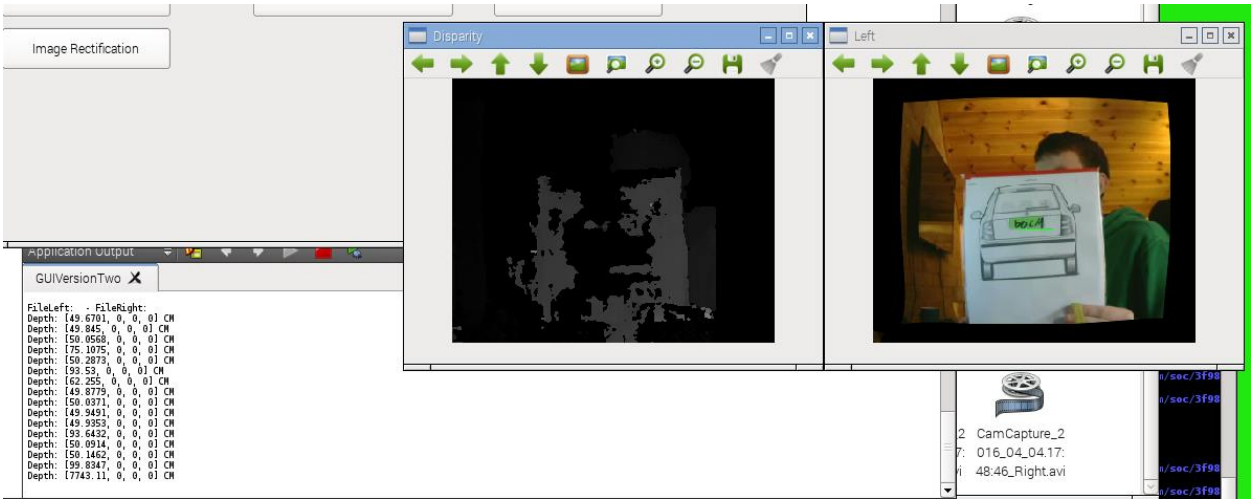


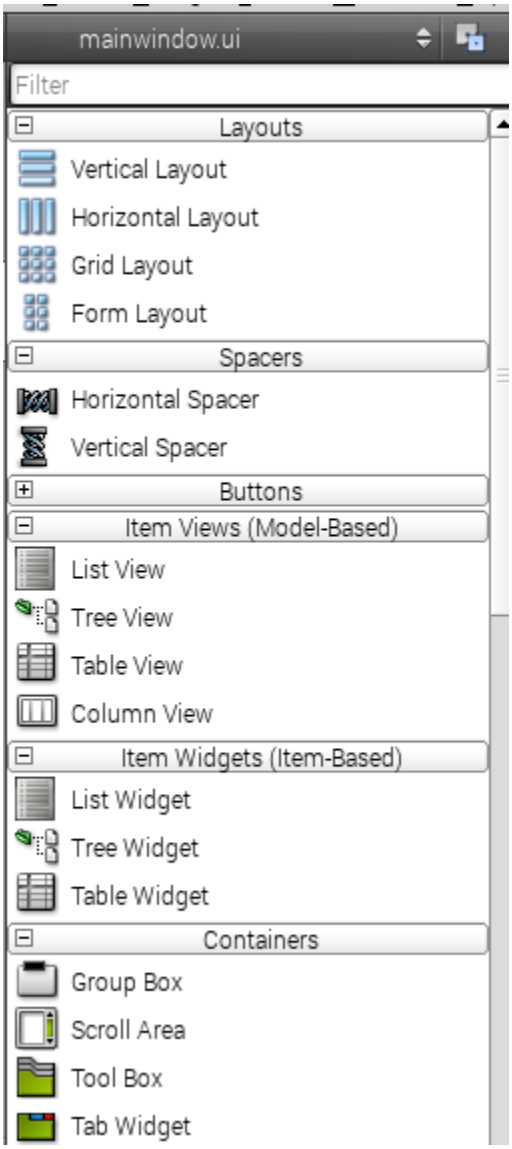
Figure 40- Getting Distance Example

```
FileLeft:  - FileRight:
Depth: [54.8461, 0, 0, 0] CM
Depth: [54.6556, 0, 0, 0] CM
Depth: [54.671, 0, 0, 0] CM
Depth: [54.4954, 0, 0, 0] CM
Depth: [54.3988, 0, 0, 0] CM
Depth: [54.6786, 0, 0, 0] CM
Depth: [54.8218, 0, 0, 0] CM
Depth: [54.6666, 0, 0, 0] CM
Depth: [79.5386, 0, 0, 0] CM
Depth: [54.962, 0, 0, 0] CM
```

Here in the figure on the left we can see that the results are all falling around 55cm. The actually distance to the picture of the car is 60cm. So in this example there is an error range of 5cm. This is defiantly a good result, based on the equipment being used.

5.10. Creating a GUI

As the project at this stage was in good shape. The author decided that it would be the best to create a GUI based off earlier designs, and link all the working features to one main GUI. QT makes creating a GUI quite easy in C++. It provides the user with a multitude of options, layouts, spacers, buttons, item widgets, containers etc. This project however would not be needing all of these. A simple and etiquette design was decided on earlier, so for now only buttons and spin boxes will be used.



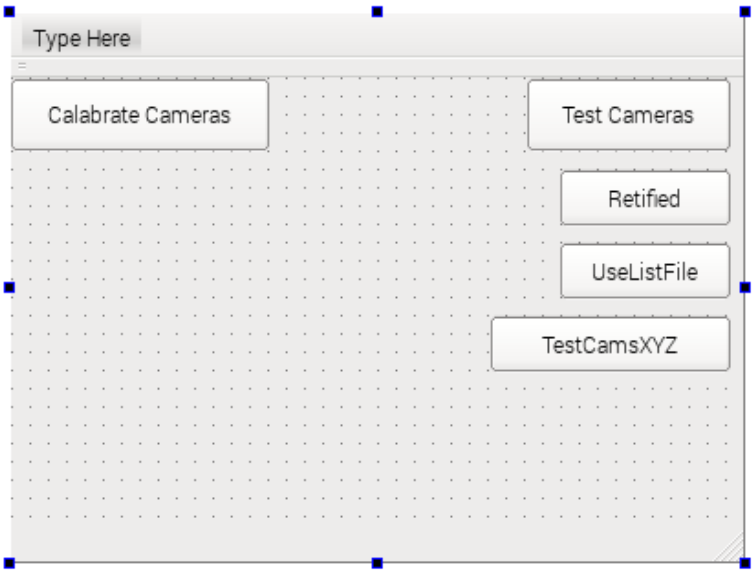


Figure 41- GUI Version One

Here is an early example of the first design, this one was leaning much more in the direction of functionality rather than appearance. I used this first design as a good learning experience on how to structure the files that were being used, and how to structure the code in the mainwindow.cpp so that it would make sense to an outside reader.

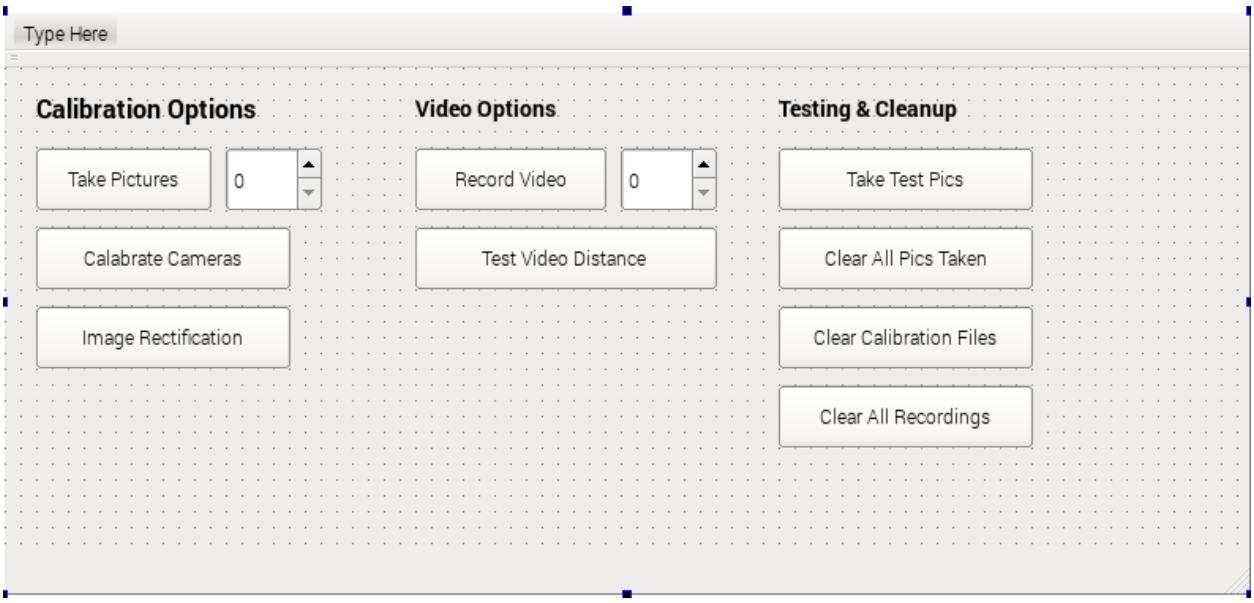


Figure 42 - GUI Version Four

This design above was the fourth design layout, as you can see there is a bit more detail, and the layout is easier to follow then the first one. This design is what lead into the actually design of the project, the image of that has been added below.

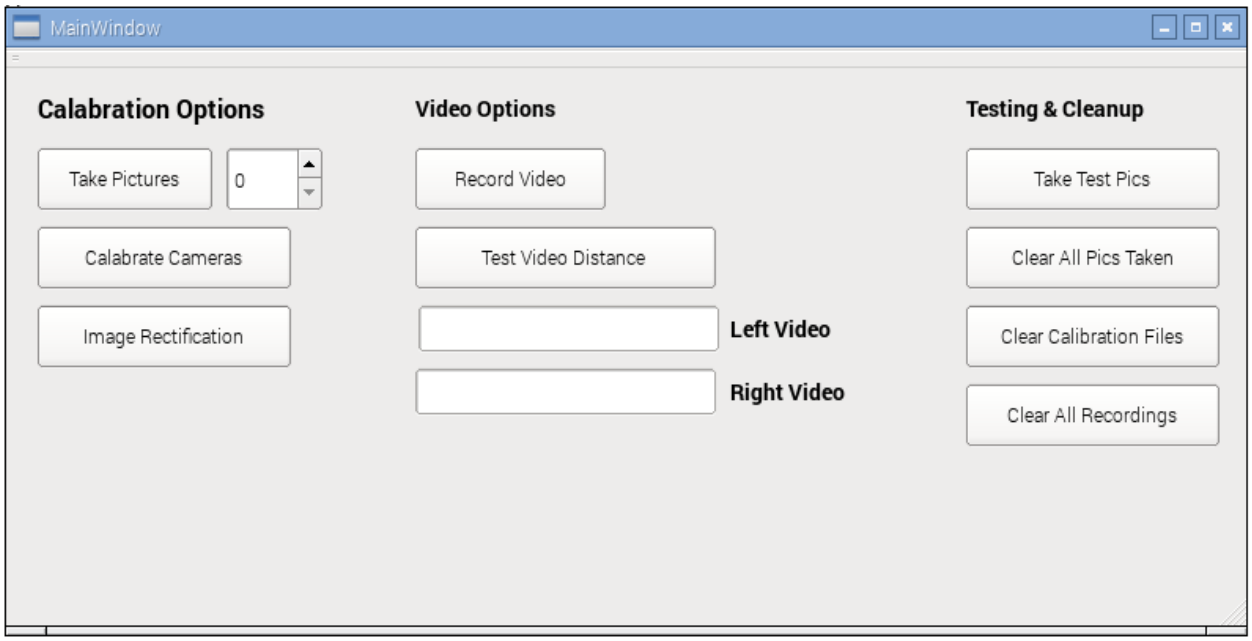


Figure 43- GUI Final Version

This is the current design that is being used for this project. There are three clear sections. There are all labeled with their purposes. If you have read all the following sections, then you will be somewhat familiar with the name of functions. The author felt it would be in the best interest of the reader if they also added an explanation of what each individual button does what.

5.10.1. Calibration Options

Take Pictures – When this button is clicked. It will take the value of the spin button next to it and then check to see if it is zero. If it is not then it will take that number of pictures, with both cameras. If it is zero, then nothing will happen. For more info on this look back to section 5.5 Taking Pictures.

Calibrate Cameras – When this button is clicked, it will call the function StereoCalib(). In that function the text file ListOfPIcs.txt will be used for calibration. More info on this can be found in section 5.7 Calibrate Cameras

Image Rectification - When this button is clicked, the function getRetified() is called. This function will display both a disparity map and a rectified image for the last two test images that were last taken. More info on these function can be found in sections 5.8 and 5.9.

5.10.2. Video Options

Record Video – When this button is clicked, both cameras will start recording (more info at Section 5.6) until the user clicks q, at this point the cameras will stop and a new avi files will be added to the videos folder.

Test Video Distance – When this button is clicked, the function testDistanceInVids( ) is called. This function will take the two video names that the user entered into the text boxes below and use them in testing. You can find out all about this in section 5.9 Getting Distance.

Text Boxes – These will store the names of the files that are going to be used when returning the distance to the user. If these files cannot be opened the user will be notified and then returned to the main GUI.

5.10.3. Testing & Cleanup

Take Test Pics – This button when clicked will turn on both cameras and take a picture. This picture is labeled test; it also overwrites the last test picture that was taken. This picture name is not added to the text file that is used for calibration as the user is only testing the cameras. More info on taking pictures can be found in section 5.5 Taking Pictures.

Clear All Pics Taken – This button is pretty self-explanatory. Once it is clicked, a system command is expected through C++. This command clears all the files inside the Images folder.

Clear Calibration Files – This button similar to the last will execute a system command that will remove all the current calibraiton results that are stored in the folder CameraCalabration.

Clear All Recordings – This button when clicked will execute a system command that will remove all the contents inside the folder Videos.

```
// Remove all contents from
// ListOfPics.txt file
void clearListOfPics(){
    std::ofstream toFileTXT("/home/pi/OpenCV_Project/ListOfPics.txt", std::ofstream::out | std::ofstream::trunc);
}

// Remove the current calibration
// files
void clearCalibrationFiles(){
    system("exec sudo rm -rf /home/pi/OpenCV_Project/CameraCalabration/* ");
}

//Removes all the pictures
// that were taken
void clearAllPicsTaken(){
    system("exec sudo rm -rf /home/pi/OpenCV_Project/Images/* ");
}

// Removs all videos recorded
void clearAllVids(){
    system("exec sudo rm -rf /home/pi/OpenCV_Project/Videos/* ");
}
```

5.11. Making the box

The cameras need a container to hold them still, this container needs to be portable and also allow for good cable management. Following the designs from earlier, a box was put together using some cheap MDF wood, and glued together to hold it all in place. Two holes were cut out in the front of the box. The cameras will be able to stick out of these holes. There is also a piece on wood holding the cameras down on the inside of the box. This allowed for more stability.

Figure 44- Box Implementation Examples

The hole in the top will allow easy access to the power and HDMI plugs. This was not in the design when plaining out the container for the box, but when faced with the decision of cable management, this seemed like a much better option then drilling holes in the box.

5.12. Issues Encountered

5.12.1. Missing Library's

During the early stages of this project, and mostly though the prototyping stage. The author of this project kept running into errors when calling certain classes and functions that were a part of the openCV libraries. This was due to fact that the openCV that was installed at the time was not an up to date version and it was not properly installed. This lead to some confusion when the author was trying to figure out if an error was due to their code or because of a missing library. This issue was resolved once openCV 2.4.9 was installed. But that itself was also quite time consuming.

5.12.2. Frame Size

The performance of the web cameras got worse over time; this was first noticed when recording video. There was a serious amount of stuttering going on with no obvious cause. This was causing issue when grabbing frames as they would not be synced up. This would then lead to the disparity map looking wrong. Once the frame sizes were lowered from 680x600 to 440x360, these problems stopped.

5.12.3. FPS & Video Codecs

The default setting when using videoWriter is 30 FPS. This caused issues when playing back the video as time appeared to be going by faster.

```
// Set up video writers
VideoWriter Rightwriter(nameRight.str(),CV_FOURCC('M','J','P','G'),10,
                        Size(capRight.get(CV_CAP_PROP_FRAME_WIDTH),
                             capRight.get(CV_CAP_PROP_FRAME_HEIGHT)),true);
VideoWriter Leftwriter(nameLeft.str(),CV_FOURCC('M','J','P','G'),10,
                       Size(capLeft.get(CV_CAP_PROP_FRAME_WIDTH),
                            capLeft.get(CV_CAP_PROP_FRAME_HEIGHT)),true);
```

Once the FPS was lowered to 10, the videos look and played normally. This was due to the fact that the web-cameras that are being used record at 10 frames per second. There was also an issue when setting the video codec here. There were a lot of times where an error message saying that a parameter was not set or could not be found would pop up. It was a bit of trial and error with each codec, but once three were found that worked. The author was able to select the one that was the best. This issue was very time consuming.

5.12.4. Devices number changing

Every so often when recording a video, the device number will change on the camera mid-recording. This will then lead to a large number of errors. This error has appeared a number of times when using remote desktop to control the raspberry pi from the

authors laptop. When the raspberry pi is connected to a television through the HDMI cable, this error does not appear. This leads the author to believe that for whatever reason, using a WIFI connection to control the raspberry pi leads to this error.

5.13. Risks

5.13.1. Hardware breaking

While working on this project, the author has been under a constant state of dread. Since he is using three pieces of hardware, two web-cameras and a raspberry pi. There is always a chance that something will break or malfunction. This would stop development as this project is so dependent on the hardware's ability to operate. This has been an incredible risk as it could cost the author an excruciating long amount of time, if any of these were to break.

5.13.2. Area I am unfamiliar with

One of the reasons why the author chooses to take on a project like this was the fact that they have very little experience in computer vision. This is a risk as they could be underestimating how difficult the task is and how time consuming it will be. Over the course of the last few months the author has become well aware of how challenging this project has turned out to be.

5.13.3. Getting the correct results

After the program is run, and the distance that it estimates from the other object is wrong. They what is there to fall back on, the author feels as though this is the most important part of the project and will place its priority above all other features until it is working.

5.13.4. Not a lot of experience with C++

This will be the first time that the author has coded in C++, at the beginning of the project he was confident in his ability to pick it up. That has proved to true over the course of the last few months. However, he has had trouble with some of the features offered by openCV as this also a new area for him. openCV has not been as easy to learn as C++, and takes a lot more time to even get the fundamentals down.

5.14. Delayed Features

Unfortunately, over the last few months, it became apparent that some of the feature were going to be pushed behind on the schedule. This was due to the complexity of setting up the cameras. The author underestimated the time that would be required to complete a lot of

these tasks. The most demanding tasks were also all the ones that were needed to be done for the distance detection to actually work.

Here are the timelines that the author was trying to follow, as you see in the second image there were a lot of features that will not be added that were planned, this is annoying as they were goals that the author expected to hit. However, the author does feel proud of the goals that he did reach as of this date. He has learned a great deal and if there were another three of four weeks, it would be possible to cross a few more tasks off the list.

Basic Timeline

Activity – Task	Dec	Jan	Feb	Mar	Apr
Get Camera Calibration Working					
Get Disparity/Depth Map Working					
Get Motion Tracking Working					
Motion Tracking **Differentiate between objects (EG are they bigger or smaller)**					
Create GUI for user to select Files					
Create GUI to show results to user					
Write results into CSV files					
Combine the Disparity Map and Motion Tracking programs together					
Write up Final Year Report					

Week + Dates	Tasks	Priority (low / mid / high / Very High)	Status (Not Star
8-14th February	Calabrate the cameras and store the results for later use	High	Completed
	Make sure that both cameras can record video in sync	High	Working On
15-21st February	Verify that the car power supply can support the Raspberry Pi	High	Completed
	Set up apache to run a C++ program (hello_world.cpp)	Mid	Working On
22-28th February	Create / Make a support to hold cameras in place	Mid	Completed
	Create frame(s) to display recorded video	Mid	Completed
29-6th March	GUI: Main create a square block to start	Mid	Completed
	GUI: Add a text box so that the user can add the path to recorded videos	Mid	Completed
7-13th March	GUI: Add a button that will start a c++ program (hello_world.cpp)	Mid	Completed
	GUI: Link the button to the start Record program	Mid	Completed
14-20th March	GUI: Add button that will display a csv file (Results)	Mid	Working On
	Detect Motion in a video, post recording	High	PushBack
21-27th March	Detect Motion in a video with motion (as in the the cameras are moving), po	High	PushBack
	Get the depth/distince of an object in a image/video	Very High	Completed
28-3rd March	Create list of rules for distance/speed user can be behind vechicals	Mid	PushBack
	Get a C++ program writing results to a csv file	Mid	Completed
4-6 th April	Combine the motion tracking and depth detection into one programe	High	PushBack
	Clean up the way the csv results are presented	Low	Stopped
	Clean up GUI where needed	Low	Completed
	Get results of number of vechicals tracked	Low	Not Started
	Get the results for MIN/MAX/AVG distance the user is from other vechicals	Low	Not Started
	Now able to write to txt file through c++	Low	Completed

Figure 45- Example of Timelines

The main features that were delayed were the motion tracking and the setting up of the apache website. All other feature was completed and are working.

5.15. Conclusion

This chapter has covered this projects complete implementation and development cycle. It has gone through each of the main features that this project can offer, taking picture, recording video, creating a rectified image, create a disparity map and finally working out an estimation of distance. All of which is on a GUI built in QT.

The methodology that was chosen at the design stage was a massive help throughout this stage as it allowed for lots of testing and quick fixes for all the small issues. Some of the more major issues however, like getting the rectified image to display correctly and getting the distance estimation took much longer.

This chapter also covered some of the risks that come with a project like this. Now that the implementation is near finished it is easier to see that must of the risks taking have payed off.

6. System Validation

6.1. Introduction

This chapter will go over the different testing strategies that were done on this project. It will also cover the different validation tests done on the features offered by the project at this point. Such things as the camera calibration and the distance estimation will have evaluation tests to show how effective they are.

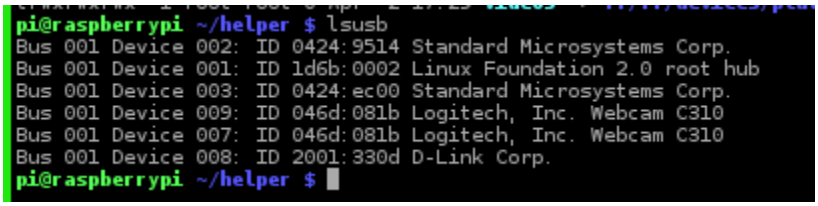
6.2. Hardware Testing

6.2.1. Raspberry PI

Testing the raspberry Pi was very straight forward, once the noobs operating system was written onto the SD card and placed into the pi. The first thing was to make sure it actually turned on. Then once it was on, each of the USB ports were checked to make sure they all work, with both cameras. Then the HDMI cable was connected from the raspberry pi to the television. Once this was verified to be working, the raspberry Pi could be placed to the side as it was ready to for use.

6.2.2. Logitech C310 webcams

Testing the webcams was fairly simple, they were both plugged into a laptop that had windows 10 running on it. Once the drivers finished installing, the cameras could be found in the device listings. The cameras were then plugged into the Raspberry Pi and after a firmware update, they appeared under the USB devices as well.



Shortly after this the web cams were used to take some pictures to confirm that they worked with the raspberry pi, once the pictures were taken, and confirmed to be saved. The cameras were confirmed to be working and ready to use.

6.3. Software Testing

There were many different approaches that could be taken in the testing of this project. This section will give a brief explanation of each along with the actually tests that were carried out.

6.3.1. White Box

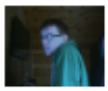
This methodology of testing is called white box; this is because the user knows what the code is doing while the test is happening. The tester is looking at and analyzing how the code is flowing and is there just to verify that it is working correctly. It is easier explained by thinking of it being a room with the light on. Now you can see all the furniture in the room. The author has added some examples of white box testing below, that are related to this project.

```
std::string input = currentDateTime(); // getCurrentDateString
cout << "Name of File will be: " << input << " :" << endl;
takePic(input); // pass in dateString
```

In the code above input is being set to the current date. This is then passed into the takePIc() function. It is used for the naming of the file. So to test that it works, a new line of code has been added right before it passes the string to takePic(). This new line of code will print out the string that was returned from currentDateTime(). The expected results for this test is that the string printed out will match the title of the new picture that it takes. The tester now runs the project and watches the output from the consol. As you can see a string has been printed.

```
Name of File will be: 2016_04_05.12:14:22 :
```

Now that the tester has the string output, he can go and check that the filename does indeed contain that very same string. Once this is confirmed then that test is considered a pass, as the expected output was there.



Right_2016_04_05.12:14:22_.ppm

```
/home/pi/OpenCV_Project/Images/Left_2016_04_05.12:14:22_.ppm
/home/pi/OpenCV_Project/Images/Right_2016_04_05.12:14:22_.ppm
```

Since it is clear that the files were created, this test can now be considered a pass. You should now have a better understanding of how these tests are being carried out, if you look at the next table, you will see other tests that were also carried out.

Table One – White Box Unit Testing

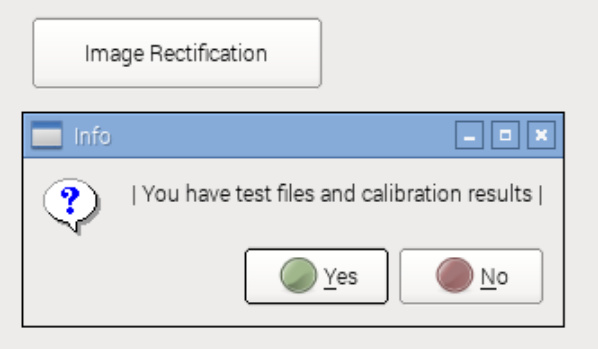
Unit Test	Description	Pass / Fail
1	Confirm that the spin box takes users the number chosen as a limit on the amount of pics taken.	Expected: Pass Results: Pass
2	Confirm that “Test” string is passed in when user only takes a test picture	Expected: Pass Results: Pass
3	Confirm that ListOfPics.txt files are updated with names of new files being created, if used chooses to take pictures	Expected: Pass Results: Pass
4	Confirm that the CalibrationPIcsErros.txt files are created if any images cause errors in the calibration	Expected: Pass Results: Pass
5	Confirm that the calibration results are stored in individual XML files	Expected: Pass Results: Pass
6	Confirm that the video files are being saved as avi file type	Expected: Pass Results: Pass
7	Confirm that the image files are of type ppm	Expected: Pass Results: Pass
8	Confirm that the path to images is /pi/home/OpenCV_Project/images/	Expected: Pass Results: Pass
9	Confirm that the path to Videos is /pi/home/OpenCV_Project/Videos/	Expected: Pass Results: Pass
10	Confirm that the path to calibration results is /pi/home/OpenCV_Project/CameraCaliabration/	Expected: Pass Results: Pass
11	Confirm that the program crashes if it tries to use a test image, when one does not exist	Expected: Fail Results: Fail
12	Confirm that program crashes if the user tries to calibrate cameras, and images listed in the text file do not exist	Expected: Fail Results: Fail
13	Confirm that the program crashes if the user tries to calibrate the cameras, and the list files empty	Expected: Fail Results: Fail
14	Confirm that the program crashes if the user attempts to test the distance estimation feature while there are no videos saved on the device	Expected: Fail Results: Fail

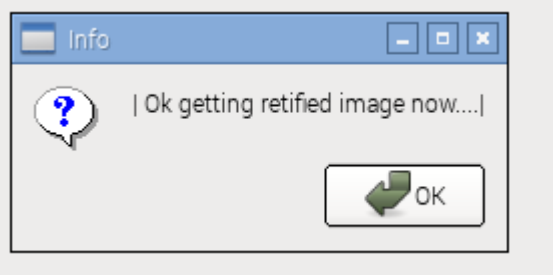
The tests above were carried out on the system, and the results can now be used to improve the overall quality of code. One of the main issues at the moment is that the program will crash when certain files and not present, when they are expected to be. The solution to this is to add a check that can be called before a feature is used. This check will ask the user if they have

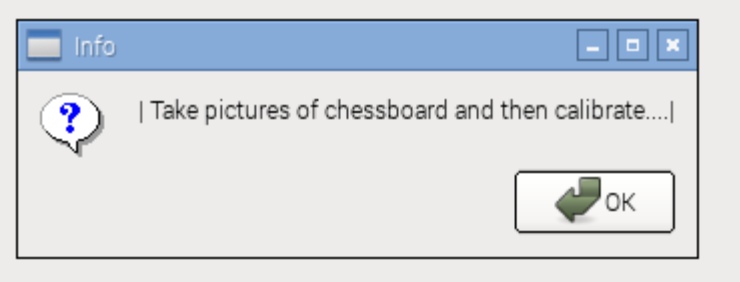
taking the necessary pictures and if they have calibrated the cameras. The user will then the option yes or no. This is not the most ideal way. It would be better if the file path was checked and then the user would be forced to complete all the necessary steps. Due to unforeseen complications with the code in this project, that was not feasible to do with the time that was left to work on this project. However, a compromise was made and now the user will at least be reminded to have all the necessary files.

```
QMessageBox::StandardButton check;
check = QMessageBox::question(this,tr("Info"),tr("| You have test files and calibration results |"),
                        QMessageBox::Yes|QMessageBox::No);

if (check == QMessageBox::Yes ){
    check = QMessageBox::question(this,tr("Info"),tr("| Ok getting retified image now....|"));
    getRetified();
}else
    check = QMessageBox::question(this,tr("Info"),tr("| Take pictures of chessboard and then calibrate....|"));
```

In the code above we can see that a message box is created asking the user if they have the correct files. They user will then respond with either yes or no. Depending on their answer, they will be brought back to the main GUI, or the program will continue and start the selected function.





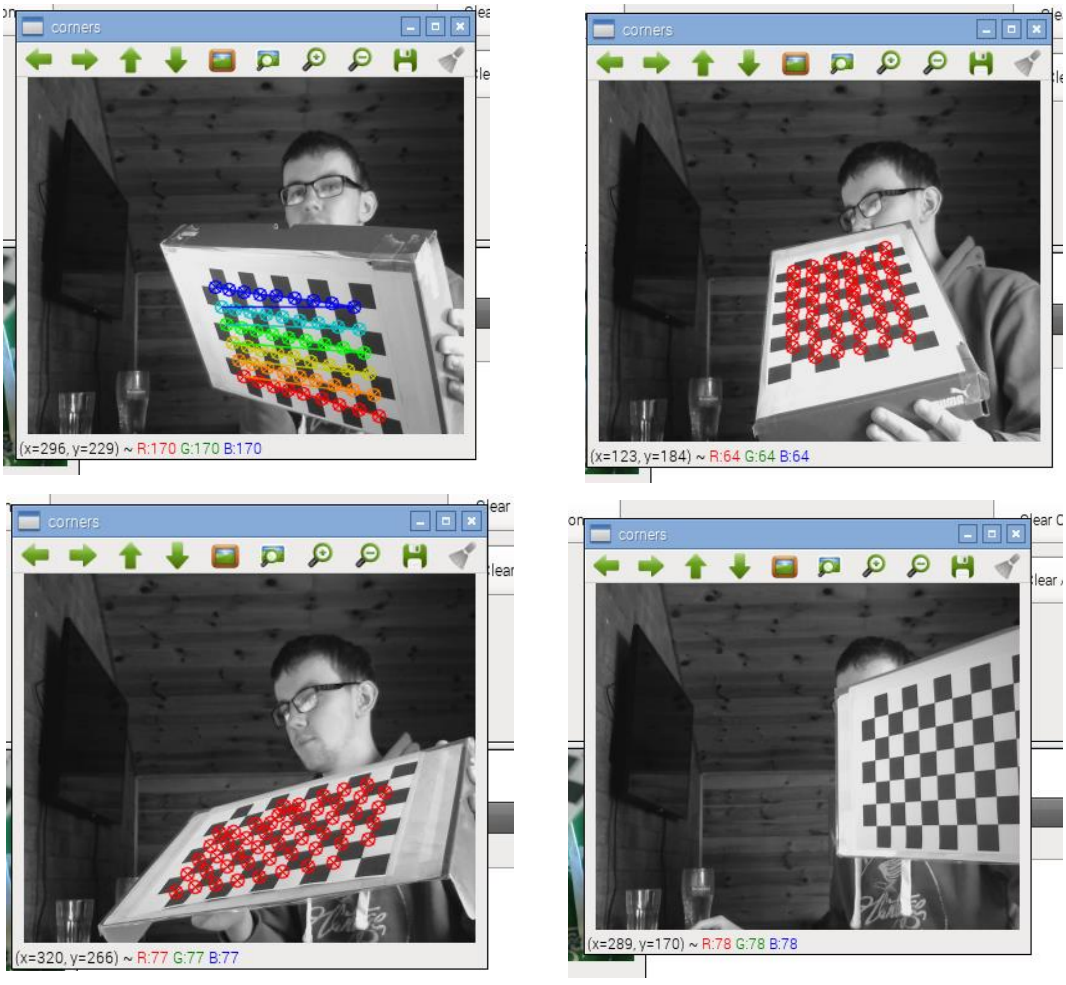


Above are all the different outputs from the buttons, top right is if the user clicked yes, while the image at the bottom of the page is when they click no. This code will be applied to other places where the user will be expected to have certain files ready.

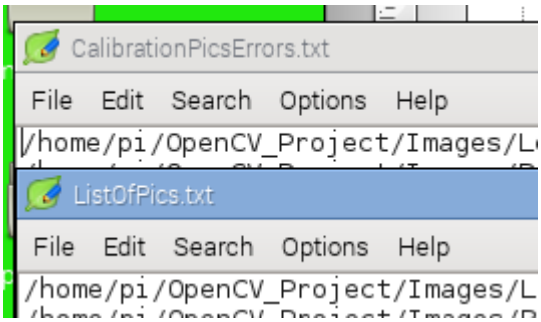
6.3.2. Black Box

This Methodology of testing is call black box testing; this is due to the user not knowing what the code is doing with the code as the test is carried out. The user understands what settings he has put in place, but when testing he is just sending in inputs and expects certain outputs. There is a similar way of thinking about it, just like white box. Imagine that you are in a room, but the light is turned off. You know the furniture is in the room but right now you can't see it.

Most of the black box testing that was done for this project, was during the calibration faze. You can read more about that in section 5.7 calibrating cameras. The way it was tested was simple. A large number of pictures were taken, around 60 per camera. So overall they were 120 pictures. The calibration was started with some simple parameters in place that were expected to catch any errors, and also write the name of any images that caused errors to a text file. Once the calibration is started, it is left to run until it completes, or stops due to errors.



The images in the last page, are the output of the calibration results. The top left image is what is to be expected while the other images are not expected. Two of the images contains lots of red dots, this is due to the findchessboard() call not being able to find all the points in the image. Then there is an image where it did not even detect the image. The code that is currently in place is expected to get the names of all the images that did not work and add them to a file. This way the user can just remove them manually and restart the calibration.



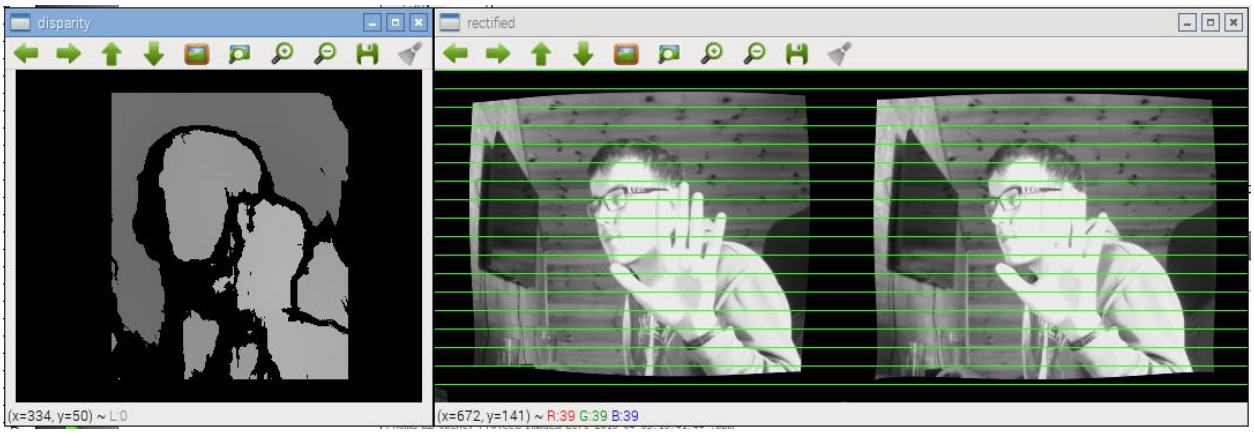
Once the user removes all the image names from the ListOfPics text file that are in the error log, then that user will be able to run the calibration again. This time there will be no errors however.

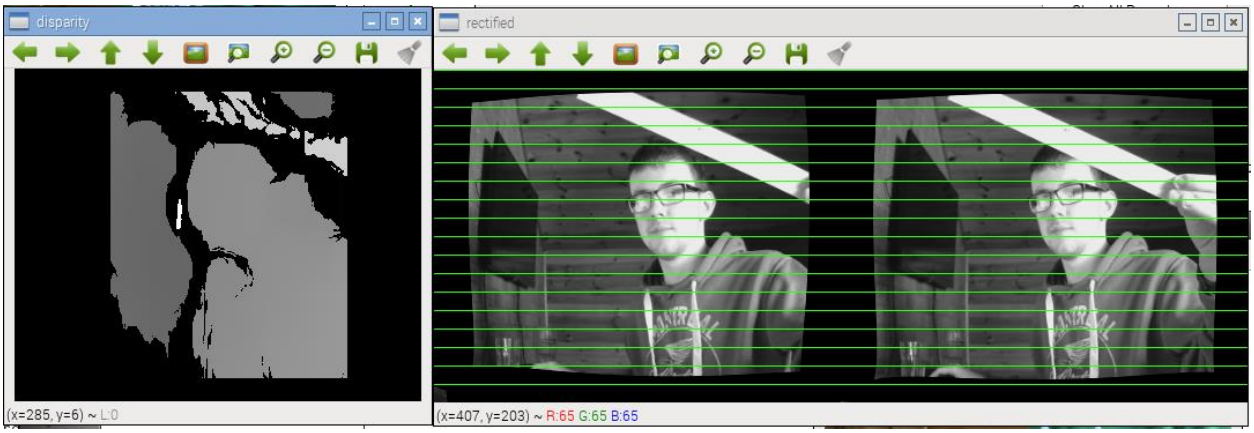
6.4. Validation Tests

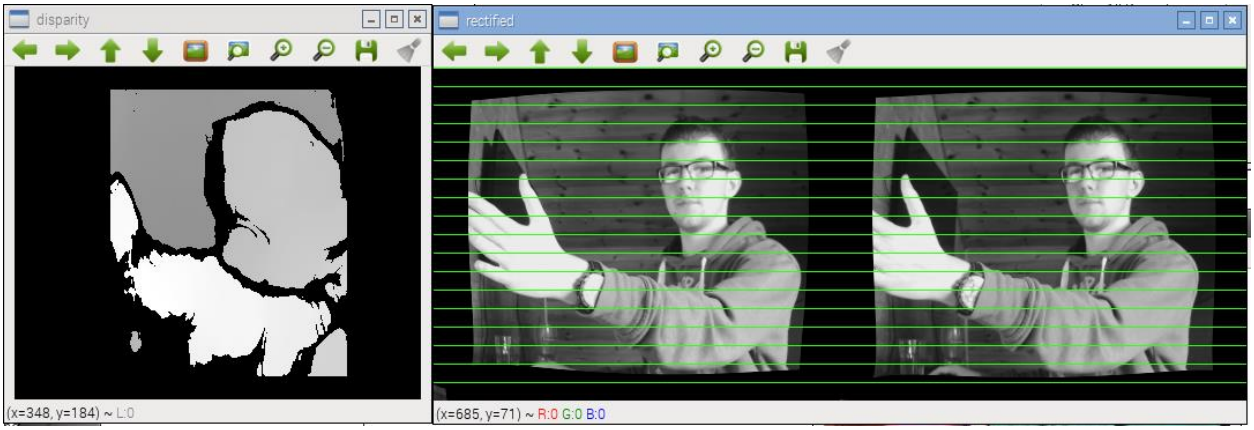
This is where the features that were implemented earlier on are now tested to see how efficient they are. The features that this section will be going over are the depth map, calibration results, and the distance estimation.

6.4.1. Depth Map

Below I will add a number of images that were taking after camera calibration. The purpose of a depth map is to get 3D information from a 2d image/images. This images below will show that this project was able to achieve that goal. They will also be images that show the results if the system is not calibrated correctly, and in some cases where the user has not taken enough picture of the chessboard in different positions.

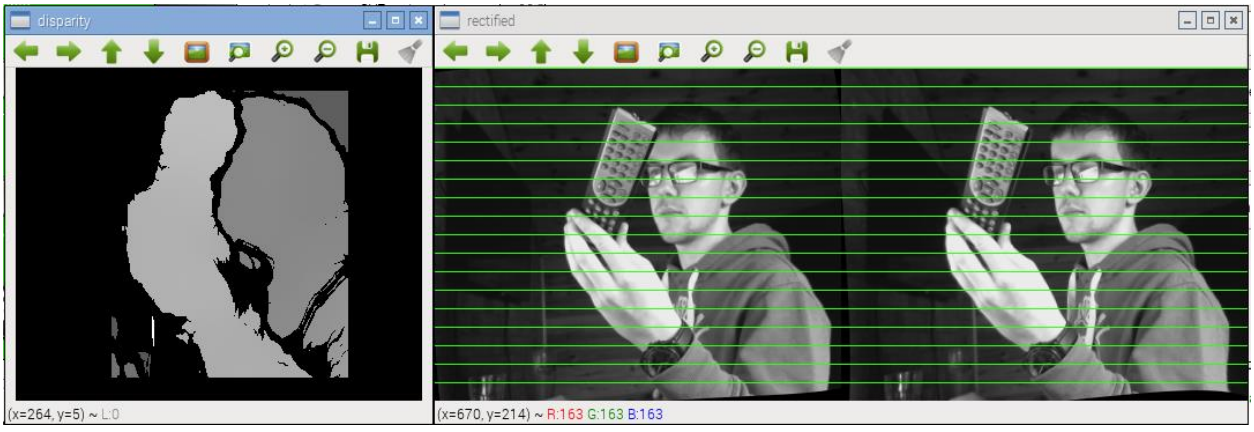


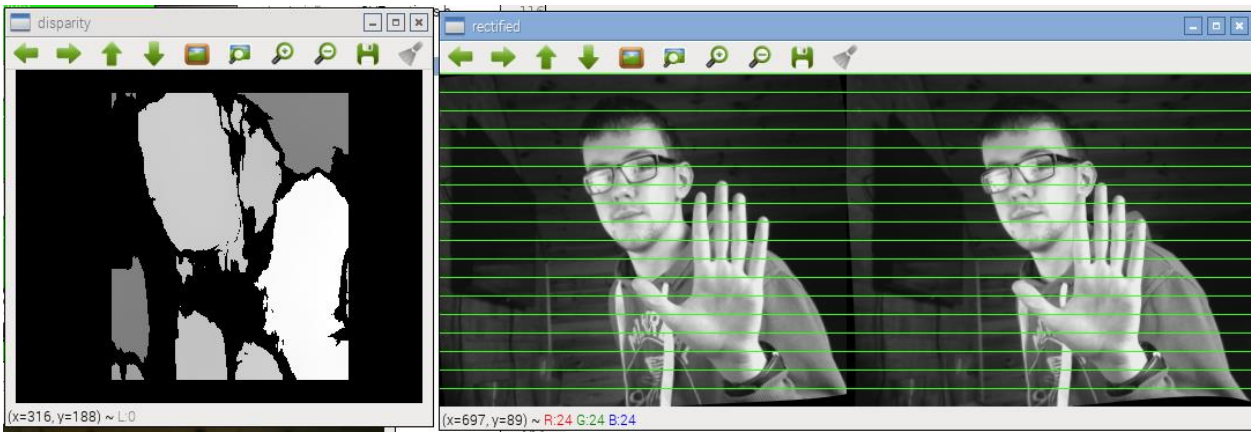




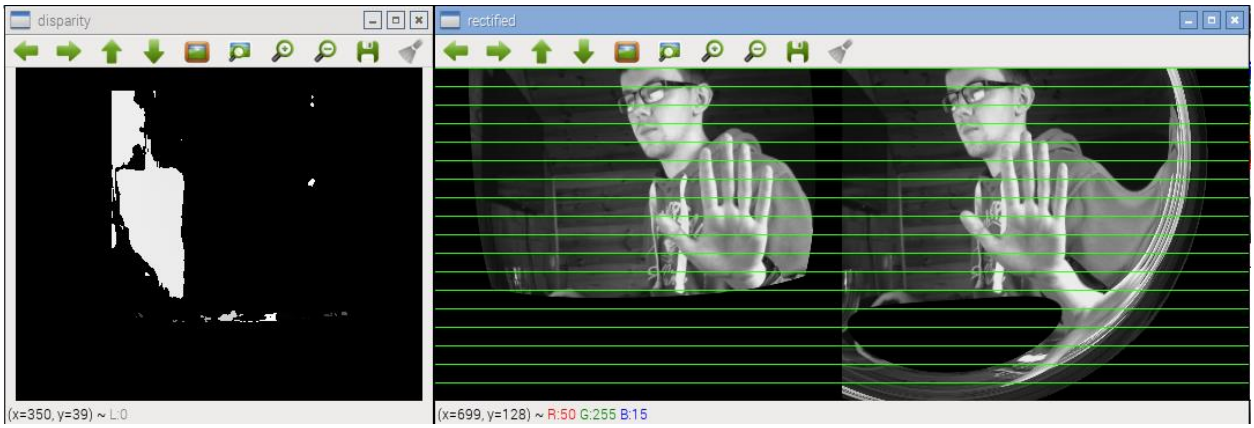
Disparity Map created – Cameras are recalibrated

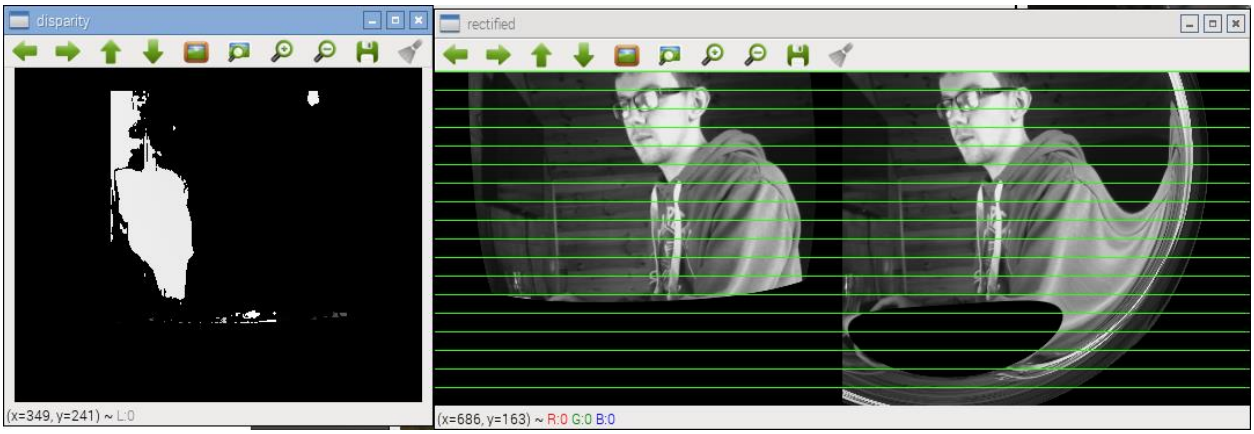
After a few good results, the camera was recalibrated. This done as the box that held the cameras was moved, as you can see in the images above it creates a white noise affect. While all the images after the re-calibration are much more clear.



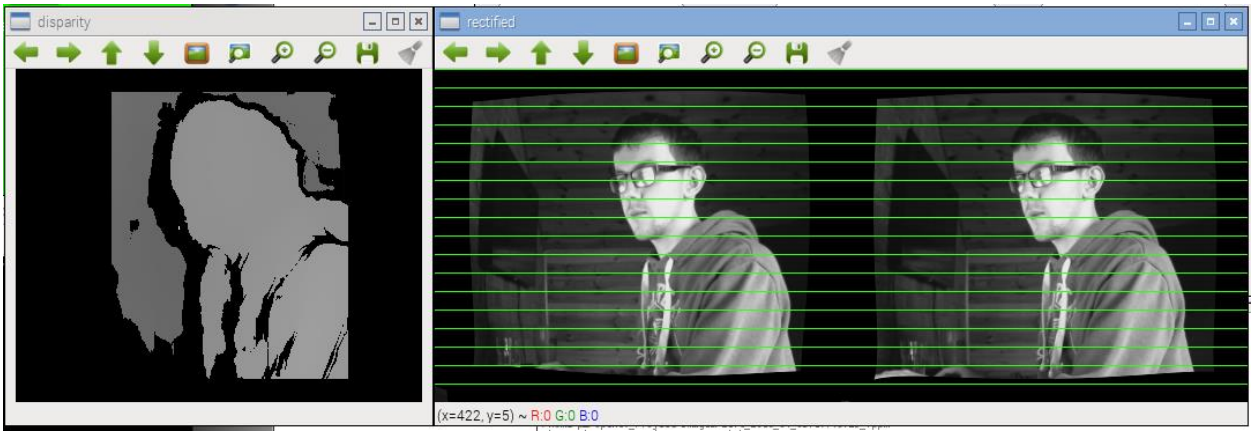


Below is some example of the depth map, but this time the results are not good. This is due to a small amount of images taken to get the calibration. There were only 6 images used when setting up the cameras for the images. This is clearly visible as the image is distorted too much to be useful.





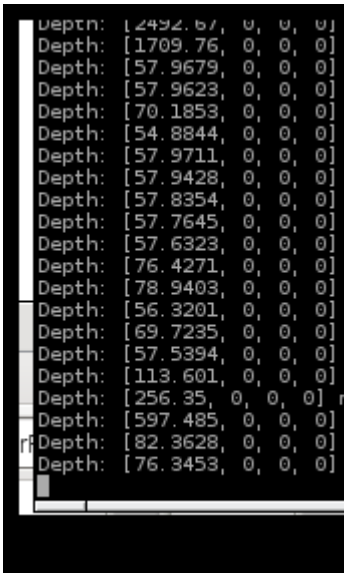
Here is the same picture as last time, but in this case the cameras have been calibrated using well over a 100 images.



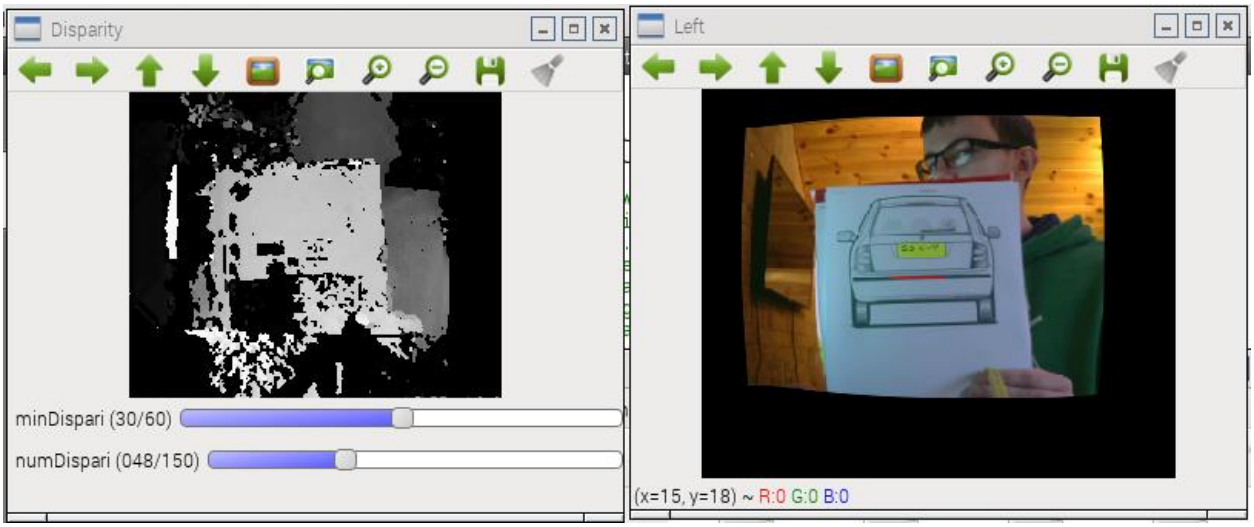
This clearly shows that the depth map is working as intended, it is just a case of the cameras being calibrated correctly.

6.4.2. Distance Estimation

Below I will demonstrate the accuracy of the distance estimation and show you what the range of error is. A number of images were printed out; these images are going to be used as points of distance. The distance is written on the image; this is the correct distance that this image is away from the cameras. This was distance was measured using a measuring tape.

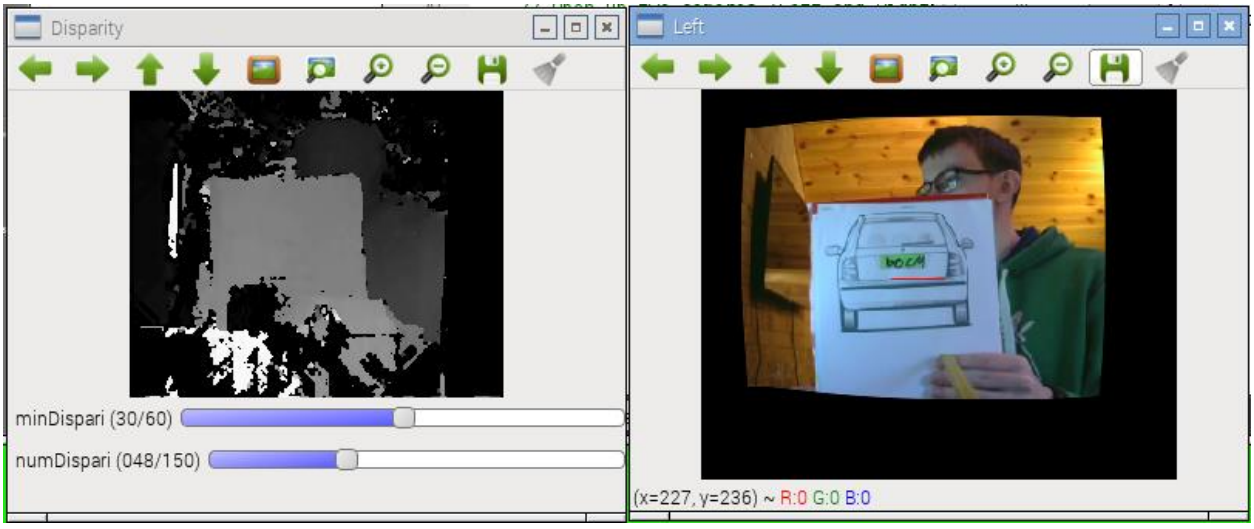


The image above was at a distance of 65cm away from the cameras. The results to the left show what the estimation was. As you can see these results jump up and down by a large amount. This happens where there is a quick movement. These quick movements seem to cause the estimation to go completely off. When holding the image as still as possible the cameras are able to pick it up much better.




```
Depth:  [44.2654,  0,  0,  0]
Depth:  [55.5118,  0,  0,  0]
Depth:  [79.6086,  0,  0,  0]
Depth:  [122.386,  0,  0,  0]
Depth:  [115.832,  0,  0,  0]
Depth:  [235.215,  0,  0,  0]
Depth:  [1566.33,  0,  0,  0]
Depth:  [757.36,  0,  0,  0]
Depth:  [1218.01,  0,  0,  0]
Depth:  [291.08,  0,  0,  0]
Depth:  [1946.1,  0,  0,  0]
```

This image caused a lot of errors when testing, it is the closest image to the cameras. The exact distance was 55cm. There was a lot of distortion and white noise in the image, that would not go away even when holding the image completely still.




```
Depth:  [52.7657,  0,  0,  0]
Depth:  [52.798,  0,  0,  0]
Depth:  [52.5082,  0,  0,  0]
Depth:  [52.4135,  0,  0,  0]
Depth:  [77.2216,  0,  0,  0]
Depth:  [52.9341,  0,  0,  0]
Depth:  [84.1472,  0,  0,  0]
Depth:  [78.1743,  0,  0,  0]
Depth:  [53.7196,  0,  0,  0]
Depth:  [109.778,  0,  0,  0]
```

This image got the most accurate results. It was at a distance of 60cm from the cameras. This is also the usually distance away that the chessboard is at when calibrating the cameras. The author believes these two things are directly related, and thinks that if he were to calibrate the cameras using more images from a wider range of distances it would improve the results of the other images.

6.5. Conclusion

This chapter has gone over the different testing methodologies that were used on this project. It also gave an idea of the hardware testing that was done before the coding had even begun. It has also given an example of a few unit tests that were carried out. These units helped find and fix a few errors that appeared after testing. This chapter also went over the system validation tests that were carried out to see the accuracy of the project. The results are varied but with more camera calibration, these results will more the likely become more accurate.

7. Project Evaluation and Conclusion

7.1. Introduction

This chapter will evaluate the work that was done over the last few months. It will compare the goal and aims outlined in chapter one to the goals and aims that were met at the end. This chapter will go over the project achievements and how the project changed during its time in development. This chapter will also discuss possible future work and what features could be implemented to the project.

7.2. Projects Aims and Goals

At the beginning of this paper you will see the overall aims and goals that were set out. They have been added down below to refresh your memory.

Goals

- Determine depth of object in image
- Setup low end cameras and hardware to record and take pictures
- Setup network to allow user to start cameras recordings
- Track an object that has motion in video
- Learn and developer a better understanding of computer vision

Objectives

- I want to start and develop new skills in C++
- Calibrate cameras for more accurate results
- Create a GUI in C++ to allow user interaction
- Give user feedback on driving ability (Score / Grade)

At the beginning of this semester when I came up with these goals, I was hopeful that I would be able to complete them, however that was not the case. The two goals that were not met were. Setting up a network to allow the user to record videos remotely, and to also track motion in video. These two features were pushed back as the deadline came closer and other feature needed more work. These features were however, tested in the prototype stage and seemed very plausible at the time. I personal believe that if I had another two months I would all the goals reached, and time to spare to iron out any kinks.

The goals that I did meet were getting the depth from a 2D image. I did this using simple low end equipment. I also learned an enormous amount in relation to computer vision. I also met most of my objectives. It was my first time using C++ and I am actually very happy that I picked it over python. It was fun to use and I felt that I really have

broadened my own knowledge of coding languages by learning it. I was also able to calibrate the cameras. While calibrating them I learned a great deal about computer vision. The OpenCV libarys were very useful and community online was very helpful as they had answer to almost every question.

I was able to create a GUI in C++. I did this using the QT developer tool. I found it to be a bit tricky when setting it up, but once I had it going everything seemed to fall into place. The one objective that I did not complete was giving feedback to the user. The current state of the project would not make is feasible to give the user feedback as the results are not accurate enough.

7.3. How Project has changed over time

At the beginning of the year, I had this clear plan that I thought I would be able to follow. As the months went on, and I made progress slowly. I quickly realized that I had a lot more work cut out for myself then I had previously anticipated. This first time it struck me that I might be out of my depths was when I was trying to calibrate the cameras. I was researching the math's behind the calibration results and it was very difficult to understand. Almost to the point where I couldn't understand ever second word and had to look it up on google to get the definition.

Then as time went on, I felt that I could no longer keep certain goals. One being getting the motion tracking working and other of setting up the network on the raspberry pi to allow the user to start recording remotely. It was a bit disheartening to just let these go, but I was better for it. I was then able to focus up on the main goal of working out the distance to an object.

Overall I would say the project has been sized down to just the basic proof on concept. Apart from Motion detection and the network setup. I have everything else working on low end equipment, with somewhat accurate results.

7.4. Future Work

I think that with some extra time and a bit more room to breathe so to speak, the motion detection and the network could both be done and up and running in around 6 weeks. The idea of working on something like this but with a little more processing power is also an interesting though. I would like to test it out on the raspberry pi 3. I would be curious to see the differences that the hardware would provide. I would also like to try out different cameras. So lower and some higher end then the ones I am using at the moment. I would like to see if the FPS that the cameras are recoding at make an impact on the overall quality of the depth map, and thus improve or worsen the distance estimation.

I would stick with C++, as I feel that there is a lot of untapped potential there. With more time to get the basics down. I found myself constantly learning with C++. But I never had any downtime to just sit back and try a few tests using certain classes and functions that may have proven useful. Instead I would test it once or twice and if I couldn't get what I wanted. I would just have to throw it out and move on, as I was on a time limit.

The future is very interesting when it comes to computer vision, it has been coming closer and closer into the spotlight as company's reveal their brand new cars that can drive themselves. It is also being seen in robotics where they are training different animatronics to avoid obstacles. This project had given me a chance to appeal to those employers now, so future work could possible leak outside of this project and into a job.

7.5. Conclusion

The end results might not match the goals and objectives that I set it with in the beginning. But it does accomplish a great deal of them on a smaller scale. I have never worked on a project to this scale before and it was everything I expected and more. I can look back now at all the different hurdles that I overcame with a smile. I can see all the problems I encountered with what seems such simple solutions, but at the time where near impossible.

I am very proud of what I have accomplished and I have developed my own interest in new technologies. I actually want to thinker with these bits of hardware, I want to see what else I can get them to do. Even though this project is by far the most stress inducing thing I have ever done in my life. I am very glad that I did it, and I am delighted with the way it turned out.

8. Bibliography (research sources)

[1] LiDAR for Dummies [Book]. [Cited 2015 Nov 06]; From: James Young

[2] Microsoft Kinect Sensor and Its Effect [Book]. [Cited 2015 Nov 06]; From: Zhengyou Zhang

[3] Learning OpenCV [Book]. [Cited 2015 Nov 09]; From: Gary Bradski & Adrian Kaebler

[4] Distance measuring based on stereoscopic pictures [Book]. [Cited 2015 Nov 9]; From: Jernej Mrovlje & Damir Vrancic

[5] Apache HTTP Server Documentation Version 2.2 [Website]. [Cited 2015 Nov 10]; From: https://httpd.apache.org/docs/2.2/install.html

[12] Python and Tkinter programming [Book]. [Cited 2015 Nov 09]; From: John E. Grayson

[34] Real-Time Disparity Map ex traction in a duel head stereo vision system [Book]. [Cited 2015 Nov 09]; From: G.Cailn & V.O.Roda

[6] LiDAR for Dummies [Website]. [Cited 2015 Nov 09]; From: https://www.raspberrypi.org/products/model-b-plus/

[7] A link to the home page for Derek Molly [Website]. [Cited 2015 Nov 12]; From: http://www.eeng.dcu.ie/~molloyd/

[8] The survey I created on survey monkey [Website]. [Cited 2015 Nov 10]; From: https://www.surveymonkey.com/r/THSYKBJ

[9] OpenCV supported formats [Website]. [Cited 2015 Nov 12]; From: http://docs.opencv.org/2.4/modules/highgui/doc/reading_and_writing_images_and_video.html

[10] Introduction to OpenCV-Python Tutorials [Website]. [Cited 2015 Nov 12]; From: http://opencv-python-utroals.readthedocs.org/en/latest/py_tutorials/py_setup/py_intro/py_intro.html

[11] Rules of The Road published by the Road Safety Authority [Image]. [Cited 2015 Nov 15]; From: http://www.rsa.ie/Documents/Learner%20Drivers/Rules_of_the_road.pdf

[13] Blog by The Tesla Motors Team [Website]. [Cited 2015 Nov 09]; From: https://www.teslamotors.com/blog/your-autopilot-has-arrived

[14] MATLAB Student price [Website]. [Cited 2015 Nov 09]; From: http://uk.mathworks.com/academia/student_version/?s_tid=tb_sv

[15] Kinect from codeproject.com [Website]. [Cited 2015 Nov 07]; From: http://www.codeproject.com/KB/audio-video/317974/kinect.JPG

[16] Ultra-sonic module from myduino.com [Website]. [Cited 2015 Nov 06]; From: http://www.myduino.com/image/cache/data/hc-sr04-500x500.jpg

[17] Logitech Webcam by Logitech.com [Website]. [Cited 2015 Nov 06]; From: http://www.logitech.com/assets/32825/5/hd-webcam-home-banner.jpg

[18] Waterfall model by technologyuk.net [Image]. [Cited 2015 Nov 13]; From: http://www.technologyuk.net/computing/sad/images/waterfall_model.gif

[19] Googles driverless car by hybridcars.com [Image]. [Cited 2015 Nov 06]; From: http://www.hybridcars.com/wp-content/uploads/2014/12/google-self-driving-car1.jpg

[20] Agile Model [Image]. [Cited 2015 Nov 13]; From: http://www.sabusiness.co.uk/wp-content/themes/SA/images/AgileDiagram.jpg

[21] Spiral Model [Image]. [Cited 2015 Nov 13]; From: https://qualityguru.files.wordpress.com/2010/08/spiral_model_basic1.png

[22] Online guide on how to setup OpenCV by opencv.org [Website]. [Cited 2015 Nov 06]; From: http://docs.opencv.org/master/d5/de5/tutorial_py_setup_in_windows.html#gsc.tab=0

[23] Unoffical Windows Binaries for Python Extension Packages [Website]. [Cited 2015 Nov 06]; From: http://www.lfd.uci.edu/~gohlke/pythonlibs/#jpype

[24] Python/OpenCV Computing a depth map from stereo images [Website]. [Cited 2015 Nov 13]; From: http://stackoverflow.com/questions/27726306/python-opencv-computing-a-depth-map-from-stereo-images

[25] Stereo Vision Gallery by Yuri Boykoy [Website]. [Cited 2015 Nov 15]; From: http://www.csd.uwo.ca/~yuri/Gallery/stereo.html

[26] Stereo/Image Database by Chieh Chih Wang [Website]. [Cited 2015 Nov 15]; From: http://vasc.ri.cmu.edu/idb/html/stereo/index.html

[27] Computing a disparity map in OpenCV [Website]. [Cited 2015 Nov 15]; From: https://dzone.com/articles/computing-disparity-map-opencv

[28] Link to sample C++ code written by Martin Peris [Website]. [Cited 2015 Nov 20]; From: http://www.martinperis.com/opencvstereowebcam/OpenCVStereoWebcam-1.0.tgz

[29] Installing OpenCV on Debian Linux by Indranil Sinharoy [Website]. [Cited 2016 Jan 01]; From: http://indranilsinharoy.com/2012/11/01/installing-opencv-on-linux/

[30] Source code for tracking objects written by Kyle Hounslow [Website]. [Cited 2015 Nov 20]; From: https://raw.githubusercontent.com/kylehounslow/opencv-tuts/master/object-tracking-tut/objectTrackingTut.cpp

[32] Stereo Vision - by Martin Peris [Website]. [Cited 2015 Nov 06]; From: http://blog.martinperis.com/2011/08/stereo-vision-why-camera-calibration-is.html

[33] Stereo Vision Image from the website scielo.org [Image]. [Cited 2015 Nov 06]; From: http://www.scielo.org.ar/img/revistas/laar/v37n1/1a04g856.gif

[34] OpenCV Version Downloads [Website]. [Cited 2016 Jan 01]; From: https://sourceforge.net/projects/opencvlibrary/files/opencv-unix/

[35] list of required packages for OpenCV install [Website]. [Cited 2016 Jan 01]; From: http://docs.opencv.org/2.4/doc/tutorials/introduction/linux_install/linux_install.html

[36] online steps to installing openCV [Website]. [Cited 2016 Jan 01]; From: http://rodrigoberriel.com/2014/10/installing-opencv-3-0-0-on-ubuntu-14-04/

[37] OpenCV Module Descriptions [Website]. [Cited 2016 Jan 01]; From: http://docs.opencv.org/2.4.9/modules/core/doc/intro.html - modules descriptions

[38] Back of Car [Image]. [Cited 2016 March 20]; From: http://www.clipartbest.com/cliparts/aiq/oGb/aiqoGbEAT.png

[39] Chessboard [Image]. [Cited 2016 Jan 15]; From:
http://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html

[40] Comparison of some stereo vision algorithms [Website]. [Cited 2016 March 20th]
https://cseautonomouscar2012.wordpress.com/2012/11/14/comparison-of-some-stereo-vision-algorithms/