Jousting Robots

Final Report

Submitted for the BSc in
Computer Systems Engineering (with Industrial experience)

March 2016

by

Jason M Duckworth

Word Count: 13,814

Table of Contents

1 Introduction

1.1 Report Content

This final report will elaborate on the aims and objectives for this project. The background section explains the reasons behind making the project and highlights in what way the user will be interacting with the robot. There will be a thorough description of all the technologies so the reader can fully appreciate how the jousting robot functions. The technical development section will involve showing the reader through the entire design of the robot, how it will be tested and the implementation of the robot. This will consist of visual representations to help understand the concept behind making a Raspberry Pi robot. The project management section explains how Scrum has been used to manage the project, an explanation of how time plan revisions were altered and how the hardware and software was maintained and managed. An evaluation of the project detailing the outcome of the project as a whole and how this can be used and developed further in the future is included at the end of the report.

1.2 Project Overview

The idea behind Jousting Robots is to create a robot encompassing different behaviours, these behaviours will give the robot the ability to joust with another robot. The robots will be autonomous. The challengers will select different behaviours, these behaviours will decide how the robot goes about attacking or defending the target, using different behaviours may change the attributes a robot has, for example inputting a different set of variables that improves the agility of the robot will reduce the top speed of the robot. The user will require to use timing and the power of the robot to seek out and attack a target successfully. A set of challenges will be made to engage the user and introduce them into Raspberry development. A student may also use this to hone their maths skills.

1.3 Raspberry Pi’s in Education

Around the world new technologies are created using complex pieces of software and sometimes understanding this technology can be difficult. Jousting robots aims to be something that can be used within education to help understand how software runs alongside hardware and also introduce pupils in schools or in other educational facilities to electronics and programming. Providing people with these skills at young ages will help give an insight into how different devices are created. This project can help jump start somebodies career into computer science and assist in increasing awareness of how useful these skills are. Raspberry Pi’s are the perfect devices for learning a programming language as it provides a fun and educational environment for the user. The Pi was initially created because there was a lack of students applying for computer science courses, as Swapnil Bhartiya (2013) says in his article about how raspberry pi’s came to be. This article also explains how devices that could be used for education have been made very expensive and therefore parents don’t feel that paying high prices are worth it. This is what jousting robots hopes to help improve by providing a fun and cheap robot that can be used to help attract people into the world of computer science.

1.4 Raspberry Pi’s in Industry

Raspberry Pi’s have great use commercially. There are many models on the market with different uses and characteristics which can be utilised in different ways. Companies around the world have even been asking for specific customised boards which has led to many different components being made for the Raspberry Pi. Monique DeVoe (2015) explains in her article about how Raspberry Pi’s are becoming part of companies, she says:

“Customized boards will be suited for use in applications like IoT *and energy management to industrial and end-consumer devices, and can be ordered in quantities starting around 3,000-5,000, depending on the customization.”*

All of these companies are finding different uses for this equipment. The cheapness, size and amount of resources makes these boards useful for all types of applications and systems. Many libraries and open source projects are easily accessible on the Pi. This gives companies the chance to make use of this hardware and create brand new helpful technologies.

2 Aim and Objectives

Create a jousting robot that can be used to help introduce people into computer science and for entertainment

Below is a list of objectives and some optional objectives that need completing in order to reach this goal:

Objective 1 – Fully functioning tracking system

The first core component of the robot is an object tracking system that tracks enemy robots and other targets that may be a threat to the robot. The robot will be able to identify an enemy robot using the light attached to the enemy, and the jousting robot will then attack depending on the behaviours set by the user. As the jousting robot attacks the enemy robot it can keep track of the location of the enemy using the on-board camera. The camera is the only way of identifying the enemy robot and therefore is an extremely important feature. This is the first step in the project and also the leading stopper.

Objective 2 – Correct and precise movement of the motors

Before the motors can be started the camera must track the enemy which is why the camera tracking must be completed before the motors are utilized. Once the robot has found the enemy it needs the ability to travel towards said enemy. Whilst the robot is carrying out its attack the enemy might attempt to flee or make a tactical dodge, this problem can be resolved by having the jousting robot adjust to accommodate for the enemy's movements. The code for this part of the robot will be available for the user to alter, this means they can teach themselves how the motors work with the Pi and may figure out how to make a much more effective jousting robot.

Objective 3 – Behaviours

Behaviours are the next key aspect of the project after tracking the enemy and moving the jousting robot. It needs to decide how to act when the enemy is in sight. There will be a number of behaviours that will change how the robot interacts with the other target. For example, the robot could either use the defensive route which may make the robot slow but accurate or the offensive route which could make the robot fast but less accurate. The reason why this feature is so important is it can determine how the robot acts when jousting another robot which could ultimately determine the winner.

Objective 4 – Fully automated

This objective links in with the behaviour objective. The robot will require no input once it has started. Using a state design which will encompass different levels of interaction with a target the robot will work on its own to complete its objective then fall back to get ready for the next target. This will require the robot to be constantly updating which means the camera and motors will need the resources to be utilised constantly while the programme runs on the Raspberry Pi.

Optional Objective 5 – User Engagement

The user will want to easily change the basic behaviours of the robot. The code provided will call a method from the motors module that will control how the robot behaves in different situations. A set of parameters will be used to decide the speed of the robot in certain situations. The user can also choose different methods from the motors module and add their own methods which can be called. The user will learn to control the speed of the motors and control the timing of the motors to make sure the least amount of resources is used so the robot still performs well. The user will need to make sure the robot doesn’t miss the target when attacking and will accurately identify the robots position. A challenge sheet will be provided which the user will have to attempt to complete.

3 Background

3.1 Why this Project is Beneficial to Education

This robot will be fun for people of any age who are wanting to learn about electronics. All the code written will be available for users to edit and some code will contain templates to help the user. The robot could be used as a kit that somebody would have to assemble themselves and then customise using different libraries or even implement their own. This would be extremely helpful in schools to help introduce pupils to programming. Programming is an important skill that may not be introduced to pupils until later on in life. Alison Ebbage (2014) says in her article about how Raspberry Pi's can help in education "The reasoning is that today's world demands that computational skills such as programming should form part and parcel of every pupil's core skillset". Unfortunately, this is not the all the time.

Programming is just one of the skills that make up the subject of computer science; This is why raspberry Pi's are so useful as educational tools; The fact that the hardware is so limited means that there are no complex IDE's which make programming simple, somebody programming the Pi in python has to learn the structure and aspects like keywords, before creating anything. This project does not only focus on learning to programme however. There are many other key skills that the user will be exposed to that are essential to introduction to computer science. Debugging and testing will be a key element to the project and the users understanding of software, as well as understanding the underlying parts of the Raspberry Pi. Once students can understand technologies and respect them they will be able to fix and maintain their own technologies. There has been a massive shift recently on the opinion of computing in schools and in the UK we have started to incorporate lessons such as computing which could in the future use projects such as these to advance learning.

ICT – Information and Communications Technology – is out, replaced by a new "computing" curriculum including coding lessons for children as young as five. (Stuart Dredge. 2014).

3.2 How the Robot Will Function

The jousting robot will provide a fun and challenging experience for users using a magic chassis, Pi camera board and a RaspiRobot board. The job of the camera is to look for any enemies and identify them. The enemy will require a light of a certain colour. Once the enemy has been identified it adjusts to the enemy's position then attacks. The robot board decides the motors speed and direction. The manner in which the robot attacks and surveys the environment depends on how the user specifies the variables which affect the behaviour of the robot. This can be done by adjusting certain variables or the user may even decide to create their own methods by using the sudo code provided. This requires the user to learn python and understand how the pi works. The challenge sheet will contain sudo code which means prior knowledge of python is crucial for this aspect of the project.

3.3 Raspberry Pi

The Pi 1 and Pi 2 are both used within this project. The Pi 1 was the original board used, however it contained a large flaw, it had a limited CPU. The CPU on the first Pi only uses one core and has speeds of up to 700MHZ (Jon Mundy. 2015). The robot needs to be able to process two threads, one for the camera and another for the motors, the lack of resources makes this complicated.

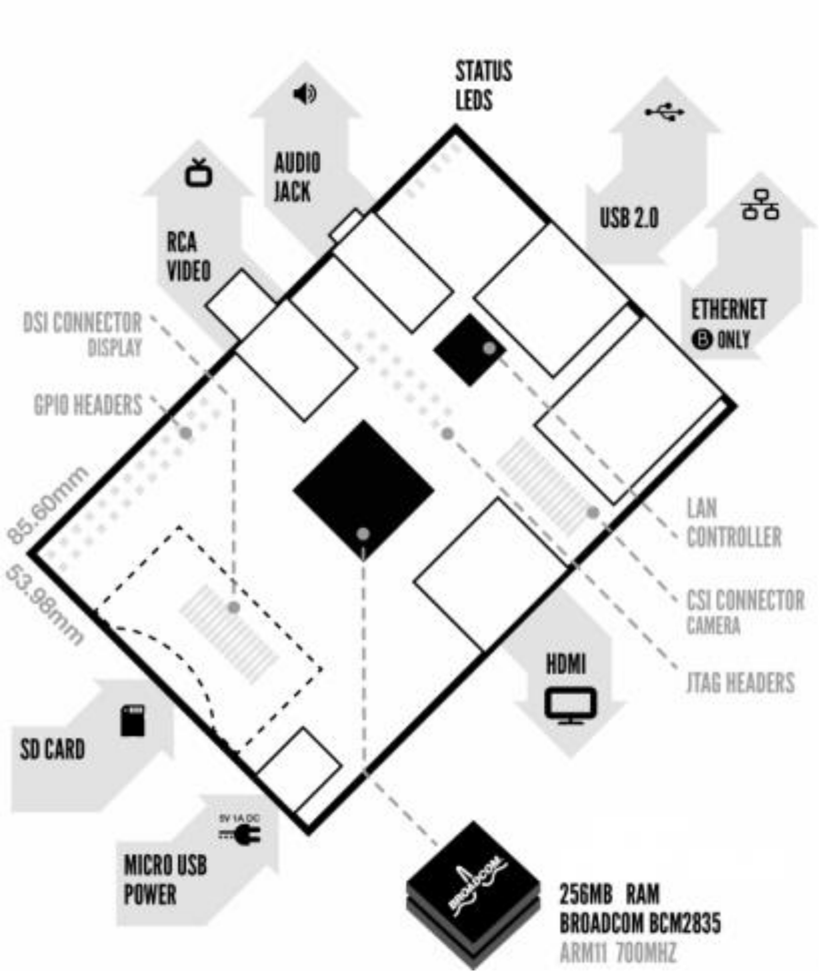


Figure 1: Raspberry Pi 1(eLinux.org. 2016)

The Pi 2 however contains a slightly faster CPU with speeds of up to 900MHz and a quad core processor which provides a much more powerful board to handle these threads. The Pi 2 also holds 500MB more RAM than the original Pi which makes using it to develop on much smoother. Overall the Raspberry Pi 2 provides a much faster robot which makes developing and running the project much nicer than on the original Pi. A data sheet can be found in appendix F which outlines the overall capabilities of the Pi 2.

There are other boards that could be used as an alternative such as the Arduino. The Arduino comes with many components which could be utilised for this project. It can even use Open CV for detecting the target. Arduinos are micro-controllers which are used as single purpose devices. Not many resources are available for an Arduino. The one main problem with Arduino is it would fail providing the user with a friendly environment.

The Raspberry Pi can be used as a desktop. You can write code on the Pi, and easily install packages with help from an operating system such as Raspbian. A Raspberry Pi doesn’t necessarily have to be a single purpose device and in this particular project utilises a nice UI which makes developing much friendlier. The Arduino however has very little resources, it only uses a processor with a speed of 16MHz (Brad Bourque. 2015.). As the robot needs to process images and the motors at the same time this just isn't enough.

3.4 Python

Python is the main language used in this project. It is a high level programming language that was made to sustain code readability. It's the obvious choice as python is the most used language on the pi and has a wide range of convenient libraries. Python is also useful for teaching, it helps with learning to maintain certain programming practises such as consistent indentations and easy to read variable names. This makes learning different aspects of programming even better for beginners. Being able to structure a programme well helps when it comes to debugging, the user can read their code much easier, therefore finding and fixing bugs should be significantly easier. Other languages such as C++ don't force you as much to indent statements, and keywords such as `cout` can sometimes be confusing, this basically means console output which may be more confusing than in python where you can quite simply use `print` which is a more obvious keyword. This doesn't necessarily mean that python programs can't be complicated. Python is a widely used language and is often used in big projects in industry. It has been created for anyone to easily understand how a programme works as is stated in the quote below.

"Python's clean, readable syntax makes code eminently readable, even by programmers other than those who worked on the original project. While some programmers object to the significant use of white space in Python code, almost everyone who sees Python code agrees that it's far more readable than C or Java." (Dan Shafer. 2002).

It is a powerful language, it uses object orientation, a dynamic type system so an application can understand the type of a variable at runtime which has a wide range of benefits and has a large and widespread standard library. This language is perfect for this project, it has many useful aspects such as it can run code on separate threads, interact with the motors and the camera using pre made libraries and there are many more advantages.

Many other languages could be used to develop on the Pi such as C, C++ or even Perl. Here is a table of what was taken into account when choosing the language:

Language	Libraries	Complexity of Project	Readability
C	Many of the libraries the developer would have to make himself. Not much support for using C with Open CV on the pi.	The code has a very good chance of being complex therefore debugging will be complicated. The project would take much longer to finish	To the user the code would be difficult to understand therefore using this language to make an educational project would be pointless.
C++	Many useful libraries that can be used on the PI. Open CV is much easier to use with C++. A lot more support for libraries.	A lot less complex that C however the project will still take much longer to complete.	Average readability, there's still a lot that could confuse a user trying to learn a new programming language.
Python	A lot more support for the Pi than any other language. Can be used with	The project shouldn't be too complex and debugging will be	The syntax is much more readable than C and C++. Python is a great language

	Raspirobot library, Open CV and has many more useful libraries to control things such as PWM	much easier than C and C++.	for beginners because it's so easy to read.

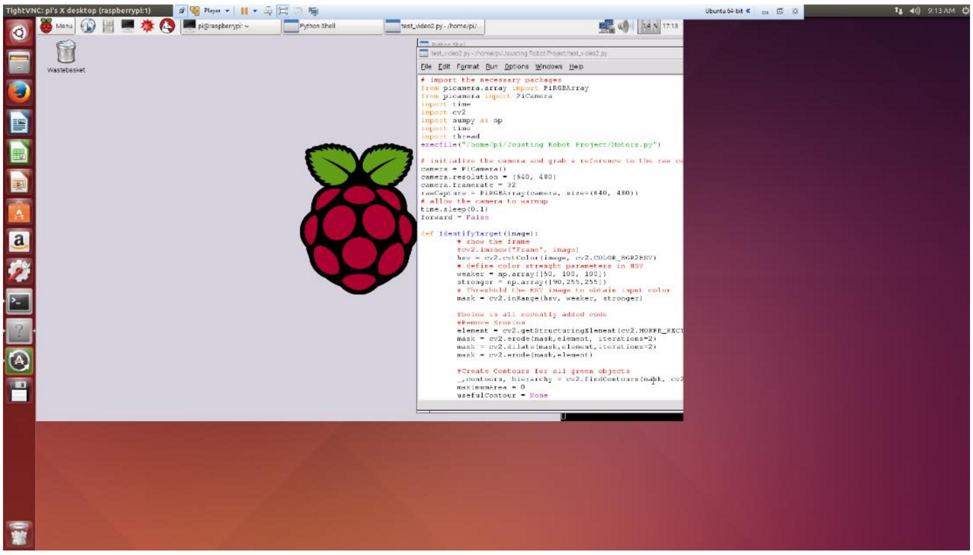
The aim was to find a language that would be advantageous for the users learning but powerful enough to finish the project with all the features implemented efficiently. In the end Python is the obvious choice.

3.5 Programming IDE - IDLE

IDLE is a very smart IDE, the fact that it runs on the PI makes it instantly useful. It's also known to be beneficial in an educational environment. There is some arrangement of intellisense that can be used. It is good that there is not too much help from this as it forces programmers to write and understand all the python keywords. It is quite a simple IDE to use, breakpoints can be employed to slowly go through the code when debugging, you can modify code on the fly, this is more of a python feature, nonetheless IDLE does give you a nice window to use to do this. Syntax errors are identified and highlighted when the code is run. It's the clear choice for writing programmes on the Raspberry Pi.

3.6 Development Environment

Working with hardware such as the raspberry Pi means there are limitations when it comes to the environment in which a robot can be developed. Working directly on the Pi requires a spare screen and HDMI cable, it makes starting work on the Pi slow. VNC ended up being the preferred option as it means remoting onto the Pi easily from a desktop (TightVNC Software. 2015). It also means the Pi won't have any unnecessary wires hanging around. Figure 2 shows a preview of how VNC displays the Pi. The window is small, this isn't much of a problem though. The Robot can freely move around when remoting meaning more accurate testing can be done without anything obstruction of the robots vision or movement.


TightVNC: pi's X desktop (raspberrypi:1)
Player
Ubuntu 64-bit
9:13 AM
Menu
pi@raspberrypi: ~
Python Shell
test_video2.py - /home/pi/
17:13
Wastebasket
Python Shell
test_video2.py - /home/pi/Jousting Robot Project/test_video2.py
File Edit Format Run Options Windows Help

```
# import the necessary packages
from picamera.array import PiRGBArray
from picamera import PiCamera
import time
import cv2
import numpy as np
import time
import thread
execfile("/home/pi/Jousting Robot Project/Motors.py")

# initialize the camera and grab a reference to the raw c
camera = PiCamera()
camera.resolution = (640, 480)
camera.framerate = 32
rawCapture = PiRGBArray(camera, size=(640, 480))
# allow the camera to warmup
time.sleep(0.1)
forward = False

def IdentifyTarget(image):
        # show the frame
        #cv2.imshow("Frame", image)
        hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
        # define color strenght parameters in HSV
        weaker = np.array([50, 100, 100])
        stronger = np.array([90,255,255])
        # Threshold the HSV image to obtain input color
        mask = cv2.inRange(hsv, weaker, stronger)

        #below is all recently added code
        #Remove Erosion
        element = cv2.getStructuringElement(cv2.MORPH_RECT
        mask = cv2.erode(mask,element, iterations=2)
        mask = cv2.dilate(mask,element,iterations=2)
        mask = cv2.erode(mask,element)

        #Create Contours for all green objects
        _,contours, hierarchy = cv2.findContours(mask, cv2
        maximumArea = 0
        usefulContour = None
```

Figure 2: VNC preview

3.7 Open CV

Open CV is an extremely useful library making vision software easier to develop. On the Open CV website, it states that this software can be used for identifying an object and tracking the object (Open CV. 2015. About). It works perfectly with the Raspberry Pi camera board. As this project requires some way of identifying an object or another robot, these libraries fit in perfectly. The robot will take in each frame and process it; the process will consist of looking for a certain coloured object. Open CV has many useful methods, such as a method which uses the HSV (Hue Saturation and Value) range to find a range of colours between the values specified in a program. There is a function that can be used to draw a bounding rectangle around an identified object which is convenient for when the jousting robot to identifies an enemy. There is also simple functions such as imshow() which will display an image on the screen to help with testing the robot.

3.8 Raspberry Pi Camera Board

The Pi camera board will simply collect video feed. The program will receive a frame from the camera board and then process it. As you can see in figure 3 the camera board is sat on the front of the magic chassis, the board connects to the CSI port on the Raspberry Pi. Open CV can be utilised with this camera board. The resolution can be set easily which helps with the performance of the robot, the board will be processing many frames in a short amount of time therefore this is important. One problem could be if the resolution is set to low the image may be to blurred to identify any objects that are far away.

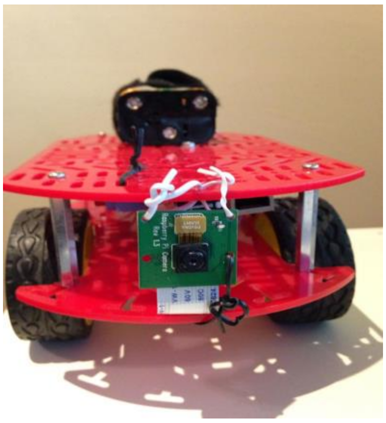


Figure 3: Raspberry Pi Camera Board

3.9 Raspi Robot board

The robot's motors need to be controlled correctly without any way of producing issues with the hardware, this is why a Raspi Robot board has been added. Figure 4 shows the board, the blue modules on the side are used to connect to the motors. The actual board connects to the Raspberry Pi's GPIO pins using the module underneath the board. This board doesn't provide any way of varying the speed unfortunately. Instead of using the provided libraries the Robot requires to use a custom library that implements PWM (Pulse Width Modulation) to control the voltage the motors will be using. There are other versions of the Raspi Robot board which do implement speed control such as the Raspi Robot v2 and v3. These boards are not available to use due to the price and as PWM can be implemented anyway there is no need to invest in another board, that would be overkill for the project.

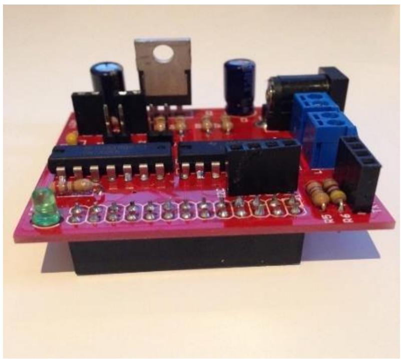


Figure 4: RaspiRobot board

3.10 PWM (Pulse Width Modulation)

Pulse width modulation is used in a wide range of applications, it can be used for calculating measurements, communications, power control and conversion (Michael Barr. 2001). In this project PWM is used for controlling the robots speed. The Raspi Robot board didn't come with PWM implemented which meant extra research needed to be done in order to understand and implement this feature.

PWM is a way of digitally encoding analogue signal levels (Michael Barr. 2001). This means we can talk to the motors in a more distinguished way creating the chance to control the voltage going to the motors. In terms of the raspberry Pi applying PWM is simple as the GPIO library comes with a function that can be called to prepare a pin to use PWM. Once this is done a value can be applied to a pin that will deliver a certain amount of pulses depending on the value. The timing between the pulses and the voltage applied when the pulse is sent determines the average voltage, depending on the voltage the motors will either move faster or slower. The batteries can only supply so much voltage and the motors have a maximum of six volts therefore the speeds are limited.

4 Technical Development

4.1 Robot Architecture

Within this section there are details of the robot's architecture. It outlines the robot's physical designs. There are a number of technological hurdles that only the correct hardware can overcome.

4.1.1 Robot Design

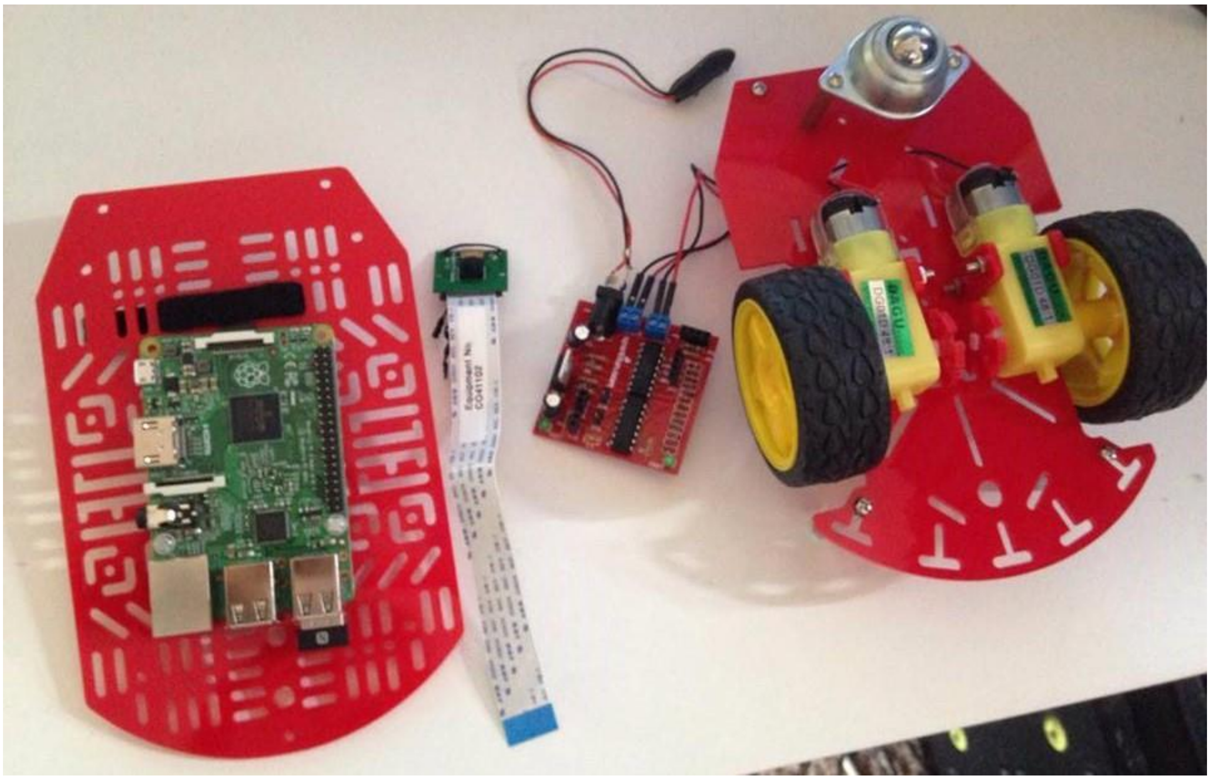


Figure 5: Jousting Robot Components – Original Design

Jousting robots uses four main components; the Magic Chassis, Raspberry Pi camera board, Raspberry Pi 2 and the Raspi Robot board. The Magic Chassis on the right (The top half of the chassis is on the left with a Raspberry Pi attached to it) comes with two motors, two wheels and stabiliser which is the silver module at the top of the chassis. This chassis keeps all the other components of the robot together, the Raspberry Pi screws onto the top half of the chassis to keep it in place. The chassis has no limitations to direction when moving however getting to a slow speed can be an issue due to the fact that the wheels require a certain amount of power before they move. To overcome this the robot requires some momentum before moving slowly.

The Raspberry Pi 2 which is the green board to the left of the image has a GPIO port (General purpose input/output) which on the image is to the top right of the Pi. An original Pi can also be used with this project. The GPIO port on the Pi 2 has more pins than the original one. This doesn't affect the connection between the Pi and the Raspi Robot board however as the pins are mapped the same. The Raspi Robot board which is the component to the left of the motors has a GPIO socket in which the Raspberry Pi GPIO pins simply plug into. There are many other components that can be attached to the Raspi Robot board such as a range finder or a switch, for now the motor screw terminals are the only useful modules. These two boards fit nicely together inside the chassis preventing any collisions from affecting the boards which are fragile, this is useful as the primary objective of the robot is to collide with another robot.

The camera board which is to the left of the Raspi Robot board simply plugs into the CSI port on the PI and is attached to the front of the chassis so the robot can look directly ahead for any targets. The resolution of the camera is large enough to identify a target within a certain area. The cameras hardware features can be found in appendix G. There was a possibility of using a USB camera however as it states on the Intorobotics (2015) website about Pi cameras, the Raspberry Pi camera board impacts the CPU's performance very little because it connects directly to the GPU. The USB uses more of the CPU since there is no direct connection to the GPU, thus using a USB camera would be problematic as this will affect resources which are also required for the motors to run smoothly.

Figure 6: Batteries

The motors require a voltage of at least five volts in order to reach acceptable speeds. Using a USB cable doesn't get anywhere near enough power to the motors. The batteries are placed on top of the chassis using Velcro to keep it secure. To the far right of figure 5 you can see that there are two DC motors. These motors are easily controlled using PWM. The speeds that these motors can run at are sufficient in making the robot move towards its target fast enough to cause damage.

4.1.2 Target Design

One of the biggest stoppers within this project was the ability for the robot to actually identify its target. The camera board collects the images, then the camera module processes the images and identifies each pixel within an image. Using the algorithms from open CV the robot can find certain colours within an image and tell the software where on the image the identified colour is. Colours change depending on the intensity of light, this means there isn't any consistency when it comes to the robot looking for a target. If a green piece of paper is shown in a certain light the robot might notice it and process the image appropriately, however if the light source is taken away the camera wouldn't be able to see the green paper therefore no target would be identified, this would change depending on the room the robot was being tested in. To solve this matter different targets have been tested.

4.1.2.1 Magic Chassis Target

The first target was simply the Raspberry Pi robot. Figure 7 shows the distinctive red colour of the chassis. This target was mostly used for testing. It would have been a simple solution yet as said before, the colour of the chassis is not consistent enough to have the robot identify the colour each time. The next step was to add light; a torch was used on the red chassis which gave a more accurate colour for the robot to identify. Use of the torch was inconvenient for the user, nonetheless it provided a useful way to test the camera in the early stages of the project.

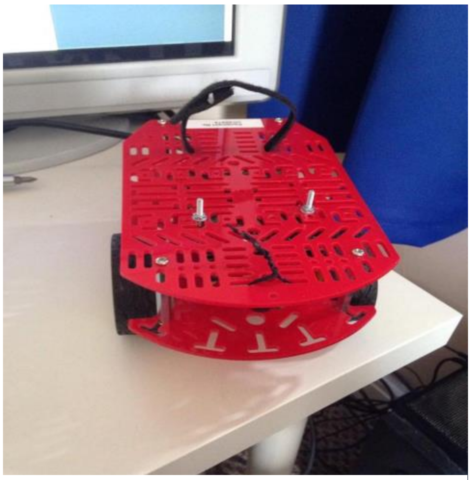


Figure 7: Raspberry Pi Target

4.1.2.2 Quality Street Light Target

Although the previous method of adding light worked for testing, as stated before it would not be convenient for the user. The next step was to find a light source that could be attached to the robot and provide a certain coloured light at a certain luminosity. To achieve this a USB light was used which provided an extremely bright light that was clearly visible for the jousting robots camera. This light would be visible from many ranges which meant the robot could easily distinguish the target. The light needed to be a colour which the robot could identify easily and the colour also had to differ from the robot's environment.

The answer was to use a green quality street wrapper and tie this around the light. The colour and visibility of the light was perfect when using this target which meant the user could simply attach the light and the robot would certainly find it. There was, however, one simple problem; the mobility of the light and positioning was not practicable enough. When using this light, it would only be visible from one direction. If the other target was mobile, then there was a chance when the robot looked at the target the light wouldn't be facing that direction if the target had the ability to move.

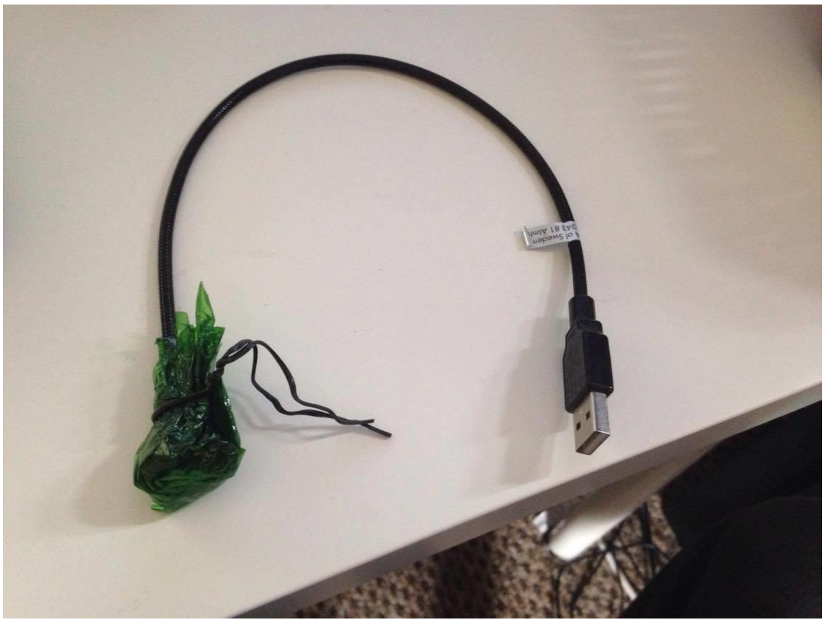


Figure 8: Quality Street Light

4.1.2.3 Finger Light Target

One solution could have been attaching LED's to the robot, these could be in the form of some small bright lights that could be placed on each side of the robot. The problem with the use of LED's is that there are no more available slots on the Raspberry Pi. Instead of lights that are wired to the Pi the solution was to use a light that would be powered on Its own. This meant using finger lights. Finger lights are small and are usually wrapped around a person's fingers when participating in a rave.

A much better use for the finger lights has been found within this project. As you can see in figure 9 there are four lights, each on one of the four sides of the robot. This means from whatever angle the target is facing the jousting robot will be able to identify the target. Another problem may arise from the use of these lights, as one of the green lights is placed right above the camera, there is a very good chance the robot will constantly be seeing its own green light. Even if the lights were placed somewhere else on the front there is a chance the robot will still see the colour green. In order to fix this the user has the chance configure the robot target colour and different coloured lights will be used on the other robot.

Initially red finger lights were used on the target. Whether the robot used red lights or tried to identify the distinctive red colour on the chassis it did not work. The camera uses a red light just above the lens to display whether the camera is currently being used. This light can sometimes shine in front of the camera and cancel out any other red colours within an image which meant the robot had difficulty finding a red target.

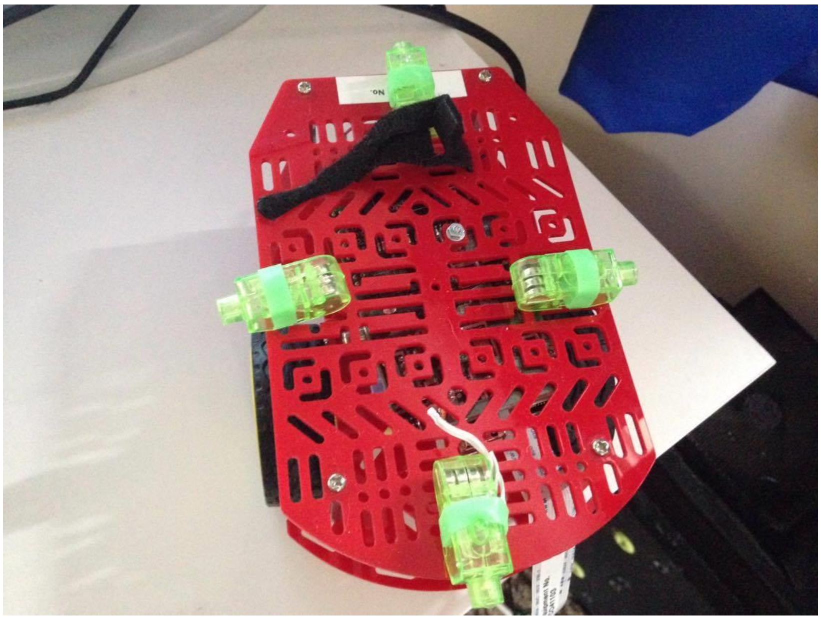


Figure 9: Finger Lights Target

4.2 Software Architecture

This section explains how the software has been designed. The software design has been done in a way that the changing of behaviours is effortless and the data flow is understandable.

4.2.1 Data Flow

Figure 10 shows a block diagram of the Jousting Robot, the basic code design for this robot and shows the interactions between the hardware.

The centre block shows the interaction between the python files running on the Pi. The diagram clearly shows the main file which is used to interact with the motors and camera modules. This main file shares data between these modules. The motor software will be running on a separate thread to the camera software so that they can run synchronously. The motors need to be running as the Pi is processing the frames from the camera, so if an enemy is sited then the Pi can stop the motors and respond.

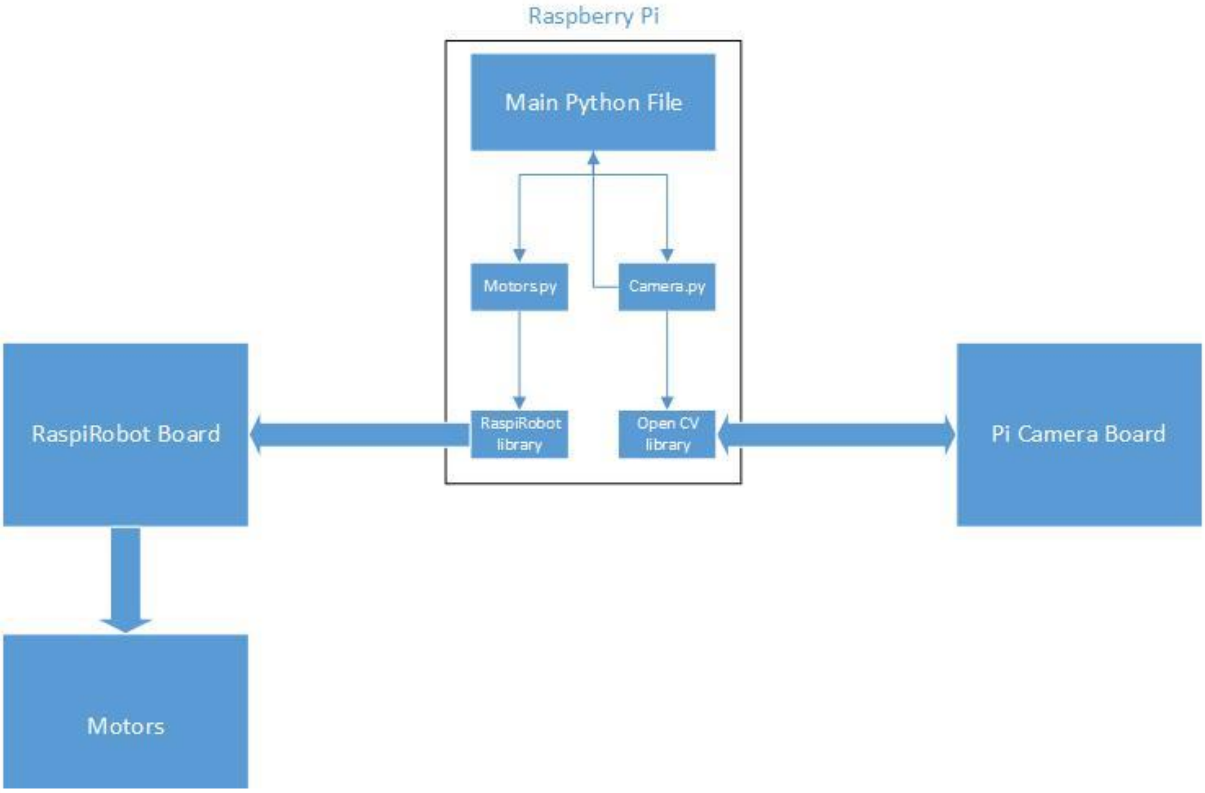


Figure 10: Data Flow

The Motors.py file uses a library the developer created. The original library does not contain PWM for controlling the motors speed however this alternative library was made to fulfil that requirement. Speed control is needed in order to control how the robot patrols an area. If the robot is always on full speed the camera will find it difficult to identify any objects. The Raspi Robot library controls the GPIO pins on the Raspberry Pi board, these pins control different parts of the Raspi Robot board. By applying PWM to the GPIO pins on the Raspberry Pi the voltage can be sent using pulses to vary the speed. When a sequence of pins on the Raspi Robot board has been set to high depending on the sequence the left or right motor moves forwards or backwards.

The open CV libraries talk directly to the Pi camera board and accumulates the video feed. Each frame is collected on the Pi. The data flow in figure 10 for the camera board travels both ways as the camera is sent data to initialise it and then sends the video feed back to the Raspberry Pi.

4.2.2 Robot States

Originally the robot was going to have four states. These states are shown in figure 11. You can see that there was a patrol state, attack state, post attack and a free mode. It was decided this solution would not be effective as the design was made when the robot would stop and then attack the target straight away. Unfortunately, the hardware used just isn't fast enough to achieve this standard. For the camera to maintain a framerate fast enough and the robot to act as soon as it sees the target the hardware used would have to respond in a much quicker time.

Patrol
Attacking
Post Attack
Free Mode

Figure 11: Original States

The free mode idea was removed altogether as this takes away the enjoyment behind the behaviours that the robot uses. The robot is supposed to be entirely autonomous, this provides a more complex problem for the user to solve. With the addition of a free mode where the user can control the robot this would take away the fun part of making a robot that completes its object by itself.

Figure 12 shows a visual representation of the actual states which the robot uses, these states provide a continuous cycle in which the robot decides the state transitions. This system means the user won't have to interact with the robot at all whilst the software is running. The bottom two states contain conditions which when met transfer the robots state to the previous state. For example, in the attack state if the robot is unable to centre the target it will change back to the adjust state. When it centres the target again the robot will transition to the attack state.

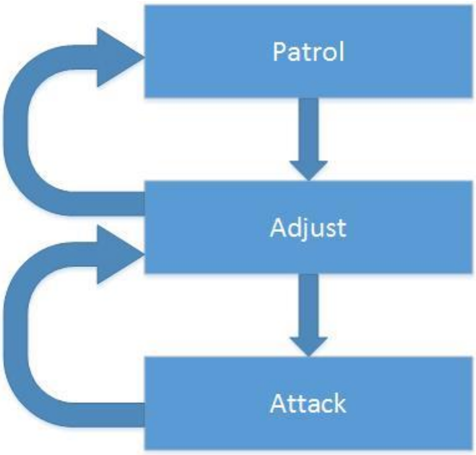


Figure 12: Actual States

4.2.2.1 Patrol

Patrolling is the initial state, in this state the speed and intervals are defined for controlling how the robot turns on the spot to patrol an area. This state is designed for the robot to look for its target and identify where it is. The factors affecting the robot's behaviour in this state are the:

- Method used
- Speed
- Interval

An interval is defined as the amount of time the motors will run for in that one iteration. The speed is applied straight to the motors using PWM. This is the base state therefore the robot won't traverse to any other states below this one. When patrolling a certain method can be used which might patrol in a certain way. The robot will only patrol on the spot however it will circle in different directions depending on the tactical advantage the user wants.

4.2.2.2 Adjust

The adjust state is important for directing the robot towards the target. Once again the inputs define whether this will be successful. The behaviours which can be changed in this state are the:

- Speed
- Interval

In this state the robot knows where the target is and will adjust so it can lock on to the target. If the target moves off screen the robot will use the last known position to direct the camera back to the targets position, or even follow the target if it moves. In figure 13 we can see how this works. The target is in view of the camera however it is not centred. The robot can now adjust to the right and centre the target.

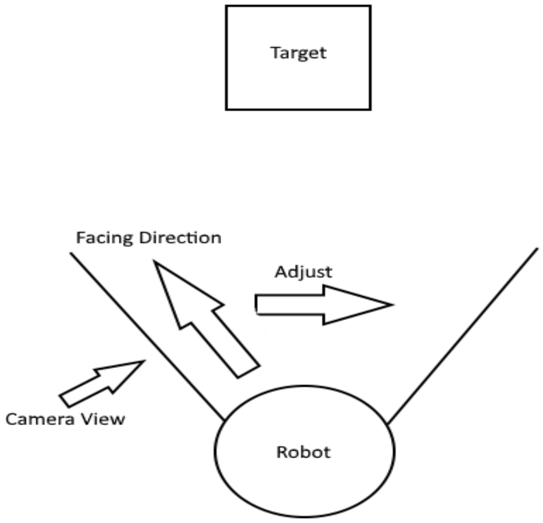


Figure 13: Adjust Diagram

In this state there is a timer which will send the robot back to the patrol state if it goes above a set value. This is so the robot can decide itself to change back to patrolling without any user input. Ultimately the user will decide how the robot will work in this situation.

4.2.2.3 Attack

When it comes to attacking the target, the robot uses a mix of adjusting and attacking. For as long as the target is centred the robot will move forwards. If the target goes out of range the robot can move back to the adjust state and turn in the direction of the target until it is centred, once this is done it will transition to the attack state again.

There are three variables that can be customised in this state:

- Left Motor Speed
- Right Motor Speed
- Interval

The robot will check whether the target is in range after every interval. The left and right motor speed variables are separate for calibration.

4.2.3 Code Design

The main Python code is set out simply in three files which will be the Main.py file, Motors.py file and the camera.py file. The user will use the main jousting robots file to change variables and adjust the behaviour of the robot. The code layout is made to be simple so the user can easily understand how to use the modules if they wish. The main file is the API that glues together the use of the camera and the Motors. The motor file or camera file can be imported into any code and use as needed. This gives the user a chance to create their own API if they wanted.

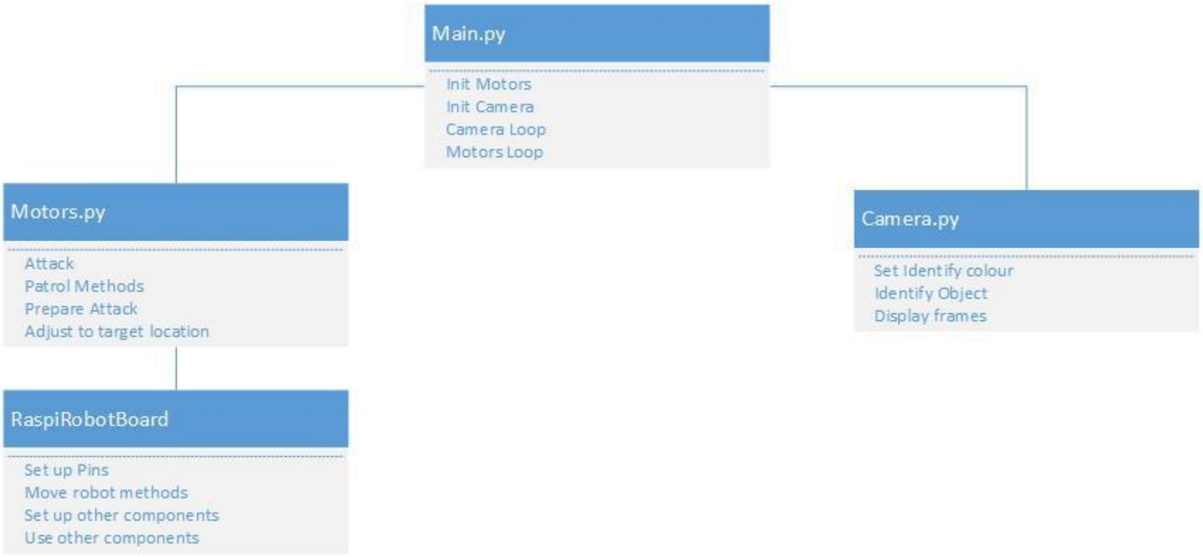


Figure 14: File Layout

Each method in each file will have the required parameters to alter things such as the speed in which the robot attacks, this means the user can call a method and simply set the values to whatever they like. This is a design for more advanced use of the robot. The Raspi Robot Baord class is contained within the Motors module so all of the motor aspects are within the same file. The camera loop and motor loop are both called from inside the Main.py file. These loops are used as separate update methods for the robot.

4.3 Software Implementation

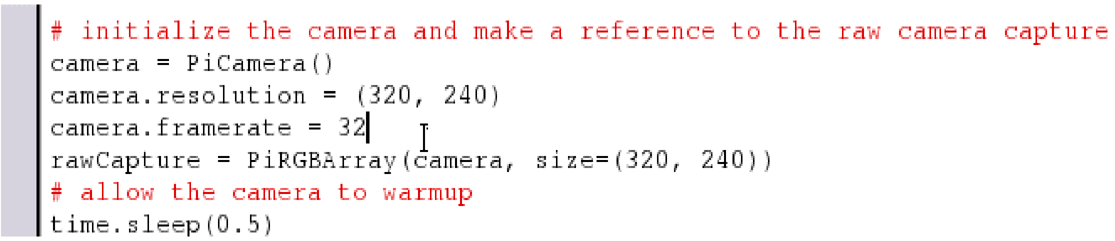
In this section there are details about how the software has been implemented. Certain parts of the code will be explained in detail.

4.3.1 Open CV and the Raspberry Pi Camera Board

Open CV contains a large amount of libraries; the install time is exceptionally long. Initially the raspberry pi one was used to install the libraries however later in the project a raspberry pi two was used, this meant the installation was much shorter (Adrian Rosebrock. 2015) however still a complex process as there are many different configurations for Open CV.

4.3.1.1 Camera Initialisation

The camera is initialised in the main file; this has been done to give the user the ability to change the different attributes of the frames to their liking. Figure 15 this shows the different camera settings that can be altered. These settings only effect the frame size and resolution, however there can be a large performance issue if this changes.



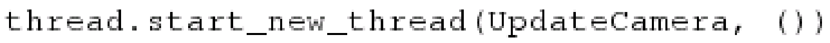
```
# initialize the camera and make a reference to the raw camera capture
camera = PiCamera()
camera.resolution = (320, 240)
camera.framerate = 32
rawCapture = PiRGBArray(camera, size=(320, 240))
# allow the camera to warmup
time.sleep(0.5)
```

Figure 15: Initialise Camera

The resolution that can be seen in figure 15 works well at a medium range, conversely if the resolution is increased the robot will be able to notice a target much further away and the opposite happens if the resolution is decreased. This happens due to the amount of pixels within a frame. If a frame has a higher resolution the robot can see more pixels therefore it can identify the different colours within those pixels more accurately. Less pixels means loss detail, consequently objects further away are either distorted to much or don't appear in the frame.

Towards the bottom of figure 15 there is a line of code that sleeps for half a second. This is required to warm the camera up before it is used. The camera board requires this warmup time to calibrate the sensors.

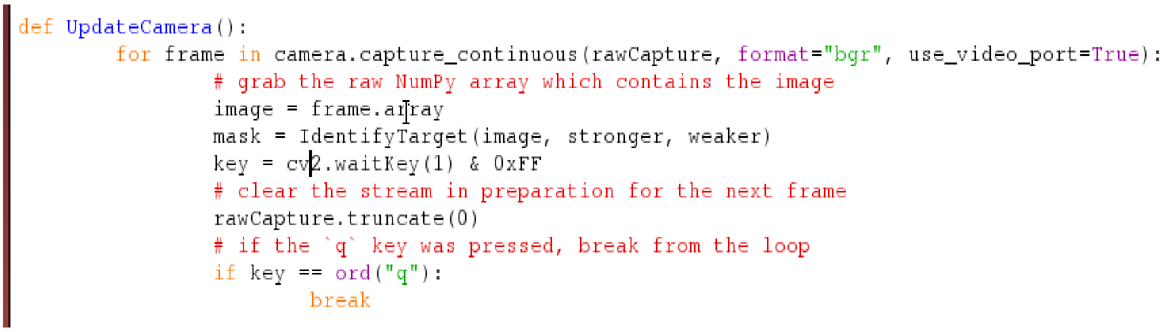
When the camera has been initialised the camera thread is then started. Initially the camera was made to run alongside the motors on the main thread however many issues were found when doing this. Frames would lag behind by quite a few seconds which meant if the camera saw a target the motors would not be notified until it had rotated too far. This is also one of the reasons why a Raspberry Pi two was used. The single core chip on the first Raspberry Pi just didn't contain the power to drive two threads at once.



```
thread.start_new_thread(UpdateCamera, ())
```

Figure 16: Camera Update Thread

When the thread is started the camera update method is called, this method is used to obtain each frame from the Pi camera board. The programme needs to grab each frame and quickly process it so that the robot reacts to the identification of a target swiftly.

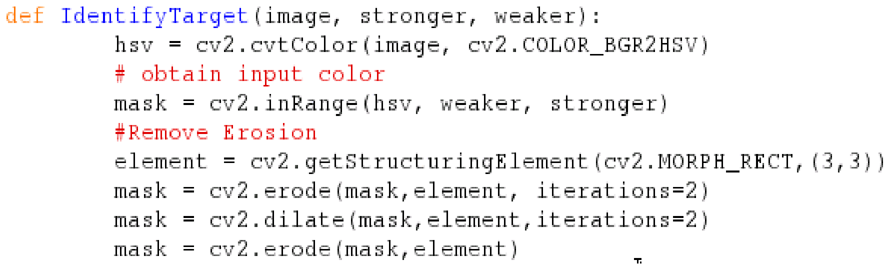


```
def UpdateCamera():
        for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
                # grab the raw NumPy array which contains the image
                image = frame.array
                mask = IdentifyTarget(image, stronger, weaker)
                key = cv2.waitKey(1) & 0xFF
                # clear the stream in preparation for the next frame
                rawCapture.truncate(0)
                # if the `q` key was pressed, break from the loop
                if key == ord("q"):
                        break
```

Figure 17: Update Camera Method

4.3.1.2 Image Processing

In order for the image to be processed it is passed to the identifyTarget method within the Camera module. When the image is being processed it requires to be formatted so the colours can be recognised easier. The image variable is sent through to the cvtColor method so the image can be converted to HSV (Hue, Saturation and Value) from BGR.



```
def IdentifyTarget(image, stronger, weaker):
        hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
        # obtain input color
        mask = cv2.inRange(hsv, weaker, stronger)
        #Remove Erosion
        element = cv2.getStructuringElement(cv2.MORPH_RECT,(3,3))
        mask = cv2.erode(mask,element, iterations=2)
        mask = cv2.dilate(mask,element,iterations=2)
        mask = cv2.erode(mask,element)
```

Figure 18: Formatting Image

Open CV requires the colour space to be changed, this is because HSV separates the intensity of the image from the colours information to make a much nicer approach to identifying colours in a given image. This splits up the colours components into a more readable set of values which can be used in algorithms such as the one used here. The mask is then used to find the colours within a certain HSV range, initially the robot will be set to identify the colour green, however the user will have full control of which colour the robot sees.

The colours in the image may be blurred or thick which could cause problems when adding a bounding box to the image. To combat this problem, the erosion and dilation methods utilised to give a more defined outline to the colours (OpenCV 2.4.12.0 documentation. 2016).

Here is an example of how erosion and dilation can affect an image:

- Original Image – The letter is quite thick and is not uniform as there are some bumps and gaps in the image.

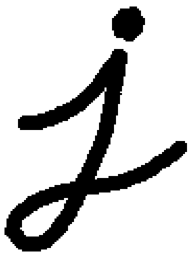


Figure 19: Original Image (Open CV. 2016. Original)

- Dilated Image – Dilation has smoothed out the edges here to make the image a lot sharper which means colours can be much more constant with less noise.

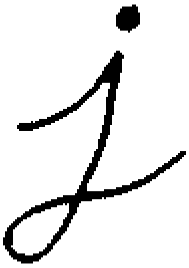


Figure 20: Dilated Image (Open CV. 2016. Dilated)

- Eroded Image – Erosion has made the letter much thicker and much more visible. This means if the target is far away we can bulk out the colours to show more of the colour the robot wants to see, therefore making the target much more visible.

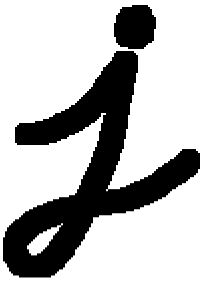
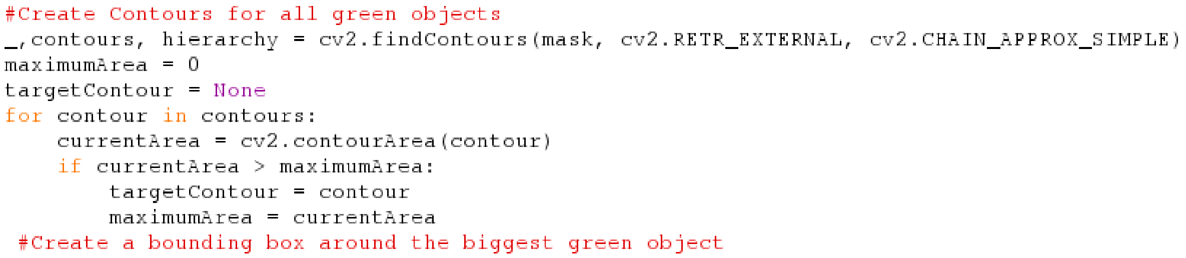


Figure 21: Eroded Image (Open CV. 2016. Erosion)

4.3.1.3 Attaining the Contours

What the program has managed to do so far is get the image, grab the mask and highlight where the colour is within range of the HSV's provided and then formatted the image to reduce any noise. The next step is to make this image understandable for the robot.



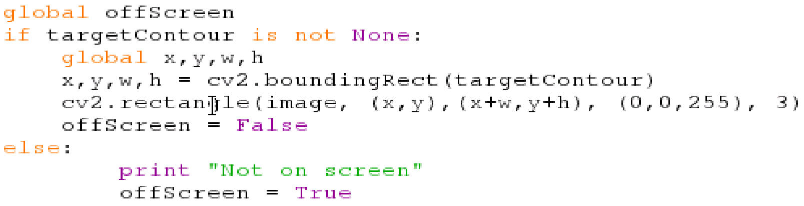
```
#Create Contours for all green objects
_,contours, hierarchy = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
maximumArea = 0
targetContour = None
for contour in contours:
    currentArea = cv2.contourArea(contour)
    if currentArea > maximumArea:
        targetContour = contour
        maximumArea = currentArea
 #Create a bounding box around the biggest green object
```

Figure 22: Contours

Open CV can be used to return the contours on an image, the robot can use this information to show where the object is on the screen and tell the motors how to behave depending on the position. The contours are returned as coordinates (x,y), these coordinates are the boundary points of the object (OpenCV . 2015. Contours : Getting Started). Once the contours have been grabbed a for loop is used to iterate through each one. If there is more than one coloured object on the screen than we want, then the program has to make sure it chooses the closest object otherwise the wrong object could be given a bounding box. The contourArea method is used to determine the closest area to place the bounding box around.

4.3.1.4 Bounding Box

Next we need to place a red bounding box around the target. To do this we need to know the exact coordinates of the robot. If we send through the targetsContours variable to the boundingRect method in open CV it will return the coordinates of the target.
These coordinates are used for two things. The first being to set a red border around the target on the image so the user can see that the robot has targeted the object. The second thing is to collect the coordinates as global variables which can then be used in the motors module to determine how the robot moves.



```
global offScreen
if targetContour is not None:
    global x,y,w,h
    x,y,w,h = cv2.boundingRect(targetContour)
    cv2.rectangle(image, (x,y),(x+w,y+h), (0,0,255), 3)
    offScreen = False
else:
        print "Not on screen"
        offScreen = True
```

Figure 23: Add Bounding Box

In Figure 23 we can see that there is an if statement around the section that draws the border. If the target is not on the screen, then a Boolean is set to true to notify the motors there is no target so it can use the correct behaviours. This is useful as the robot can decide which direction to move if it has seen the target but is no longer on the screen. This prevents the robot from losing track of the target by accidently turning too far when it’s preparing an attack.

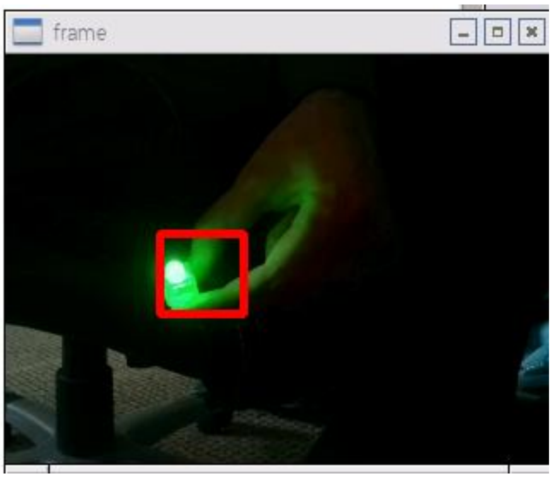


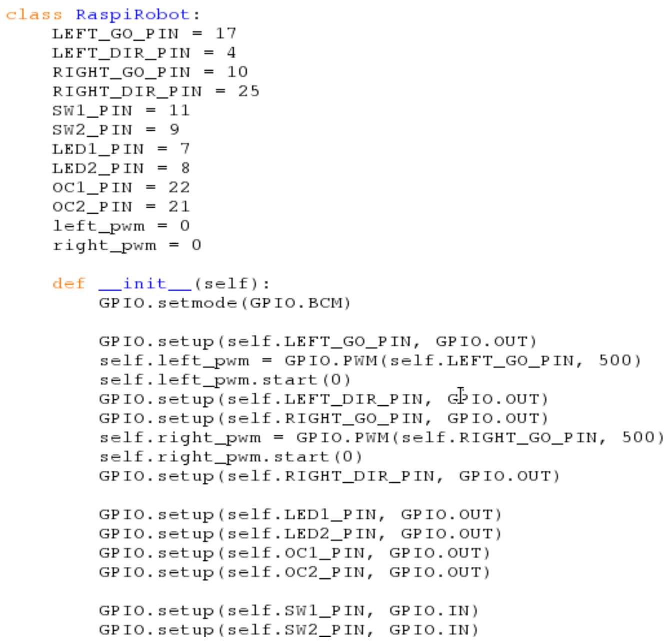
Figure 24: Bounding Box

4.3.2 Raspi Robot board

This part of the codebase wasn’t originally planned to be written by me, however due to the needs of the project the libraries provided by Simon Monk (2013) required to be edited. This meant re-writing the entire class using PWM. This library is contained within the Motors module and is extremely simple to use.

4.3.2.1 GPIO Pin Setup

Figure 25 shows how the pins are setup when an instance of the RaspiRobot object is made. PWM is then applied to the pins connected to the robots DC motors. There is no need to do anything else with the GPIO pins once this is done which makes interacting with the Raspi Robot board much more efficient.

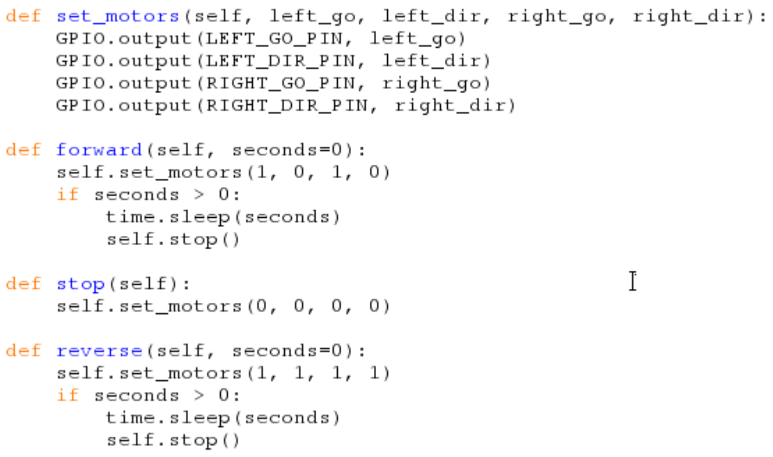


```
class RaspiRobot:
    LEFT_GO_PIN = 17
    LEFT_DIR_PIN = 4
    RIGHT_GO_PIN = 10
    RIGHT_DIR_PIN = 25
    SW1_PIN = 11
    SW2_PIN = 9
    LED1_PIN = 7
    LED2_PIN = 8
    OC1_PIN = 22
    OC2_PIN = 21
    left_pwm = 0
    right_pwm = 0

    def __init__(self):
        GPIO.setmode(GPIO.BCM)

        GPIO.setup(self.LEFT_GO_PIN, GPIO.OUT)
        self.left_pwm = GPIO.PWM(self.LEFT_GO_PIN, 500)
        self.left_pwm.start(0)
        GPIO.setup(self.LEFT_DIR_PIN, GPIO.OUT)
        GPIO.setup(self.RIGHT_GO_PIN, GPIO.OUT)
        self.right_pwm = GPIO.PWM(self.RIGHT_GO_PIN, 500)
        self.right_pwm.start(0)
        GPIO.setup(self.RIGHT_DIR_PIN, GPIO.OUT)

        GPIO.setup(self.LED1_PIN, GPIO.OUT)
        GPIO.setup(self.LED2_PIN, GPIO.OUT)
        GPIO.setup(self.OC1_PIN, GPIO.OUT)
        GPIO.setup(self.OC2_PIN, GPIO.OUT)

        GPIO.setup(self.SW1_PIN, GPIO.IN)
        GPIO.setup(self.SW2_PIN, GPIO.IN)
```

Figure 25: Raspi Robot setup

4.3.2.2 Initial Raspi Robot library

Figure 26 shows the original library. As we can see there is no speed control within any of the methods. This would make getting the robot to achieve its primary objective extremely difficult. The motors move at such a fast speed that the camera would have trouble analysing its surroundings and identifying any targets.

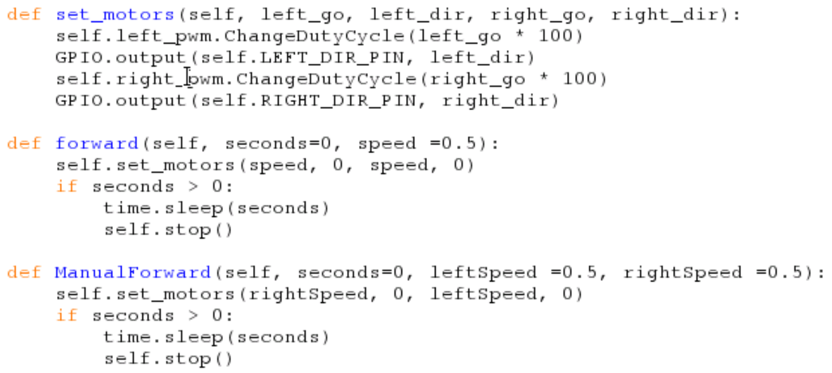


```
def set_motors(self, left_go, left_dir, right_go, right_dir):
    GPIO.output(LEFT_GO_PIN, left_go)
    GPIO.output(LEFT_DIR_PIN, left_dir)
    GPIO.output(RIGHT_GO_PIN, right_go)
    GPIO.output(RIGHT_DIR_PIN, right_dir)

def forward(self, seconds=0):
    self.set_motors(1, 0, 1, 0)
    if seconds > 0:
        time.sleep(seconds)
        self.stop()

def stop(self):
    self.set_motors(0, 0, 0, 0)

def reverse(self, seconds=0):
    self.set_motors(1, 1, 1, 1)
    if seconds > 0:
        time.sleep(seconds)
        self.stop()
```

Figure 26: Original Library

4.3.2.3 Updated Raspi Robot Library

The overall format of the file has stayed the same. Each method that makes use of the motors now contains a speed variable which is applied to the required pins. The default speed is set to 0.5, this represents half of the highest amount of pulses applied to the motors. This speed is generally quite slow and the slowest the motors can move. A lower value could be used, however due to factors such as friction and the fact that the motors require a quick spike in voltage before they set off these values are not advisable.



```
def set_motors(self, left_go, left_dir, right_go, right_dir):
    self.left_pwm.ChangeDutyCycle(left_go * 100)
    GPIO.output(self.LEFT_DIR_PIN, left_dir)
    self.right_pwm.ChangeDutyCycle(right_go * 100)
    GPIO.output(self.RIGHT_DIR_PIN, right_dir)

def forward(self, seconds=0, speed =0.5):
    self.set_motors(speed, 0, speed, 0)
    if seconds > 0:
        time.sleep(seconds)
        self.stop()

def ManualForward(self, seconds=0, leftSpeed =0.5, rightSpeed =0.5):
    self.set_motors(rightSpeed, 0, leftSpeed, 0)
    if seconds > 0:
        time.sleep(seconds)
        self.stop()
```

Figure 27: Motors Methods

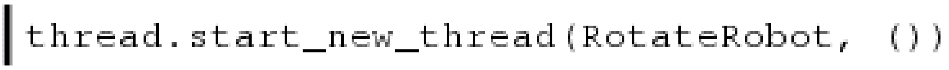
If we observe the forward method in figure 27 we can see that the set_motors method is called. This method requires four variables. Each one sets the value of a certain pin. In this case the first and third pin is set to a given value which is applied when the forward method is called. In the set_motors method the left motors duty cycle is set to the PWM value multiplied by one hundred. Having a value which is smaller makes more sense when using PWM, there is not much of a difference when only changing the value by a small amount. The direction and pin in which is used to move the correct motor is set which causes the motor to start moving. The ManualForward method is a new method which is used to help the user cancel out any calibration problems with motors when it moves forwards at a fast pace, such as when commencing an attack. The user can specify the speed of both wheels when going forwards. The robot can quite easily veer off to the side which can put the robot in danger of losing the joust.

4.3.3 The Motors

The motors module contains the complex behaviours of the robot. This is the section in which the user will interact with the most. The behaviours of the robot are defined here and updated. Each state transition is also made here.

4.3.3.1 Motor Initialisation

To start the patrol state, the RotateRobot method is started in a new thread. This is done in the robots main file alongside when the camera thread is started.

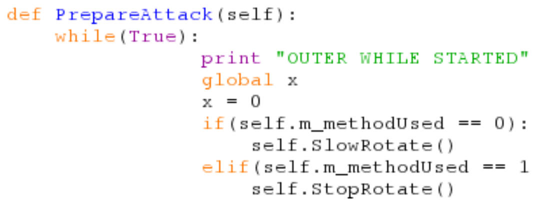


```
thread.start_new_thread(RotateRobot, ())
```

Figure 28: Rotate Thread

4.3.3.2 Motor Outer Update Loop

The robot rotate method then calls the prepare attack method from within the JoustingRobotMotors class from the Motors module. This method essentially does the updating of states for the robot. The user will require to modify this method if they add any other methods for patrolling. The first if statement chooses which method is to be used for surveying an environment, this section of code is where the patrol state begins.



```
def PrepareAttack(self):
    while(True):
                print "OUTER WHILE STARTED"
                global x
                x = 0
                if(self.m_methodUsed == 0):
                    self.SlowRotate()
                elif(self.m_methodUsed == 1):
                    self.StopRotate()
```

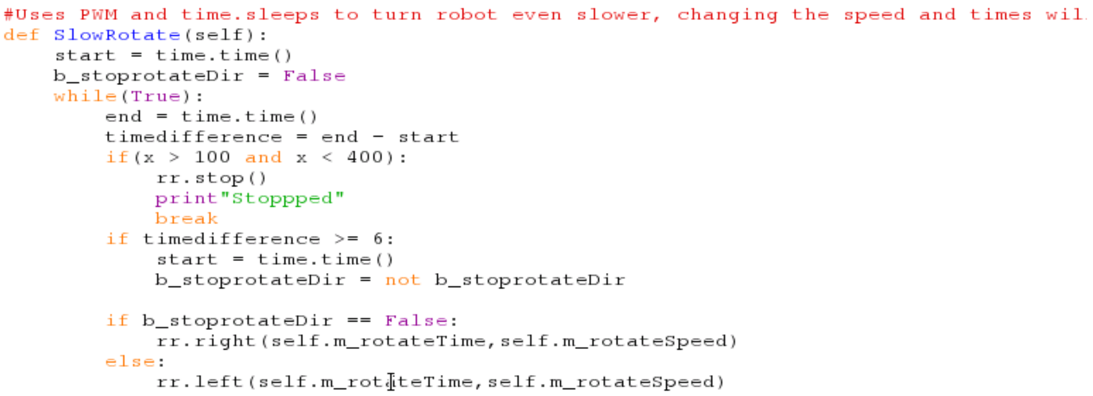
Figure 29: Prepare Attack

You can see this this code is inside a while loop, the while loops parameter is always set to true. This is the motors main update loop. This loop holds the robot within the three state design, it ensures the robot can transition backwards through the states and forwards.

If the user sets m_methodUsed to 0 then we can see in figure 29 the SlowRotate method is called. If they choose set it to 1 then the StopRotate method for patrolling is used instead. Within this section the user can use their own patrol methods to patrol an area. They can simply add another if statement to call their method in which they have created. The global variable x contains the x axis value from the camera module which is used to set the bounding box of the target. The robot sets x to 0 so that if the robot does start patrolling after transitioning back from the attack state or adjust state, it re-initialises the targets position to prevent the robot from moving back into the adjust state even though there is no target within range

4.3.3.3 Motor Patrol Method Example

Figure 30 shows an example of one of the rotate methods which is called when the robot is patrolling. A timer is used to determine how long the robot has been rotating in a direction. When the timedifference variable is greater than 6 seconds the robot will switch direction. Once the timer if statement’s condition is true the Boolean is flipped. This happens until x (the targets position) is between the values of 100 and 400. At this point the program breaks from this while loop and returns to the PrepareAttack method.

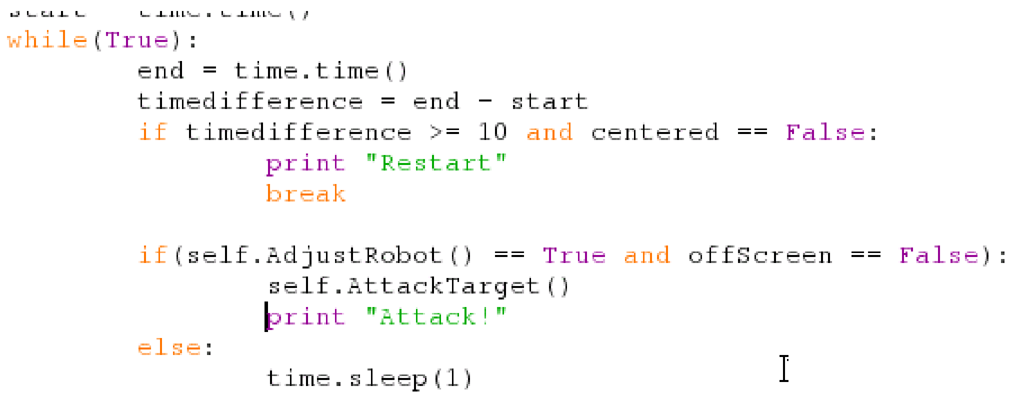


```
#Uses PWM and time.sleeps to turn robot even slower, changing the speed and times wil
def SlowRotate(self):
    start = time.time()
    b_stoprotateDir = False
    while(True):
        end = time.time()
        timedifference = end - start
        if(x > 100 and x < 400):
            rr.stop()
            print"Stoppped"
            break
        if timedifference >= 6:
            start = time.time()
            b_stoprotateDir = not b_stoprotateDir

        if b_stoprotateDir == False:
            rr.right(self.m_rotateTime,self.m_rotateSpeed)
        else:
            rr.left(self.m_rotateTime,self.m_rotateSpeed)
```

Figure 30: Slow Rotate Method

4.3.3.4 Adjust and Attack Update Loop

Once the robot has seen a target it’s time to adjust to the targets position. This while loop loops through continuously and checks if the target is centred. To prevent the robot getting stuck in the while loop a timer is set for 10 seconds. If the timer is greater than ten, then the robot transitions back to the patrol state and will call the survey method.



```
start = time.time()
while(True):
        end = time.time()
        timedifference = end - start
        if timedifference >= 10 and centered == False:
                print "Restart"
                break

        if(self.AdjustRobot() == True and offScreen == False):
                self.AttackTarget()
                print "Attack!"
        else:
                time.sleep(1)
```

Figure 31: Adjust Update Loop

If the timer hasn’t reached ten seconds, then within the second if statements condition the adjustRobot method is called. The adjust method returns a bool which is turned to true if the robot has centred on the target. We can also see there is a Boolean called offScreen. This Boolean is true whenever the camera doesn’t see a target on the screen. The offScreen Boolean is extremely important as the x variable which is returned from the camera module can sometimes stay the same after seeing the target in the centre of the screen, if the robot over shoots the target and the target is no longer on screen the robot can sometimes think it’s still in the same position, as x is still set to a particular value. If the robot has seen an object, then the attack method is called. The camera may have processed many frames between adjusting the robot and attacking therefore this has to happen quickly to make sure the robot is accurate.

The adjust method is extremely important for the accuracy of the robot. The initial design was for the robot to stop as soon as it saw the target. A problem occurs when the robot sees the target for the first time. Due to there being a small amount of resources on the Raspberry Pi and the fact that the image is only processed so fast there is no way for the motors to react in time when it first sees the target.

When using the raspberry Pi 1 the problem is much worse as the lag on the camera is far greater, this causes the framerate to decrease considerably and sometimes the camera module will even crash.

In figure 32 we can see a diagram showing how adjusting works. The direction the robot is facing when the frame is processed is not accurate for the physical robot. Using the initial state design of the robot that doesn’t contain an adjust function, the robot would attack straight away without knowing the target actually isn’t in range.

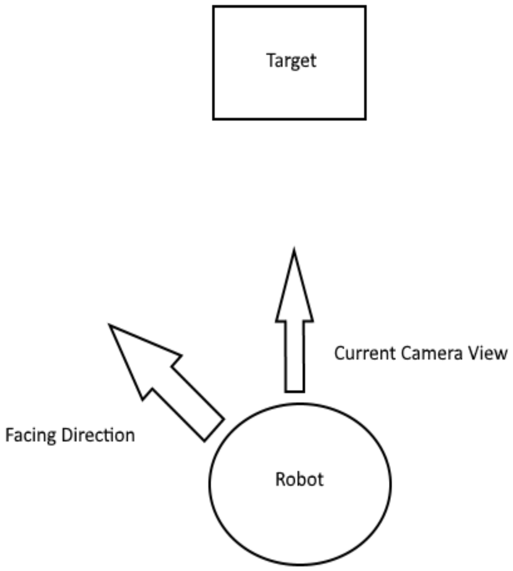


Figure 32: Camera lag problem

Implementing the ability for the robot to adjust gives the robot the capability to correct the lag. The success of the adjust method is determined by the values passed to the robot by the user. The robot can adjust using intervals for each movement, slowly rotate or even rotate fast. There are many different configurations for the behaviour of the adjust state. If the robot is given full speed for half a second it will turn to the target by jumping right or left. If it is given a speed of 0.5 and a longer time for each interval it will turn slowly.

Back to the python code In figure 31 right at the end time.sleep is called to give the camera time to assess the targets location, this has to be done to ensure the exams the frame well. This means the robot will always have to stop for a given amount of time after it adjusts once.

4.3.3.5 Adjust Method

The adjust method returns a Boolean to identify weather the target is centred or not. When the method is called the most recent frame processed is split into three blocks. The outer two blocks are used to determine if the robot needs adjusting by deciding whether the robot is within the area.

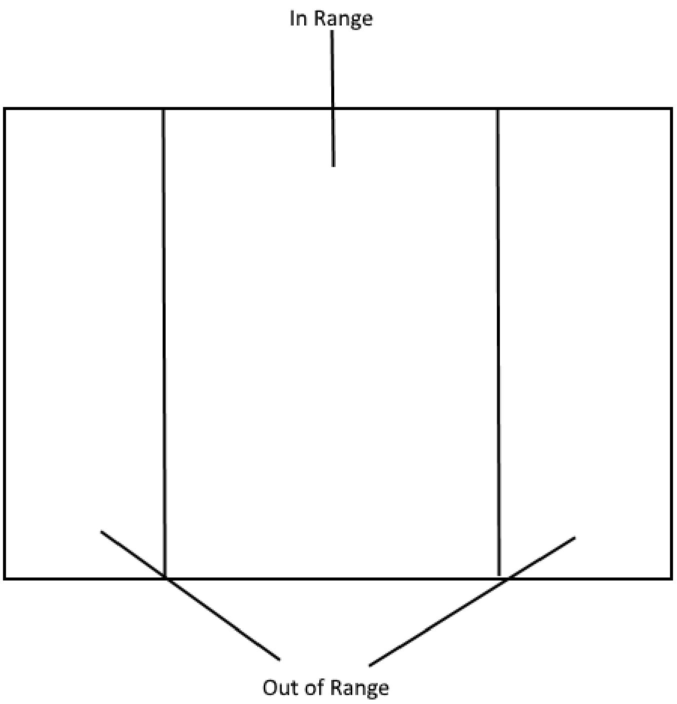
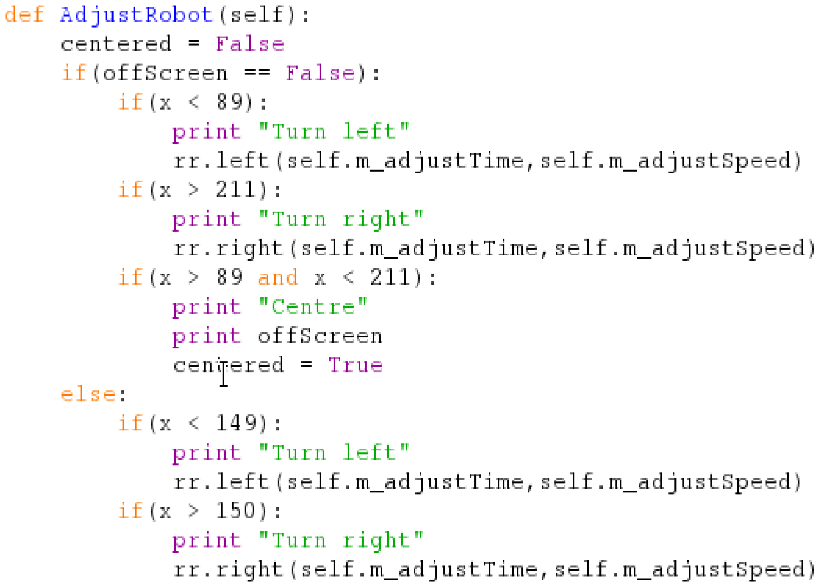


Figure 33: Camera Ranges

The image above shows a rough view of the different section the camera perceives. The left and right section of the frame are the out of range zones. If the target is within these two sections, the robot will take action and turn to the left or right depending on the section of the frame which the target populates. When the target is within the in Range section this is when the adjust method returns the centred Boolean as true.



```
def AdjustRobot(self):
    centered = False
    if(offScreen == False):
        if(x < 89):
            print "Turn left"
            rr.left(self.m_adjustTime,self.m_adjustSpeed)
        if(x > 211):
            print "Turn right"
            rr.right(self.m_adjustTime,self.m_adjustSpeed)
        if(x > 89 and x < 211):
            print "Centre"
            print offScreen
            centered = True
    else:
        if(x < 149):
            print "Turn left"
            rr.left(self.m_adjustTime,self.m_adjustSpeed)
        if(x > 150):
            print "Turn right"
            rr.right(self.m_adjustTime,self.m_adjustSpeed)
```

Figure 34: Adjust Method Example

The method shown above shows how if statements are utilised to decide which way the robot turns. The global x variable holds the pixel position on the screen of where the target is on its x axis, it is the same variable which is used in the camera module and the PrepareAttack Method. The first two nested if statements decide whether the target is within the left out of range zone or the right out of range zone.

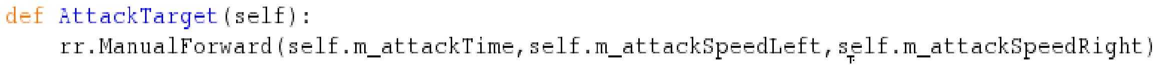
The final nested if statement tells the robot that the target is within range and the robot should commence the attack.

The offScreen variable is also used to determine the last known position of the target and turn in the correct direction when the targets off screen. The screen is split in half when the robot is off screen, so if the last know location is less than half the screens pixel width then the robot turns left. If the last known location is greater than half the width, then the robot will turn right.

4.3.3.6 Attack Method

The attack method is called within the adjust loop. The time in which the robot stays in the attack state depends entirely on if the centred bool is true. This is an advantage as the robot has the ability to adjust whilst attacking. This is because if the centred bool returns back to false (meaning that the target is not in the centre), the robot can adjust in the correct direction to fix this and then continue attacking. The transition between attacking, adjusting and back to attacking happens very often however both states use each other to reduce the possibility of being inaccurate.

The attack method is fairly simple; it contains a line of code which moves the robot forwards. The motor function which is called has the ability to choose the speed of the separate motors. The decision to do this was made to prevent the robot from veering off to the left or right. The user now has the chance to calibrate the motors to make the attack method more accurate. The problem here is that two motors can’t accurately move at exactly the same speeds every time on every robot. The speeds of each motor are adapted to fix this problem by amending the ManualForward PWM parameters.



```
def AttackTarget(self):
    rr.ManualForward(self.m_attackTime,self.m_attackSpeedLeft,self.m_attackSpeedRight)
```

Figure 35: Attack Method

Staying in the attack state can also depends on the interval the robot attacks for. If the time specified is 5 seconds, the robot will travel forwards at a given speed for that amount of time before running the attack method again. If the robot is given 0.1 seconds, then the robot will run through the attack method many times which will slow down the speed but will give the robot time to adjust if it is needed.

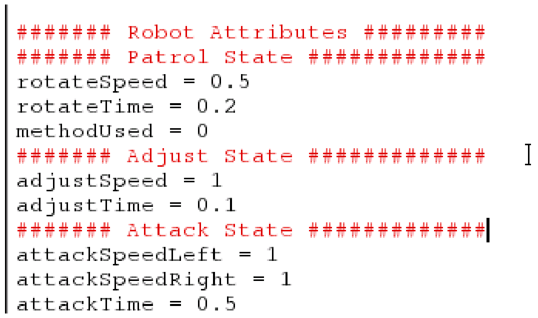
4.4 User Interaction

This section explains how the user will interact with the robot. It outlines how the user can use this robot to learn about Raspberry Pi development. The user will be supplied with a challenge sheet which can be seen in appendix C.

4.4.1 Main Jousting Robots Module

The user’s interaction with this file can be complex as there are many variables which change the behaviour of the robot. The user will set the attributes of the robot when it is initialised. This will be done by inputting preferred values into the variables provided.

This is key to the success of the robot when jousting. The user will set the variables on one robot and another user sets the variables on another robot. Once this is accomplished the user’s robot with the smartest set of attributes will attack the other robot first.

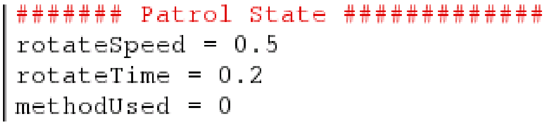


```
####### Robot Attributes #########
####### Patrol State #############
rotateSpeed = 0.5
rotateTime = 0.2
methodUsed = 0
####### Adjust State #############
adjustSpeed = 1
adjustTime = 0.1
####### Attack State #############
attackSpeedLeft = 1
attackSpeedRight = 1
attackTime = 0.5
```

Figure 36: Robot Attributes

4.4.1.1 Rotate Behaviour

The rotate variables affect the robots patrol state. The speed determines how fast the robot rotates at each interval.

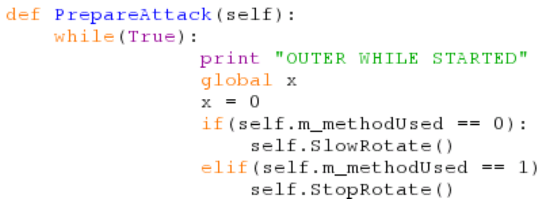


```
####### Patrol State #############
rotateSpeed = 0.5
rotateTime = 0.2
methodUsed = 0
```

Figure 37: Robot Attributes

The rotateTime variable can be set for as long as the user likes, however if the time is too long there will be a large performance hit which will significantly affect the use of the camera. This happens as the thread sleeps after the motors are started when using a specific time, which hogs a lot of the CPU. Using the variables from figure 37 the robot will start surveying the room at a speed of 0.5 which is the slowest speed the robot can move. This speed will be delivered for 0.2 seconds at a time. This works as sort of a PWM on top of a PWM. The power provided for 0.2 seconds is enough to get the motors moving, when the interval has finished the motors will stop spinning for a very short time, then when the while loop comes back around and runs the motors again the robot will move, giving the affect that it is moving very slowly.

The methodUsed variable determines which patrol method is used. On this example it is it is set to 0. If it is set to 1 the program will choose another method depending on how the if statements are set up in the main motors update loop.



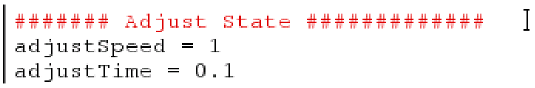
```
def PrepareAttack(self):
    while(True):
                    print "OUTER WHILE STARTED"
                    global x
                    x = 0
                    if(self.m_methodUsed == 0):
                        self.SlowRotate()
                    elif(self.m_methodUsed == 1):
                        self.StopRotate()
```

Figure 38: Survey Method Choice Example

In figure 38 we can see if the method is set to 0 the slow rotate method is called. This part of the code is where the user can add their own patrol methods. The manual and challenge sheet will explain how to do this to the user. The user may find using the stop rotate method much more accurate as it pauses every time an interval has done which will give it time to process an image. The slow rotate method however could be utilised to be faster therefore this would speed up the chance to find the target quicker. This is a small example of how the user could decide what method to use.

4.4.1.2 Adjust Behaviour

The accuracy of the robot's target identification process is determined in here. The adjust variables affect the robot's behaviours the adjust state.



```
####### Adjust State #############
adjustSpeed = 1
adjustTime = 0.1
```

Figure 39: Adjust Variables

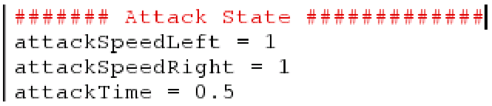
The robot can easily get stuck if the user sets the behaviours to exceed the targets x position every time it adjusts. If the interval is set for too long there is less of a chance for the robot to stop when the target is in range. The robot will always adjust using the same speed and intervals for each direction it rotates. Ideally the user will have to learn to make the robot adjust very slowly or in small intervals so it jumps left or right. There is always a one second sleep between each interval so the robot has time to observe the frame before using the motors again. Originally there was still a small amount of latency when the robot moved, putting the motors thread to sleep seemed to reduce lag and increase the robot accuracy.

Getting the right speed and interval can be a tricky task, the robot needs to be able to move at a quick pace but not so quick that it suffers from a lack of precision. This may mean thinking into how much per pixel the robot needs to adjust for the target to move into the in range zone.

4.4.1.3 Attack Behaviour

As you have seen previously all of the different state variables have similar formats. They basically contain a speed variable and a time variable. The accuracy of the motors in the patrol and adjust state are generally pretty good. If the robot turns using a set speed one way, when it turns the other the speed will be exactly the same even if the motors don't turn at exactly the same speeds.

The problem which occurs when the robot attacks is that it requires to move straight. This is where the robot starts to slow down as it has the problem of having to change states constantly. Changing states can be useful as the robot can still track the target if it fails to attack through that interval. The user may want the robot to be accurate the first time and not have to adjust every time it moves forwards. To resolve the issue here, the user can input the speed of both motors meaning that they can calibrate the motors to force the robot to travel in a straight line.



```
####### Attack State #############
attackSpeedLeft = 1
attackSpeedRight = 1
attackTime = 0.5
```

Figure 40: Attack Variables

There are various ways in which the user can decide to set the attack variables. The robot could travel forwards at full speed for a few seconds then check for the target and adjust to it or even hit the target at full speed, or it could drive forwards slower and check the target often which would increase the accuracy.

4.4.2 Motors Module

There are two sections to this module which the user can utilise. The first section is the JoustingRobotMotors class. The user may want to add some methods to this class. The challenge sheet contains a section providing the user with sudo code to write their own methods and an explanation of how it will work.

The Raspi Robot class is similar to the JoustingRobotMotors class. The user can simply create an instance and call which ever method they wish. This could include calling a method to turn the robot left or even use other aspects of the Raspi Robot board such as the LED lights.

4.5 Testing

As the robot cannot always be predictable and there are many other environmental factories which affect the robot's movement and sight, testing is rather difficult. In the end the only way to formally test the robot was to judge how it behaved there and then. Initially FAT testing was going to be the main form of testing. This would not work however as the expected output is unknown. To properly test the robot a predicted output was recorded however this would still be unpredictable. The tests for the project are provided in the appendix.

There are three tables; patrol tests, adjust tests and attack tests. These purely test the behaviours of each state in certain situations. The idea behind this is to make sure the robot actually works the way it is thought to and also get a impression on how using different behaviours affect the robot's performance.

5 Project Management

5.1 Time Plan

The project changed quite a lot throughout, many issues occurred which would affect development. Planning was key to overcoming this issue, therefore many time plan revisions were made to overcome this problem. In the appendix of this document you will find the various revisions of time plans. As you can see certain features were removed and replaced with other features, for example the follow target task was removed in the fifth time plan. This was done as it became clear this task was not necessary as the project moved forwards. There was no need to follow a target as the adjust state was introduced which would make this task pointless.

As the project moved along problems such as the Raspi Robot board not having the required features or the Raspberry Pi was not powerful enough to run this project would occur. Alterations were made to the time plans to adapt to this.

5.2 Scrum and Vertical Slicing

Scrum is an agile development methodology used in industry around the world. Using Scrum to manage a project means that the project can be altered depending on the specification changes. To manage jousting robots, I had set up my own variation of Scrum. This needs to be my own variant as I am not working in a team and I am also working against a gantt chart.

In this variation of Scrum, weekly reviews were carried out to determine the progress that week and update my diary. If any problems were found, further investigation was needed and the time plan would be rearranged. As Scrum is so flexible I can easily remove a task or change it without having any other problems. If the waterfall methodology was to be used, then any changes would mean going back to the design stage of the project which would make any code I had already done obsolete. Every two weeks I planned to have something to demo which could potentially be a finished product. I have managed to stay organised by using a Scrum board to keep track of my current tasks in a sprint.

Overall this methodology of agile development has been extremely useful. It was very affective when the project came to a halt due to some unplanned obstacle. I managed to rearrange the time plan and incorporate the task to fix this into my next sprint.

The other method utilised here which works well with Scrum is vertical slicing. The project has many features that entirely depend on certain functionality to be implemented. Using vertical slicing the project was completed by splitting the robot into separate sections. For example, the camera section was completed to its entirety, the next stage was to implement the motors. Using Scrum at the same time, even though I may have completed the camera, I could still move back to that section of the project as the overall plan was adaptable. The project was built up a section at a time until completed

5.3 Managing the Software

With the project being based on introducing people into Raspberry Pi development the user has to feel like the layout of the project is helpful and user friendly. This meant managing the code with care and making sure it was readable enough for the user. All of the files have been set out so the user can easily identify which variables can be modified to change the behaviours of the robot. Everything else in the files are neatly wrapped in methods which then talk to each other to share data. The camera and motors module can be utilised in many ways.

Here are some examples of how the different files could be used and why they have been managed this way:

- Camera module - can be used to identify a certain colour on any image. All the user would have to do is call the identifyTarget method and pass through the image with the colour range. This could be useful in many other projects.

- Motors Module - The user can easily use this module to call any method from the JoustingRobotsMotor class. All the user has to do is make an instance of the class and call any methods from within it. The methods are named so the user can easily understand what each method does.

- Updated Raspi Robot Library – This can be used exactly the same as the original library which Simon Monk wrote, except the PWM can be set to a specific value which gives the user the chance to control the motors speed. The Raspi Robot class is kept within the same module as the JoustingRobotMotors class to ensure that everything is tidy and kept together. None of the Raspi Robot library methods need to be changed as everything that can be done in this library has already been finished.

There is some object orientated programming used, however most of it is procedural. The main file runs instruction after instruction, when the motor and camera modules are used the object orientated side of python is exploited. It is written this way to show clearly each state change and what parts of the code result in changes to the robot’s behaviours.

The project comes with an installation file which the user can run using the terminal on the Pi. This file can be used at any time to set the robots behaviours back to default, the whole idea of the project is to use trial and error to make an effective robot, the user will find themselves wanting to reset the project many times, thus an installation file will be convenient.

5.4 Managing the Hardware

Creating a project which only requires software can be fairly simple when it comes to maintaining an effective development environment, depending on the project obviously. The management of hardware when it comes to developing a robot on the other hand can be slightly trickier and more time consuming. To make matters worse certain aspects of the robot may not work in certain situations, such as the motors not working when only using a USB cable to run the Pi.

To manage these issues every time development was to take place on the project, the requirements for that stage of development had to be inspected in order to maintain a constant work flow and not halt development. In the first stages of the project the only power sources that could be used were a USB cable or a battery pack. When setting up the project the USB cable would be enough to run simple parts of the code. As the project became more complex the Raspberry Pi would struggle with just using a USB cable as it provided a small amount of voltage, this meant using the battery pack. Halts in development would occur when the batteries would deplete there was no way to test the robot, this meant waiting hours for the pack to recharge. To overcome this a second battery pack was obtained. The packs would alternate being charged to make sure that every time the robot was needed for development it would work.

There were also many risks when it came to the storage of data on the Raspberry Pi. Mini SD cards were used to run Raspbian and store the project. There is always a chance that this data could corrupt. On many occasions the robot would turn off when the battery was low without any warning, this could affect the data on the Pi. To ensure this would never be a problem, images of the SD card were stored onto a desktop and the code uploaded to source control so everything was backed up.

Overall as the robot’s hardware is nowhere near as advanced as a desktop, writing code on the device and testing the robot proves to be much slower than the average software project. This is why managing the hardware correctly proved extremely useful as every second of development time was crucial to the project.

6 Evaluation

6.1 Objectives Accomplished

The objectives of the project have changed over the course of the projects lifecycle due to the feasibility of the objectives. This section explains which ones have been accomplished.

6.1.1 Fully functioning tracking system

The tracking system has been implemented successfully and provides very good feedback which the Pi can use. The robot can see the target and keep track of it even when the target moves out of sight. This objective was essential to meet the overall aim of the project as the vision system is the only way of knowing where the target is. Initially this objective was not going to be accomplished in the way which was planned as the Raspberry pi 1 just was not powerful enough to cope with a vision system. The framerate was much to low, causing the camera to be inaccurate. This was overcome by replacing the Pi 1 with a much more powerful Pi 2. The frames per second managed to be maintained at a usable level which has helped increase the accuracy of the robot to a point where it can be effective and therefore made this objective a success.

6.1.2 Correct and precise movement of the motors

This objective has only been partially met as the movement of the robot is not as precise as originally intended. Luckily anybody using the robot can change the alter the motors behaviours. This means depending on the situation the robot’s movements can be adapted by the user. The reason for the lack of precise movement is that the motors require a spike in voltage in order to move. Momentum is utilised here to move the wheels. This would mean if the user set the wheels to move at a slower speed sometimes the motors just won’t move causing the robot to stand still. In a situation where the robot is fighting another robot this may cause it to fail its objective and be attacked first. This is why intervals were implemented. If the user wanted the robot to move a small amount they can set the speed to a high speed and have the interval for a short amount of time which would cause the robot to jump in a given direction, however this will never be as smooth and accurate as slowly turning. The motors still work to a level where it can be utilised to make an effective jousting robot.

6.1.3 Behaviours

Behaviours are a key aspect of the robot which is required to make the user think about how to create an outstanding jousting robot. The three states define how the robot acts and the values given to the variables within these states will affect how each state works. This objective has been met successfully. The design behind the three stages is intended to give a much more organised view on how the robot will fight. There is a learning curve here which the user must understand in order to optimise the robot. The behaviours set for each state will impact its performance depending on the state.

6.1.4 Fully automated

This objective was designed to make it clear that once the user has set the robot up, they will take no part in the actual performance of the robot. Unfortunately, this is not entirely the case. The robot will work on its own until it has attacked a target. Once the attack has finished the robot will transition back down the states without moving in any way away from the target, this means the user will need to move the robot back to a suitable position. The objective was partially successful as the robot will do most of the jousting on its own.

6.1.5 User Engagement

Engaging the user was not as entirely successful as intended. Initially the objective was to make a UI which would change the behaviours, however this just wasn't feasible when also attempting to make a fully automated robot. Added hardware would be needed such as a screen and a way of inputting data. The user will preferably use a graphical sharing system to interact and change the robot's behaviours, or they could simply plug the Pi into a monitor. The user will also have to go to the main project file to change the behaviours. The overall aim of the project will be met still, as the user will be forced to interact with the Raspbians user interface and get familiar with using a Raspberry Pi. They will also get the opportunity to use IDLE to modify the behaviour variables and even add their own code. As the user is forced to engage themselves and learn about Raspberry Pi development this objective has been met.

6.1.6 Summary

The objectives have been either completed or partially completed. Even though some have not been met fully the robot still functions well and provides a fun experience. The fully automated objective has mostly suffered however this is only a mild cost compared to the possibilities of other aspects failing within the project. The task set out to complete this objective wasn't practicable as it required more hardware.

6.2 Further Development

This section explains how the robot would be improved if the project was carry on. Some features such as the behaviours UI were to be implemented however due to certain issues this was not feasible within this project.

6.2.1 Detection System

The vision system already works well, however there is potential for a more accurate system. The robot can see the object however it has no consistent way of determining whether the target has been hit. This could be improved using ultra sound sensors to determine the distance of the target. When the distance has reduced to 0 then the robot has hit the target and should move back to its original position. When the robot lines up with the target the ultrasound sensors would determine the distance of the robot. This could also be done by determining the distance using two lights. Knowing when the target has been hit could mean scoring would be implemented as well, the Pi could relay back to a desktop when it has been hit and update a scoreboard on a desktop.

6.2.2 Improved Chassis

The current chassis motors have a limited slow speed, this is mostly due to the heaviness of the actual robot and the power needed to move the motors. This affected the precision of the motors. A more advanced chassis would be beneficial as the robot could move slower. This would mean the robot has the ability to accurately detect a target without having to patrol or adjust using intervals. For example, with the robot's current motors the only way to move truly slow is to jump and stop, ideally the robot wants to rotate very slowly without the need to stop. Stepper motors could be useful here as they provide more precise movements, the robot could even calculate how far its travelled. The price of stepper motors is expensive and they do have a limited top speed however.

6.2.3 User Interface

In the current state the robot's behaviours can be changed using an editor to edit the variables in the code. This can change the speeds, intervals and methods used for patrolling. A user interface would give the user a chance to do this in a nicer environment. If the user was using for example VNC, to interact with the robot this would mean the behaviours could be changed without having to stop and re run the robot.

6.2.4 Desktop Jousting Robot Application

At the moment the robots will be scored based on judgement. Using an advanced detection system as mentioned previously the Pi would be able to send information back to a desktop application which would show a score for that game. The games would also be managed and tracked by this desktop application. For example, if a tournament was to take place the user would set this up on the scoring application and organise the tournament through that. There would have to be some thought into what way the robot would interact with a desktop. Bluetooth could be used or even Wi-Fi, however there would be some limitations therefore a bit of research will be required in order to decide the practicability of different communication methods.

7 Conclusion

This project was set out to create a fun and educational robot application for anyone to use. It would engage the user and have them generate a robot using different behaviours which would be successful when jousting against another robot. The user will enjoy interacting with this project as it provides an interesting way of learning Raspberry Pi development. Friends could hold tournaments and use their knowledge of electronics and Raspberry Pi development to enhance the robot even more. As the user will gain experience with using this project they might even use this knowledge to kick-start their career into computer science or incorporate Raspberry Pi's into their industry. Educating is a key aspect to this project and has been helped with use of an already fun and educational board, the Raspberry Pi.

In conclusion I believe this project has been a success. Even if there was no user interaction the technological hurdle of creating a partially autonomous robot has been a thrill to overcome. I have learnt a new language, integrated a vision system, movement system and totally upgraded a professional library. Learning to create multithreaded applications in a new language has been a steep learning curve. I have had the ability to learn many things about software and hardware development such as working on pieces of hardware with limited resources and using I/O pins to interact with the hardware. This project also still has massive potential and could be made into a package used within education as well as by anyone wanting to expand their learning at home or even just to have fun.

Appendix A: Time Plan Revisions

Time Plan Revision 1

		University Calendar Weeks																																
#	Task Name	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
1	Configure Raspberry Pi			▨																														
2	VNC			▨																														
3	Add Camera to Raspberry Pi			▨																														
4	Install Open CV				▨																													
5	Get Video feed from camera				▨	▨																												
6	Camera testing					▨																												
7	Identify target					▨	▨	▨																										
8	Track Target								▨																									
9	Move Robot								▨	▨																								
10	Robot attack target										▨																							
11	Turn robot											▨	▨																					
12	Interim Report													▨	▨	▨	D																	

13	Position of target																	▨																
14	Robot follows target																		▨															
15	Follow target then joust																		▨	▨														
16	Set up second robot																			▨														
17	Enemies direction																				▨													
18	Enemies distance																					▨	▨											
19	Stop robot																							▨										
20	Adjust direction whilst attacking																							▨	▨	▨								
21	Behaviours																										▨	▨						
22	Behaviours application on Pi																												▨	▨	▨			
23	Final report																											▨	▨	▨	▨	▨	▨	D
24	Create on screen controller																			▨														
25	Move robot with controller																				▨													

Time Plan Revision 2

		University Calendar Weeks																																
#	**Task Name**	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
1	Configure Raspberry Pi			▒																														
2	VNC			▒																														
3	Add Camera to Raspberry Pi			▒																														
4	Install Open CV				▒																													
5	Get Video feed from camera				▒	▒																												
6	Camera testing					▒																												
7	Identify target					▒	▒	▒																										
8	Track Target							■	▒																									
9	Move Robot								▒	▒																								
10	Robot attack target										▒	■																						
11	Turn robot											▒	▒																					
12	Interim Report													▒	▒	▒	D																	
13	Position of target																	▒																
14	Robot follows target																		▒															

15	Follow target then joust																		▨	▨														
16	Set up second robot																			▨														
17	Enemies direction																				▨													
18	Enemies distance																					▨	▨											
19	Stop robot																							▨										
20	Adjust direction whilst attacking																							▨	▨	▨								
21	Behaviours																										▨	▨						
22	Behaviours application on Pi																												▨	▨	▨			
23	Final report																											▨	▨	▨	▨	▨	▨	D
24	Create on screen controller																			▨														
25	Move robot with controller																				▨													

Time Plan Revision 3

		University Calendar Weeks																																
#	**Task Name**	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
1	Configure Raspberry Pi			▨																														
2	VNC			▨																														
3	Add Camera to Raspberry Pi			▨																														
4	Install Open CV				▨																													
5	Get Video feed from camera				▨	▨																												
6	Camera testing					▨																												
7	Identify target					▨	▨	▨																										
8	Track Target							■	▨																									
9	Move Robot								▨	▨																								
10	Robot attack target										▨	■																						
11	Turn robot											▨	▨																					
12	Interim Report													▨	▨	▨	D																	
13	Robot Speed													▨	▨																			
14	Position of target																	▨																
15	Robot follows target																		▨															

16	Follow target then joust																		▨	▨														
17	Set up second robot																			▨														
18	Enemies direction																				▨													
19	Enemies distance																					▨	▨											
20	Stop robot																							▨										
21	Return to starting position																							▨	▨	▨								
22	Behaviours																										▨	▨						
23	Behaviours application on Pi																												▨	▨	▨			
24	Final report																											▨	▨	▨	▨	▨	▨	D
25	Implement Free Mode																			▨														
26	Move robot with controller																				▨													

Time Plan Revision 4

		University Calendar Weeks																																
#	**Task Name**	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
1	Configure Raspberry Pi			▨																														
2	VNC			▨																														
3	Add Camera to Raspberry Pi			▨																														
4	Install Open CV				▨																													
5	Get Video feed from camera				▨	▨																												
6	Camera testing					▨																												
7	Identify target					▨	▨	▨																										
8	Track Target							■	▨																									
9	Move Robot								▨	▨																								
10	Robot attack target										▨	■																						
11	Turn robot											▨	▨																					
12	Interim Report													▨	▨	▨	D																	
13	Robot Speed													▨	▨																			
14	Fix performance issue with																	▨	▨	▨														

	camera and motor thread																	▨	▨	▨														
14	Position of target																	▨																
15	Robot follows target																		▨															
17	Set up second robot																			▨														
18	Enemies direction																				▨													
19	Enemies distance																					▨	▨											
20	Stop robot																							▨										
21	Return to starting position																							▨	▨	▨								
22	Behaviours																										▨	▨						
23	Behaviours application on Pi																												▨	▨	▨			
24	Final report																											▨	▨	▨	▨	▨	▨	D

Time Plan Revision 5

		University Calendar Weeks																																
#	Task Name	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
1	Configure Raspberry Pi			▨																														
2	VNC			▨																														
3	Add Camera to Raspberry Pi			▨																														
4	Install Open CV				▨																													
5	Get Video feed from camera				▨	▨																												
6	Camera testing					▨																												
7	Identify target					▨	▨	▨																										
8	Track Target							■	▨																									
9	Move Robot								▨	▨																								
10	Robot attack target										▨	■																						
11	Turn robot											▨	▨																					
12	Interim Report													▨	▨	▨	D																	
13	Robot Speed													▨	▨																			
14	Fix performance issue with camera																	▨	▨	▨														

	and motor thread																	▒	▒	▒														
17	Set up second robot																■																	
18	Adjust Algorithm																				▒													
20	Stop robot																							▒										
21	Return to starting position																							▒	▒	▒								
22	Behaviours																										▒	▒						
23	Behaviours application on Pi																												▒	▒	▒			
24	Final report																											▒	▒	▒	▒	▒	▒	D

Time Plan Revision 6

		University Calendar Weeks																																
#	Task Name	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
1	Configure Raspberry Pi			▨																														
2	VNC			▨																														
3	Add Camera to Raspberry Pi			▨																														
4	Install Open CV				▨																													
5	Get Video feed from camera				▨	▨																												
6	Camera testing					▨																												
7	Identify target					▨	▨	▨																										
8	Track Target							■	▨																									
9	Move Robot								▨	▨																								
10	Robot attack target										▨	■																						
11	Turn robot											▨	▨																					
12	Interim Report													▨	▨	▨	D																	
13	Robot Speed													▨	▨																			
14	Fix performance issue with																	▨	▨	▨														

	camera and motor thread																▨	▨	▨															
17	Set up second robot															■																		
18	Adjust Algorithm																			▨														
20	Stop robot																						▨											
21	Restart robot patrol loop																						▨	▨	▨									
22	Behaviours																									▨	▨							
24	Final report																										▨	▨	▨	▨	▨	▨	D

Appendix B: Testing

Patrol Tests

Patrol Test	Predicted	Output
Turning using SlowRotate Method	Robot should change direction after 6 seconds	The robot didn’t change direction or around 8 seconds
Change robot survey speed to 1 and motor timing to 0.5	Robot should move fast when surveying	The robot did move very fast however due to calibration of the motors when turning left the robot was slightly slower
Change the robot Slow rotate motor interval to 4	Robot turns at the given speed for five seconds	The robot turned for 5 seconds
Robot Identifies target	The robot should transition to the next state	The robot transitions to the adjust state however there is some lag

Adjust Tests

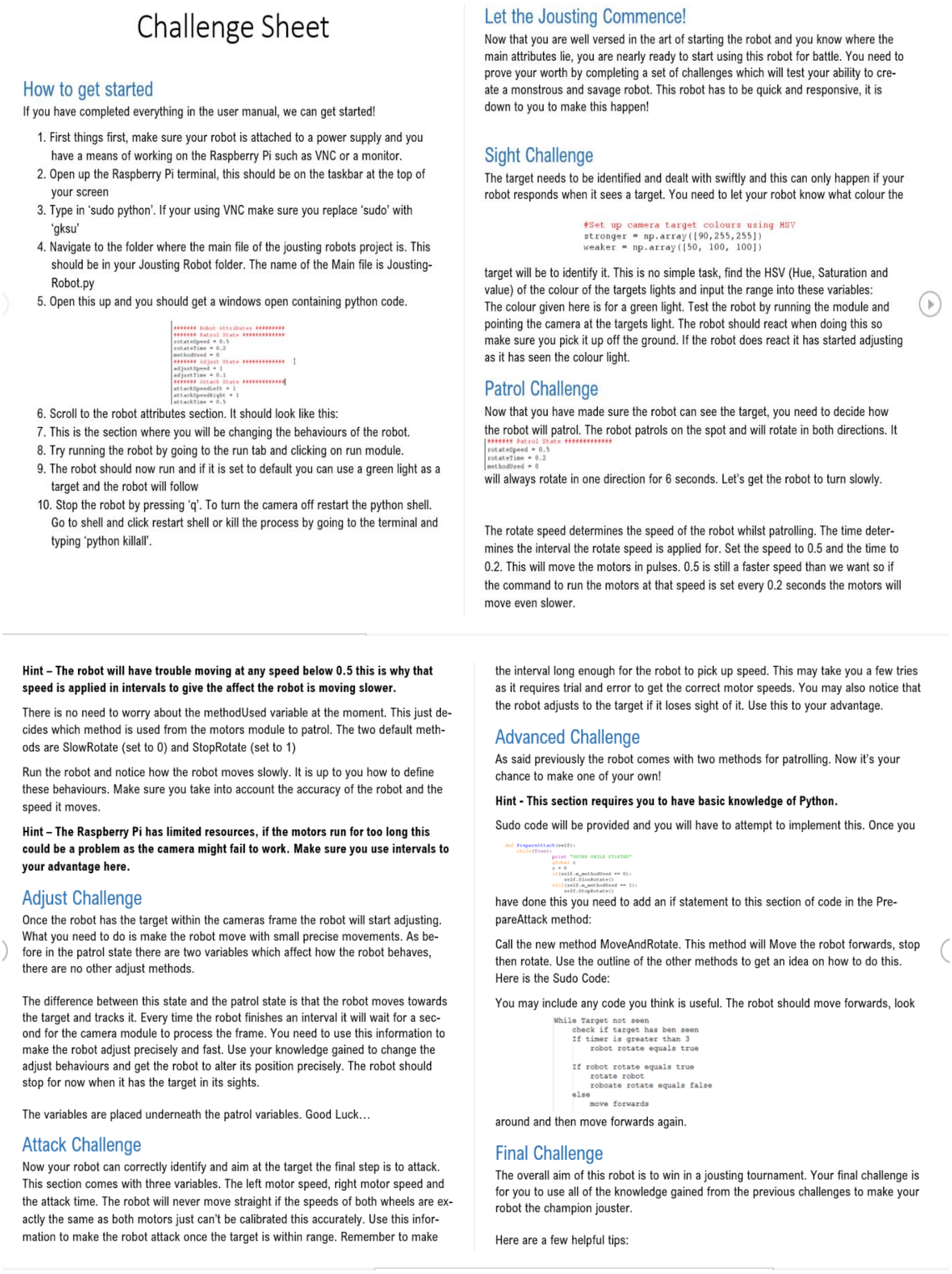
Adjust Test	Predicted	Output
Change robot adjust speed to 1 and motor timing to 0.2	Robot should jump left or right when adjusting	The robot adjusted towards the target by jumping right
Set robot patrol method to 1 (StopRotate method)	Robot should sleep for a second after every time it moves	The robot would move then sleep
Target off screen left	When the target is off screen the robot should adjust left	The robot adjusts left towards the target
Target off screen right	When the target is off screen the robot should adjust right	The robot adjusts right towards the target
Target in range	Robot changes state to attack state	The robot changes to attack state
Wait 10 seconds	Robot should move to patrol state if it hasn’t seen target for 10 seconds	Robot goes back to patrol state

Attack Tests

Attack Test	Predicted	Output
Attack speeds above 0.5 and interval of 1	Robot should move forwards	The robot transitions to the adjust state however there is some lag
Left wheel speed of 0.7 and right wheel speed of 1	Robot should move left as it goes forwards	The robot veers off to the left

Right wheel speed of 0.7 and left wheel speed of 1	Robot should move right as it goes forwards	The robot veers off to the right
Speed of 1 for wheels and interval of 0.1	Robot will move forwards very slowly	Robot moved forwards slightly
Target out of range	Robot should go back to adjust state	Robot transitions to adjust state

Appendix C: User Challenges



Challenge Sheet

How to get started

If you have completed everything in the user manual, we can get started!

1. First things first, make sure your robot is attached to a power supply and you have a means of working on the Raspberry Pi such as VNC or a monitor.
2. Open up the Raspberry Pi terminal, this should be on the taskbar at the top of your screen
3. Type in 'sudo python'. If your using VNC make sure you replace 'sudo' with 'gksu'
4. Navigate to the folder where the main file of the jousting robots project is. This should be in your Jousting Robot folder. The name of the Main file is Jousting-Robot.py
5. Open this up and you should get a windows open containing python code.

```
####### Robot Attributes #########
####### Patrol State #############
rotateSpeed = 0.5
rotateTime = 0.2
methodUsed = 0
####### Adjust State #############
adjustSpeed = 1
adjustTime = 0.1
####### Attack State #############
attackSpeedLeft = 1
attackSpeedRight = 1
attackTime = 0.5
```

6. Scroll to the robot attributes section. It should look like this:
7. This is the section where you will be changing the behaviours of the robot.
8. Try running the robot by going to the run tab and clicking on run module.
9. The robot should now run and if it is set to default you can use a green light as a target and the robot will follow
10. Stop the robot by pressing 'q'. To turn the camera off restart the python shell. Go to shell and click restart shell or kill the process by going to the terminal and typing 'python killall'.

Let the Jousting Commence!

Now that you are well versed in the art of starting the robot and you know where the main attributes lie, you are nearly ready to start using this robot for battle. You need to prove your worth by completing a set of challenges which will test your ability to create a monstrous and savage robot. This robot has to be quick and responsive, it is down to you to make this happen!

Sight Challenge

The target needs to be identified and dealt with swiftly and this can only happen if your robot responds when it sees a target. You need to let your robot know what colour the target will be to identify it. This is no simple task, find the HSV (Hue, Saturation and value) of the colour of the targets lights and input the range into these variables:

```
#Set up camera target colours using HSV
stronger = np.array([90,255,255])
weaker = np.array([50, 100, 100])
```

The colour given here is for a green light. Test the robot by running the module and pointing the camera at the targets light. The robot should react when doing this so make sure you pick it up off the ground. If the robot does react it has started adjusting as it has seen the colour light.

Patrol Challenge

Now that you have made sure the robot can see the target, you need to decide how the robot will patrol. The robot patrols on the spot and will rotate in both directions. It will always rotate in one direction for 6 seconds. Let's get the robot to turn slowly.

```
####### Patrol State #############
rotateSpeed = 0.5
rotateTime = 0.2
methodUsed = 0
```

The rotate speed determines the speed of the robot whilst patrolling. The time determines the interval the rotate speed is applied for. Set the speed to 0.5 and the time to 0.2. This will move the motors in pulses. 0.5 is still a faster speed than we want so if the command to run the motors at that speed is set every 0.2 seconds the motors will move even slower.

Hint – The robot will have trouble moving at any speed below 0.5 this is why that speed is applied in intervals to give the affect the robot is moving slower.

There is no need to worry about the methodUsed variable at the moment. This just decides which method is used from the motors module to patrol. The two default methods are SlowRotate (set to 0) and StopRotate (set to 1)

Run the robot and notice how the robot moves slowly. It is up to you how to define these behaviours. Make sure you take into account the accuracy of the robot and the speed it moves.

Hint – The Raspberry Pi has limited resources, if the motors run for too long this could be a problem as the camera might fail to work. Make sure you use intervals to your advantage here.

Adjust Challenge

Once the robot has the target within the cameras frame the robot will start adjusting. What you need to do is make the robot move with small precise movements. As before in the patrol state there are two variables which affect how the robot behaves, there are no other adjust methods.

The difference between this state and the patrol state is that the robot moves towards the target and tracks it. Every time the robot finishes an interval it will wait for a second for the camera module to process the frame. You need to use this information to make the robot adjust precisely and fast. Use your knowledge gained to change the adjust behaviours and get the robot to alter its position precisely. The robot should stop for now when it has the target in its sights.

The variables are placed underneath the patrol variables. Good Luck…

Attack Challenge

Now your robot can correctly identify and aim at the target the final step is to attack. This section comes with three variables. The left motor speed, right motor speed and the attack time. The robot will never move straight if the speeds of both wheels are exactly the same as both motors just can't be calibrated this accurately. Use this information to make the robot attack once the target is within range. Remember to make the interval long enough for the robot to pick up speed. This may take you a few tries as it requires trial and error to get the correct motor speeds. You may also notice that the robot adjusts to the target if it loses sight of it. Use this to your advantage.

Advanced Challenge

As said previously the robot comes with two methods for patrolling. Now it's your chance to make one of your own!

Hint - This section requires you to have basic knowledge of Python.

Sudo code will be provided and you will have to attempt to implement this. Once you have done this you need to add an if statement to this section of code in the PrepareAttack method:

```
def PrepareAttack(self):
    while(True):
            print "OUTER WHILE STARTED"
            global x
            x = 0
            if(self.m_methodUsed == 0):
                self.SlowRotate()
            elif(self.m_methodUsed == 1):
                self.StopRotate()
```

Call the new method MoveAndRotate. This method will Move the robot forwards, stop then rotate. Use the outline of the other methods to get an idea on how to do this. Here is the Sudo Code:

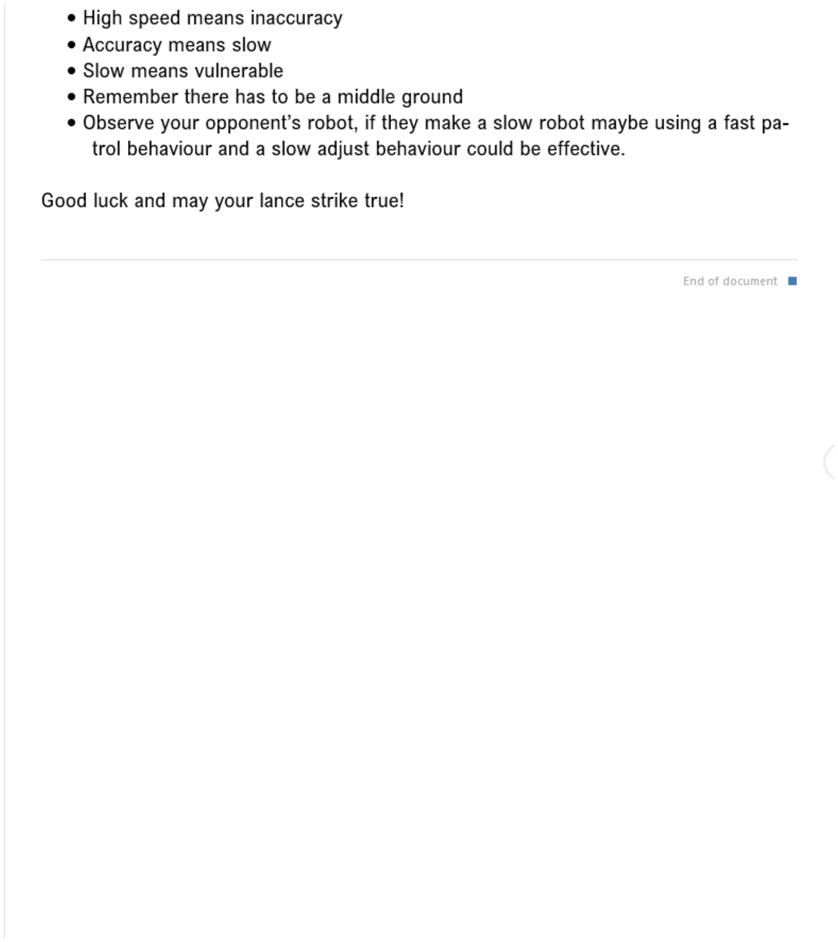
You may include any code you think is useful. The robot should move forwards, look around and then move forwards again.

```
While Target not seen
    check if target has ben seen
    If timer is greater than 3
        robot rotate equals true

    If robot rotate equals true
        rotate robot
        roboate rotate equals false
    else
        move forwards
```

Final Challenge

The overall aim of this robot is to win in a jousting tournament. Your final challenge is for you to use all of the knowledge gained from the previous challenges to make your robot the champion jouster.

Here are a few helpful tips:

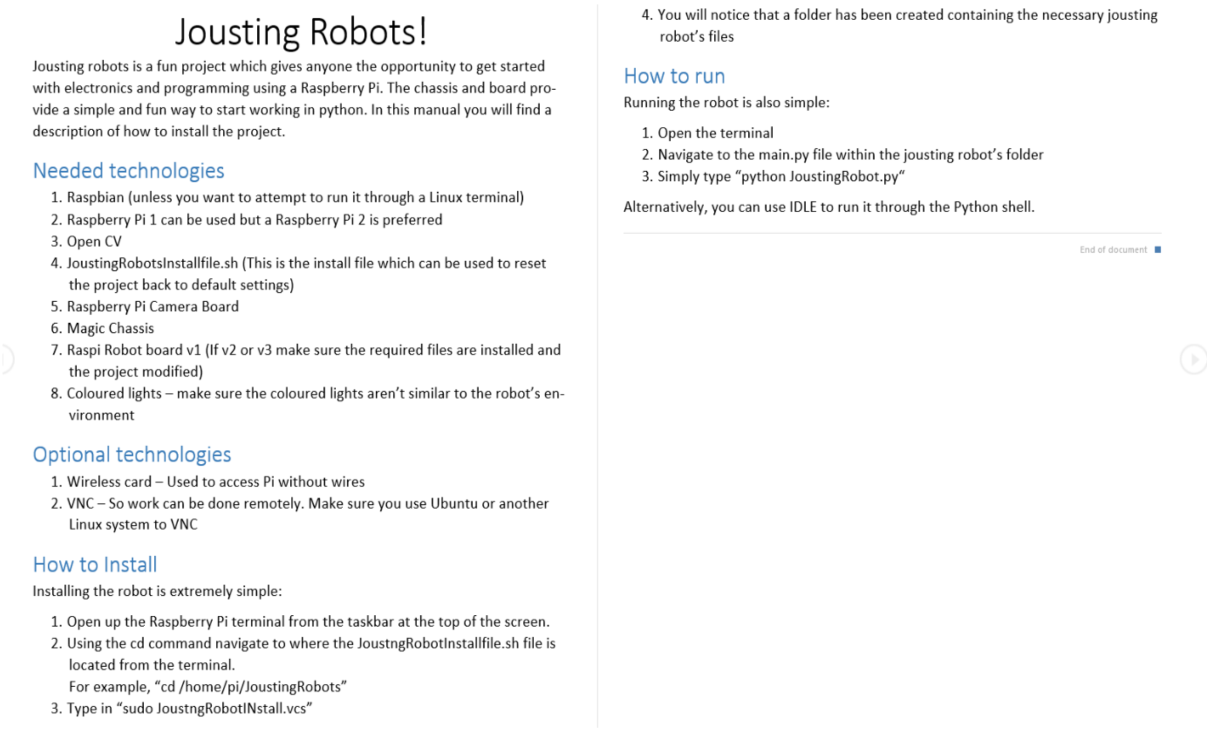


- High speed means inaccuracy
- Accuracy means slow
- Slow means vulnerable
- Remember there has to be a middle ground
- Observe your opponent’s robot, if they make a slow robot maybe using a fast patrol behaviour and a slow adjust behaviour could be effective.

Good luck and may your lance strike true!

End of document ■

Appendix D: User Manual



Jousting Robots!

Jousting robots is a fun project which gives anyone the opportunity to get started with electronics and programming using a Raspberry Pi. The chassis and board provide a simple and fun way to start working in python. In this manual you will find a description of how to install the project.

Needed technologies

1. Raspbian (unless you want to attempt to run it through a Linux terminal)
2. Raspberry Pi 1 can be used but a Raspberry Pi 2 is preferred
3. Open CV
4. JoustingRobotsInstallfile.sh (This is the install file which can be used to reset the project back to default settings)
5. Raspberry Pi Camera Board
6. Magic Chassis
7. Raspi Robot board v1 (If v2 or v3 make sure the required files are installed and the project modified)
8. Coloured lights – make sure the coloured lights aren't similar to the robot's environment

Optional technologies

1. Wireless card – Used to access Pi without wires
2. VNC – So work can be done remotely. Make sure you use Ubuntu or another Linux system to VNC

How to Install

Installing the robot is extremely simple:

1. Open up the Raspberry Pi terminal from the taskbar at the top of the screen.
2. Using the cd command navigate to where the JoustngRobotInstallfile.sh file is located from the terminal.
   For example, "cd /home/pi/JoustingRobots"
3. Type in "sudo JoustngRobotINstall.vcs"
4. You will notice that a folder has been created containing the necessary jousting robot's files

How to run

Running the robot is also simple:

1. Open the terminal
2. Navigate to the main.py file within the jousting robot's folder
3. Simply type "python JoustingRobot.py"

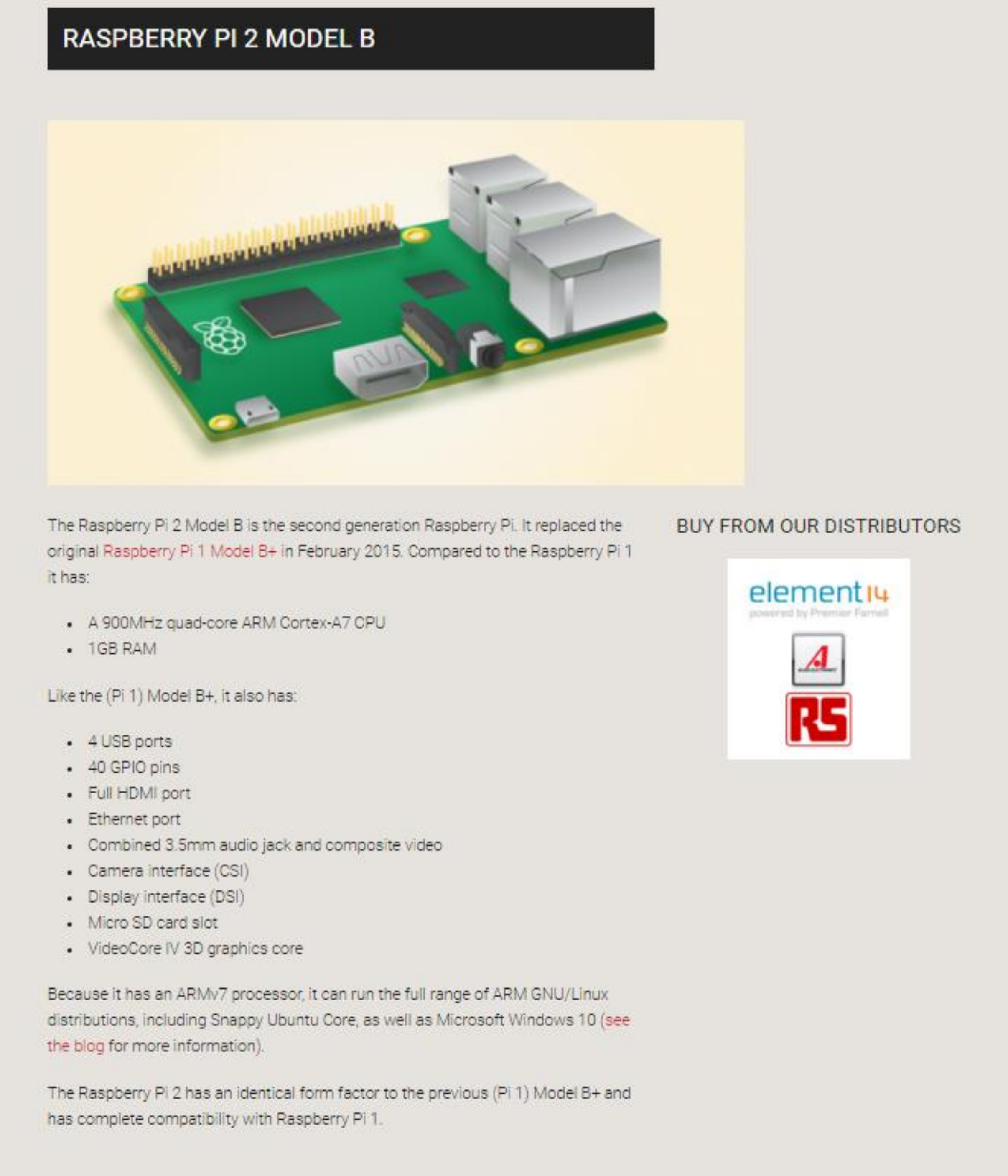
Alternatively, you can use IDLE to run it through the Python shell.

End of document ■

Appendix E: Initial Risk Analysis

Risk	Severity (L/M/H)	Likelihood (L/M/H)	Significance (Sev. x Like.)	How to Avoid	How to Recover
Open CV not working on Raspberry Pi	H	M	HM	Keep Backups	Reinstall the libraries
Motor Control board malfunction	H	L	HM	Make sure the board is connected correctly	Use another control board
Malfunction with the motors	M	L	H	Use controller board connected to GPIO	Reinstall motors or use spare Motors
Camera won’t connect to Raspberry Pi	H	L	H	Make sure camera is installed correctly	Reinstall camera or replace with a spare
Not completing project	M	L	M	Use SCRUM to manage project and give tasks appropriate time to work on	Revaluate backlog

Appendix F: Raspberry Pi 2 Model B Specifications



RASPBERRY PI 2 MODEL B

The Raspberry Pi 2 Model B is the second generation Raspberry Pi. It replaced the original Raspberry Pi 1 Model B+ in February 2015. Compared to the Raspberry Pi 1 it has:

- A 900MHz quad-core ARM Cortex-A7 CPU
- 1GB RAM

Like the (Pi 1) Model B+, it also has:

- 4 USB ports
- 40 GPIO pins
- Full HDMI port
- Ethernet port
- Combined 3.5mm audio jack and composite video
- Camera interface (CSI)
- Display interface (DSI)
- Micro SD card slot
- VideoCore IV 3D graphics core

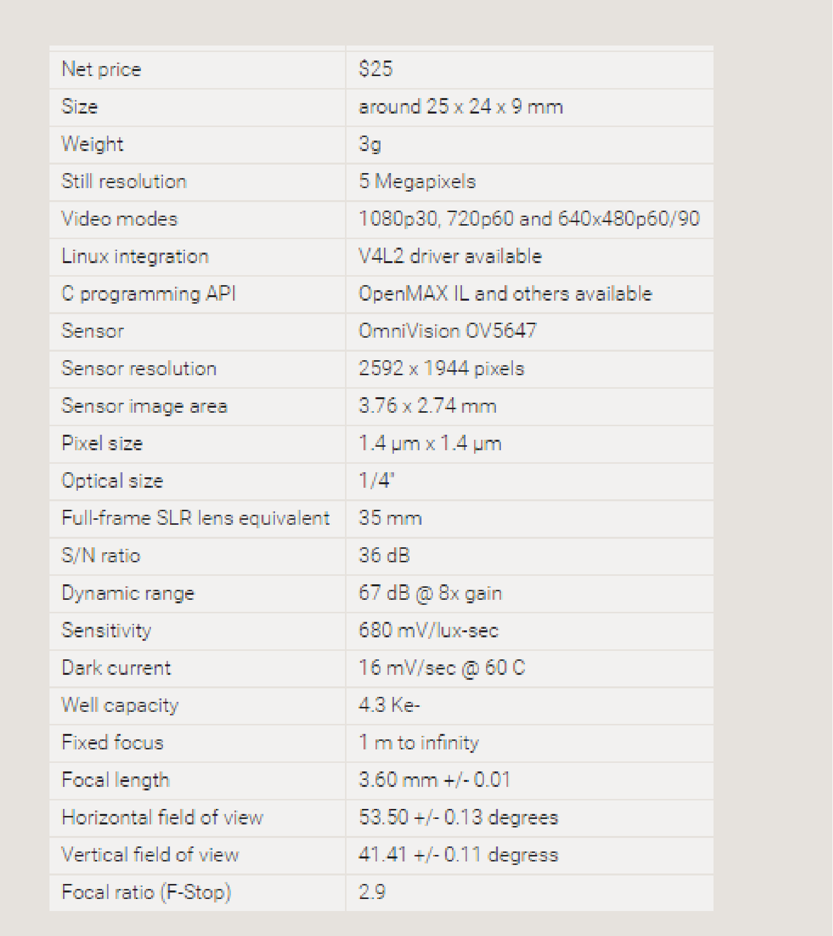
Because it has an ARMv7 processor, it can run the full range of ARM GNU/Linux distributions, including Snappy Ubuntu Core, as well as Microsoft Windows 10 (see the blog for more information).

The Raspberry Pi 2 has an identical form factor to the previous (Pi 1) Model B+ and has complete compatibility with Raspberry Pi 1.

BUY FROM OUR DISTRIBUTORS

Source: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/ (RaspberryPi.org. 2016)

Appendix G: Raspberry Pi Camera Hardware Features



Net price	$25
Size	around 25 x 24 x 9 mm
Weight	3g
Still resolution	5 Megapixels
Video modes	1080p30, 720p60 and 640x480p60/90
Linux integration	V4L2 driver available
C programming API	OpenMAX IL and others available
Sensor	OmniVision OV5647
Sensor resolution	2592 x 1944 pixels
Sensor image area	3.76 x 2.74 mm
Pixel size	1.4 µm x 1.4 µm
Optical size	1/4"
Full-frame SLR lens equivalent	35 mm
S/N ratio	36 dB
Dynamic range	67 dB @ 8x gain
Sensitivity	680 mV/lux-sec
Dark current	16 mV/sec @ 60 C
Well capacity	4.3 Ke-
Fixed focus	1 m to infinity
Focal length	3.60 mm +/- 0.01
Horizontal field of view	53.50 +/- 0.13 degrees
Vertical field of view	41.41 +/- 0.11 degress
Focal ratio (F-Stop)	2.9

Source: https://www.raspberrypi.org/documentation/hardware/camera.md (RaspberryPi.org. 2015)

References

Swapnil Bhartiya. 2013. *Raspberry Pi was created to solve talent crisis at Cambridge: Eben Upton* [Interview]. [ONLINE] Available at: http://www.linuxveda.com/2013/11/07/raspberry-pi-created-solve-talent-*crisis-cambridge-eben-upton-interview-2/. [Accessed 16 December 15].*

Alison Ebbage. 2014. Raspberry Pi education kits: how they can help develop IT skills - E & T Magazine . [ONLINE] Available at: http://eandt.theiet.org/magazine/2014/09/raspberry-pi-teaching-aids.cfm. [Accessed 12 October 15].

Stuart Dredge. 2014. Coding at school: a parent's guide to England's new computing curriculum. [ONLINE] Available at: http://www.theguardian.com/technology/2014/sep/04/coding-school-computing-children-programming. [Accessed 23 December 15].

Dan Shafer. 2002. *Python in the enterprise: Pros and cons*. [ONLINE] Available at:http://www.techrepublic.com/article/python-in-the-enterprise-pros-and-cons/. [Accessed 30 December 15].

Open CV. 2015. *About*. [ONLINE] Available at: http://opencv.org/about.html. [Accessed 12 October 15].

Intorobotics. 2015. *The Raspberry Pi camera guide*. [ONLINE] Available at:http://www.intorobotics.com/raspberry-pi-camera-guide/. [Accessed 05 January 16].

Michael Barr. 2001. *Pulse Width Modulation*. [ONLINE] Available at:http://homepage.cem.itesm.mx/carbajal/Microcontrollers/ASSIGNMENTS/readings/ARTICLES/barr01_pwm.pdf. [Accessed 14 January 16].

TightVNC Software. 2015. *Home*. [ONLINE] Available at: http://www.tightvnc.com/. [Accessed 14 October 15].

Jon Mundy. 2015. Raspberry Pi 2 vs Raspberry Pi: How the DIY computer has been improved . [ONLINE] Available at: http://www.trustedreviews.com/opinions/raspberry-pi-2-vs-raspberry-pi. [Accessed 05 April 16]

Adrian Rosebrock. 2015. *Install OpenCV and Python on your Raspberry Pi 2 and B+.* [ONLINE] Available at: http://www.pyimagesearch.com/2015/02/23/install-opencv-and-python-on-your-raspberry-pi-2-and-b/. [Accessed 13 April 2016].

OpenCV 2.4.12.0 documentation . 2016. Eroding and Dilating. [ONLINE] Available at: http://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html. [Accessed 13 April 2016].

OpenCV . 2015. Contours : Getting Started. [ONLINE] Available at: http://docs.opencv.org/3.1.0/d4/d73/tutorial_py_contours_begin.html#gsc.tab=0. [Accessed 14 April 2016].

Open CV, (2016), Origional. http://docs.opencv.org/2.4/_images/Morphology_1_Tutorial_Theory_Dilation.png. [image]. [Accessed 18 April 2016].

Open CV, (2016), Dilate. *http://docs.opencv.org/2.4/_images/Morphology_1_Tutorial_Theory_Original_Image.png*. [Accessed 18 April 2016]. [image].

Open CV, (2016), Erosion. http://docs.opencv.org/2.4/_images/Morphology_1_Tutorial_Theory_Erosion.png. [image]. [Accessed 18 April 2016].

Simon Monk. 2013. Home. [ONLINE] Available at: https://github.com/simonmonk/raspirobotboard/wiki. [Accessed 18 April 2016].

Monique DeVoe. 2015. Raspberry Pi goes custom for industrial, commercial applications. [ONLINE] Available at: http://embedded-computing.com/26496-raspberry-pi-goes-custom-for-industrial-commercial-applications/. [Accessed 18 April 2016].

eLinux.org. 2016. http://elinux.org/images/c/cf/Raspi-Model-AB-Mono-2-699x1024.png. [Accessed 18 April 2016].

RaspberryPi.org. 2016. RASPBERRY PI 2 MODEL B. [ONLINE] Available at: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/. [Accessed 28 April 2016].

RaspberryPi.org. 2015. CAMERA. [ONLINE] Available at: https://www.raspberrypi.org/documentation/hardware/camera.md. [Accessed 28 April 2016].

Brad Bourque. 2015. Arduino vs. Raspberry Pi: Mortal enemies, or best friends? . [ONLINE] Available at: http://www.digitaltrends.com/computing/arduino-vs-raspberry-pi/. [Accessed 3 May 2016]