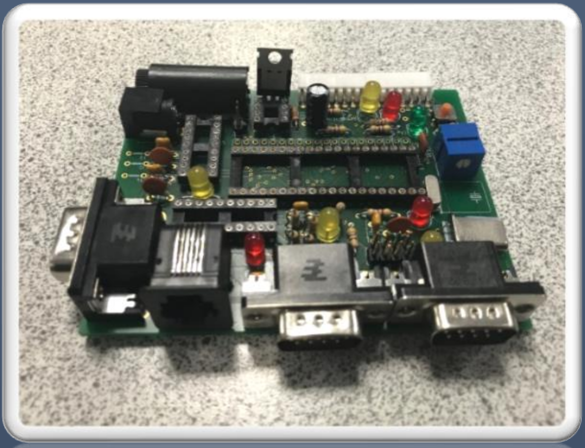
2018



Project report

PCB DESIGN & MANUFACTURE & BITBOT

Semester Two Project

Author: Loti Ibrahimi (W20015453)

Supervisor: Jason Berry

WATERFORD INSTITUTE OF TECHNOLOGY | BSc Internet of Things – Year 2 | 23/05/2018

Table of Content

Chapter One

1.1 Introduction

In this report I will be discussing and doing an overview of how I ended up designing and implementing an Embedded System, also known as an Electronic Baseboard – PCB, along with how it was used (along with others) to create BitBot.

I will explain the steps taken in creating the schematic, which was used using EAGLE, and I will review how the schematic was created, and the necessary parts required in the design of the PCB.

A brief overview of the function of each major block of this project will also be made, explaining what each part of the schematic does.

I will then further show how this schematic was developed into a physical board (the PCB), and how each part was inserted.

Examples of how the PCB's can be incorporated into larger group project will then be given, as Embedded Systems tend to be big element in IoT in this day and age, so it's important to get an insight into it.

The breadboard design was part of our semester one project development, which was a part of a larger project, which was aimed to fit an overall team objective in Semester two, which has now been accomplished.

I will walkthrough in detail, as to how we came about making BitBot in semester two, the experience we had & things we learnt, and finally a conclusion.

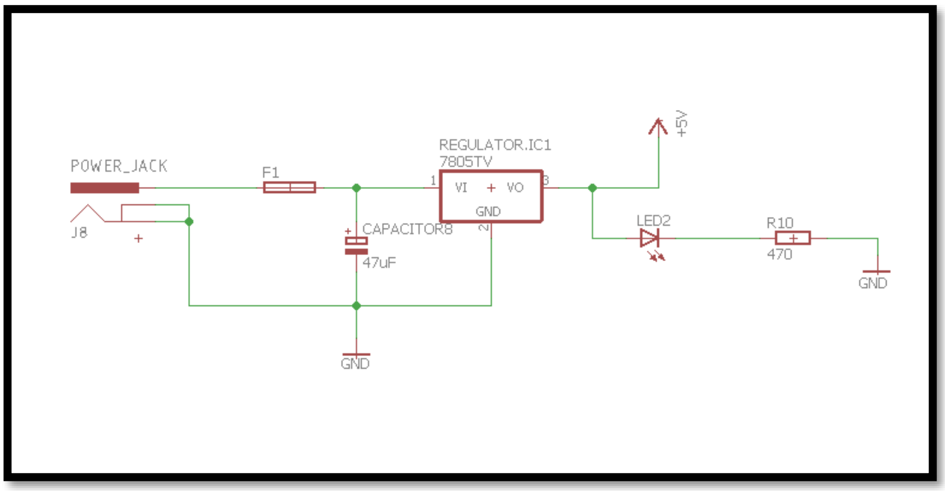
1.2 PCB Design and Manufacture

To design our PCB board, we firstly had to introduce ourselves to a new software which our class supervisor had suggested we'd try this year, EAGLE. The aim this year was that we as student take the initiative and do our own research on how the software worked, rather than go through the usual lecture way of being told how this and that works, which certainly proved to be successful, and we seemed to learn a lot more.

After some time of getting familiar with EAGLE, we began our first ever design of a PCB, exciting stuff! We were given out a schematic sample, with reference to parts & using EAGLE we made attempts to replicate and put together these jigsaw pieces, i.e. the different schematic parts.

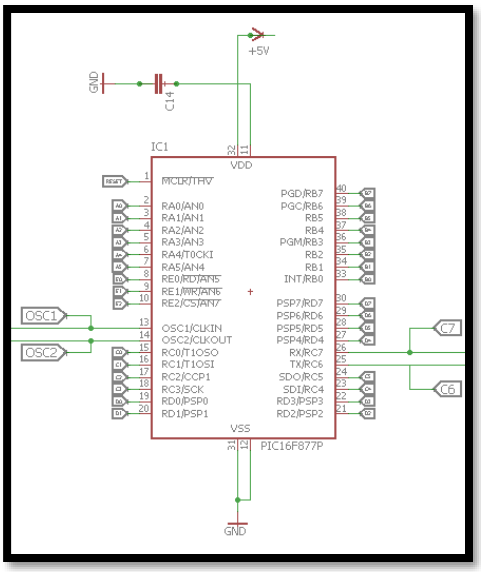
Below are all the different schematic parts of the PCB:

1. **Power**



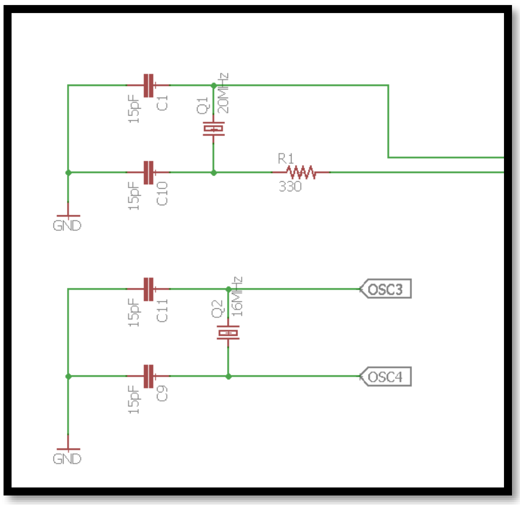
The power supply Schematic for the Board

2. PIC



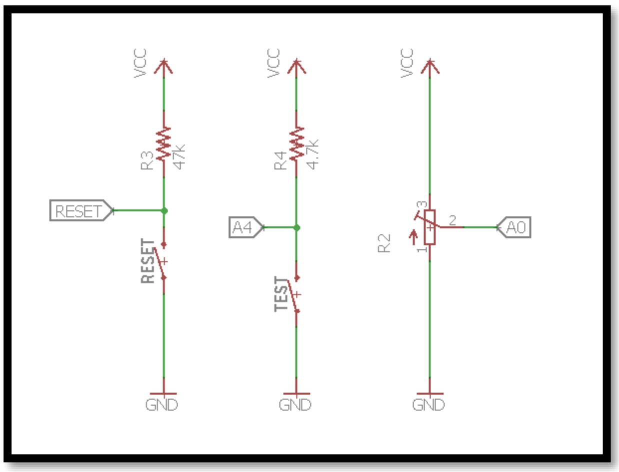
Schematic of a PIC microcontroller (Programmable Interface Controller). This is a electronic circuit that can be programmed to carry out a vast range of tasks.

3. Clock



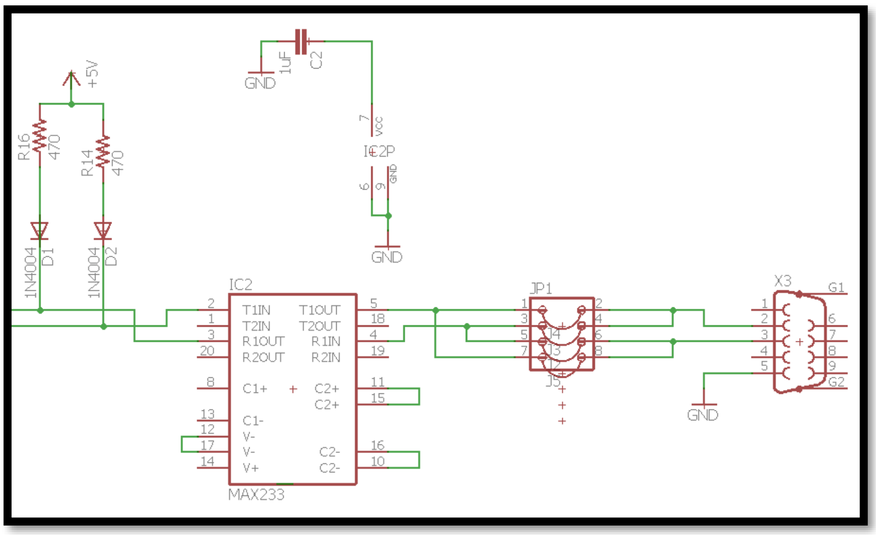
The clock on a PCB

4. Buttons



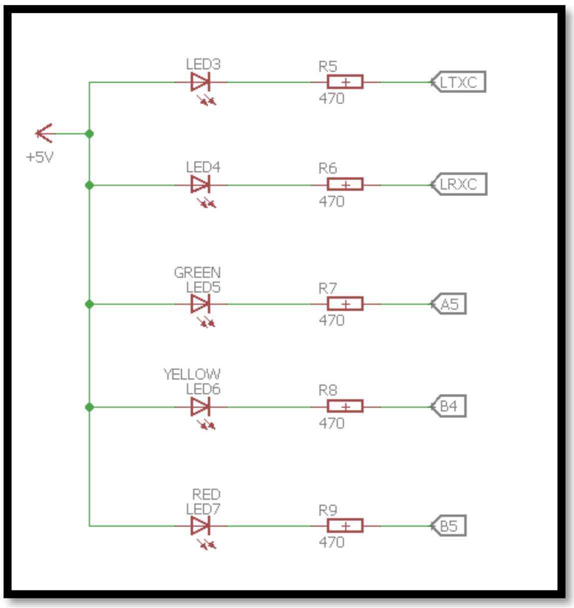
Schematic for buttons on a PCB

5. RS232



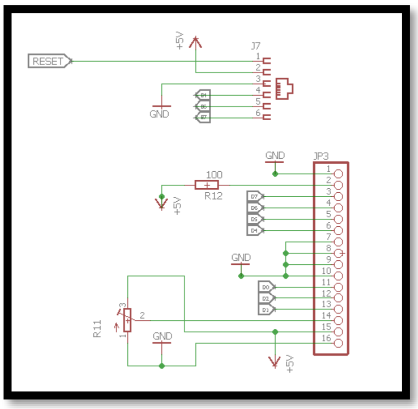
The Schematic for RS232. This is basically the signals connecting between a DTE (data terminal equipment) such as a computer terminal, and a DCE (data communication equipment) such as a modem.

6. Indicator LED



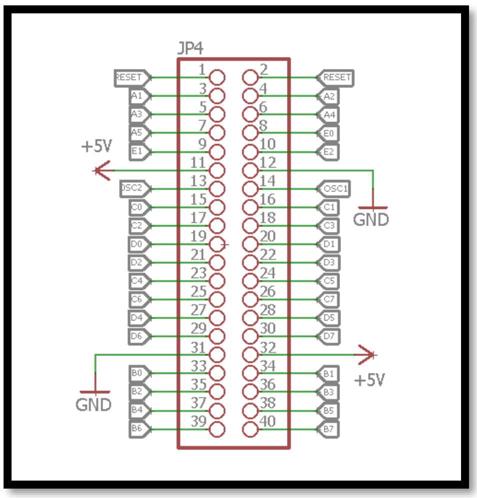
Schematic for PCB LEDs

7. LCD



Schematic for the LCD

8. PIC Header



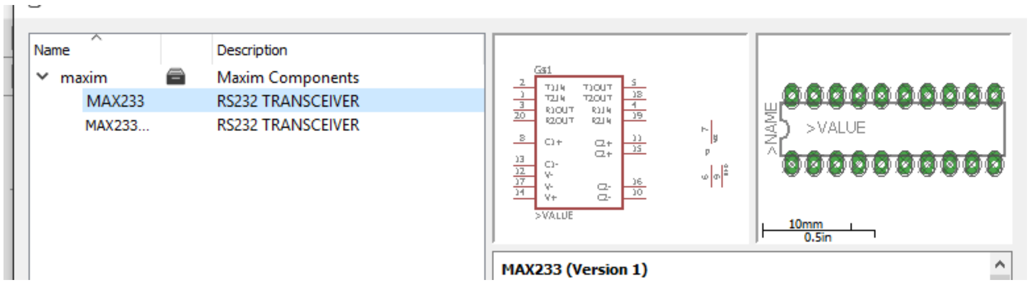
Schematic for PIC Header

Detailed instructions on creating the Schematics are linked below in the Appendix. Both site link & video instructions.

Now with the schematic complete, the next step taken was to design the PCB board layout. This was also done using EAGLE. We were free to design the board as we wished, which meant placing the blueprints of parts in our chosen locations, however, certain parts had to be suitably positioned. These included the fuse and the LED, along with the ethernet ports which had to be situated on the outskirts of the board.

Pieces were situated beside others that had links in certain schematic circuits. For example, the clock parts were located near one another, likewise with the power and so on. This helped reduce the route complexity connection between each piece.

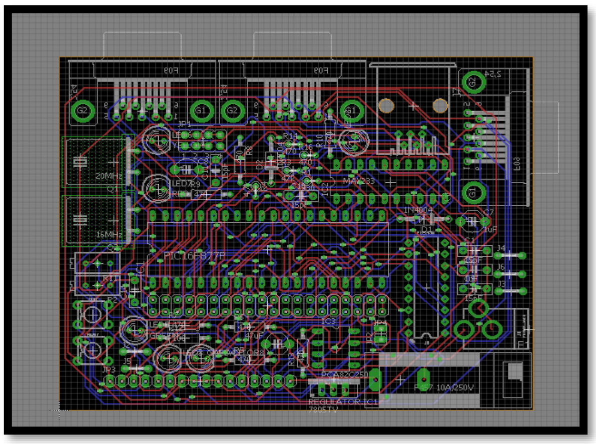
Below is an example of how each schematic piece has a blueprint which is what is set on the board design, in this case the MAX233, from the RS232 schematic:



Name	Description
maxim	Maxim Components
MAX233	RS232 TRANSCEIVER
MAX233...	RS232 TRANSCEIVER

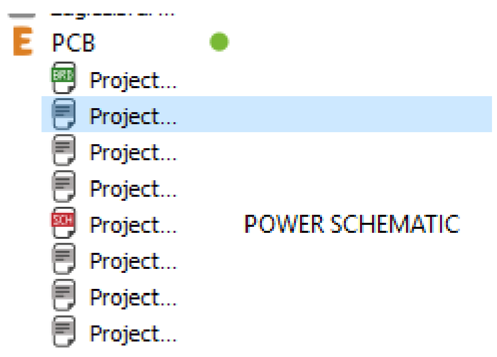
MAX233 (Version 1)

This is the finished board blueprint design:



Once the board design was completed, we had to generate Gerber and Drill files. These are files which you would basically send to a manufacturer, which tell them the necessary details and info to create a physical replica of the design you created. In this case it is the physical board design based on the board blueprint shown above.

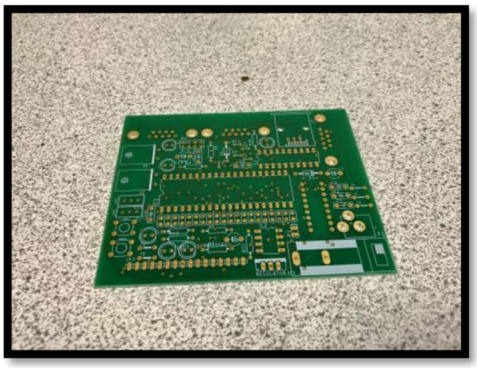
Below are the Project Board file (green), the Schematic file (red), and the 6 Gerber/Drill files (grey):


PCB ●
Project...
Project...
Project...
Project...
Project... POWER SCHEMATIC
Project...
Project...
Project...

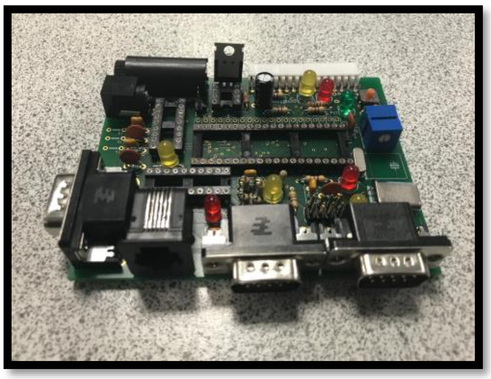
Gerber & Drill files:

- Top Copper (.cmp)
- Bottom Copper (.sol)
- Top Soldermask (.stc)
- Bottom Soldermask (.stc)
- Top Silkscreen (.plc)
- Drill file (.drd)

All of the 8 files were sent to our supervisor, who then had them sent off to be manufactured. The outcome was our very own, first ever built PCB board:



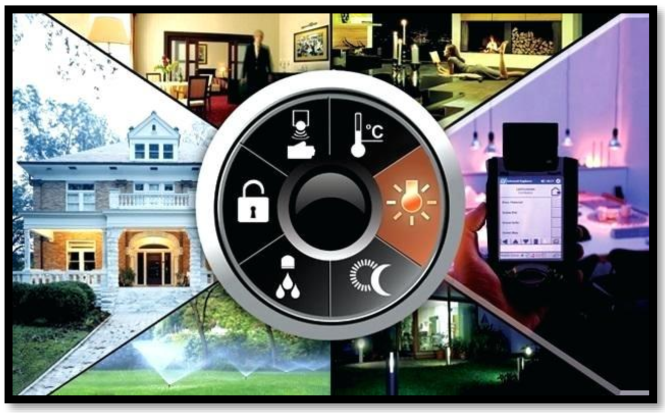
The next objective was a new learning curve for most of us, especially for me, soldering. I now had to solder the parts onto the corresponding blueprints. After a slow and steady start, with small mistakes here and there and tedious unsoldering, the speed eventually picked up and I had gotten a hang of it. Below is the final product:



Detailed instructions on creating the board design are linked below in the Appendix. Both site link & video instructions.

1.3 Overview of Baseboard Operation

Embedded Electronic baseboards have many possible operations and are usually part of a larger device which they perform specific tasks of that device.



For example, in the home, embedded systems are used as networked thermostats in Heating, Ventilation and Air Conditioning (HVAC) systems at home and are used as wired or wireless networking to automate and control lights, security, audio/visual systems, sense climate change, monitoring, etc.

Chapter Two

1.4 Brief Intro:

As highlighted towards the end of Chapter 1, many embedded systems are found in a lot of bigger devices, and they basically behave like the brain of that device - if you want to put it that way.

For our semester two project we had to implement the PCB's into a groupwork project concept idea. As a group we settled on the idea of Robo Wars! The idea sounded quite exciting and it was certainly a new challenge for us lot.

We were grouped up into 3 groups by our supervisor, in which each group was made up of at least 4 people. Each team were given a base body of a bot, of which each member was tasked in working on a certain element of the bot development in the project.

Our group bot was called 'BitBot' – and it was made up of Daniel Collins, Darren Browne, Seamus Reamonn, and myself.

Individual Tasks:

Daniel Collins / Darren Browne: Inputs/Outputs, motors, overall build, sensors.

Seamus Reamonn: Sound System to comms.

Loti Ibrahimi: Vision System/Manual Controls to comms.

We worked inside the project class aswell as in our own times to make progress on this project, continuously working together and helping one another in this development.

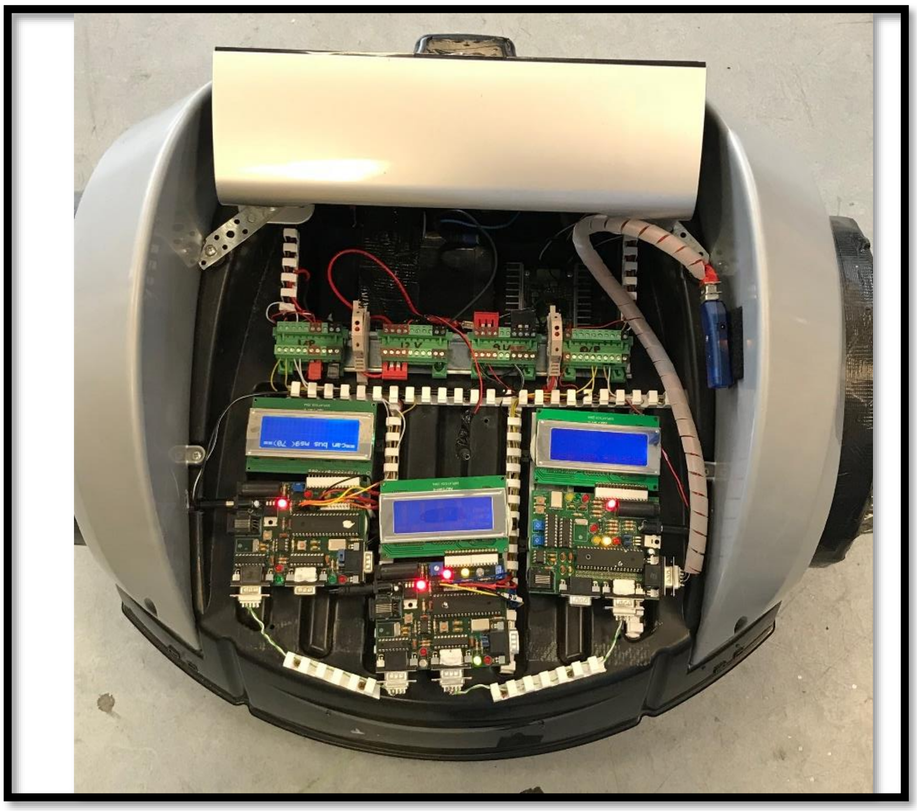
As a class we worked in sprints, a total of around 5. The first few were just getting comfortable with comms, RoboRealm, hyper-terminal, the Can Bus etc. Important things to understand before getting started on the main project which was primarily in the last 2-3 sprints.

I will now discuss over the next few pages in detail, my task in this project aswell as a brief overview of what the other guys worked on.

1.5 Application of PCB

As highlighted towards the end of chapter one, many embedded systems are found in a lot of bigger devices, and they behave like the brain of that device - if you want to put it that way.

In this project we incorporated our PCB's from semester one in our Bot, three in total.



<u>Baseboard 1 (outputs):</u>

Motors, actuators.

<u>Baseboard 2 (inputs):</u>

Object sensor

<u>Baseboard 3:</u>

Bluetooth Comms.

Darren and Daniel worked with the build and mounting up the boards/settings on the bot. Sensors were placed around the bot whereby values were read to indicate if something was near to any of the specific sensors. Values would rise from 0 – 255, sensor @ 0 meaning nothing in sensors line.

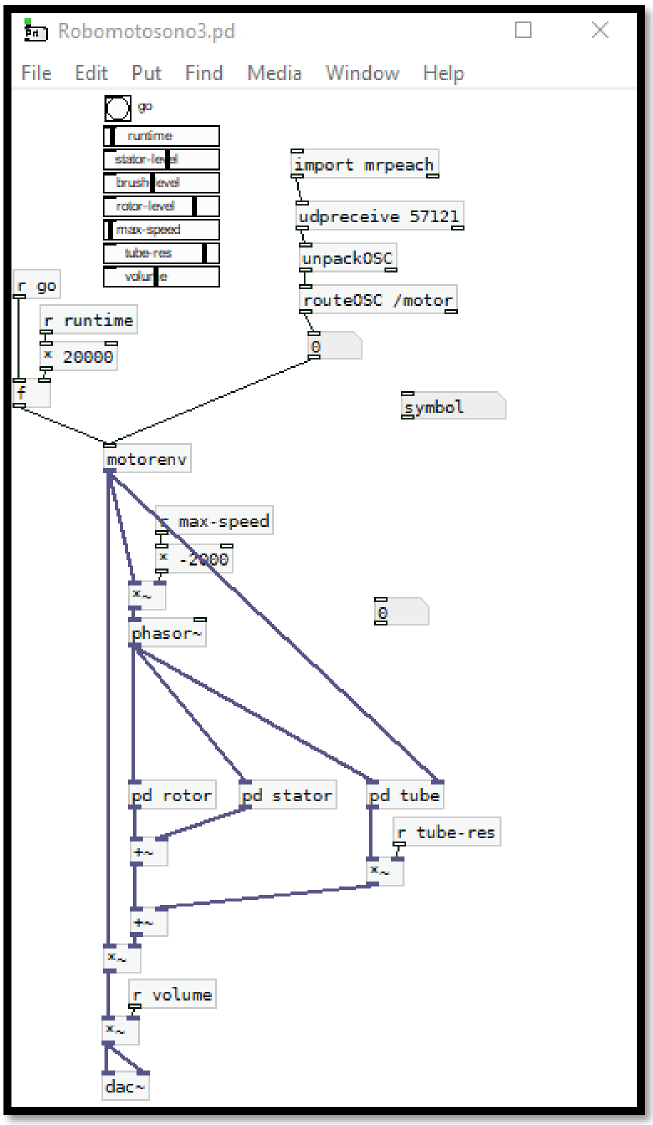
The lads also tweaked the can bus code from a previous project to suit our requirements.

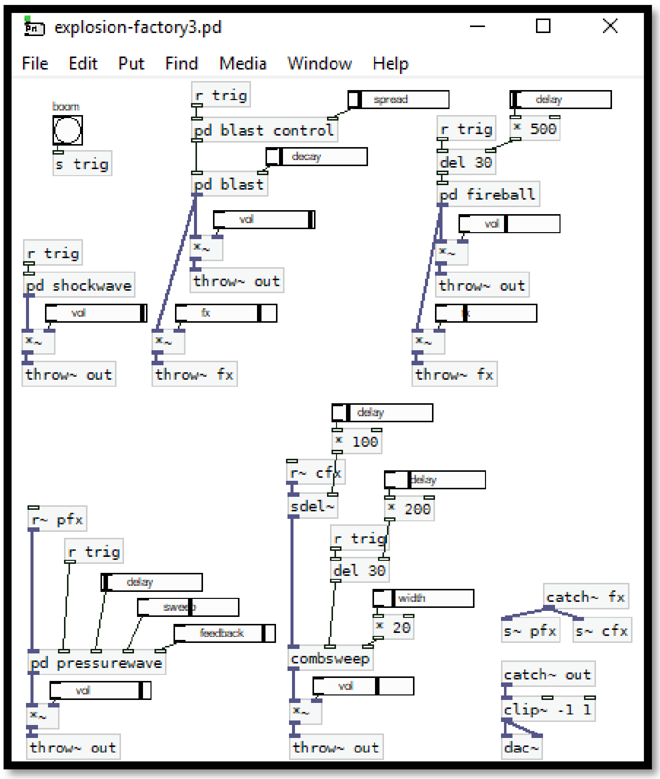
These were incorporated into the vision system display which will be shown below.

Sound played a big part of this project, as it enhanced the experience and environment of the Bot. Sound effects incorporated into the motors and other features is what made the Bot more than just a robot. It added a life-like effect to it.

Seamus worked on the Sound System using a software called Pure Data, which is basically a visual programming language for manipulating and creating sound.

Below are some of the stuff he worked on, and it shows how much work and effort is put into just making a simple motor – speeding up/slowing down, for example.





The outcome sound we used for our Bot was somewhat like a raspy motor, like that of a chainsaw. It was definitely a different and unique sound for a motor, but it enhanced the visual/sound experience of the Bot in action greatly.

Each team had their own unique and interesting sounds, which made everything that bit more exciting and also humorous, especially with random noises being played in class during testing phases.

I personally decided to work on the Vision System, as I thought it was quite interesting and I wanted to see where it leaded/what I could learn from it.

A lot of the Vision System work circled around a software application called RoboRealm, which I may have previously mentioned. This is where all the magic, but also frustrations happened.

RoboRealm consists of a main panel where different 'module' commands/instructions are mixed and matched to create somewhat of a command in which the programme will read and execute.

Different module elements ranged from, 'If statements', 'RGB filters', 'set variables', and so forth. Similar in a structure sense to that of say programming languages in general.

Two of the main aspects which I worked/developed progression in were autonomous controls/manual controls, which were both done through RoboRealm.

Manual Controls:

Manual controls are the first thing that I began working on. It would later prove to be useful in understanding exactly how some features of autonomous work, such as motors etc.

I used a joystick for our Bot manual controls, which I implemented in RoboRealm using one of the modules called 'VBScript':

As you can see, the Joystick had multiple buttons and controls, meaning a lot of possibilities with assigning different actions to different keys.

Due to some sensitivity with manual controls, I implemented 'brake' features on the Bot, assigned as follows:

Button #3: Locks the left wheel.

Button #4: Locks the right wheel.

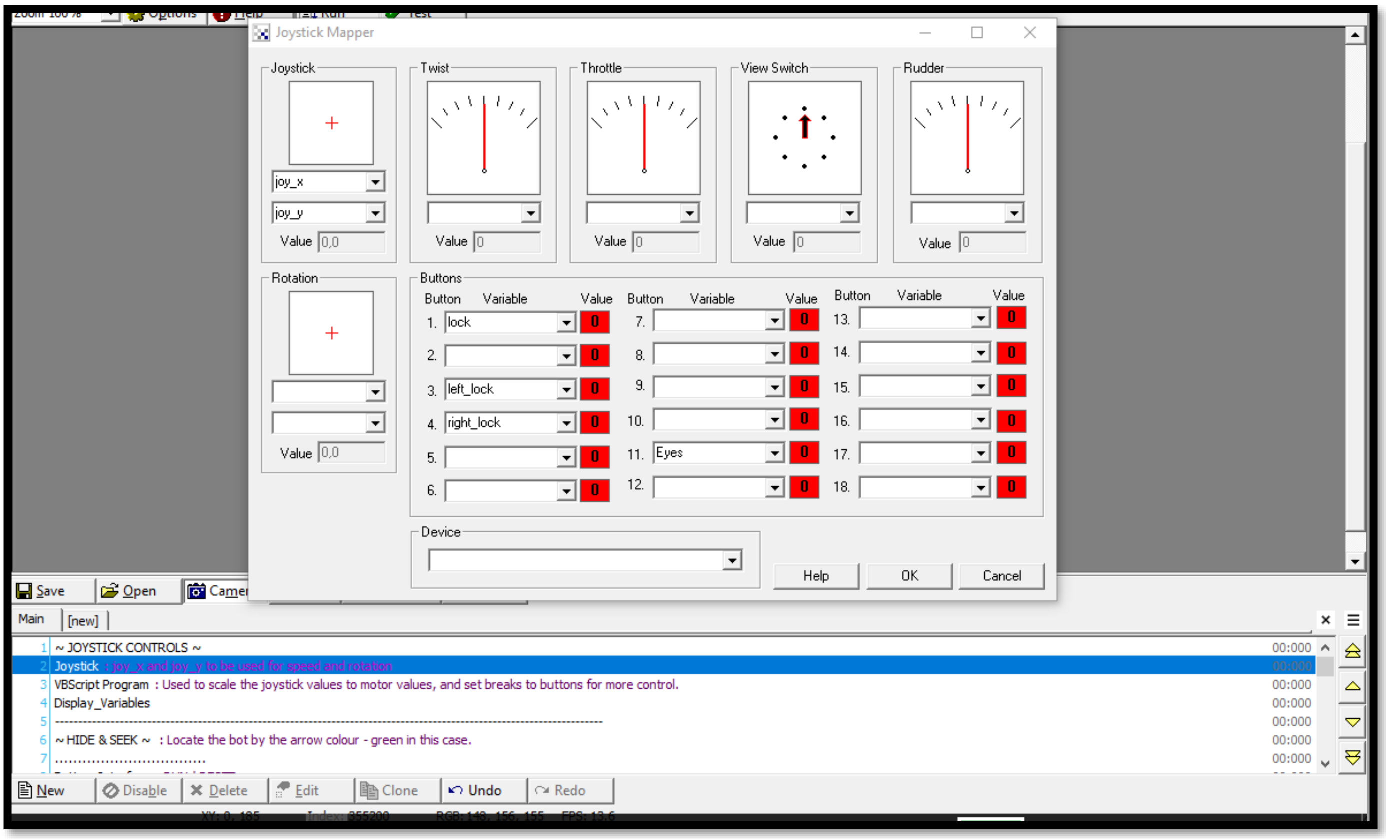
Trigger (on the other side): Locks both wheels.

This added more control in manoeuvrability.

Joystick module for Manual Controls:

As mentioned previously the VBScript module in RoboRealm was used in implementing the Joystick controls, but also a Joystick module, which allowed for the setting of certain variables to different keys etc.

The Joystick module was inserted into the command panel and the Joystick was plugged into the PC, which automatically would sync up the available Joystick controls/buttons to the keys/elements in the Joystick module, as shown below:



Joystick: joy_x, joy_y, Value 0,0
Twist: Value 0
Throttle: Value 0
View Switch: Value 0
Rudder: Value 0
Rotation: Value 0,0

Button	Variable	Value	Button	Variable	Value	Button	Variable	Value
1.	lock	0	7.		0	13.		0
2.		0	8.		0	14.		0
3.	left_lock	0	9.		0	15.		0
4.	right_lock	0	10.		0	16.		0
5.		0	11.	Eyes	0	17.		0
6.		0	12.		0	18.		0

Main [new]

```
~ JOYSTICK CONTROLS ~
Joystick : joy_x and joy_y to be used for speed and rotation
VBScript Program : Used to scale the joystick values to motor values, and set breaks to buttons for more control.
Display_Variables
----------------------------------------------------------------------------------------------
~ HIDE & SEEK ~ : Locate the bot by the arrow colour - green in this case.
.................................
```

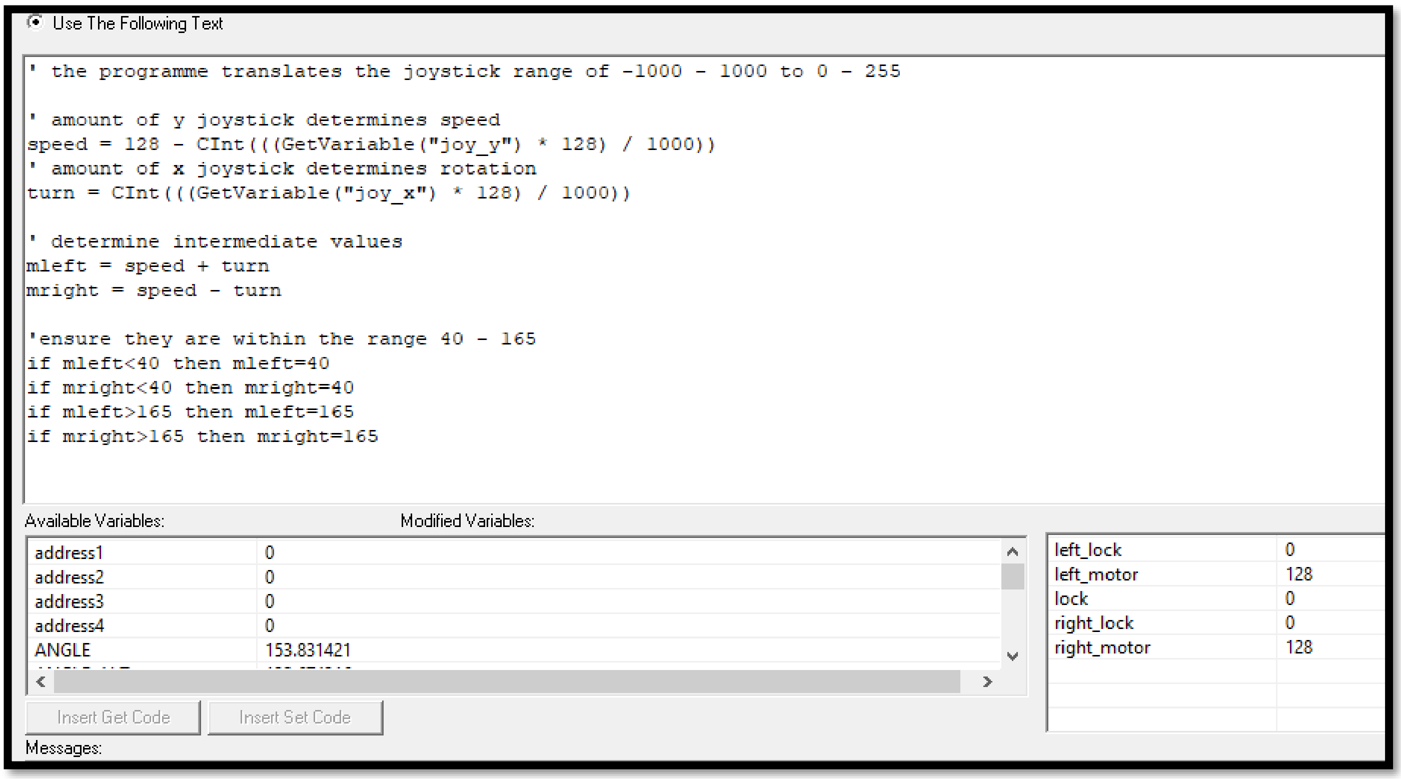
XY: 0, 185   Index: 355200   RGB: 148, 156, 155   FPS: 13.6

Variables created in this module can be called in VBScript enabling you to assign actions to those selected elements.

In the next page I will explain how the manual controls were coded using VBScript module, which you can see in the panel above.

VBScript:

First-Half



```
' the programme translates the joystick range of -1000 - 1000 to 0 - 255

' amount of y joystick determines speed
speed = 128 - CInt(((GetVariable("joy_y") * 128) / 1000))
' amount of x joystick determines rotation
turn = CInt(((GetVariable("joy_x") * 128) / 1000))

' determine intermediate values
mleft = speed + turn
mright = speed - turn

'ensure they are within the range 40 - 165
if mleft<40 then mleft=40
if mright<40 then mright=40
if mleft>165 then mleft=165
if mright>165 then mright=165
```

address1	0
address2	0
address3	0
address4	0
ANGLE	153.831421

left_lock	0
left_motor	128
lock	0
right_lock	0
right_motor	128

The first step was to convert the range of the Joystick modules values for the Joystick from their default 0-1000 to suitable 0-255, which is what values the motors in our Bot function by.

Setting the Joystick Y value as variable ‘speed’ which will be:

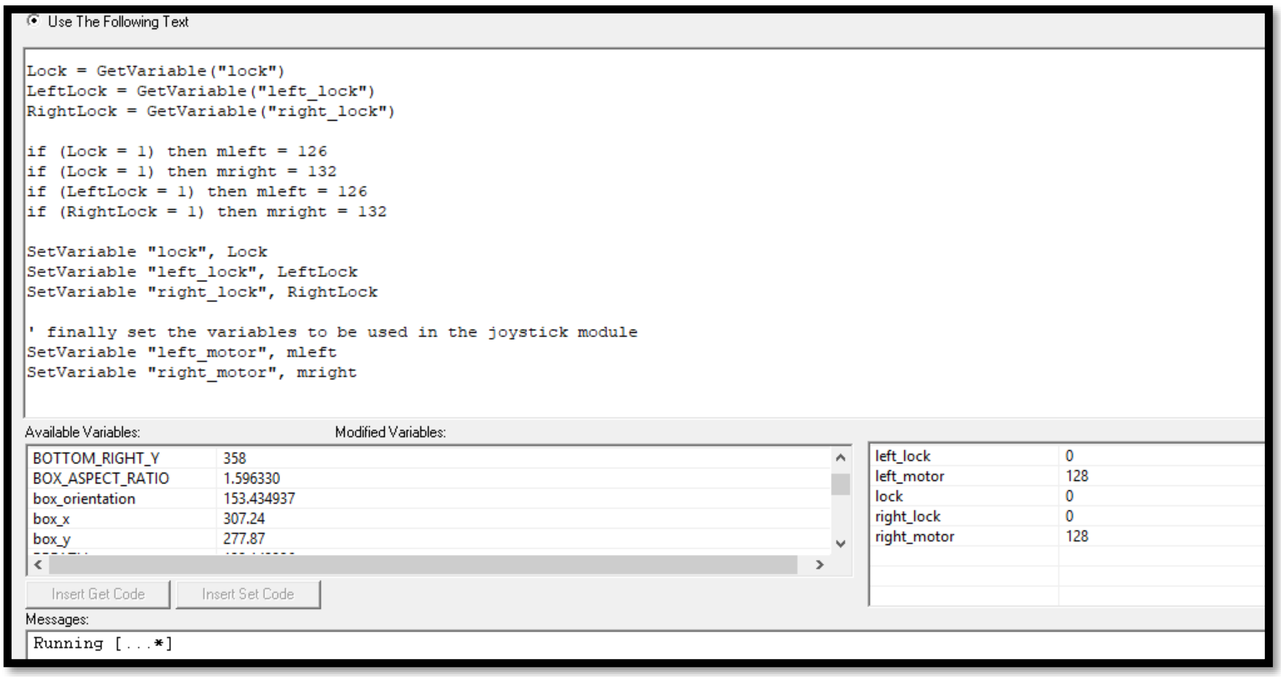
128 (the centre of 0-255) minus the variable that’s set in the Joystick module called ‘joy_y’ (which takes a value from 0-1000 depending on joystick rotation up/down), multiplied by 128 and divided by 1000. This would give a corresponding Y value for the y value of Joystick in ‘0-255’ range.

Likewise, it is done for Joystick X value, which is set as variable ‘turn’.

The next two lines simply determine the intermediate values meaning, values for each motor if the Joystick is pushed at an angle, and not just simple up, down, left, right. Left motor set to variable ‘mleft’, and right motor set to variable ‘mright’.

Motor values are then set to certain limits in order to prevent excessive power being requested from the motors, which would ultimately damage the fuse in the Bot. Rather than have all values for ‘mleft’ and ‘mright’ between 0-255, min-max, I set them to a new min-max, 40-165.

Second-Half



Use The Following Text

```
Lock = GetVariable("lock")
LeftLock = GetVariable("left_lock")
RightLock = GetVariable("right_lock")

if (Lock = 1) then mleft = 126
if (Lock = 1) then mright = 132
if (LeftLock = 1) then mleft = 126
if (RightLock = 1) then mright = 132

SetVariable "lock", Lock
SetVariable "left_lock", LeftLock
SetVariable "right_lock", RightLock

' finally set the variables to be used in the joystick module
SetVariable "left_motor", mleft
SetVariable "right_motor", mright
```

Available Variables:

BOTTOM_RIGHT_Y	358
BOX_ASPECT_RATIO	1.596330
box_orientation	153.434937
box_x	307.24
box_y	277.87

Modified Variables:

left_lock	0
left_motor	128
lock	0
right_lock	0
right_motor	128

Messages:

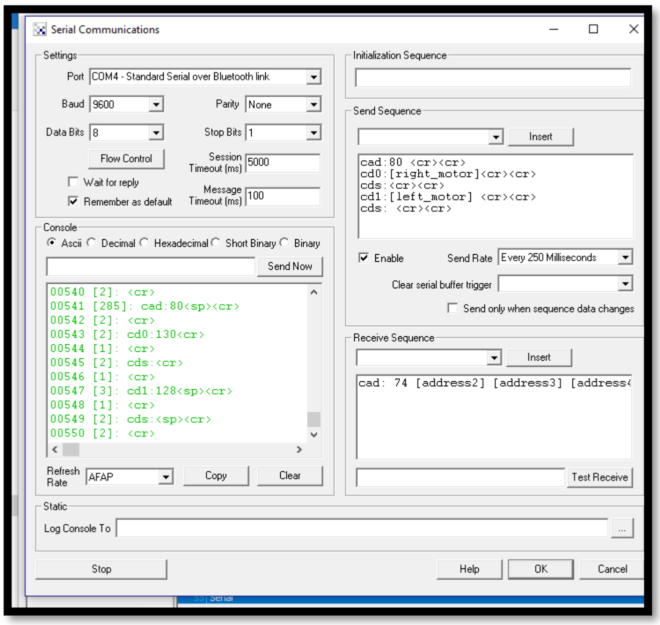
```
Running [...*]
```

The second half of the VBScript module simply sets motor values to the variables 'lock', 'left_lock', ' right_lock' by firstly calling them in using 'getters', then using an 'if statement' basically saying if for example, the button 'left_lock' is pressed (each button is valued 0, if not pressed) then lock the left motor, i.e. 'mleft'.

In this case the stationary values for the motor lie in between 0-255, which is 128, meaning by setting 'mleft' = 128 you are basically stopping it.

At the end of the VBScript is sets the 'left_motor' variable to the value under the variable 'mleft', and likewise for 'right_motor' to value 'mright'.

These are the variables set to the motors on the PCB, and are then called in the 'Serial' module, which acts like the link between Bot & RoboRealm, as follows:



Serial Communications

Settings
Port: COM4 - Standard Serial over Bluetooth link
Baud: 9600
Parity: None
Data Bits: 8
Stop Bits: 1
Session Timeout (ms): 5000
Message Timeout (ms): 100
☐ Wait for reply
☑ Remember as default

Console: [✓ Ascii] Decimal Hexadecimal Short Binary Binary

```
00540 [2]: <cr>
00541 [285]: cad:80<sp><cr>
00542 [2]: <cr>
00543 [2]: cd0:130<cr>
00544 [1]: <cr>
00545 [2]: cds:<cr>
00546 [1]: <cr>
00547 [3]: cd1:128<sp><cr>
00548 [1]: <cr>
00549 [2]: cds:<sp><cr>
00550 [2]: <cr>
```

Refresh Rate: AFAP

Initialization Sequence

Send Sequence

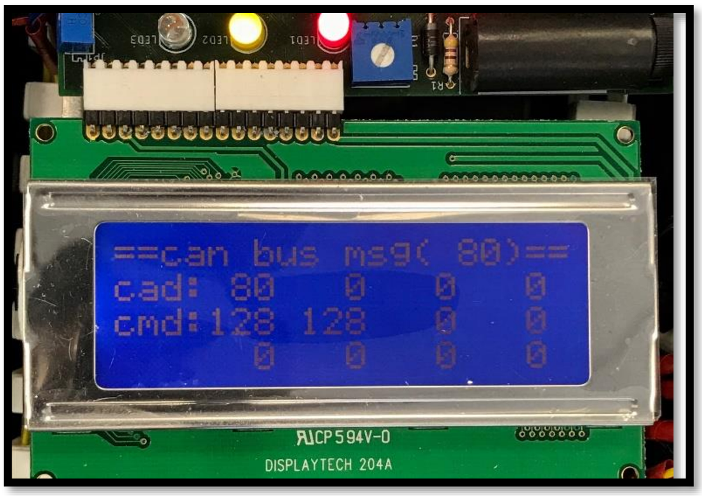
```
cad:80 <cr><cr>
cd0:[right_motor]<cr><cr>
cds:<cr><cr>
cd1:[left_motor] <cr><cr>
cds: <cr><cr>
```

☑ Enable
Send Rate: Every 250 Milliseconds
Clear serial buffer trigger
☐ Send only when sequence data changes

Receive Sequence

```
cad: 74 [address2] [address3] [address
```

Static
Log Console To



Commands are sent to the comms on the Bot using Bluetooth.

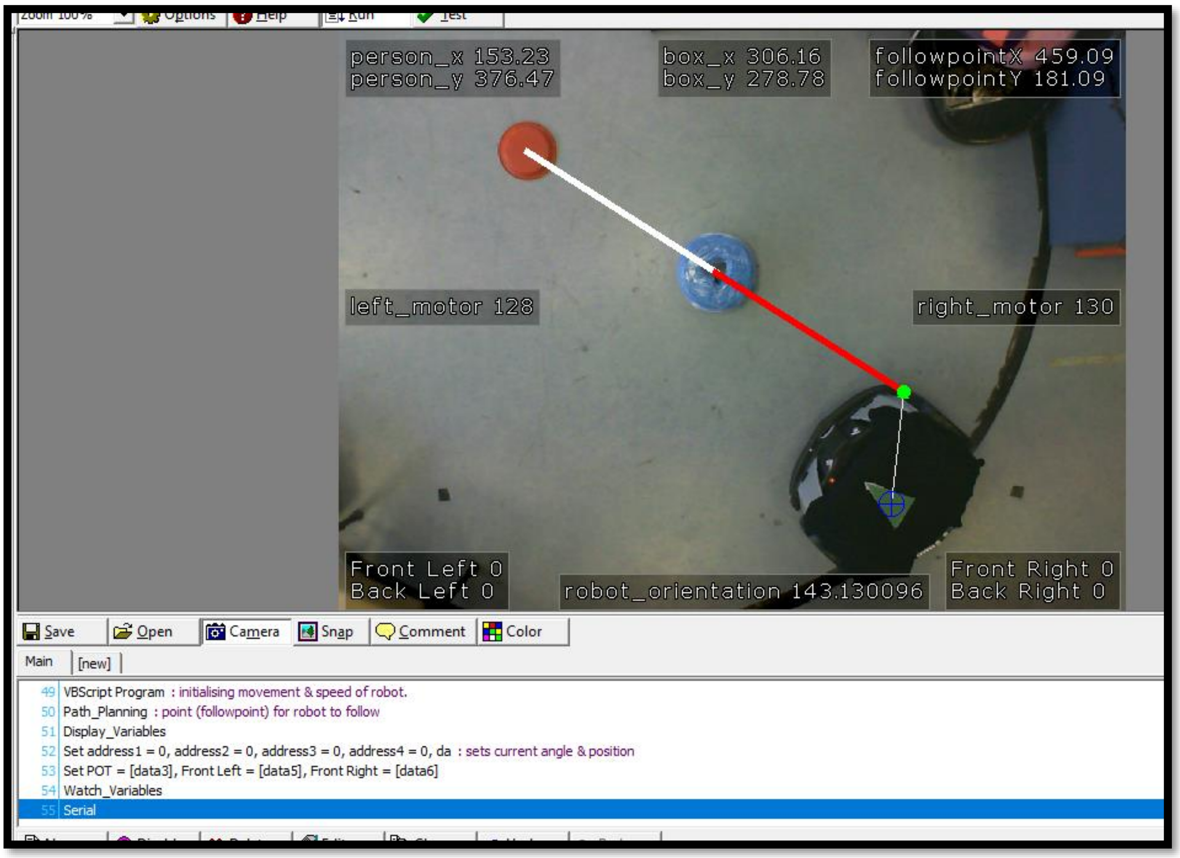
Vision System:

The Vision System is then next and final part of my role which I had to work on. Starting of it certainly without a doubt proved to be challenging, slowly but surely, I got there in the end through the use of online sources for help, teammates, and also brainstorming ideas with our supervisor.

I was tasked to create a 'Hide & Seek' concept whereby the Bot would for example go around a centre obstacle or object, depending on where the person stands. Very much like a cat and mouse scenario but with the idea of going around an object in circles.

All this was once again done through RoboRealm, but also with the use of an overhead camera in this case.

The idea was certainly there but, how to go about it was the true challenge. Through multiple brainstorms and sketches, plans, discussions, I thought of the idea as follows:



```
person_x 153.23
person_y 376.47
box_x 306.16
box_y 278.78
followpointX 459.09
followpointY 181.09
left_motor 128
right_motor 130
Front Left 0
Back Left 0
robot_orientation 143.130096
Front Right 0
Back Right 0
```

```
VBScript Program : initialising movement & speed of robot.
Path_Planning : point (followpoint) for robot to follow
Display_Variables
Set address1 = 0, address2 = 0, address3 = 0, address4 = 0, da : sets current angle & position
Set POT = [data3], Front Left = [data5], Front Right = [data6]
Watch_Variables
Serial
```

I will explain the concept briefly overall.

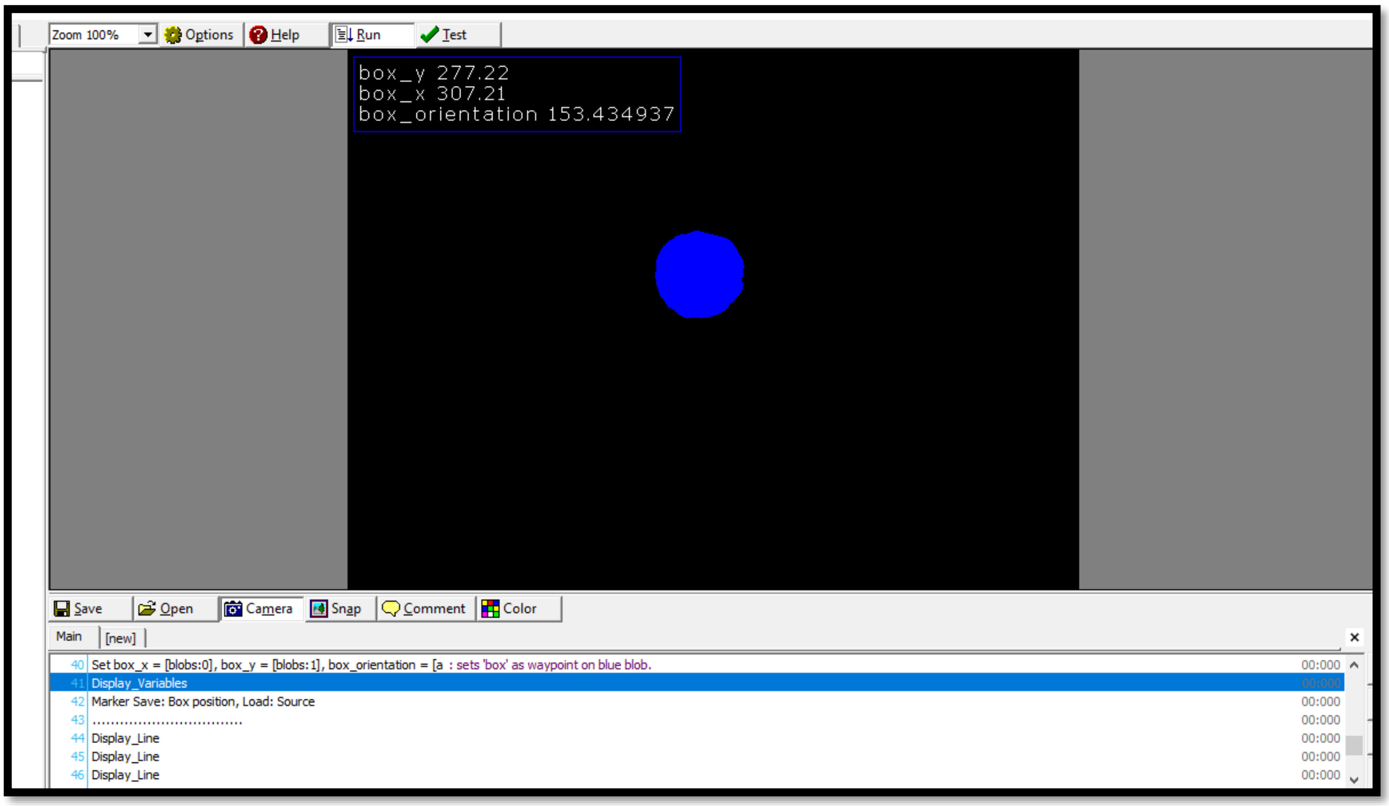
Having three points, a person (red cone), obstacle (blue object), Bot (green arrow).

Using the modules on VBScript I was able to identify the objects through the camera based on their colours, using the RGB filter module.

Using the 'display_line' module' and VBScript I was able to pinpoint the different objects, based on their colours, and identify their position (x,y coordinates), and set a variable to them.

These variables were then called in the VBScripts and implemented to bring everything together.

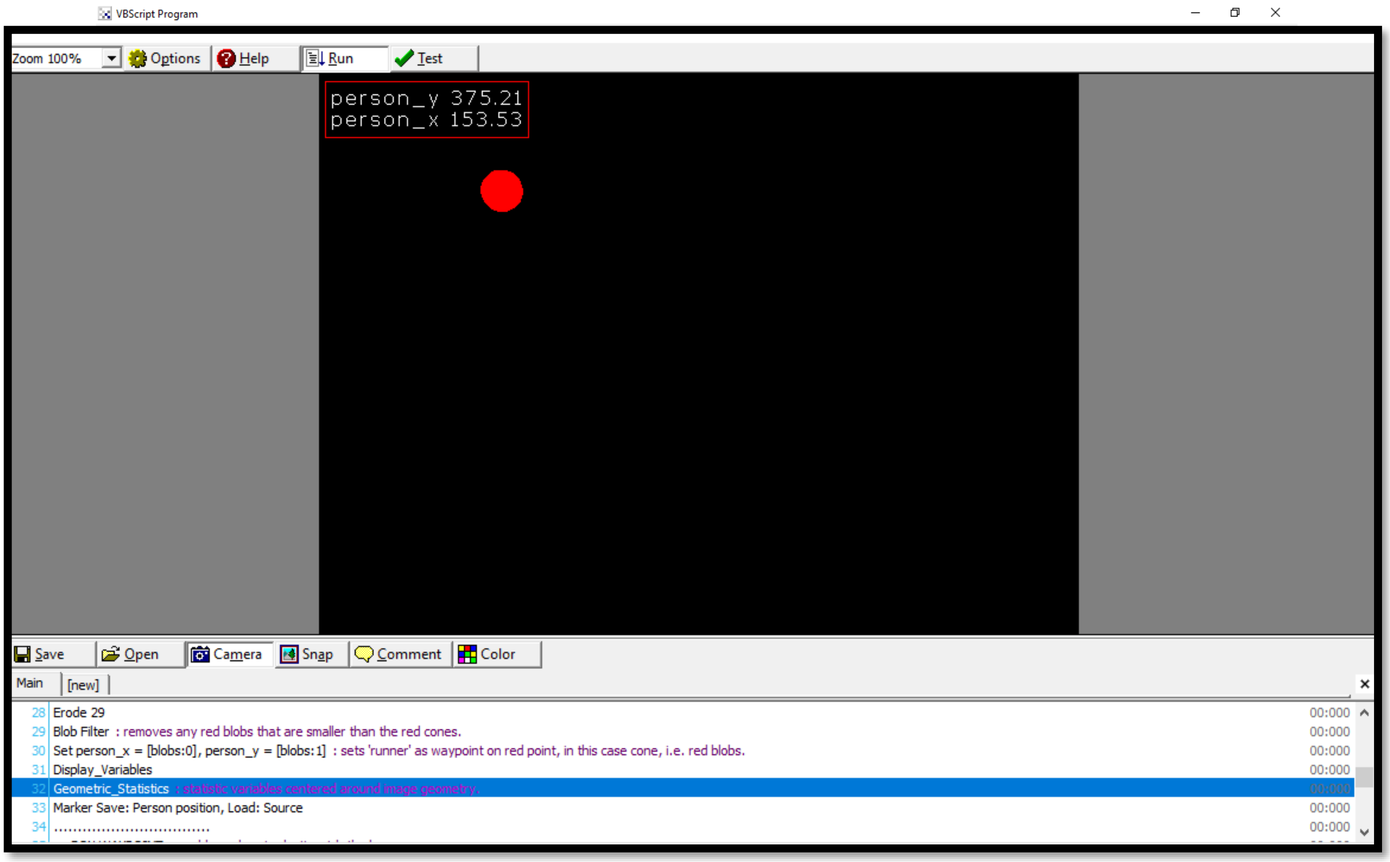
Identifying the Box/Obstacle (RGB filter blue):



```
box_y 277.22
box_x 307.21
box_orientation 153.434937
```

```
Set box_x = [blobs:0], box_y = [blobs:1], box_orientation = [a  : sets 'box' as waypoint on blue blob.   00:000
Display_Variables                                                                                         00:000
Marker Save: Box position, Load: Source                                                                   00:000
..................................                                                                        00:000
Display_Line                                                                                              00:000
Display_Line                                                                                              00:000
Display_Line                                                                                              00:000
```

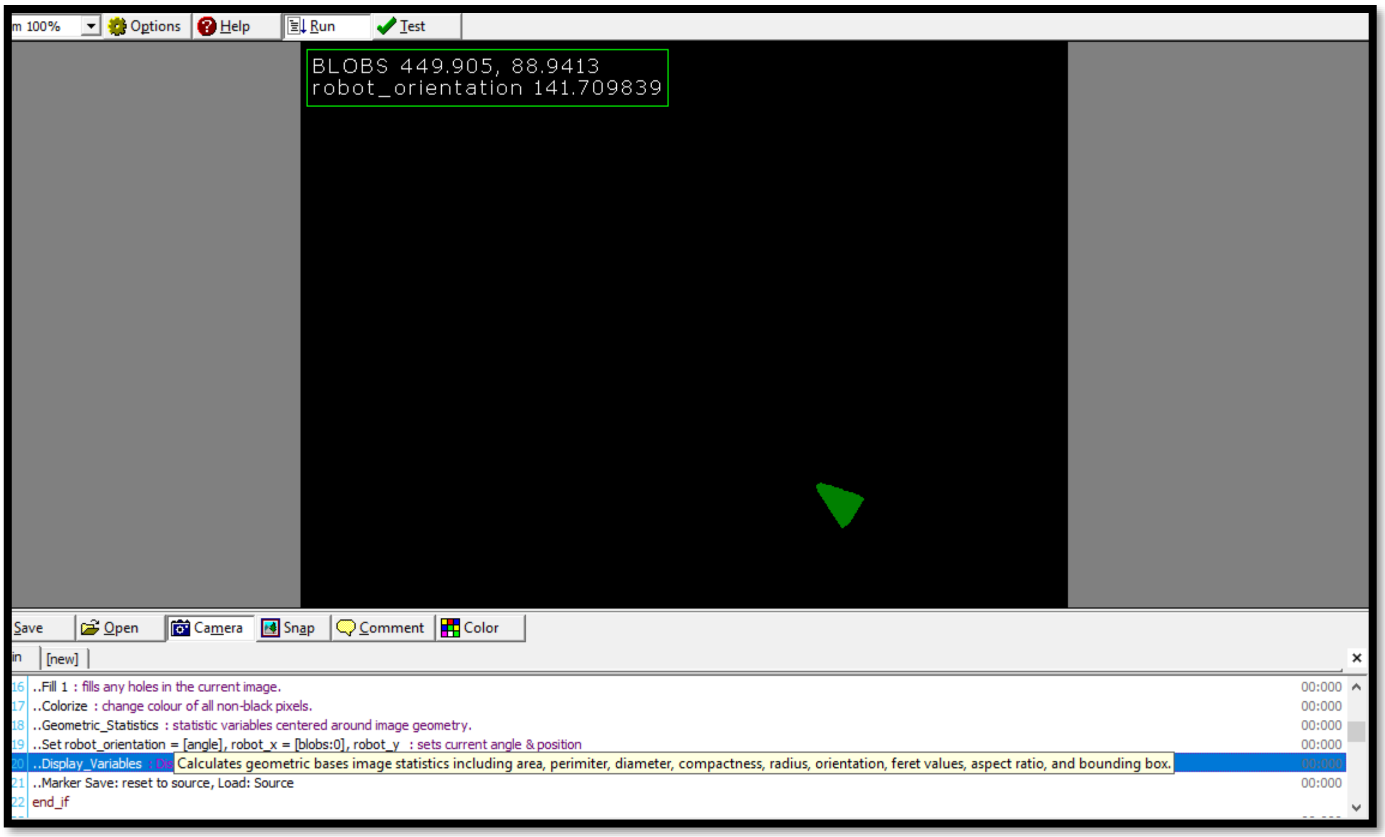
Identifying the person (RGB filter red):



```
person_y 375.21
person_x 153.53
```

```
Erode 29                                                                                                        00:000
Blob Filter  : removes any red blobs that are smaller than the red cones.                                       00:000
Set person_x = [blobs:0], person_y = [blobs:1]  : sets 'runner' as waypoint on red point, in this case cone, i.e. red blobs.   00:000
Display_Variables                                                                                               00:000
Geometric_Statistics  : statistic variables centered around image geometry.                                     00:000
Marker Save: Person position, Load: Source                                                                      00:000
..................................                                                                              00:000
```

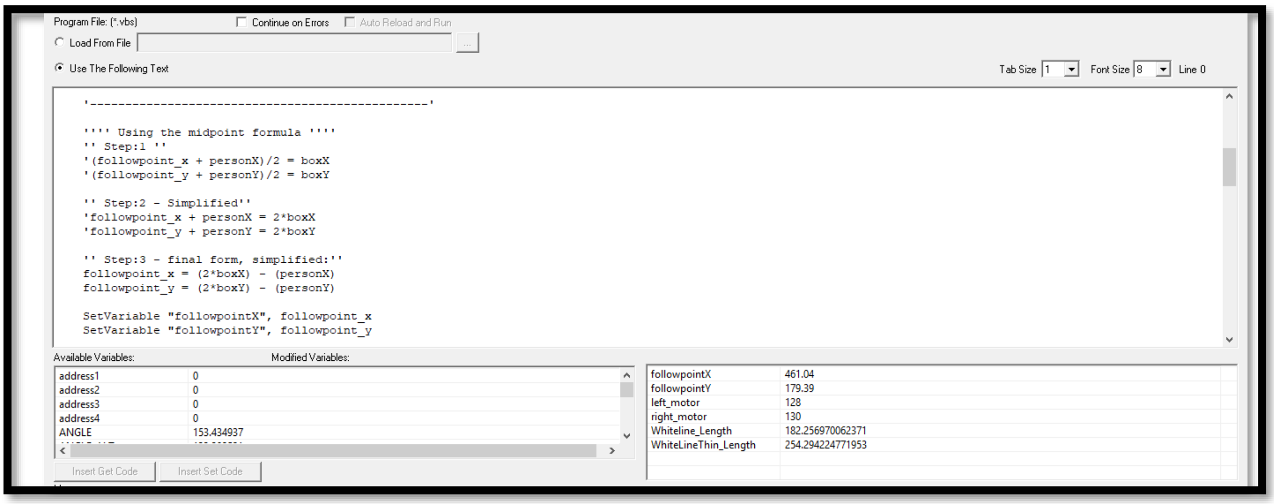
Identifying Bot (RGB filter green):



```
BLOBS 449.905, 88.9413
robot_orientation 141.709839
```

```
..Fill 1 : fills any holes in the current image.
..Colorize : change colour of all non-black pixels.
..Geometric_Statistics : statistic variables centered around image geometry.
..Set robot_orientation = [angle], robot_x = [blobs:0], robot_y : sets current angle & position
..Display_Variables   Calculates geometric bases image statistics including area, perimiter, diameter, compactness, radius, orientation, feret values, aspect ratio, and bounding box.
..Marker Save: reset to source, Load: Source
end_if
```

VBScripts (finding the followpoint, i.e. the green dot on red line)



```
'-----------------------------------------------'

'''' Using the midpoint formula ''''
'' Step:1 ''
'(followpoint_x + personX)/2 = boxX
'(followpoint_y + personY)/2 = boxY

'' Step:2 - Simplified''
'followpoint_x + personX = 2*boxX
'followpoint_y + personY = 2*boxY

'' Step:3 - final form, simplified:''
followpoint_x = (2*boxX) - (personX)
followpoint_y = (2*boxY) - (personY)

SetVariable "followpointX", followpoint_x
SetVariable "followpointY", followpoint_y
```

Available Variables:

address1	0
address2	0
address3	0
address4	0
ANGLE	153.434937

Modified Variables:

followpointX	461.04
followpointY	179.39
left_motor	128
right_motor	130
Whiteline_Length	182.256970062371
WhiteLineThin_Length	254.294224771953

After getting all required details from the RGB identified objects, and setting the corresponding variables, i.e. ‘boxX’, ’boxY’ for the blue object, etc. I used the display line module, to create lines between the ‘personX/Y’ & ‘boxX/Y’ (white line). To get the ‘followpointX/Y’ values I used the midpoint formula. If you look at the camera display above, is basically a large circle, with the box(blue) being the centre, the person(red) being on the circumference, and followpoint being on the circumference but opposite person(red). The midpoint formula states that to find the Centre(x,y) of the centre of a circle given two opposite points on the circumference P1(x1, y1) & P2(x2,y2), its simply Centre(x) = (P1(x1)+P2(x2))/2 & Centre(y) = (P1(y1)+P2(y2))/2.

To get followpoint(x, y), given centre/box(x,y) and person(x,y) you do what is shown in step 3.

When the person(red) moves around the box, the waypoint also moves, always remaining directly opposite the person at all times.

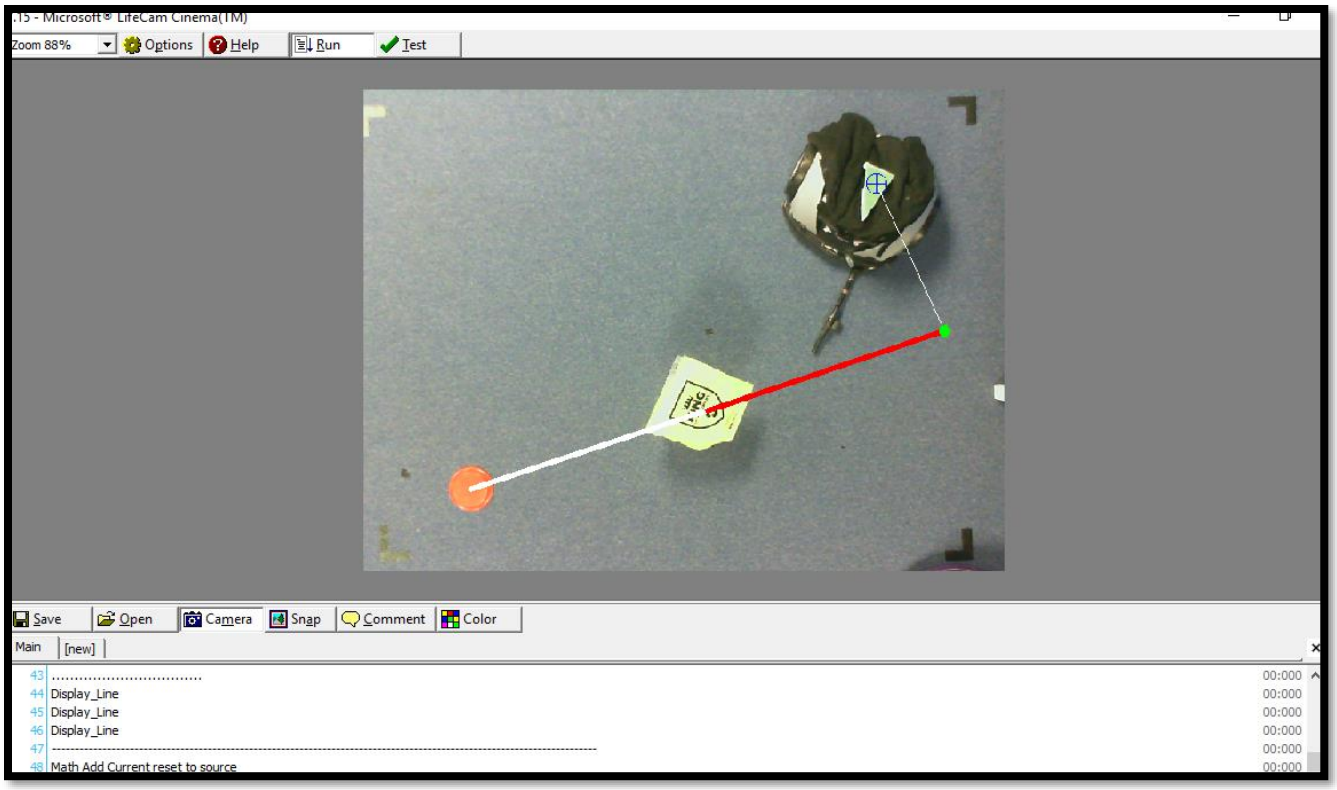
This is the point I wanted the bot to follow. Getting the robot angle, and orientation with the followpoint you can get it to line up with it, up to a certain degree and follow it, within that threshold.

Unfortunately, problems arose when testing, and due to being caught for time, the autonomous wasn't physically functioning, nonetheless the vision system detection was there etc.

Problems arose when testing due to the room carpet being blue which was on our presentation day.

This greatly affected the RGB filters and therefore no solid object recognition/distinctions could be effectively made.

<u>Presentation preview:</u>



```
.................................                                                   00:000
Display_Line                                                                        00:000
Display_Line                                                                        00:000
Display_Line                                                                        00:000
------------------------------------------------------------------------------------ 00:000
Math Add Current reset to source                                                    00:000
```

Due to the blue carpet, a yellow object had to be used instead of the blue box. Although this worked, it ended up interfering with the 'green' of the Bot recognition. Which led to all sorts of complications.

Nonetheless, the concept was explained and brought forth, as the idea is there, it just needs a bit of tweaking. With a little more time at it, there would be some good progress made.

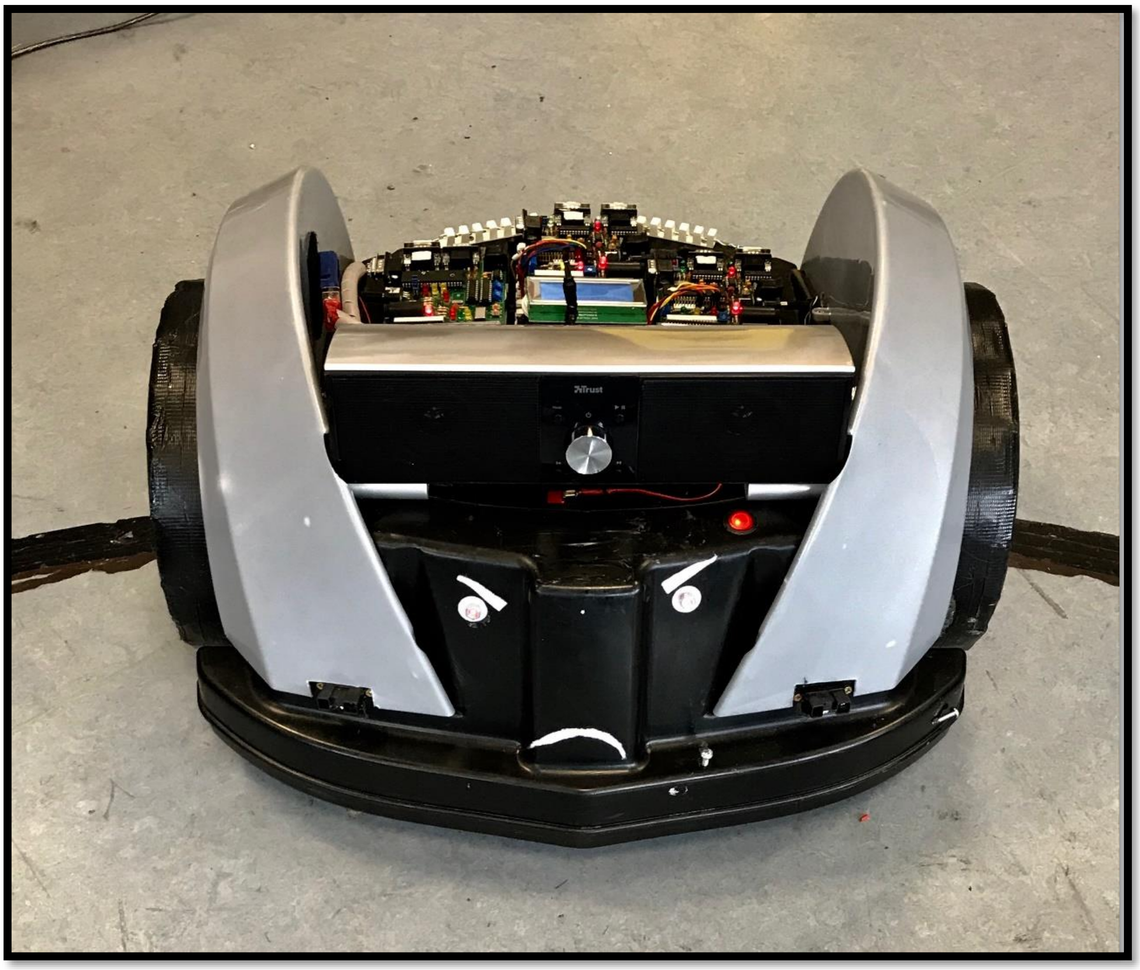
Having brought it up, we had a presentation at the end of this semester for the Transition Year students of Mount Sion, who came in to see what we have been progressively working on this semester.

It was a great day, as we got to speak to them about our journey thus far, explain things to them, etc.

Due to some problems with setting up/installing Pure Data on my computer, we couldn't get to implement Seamus's motor sound in sync with our Bot, however continued and decided to play music on the speaker, which was mounted on our Bot. It certainly lit up the atmosphere for that morning!

Although the motor controls worked quite well, and we had many of the Mount Sion guys played around.

BitBot!



1.6 Conclusion

This Project has certainly been a new learning experience and has given me knowledge and understanding of electronic embedded systems and electronics in general much more than I would have known previously. It's been quite an exciting road so far, designing my first ever PCB Schematic, the board blueprint itself, the Vision System, building BitBot, working together in our teams, but also as a whole class team effort, and presenting in the end in front of an audience, something we don't usually do.

From working on the PCB board, and being introduced to soldering in semester one, something I never did before, nor did I ever think about doing, to creating a working Robot from basically scratch, what a journey!

The experience so far was something else, and the way in which the learning tasks were achieved throughout the project as not only individuals but also as a class group, enhanced this learning curve of ours. Help was given and taken when needed and we all worked together for the most part to get up and going, which certainly proved successful.

Appendix

Chapter One

Using EAGLE: Schematic

https://learn.sparkfun.com/tutorials/using-eagle-schematic

Useful YouTube video for creating Schematic:

- https://www.youtube.com/watch?v=1AXwjZoyNno

Using EAGLE: Board Layout

https://learn.sparkfun.com/tutorials/using-eagle-board-layout

Useful YouTube video for designing your PCB board:

- https://www.youtube.com/watch?v=CCTs0mNXY24

Chapter Two

Vision System help:

Path planning.

http://www.roborealm.com/tutorial/Path_Planning/slide010.php