SWARM LOGIC AND ENERGENT BEHAVIOUR

Final Report

Swarm logic and Emergent behaviour

Final Report

Submitted for the BSc

in

Computer Systems Engineering

April 2016

by

Steven Balding

Word Count: 15,000

CONTENTS

Chapter 1 Introduction 5

1.1 Inspiration 6

1.2 Project Brief 7

1.3 Context 8

Chapter 2 Aim and Objectives 10

2.1 Objective 1 – Test basic robotics movement 11

2.2 Objective 2 – Develop colour recognition 11

2.3 Objective 3 – Communicate with the main computer 13

2.4 Objective 4 – Implement collision logic 14

2.5 Objective 5 – Implement object recognition logic 15

2.6 Objective 6 – Develop swarm rule set 2.6 16

2.7 Objective 7 – Experiment with swarm and rule set 16

Chapter 3 Background 17

3.1 Problem Context 17

3.2 Stigmergy 18

3.3 Emergent Behavior 19

3.4 Decentralised Collective Behaviour 20

3.5 BDI Model 21

3.6 Comparison of Technologies 21

3.6.1 Constraints 21

3.6.2 Propeller bot 22

3.6.3 Arduino 23

3.6.4 C 24

3.6.5 Python 24

3.6.6 Pro-log 25

3.6.7 Selection Matrix 26

3.7 Sensors And LEDs 26

3.7.1 Sonar sensor 26

3.7.2 Colour sensor 28

3.7.3 LED 30

3.8 Alternative Solutions 31

Chapter 4 Designs 32

4.1 Robot Design 32

4.2 Pro-log Code Structure Design 32

4.2.1 Connection Code 34
4.2.2 Packet manager Code 34
4.2.3 Decision Code 35
4.2.4 Send Code 35
4.3 Robot Code Structure Design 36
3.4.1 Rule Selection 36
3.4.2 void roam 37
3.4.3 void wall_check 38
3.4.4 void robot_check 39
3.4.5 void goal_check 41
3.4.6 void colour_check 41
4.4 Rules 42
4.4.1 Robots Collision Rule 42
4.4.2 Robot Mazing Movement Rule 43
4.4.3 Stigmergy Rule 44
4.4.4 Colour Seeker Rule 44
4.5 Experimental Design 47
4.6 Testing Design 49
Chapter 5 System implementation 50
5.1 Asci Converter 50
5.2 Colour Sensor 50
5.2.1 Attachment and wiring 50
5.2.2 code 52
5.3 Robots LEDs 53
5.4 Goal LEDs 54
Chapter 6 Project Management 54
Considerations 54
6.1 Processes and Methodology 56
6.2 Project success and failure criteria 56
6.3 Time Management 57
6.4 Planning and organisation 58
Risk assessment 6.5 58
Chapter 7 Testing 59
Chapter 8 Experiments Outcome 63
8.1 Data 64
8.2 Analysis 66
8.2.1 Emergent behaviour 66

8.2.2 Adding Rules 66
8.2.3 2 or 3 Robots 68
8.2.4 adding obstacle 69
8.3 Overall 69
Chapter 9 Evaluation 70
9.1 Project Achievements 70
9.2 Personal development 70
9.3 Further Work 70
Chapter 10 Conclusion 72
Chapter 11 Glossary 73
Chapter 12 Appendix 74
12.1 Appendix A 74
12.2 Appendix A – 2 75
12.3 Appendix B 76
12.4 Appendix C 77
12.5 Appendix D 79
12.6 Appendix E 81
12.6.2 Appendix E-1 81
12.6.3 Appendix E-2 82
12.6.4 Appendix E-3 83
12.6.5 Appendix E-4 84
12.7 Appendix F 85
12.7.1 Appendix F-1 85
12.7.2 Appendix F-2 86
12.7.3 Appendix F-3 87
12.7.4 Appendix F-4 89
12.7.5 Appendix F-5 89
12.7.6 Appendix F-6 91
12.8 Appendix G 92
12.9 Appendix H 94
12.10 Appendix I 95
12.11 Appendix J 96
Chapter 13 Bibliography 97

CHAPTER 1 INTRODUCTION

The purpose of this report is to outline the aims and objectives of the proposed project, in addition to detailing the context in which it was conceived. The report will also examine the project requirements and specification, as well as reviewing project management and progress made. The focus of the project is swarm intelligence and self-organisation; this report will include the design stages of the project and identify potential hazards found along the way.

1.1 INSPIRATION

This idea comes about from ideas put forth in Richard Dawkin's book *The Selfish Gene* (Dawkins, 1976). In this work, he states that genes are like the controllers and people are the machines that enable the genes; this might appear to have little relevance to robotics but looking closely could reveal key parallels.

The gene inside the machine is a controller; it works and communicates to the machine about how it should behave, its morphological state and other important factors. It does not directly control the machine but rather gives a list of instructions that are then followed when the events listed arise. So too can a robot function in this way. A high level computer can have programmed a list of instructions; for example, what to do if a sensor picks up a signal. The "machines" (in this case the individual robot) sends the sensor data to the controller. The controller then looks at the data, decides if it is close to an object or if the path is clear, and that information can then be processed returning a "behavior" (another list of instruction for the robot to execute).

This is a much stripped down example of the partnership working, but simple tasks can build to more complex ones, just as physical human reaction is simple but leads to complex emotion.

1.2 PROJECT BRIEF

Swarm intelligence comes in many forms, such as self-organisation, and often closely mirrors that of the animal kingdom. This project will attempt to replicate the type of swarm intelligence found in smaller animals such as ants or termites, by using robotics. Small, simple robots will be given a set of simple rules and together these robots will perform goals that a single robot could not.

The project started its life with the title;

"Swarm logic and instinct behaviour"

This title has not been changed as the project was falling into a places with little to no defined aims, end goal or repeatable results, this new title is one small step in helping focus this project. The new title is "Swarm logic and emergent behaviour".

The robots will start with only a few rules with more and more being added when the goal state is not reached within a time period. We will then be able to see the minimum amount of rules needed for cooperation to emerge but also the type of rules that can be taken away with little to no effect.

In this project the rules will be simple instructions that will increasingly define the goal state, giving more and more specific information to the robots as they progress.

The goal, in this case, will be moving a small plastic container across the robot's enclosure.

1.3 CONTEXT

Swarm intelligence is when simple instructions, in this case for robots, create larger, more complex actions and behaviours. In most cases these actions simply could not be performed by a single entity. Swarm intelligence is rife throughout most, if not all animals and is a key part of how they have evolved and survived. The idea of collective intelligence in a swarm can be identified in simple animals such as the termite; in particular, the way they build nests or map food travel routes. Schools of fish are another great example of a simple animal acting on simple instruction but creating complex behaviour; tune fish will, when threatened, closely swim in next to other fish drawing a circular pattern in the water. This in turn creates a tight ball of some half a million strong fish, making not only a good defence against predators but also scaring them off with the sheer size. (BBC, 2005) It is important to note that swarm intelligence does not have to be decentralised behaviour and, it can in fact be the sum of collective behaviour where the swarm works as a team but gains instruction from some type of leader source.

However, in this case this project will try to replicate self-organisation in the robots and not just swarm intelligence. Self-organisation is not the fish communicating with each other, any more than one cog talks to another inside a machine. The animals are independent and are following simple instructions, not communicating complex commands, positions and other sensory variables. This is what will try to replicated with robotics; a handful of small robots will be given simple rules to follow based on their own sensory feedback.

These rules will start out very simple and quite sparse, likely leading to the robots not being able to complete their goal, however as more tests are run more and more rules will be added to the robot’s logic. These rules will start to provide increasing knowledge about the surroundings, the goal state and how to achieve it. Building this rule set up will show how many and what type of rules are needed to start to see swarm like behaviour from the robots.

The goal state in this case will be Quantitative, Dr Davis states that “Gaols come in two types” when referencing the motivators for robots and it applies here as well. The goal is an objective that has an endpoint and as such rules can be built around it.

When looking at the rules that will be used in this project the will. After such time that a significant amount of rules has been built up; the rules will mesh together to create a form of collective information and behaviour that mirrors that of the instinct of animals and humans.

These rules could even be probabilistic and given chances of commands being triggered based on the most current evidence; in this way, the robots could even feasibly ignore the most sensible path. Of course, this is not actually ignorance but more choosing the unlikely path of code.

CHAPTER 2 AIM AND OBJECTIVES

In the original project aims, the robots were going to be mapped to a grid on the computer, but ultimately this was not reflective of swarm intelligence and self-organisation. For this reason, the aim of the mapping algorithm has now changed to the colour recognition stage, and there has been another stage implemented to add in the more complex pro-log commands that make decisions about wall collisions. See Appendix A for the project's aims in the initial planning stages.

Also in the original aims was an aim entitled '*Swarm to cooperate to move object*'. This has now been replaced with 2 new objectives relating to rules sets. This choice was made as the overall aim tying the project together was loose and a bit cumbersome, these new aims help give the project a clear goal and help the direction to has tangible re-producible outcomes. Please see Appendix A – 2 for old aim.

The aim of this project is to create a swarm that can move an object that a single robot could not move. This will be done via only Rules passed by a separate main computer. More Rules will be added to see when behaviour energies that complete the goal.

This aim will be achieved by delivering on the following objectives:

1. Test basic robotics movement
2. Develop colour recognition
3. Communicate with the main computer
4. Implement collision logic
5. Implement object recognition logic
6. Develop swarm rule set
7. Experiment with swarm and rule set.

2.1 OBJECTIVE 1 – TEST BASIC ROBOTICS MOVEMENT

This is a basic test of all the hardware involved. This will test that the propeller robot can simply move in place and send basic signals to the main computer. This step is a simple but necessary one as it ensures that all the hardware being used is ready and working as expected.

Aims:

- Test robot can be programmed and set up environment
- Test robot can move

2.2 OBJECTIVE 2 – DEVELOP COLOUR RECOGNITION

Colour recognition will be used in place of an eye for the robots; it will help the sensor know if what it is looking at is a wall or a goal object or even another robot. In the animal kingdom, there would be a lot more complex factors that are built up from lots of sensors and even the eye itself would be sending effectively thousands of bites of data to the brain in order to communicate the most likely object it perceives. The idea behind using colour recognition is to use a simple photo resistor to differentiate objects, using a different coloured LED attached to all of the objects in the robots' test area and checking the return resistance from the photo resistor; in this way, the robot has a concept of identifying one object from another.

There will still be, however, more logic than just sensing a colour and performing a command; the colour information will be sent to the pro-log script for outcomes to be determined. This logic will be implemented further in a later objective, as this stage is simply to attach a photo resistor and check that it can give different readings from different light sources.

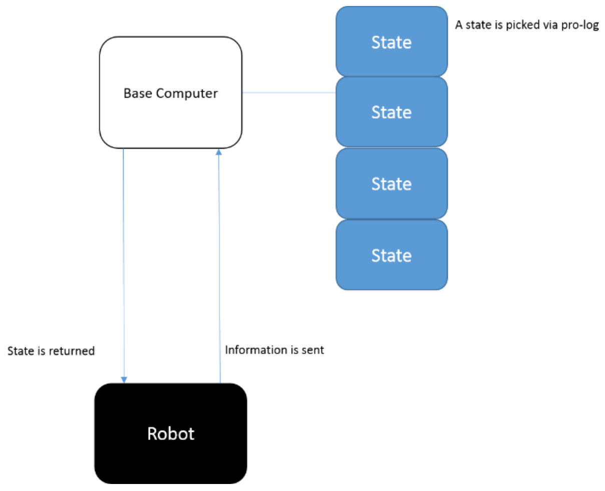
Aims:

- Test colour sensor
- Add sensor to robot
- Test sensor while on robot
- Test range of sensor
- Write functions for colour recognition

2.3 OBJECTIVE 3 – COMMUNICATE WITH THE MAIN COMPUTER

This is where the swarm logic starts to reveal itself. When the robots can communicate with the main computer, the robot can then access the commons "instinct" or logic that tells it how to interact with the world around it. However, "instinct" merely operates as a metaphor for the programmed commands that are given to replicate instincts within animals.

The aim of this objective is to set the robot on a random course and then, via the meeting of other robots or even the collision with them or a wall, the robot will send data about the nearing object back to the pro-log on the base computer. The base computer will then process the information and return a command for the robot to carry out. This simple series of events will be carried out for almost all actions and interactions the robots participate in. These processes will be unique to each robot as different information is being fed back to the base computer, but the commands that are sent back will be same for all the swarm. It was originally though that a python script would pick up the information sent from the robot, but it has since been discovered that prolog can create a socket for the robot to communicate with directly.



As more rules are added more logic will be added to the pro-log script, as this is where the rules will be accessed.

Aims:

- Create client server prolog script using c# to test functionality.
- Create client server prolog for central computer
- Connection robots through the central computer
- Send data through socket
- Manage data through a custom build package manager

2.4 OBJECTIVE 4 – IMPLEMENT COLLISION LOGIC

When the robot sensors something from the ping sensor it will stop and ask the prolog script what to do. There will be a list of response that are chosen and worked through and finally a result will be passes back to the robot.

This is where the rules come in the play. Most of the rules that will be provided will be dealing with an object upon detecting a close object.

As an example it might be that the basic rule set is simple to either check the object for colour and then either push the object or turn away from it.

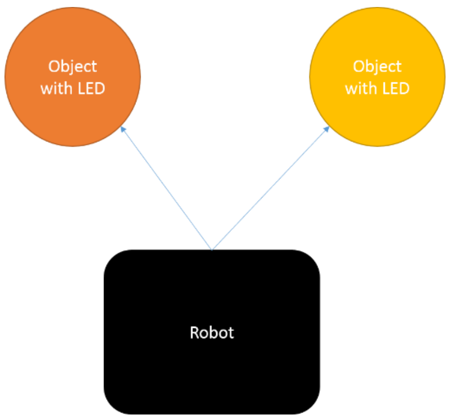
Aims:

1. Make robots stop at any close object
2. Send data to central computer
3. Implement responses and rules

2.5 OBJECTIVE 5 – IMPLEMENT OBJECT RECOGNITION LOGIC

This objective is similar to objective 4 but is the detailed construction of the functions that determine what an object is when it stops in front of it. This will only deal with the basic rules that all robots will start with.

The robot will have to make a discernment between a wall, another robot and the goal. This will be done with colour recognition. When the robot stops it check if there is any bright colour coming from it facing direction, this will give a good indication of the objects being a wall or something else. After this check is complete, the robot will feed back to the central computer and prolog will then identify bases on the reading if it is a wall, goal object or even another ro-bot. Commands can then be returned based on the variables at play with that robot made.



Aims:

- Add detailed function for sensor picking up an object

2.6 OBJECTIVE 6 – DEVELOP SWARM RULE SET 2.6

In this task the rules set, further than the basic rules, will be addresses. This will incorporate the research of potential rules as well as the implementation of both the robots and the central computer.

These rules will be turn on gradually to see at which point the robots can complete their goal.

Aims:

- Research rules set
- Implement rules set

2.7 OBJECTIVE 7 – EXPERIMENT WITH SWARM AND RULE SET

In this final stage the completed robots will be put through experiments that will add more robots and more rules set, and record the results.

The resulting data will be plotted on to graphs to show the success or failure of the results clearly. This will be a great analysis and overview to the data and hopefully show us that simple rules can in produce simple emergent behaviour.

Aims:

- Design experiments
- Record data
- Plot graph
- Analyse data

CHAPTER 3 BACKGROUND

This chapter addresses the background research and some ideas generated from that research.

3.1 PROBLEM CONTEXT

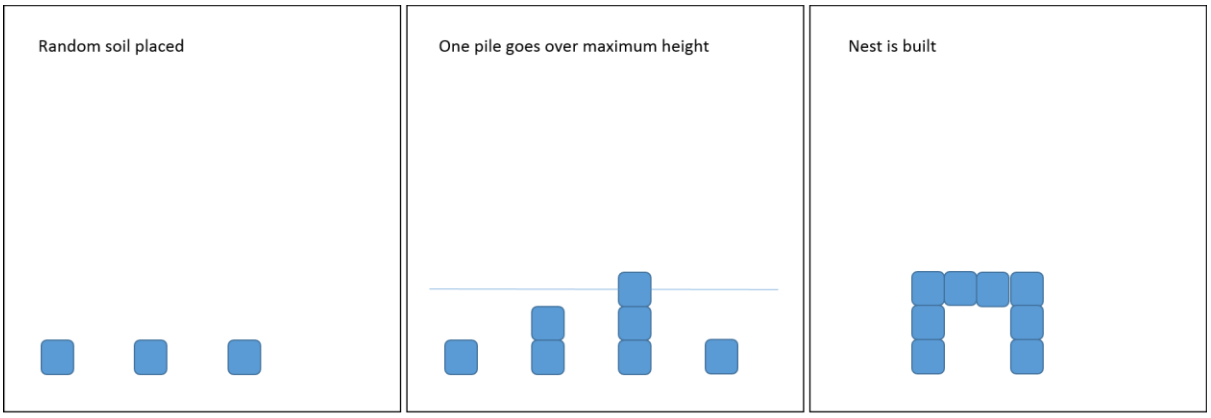
The focus of this project is swarm intelligence with a heavy emphasis on self-organisation. Swarm intelligence is a diverse field that incorporates a wide variety of research, but at its core it is simply the collective intelligence of a group of sentient entities. This, for example, could be a troop of ants; they are simple creatures with a limited intelligence but appear to be working together to achieve very vast and complex goals, like building a nest or planning a food route. This, however, is not the individual ant being blessed with the great oversight and complex planning skills needed to create such magnificent and beautiful nests, but rather the ant is following a simple set of instructions that, when carried out but multiple ants, creates the structure we observe. The instructions given to the ant are of course not literal instructions given from some leader source, although that could be a form of swarm intelligence. Rather, it is the instinct implanted inside the ant from birth; where this comes from is up for debate, but one thing is certain - this simple ant acts on simple instincts to create great wonders with the help of its swarm.

Self-organisation is one aspect of swarm intelligence, but a key one; it is in part a process in which coordination arises out of psychical interaction between smaller entities of the initially disorganised swarm. This behaviour can be triggered by many different actions, including positive feedback and negative feedback, but all behaviours rely on “applications of fluctuations” (Eric Bonabeau, 1999). Animals will always have some amount of random activity that goes outside their normal routine; This is really the key to true emergent behaviours as these random activities born out of feedback could create new behaviours that could benefit the swarm.

because of the fluctuations in the animal's patterns, new traits are reinforced or removed over long periods of time. This feedback from the swarm can come in two different forms: direct feedback, with examples such as antennation or trophallaxis; or indirectly, in the case of Stigmergy - using the environment to influence the swarm.

3.2 STIGMERGY

Stigmergy is an important part of swarm intelligence and an important part of this project. Stigmergy, in essence, is when a member of the swarm modifies the environment to feedback or trigger a response from other members of the swarm. When an animal performs an action which impacts on their environment, the resulting effects will influence other member of the swarm that come into contact with these changes, which leads the animal to change its behaviour based on the altered surroundings. A great example of this is the termite constructing a nest: "Coordination and regulation of the building activities do not depend on the workers themselves but are mainly achieved by the nest structure". (Eric Bonabeau, 1999)The termites will randomly deposit soil clumps down until one reaches a certain high; at this point the termite triggers a response and the nest is built from that point outward.



🟦 - Soil Piles

— - Height need to be reached before building point is selected

In this project we will modify the environment by extending the goal object when a robot starts to push it. This in turn will change the environment in which the robots exist. This will be achieved by using LED on the robots themselves. One of the rules that will be added will state that when a robot is pushing the goal object it will turn its read facing LED to green, the same colour as the goal, and then continue to push.

3.3 EMERGENT BEHAVIOR

This is the core philosophy of swarm intelligence and it is a process in which simple behaviours that a swarm might repeats many time in a day lead to new and sometime more complex behaviours. This is usually trigger by "applications of fluctuations" (Eric Bonabeau, 1999), or randomised routing, sometime even environmental changes will draw out new behaviours. In this project we will see what type and how many simple rules it takes to start to see this new behaviour, adding new rules will act as our fluctuation or mutation and we can observe the predicted behaviour that we are trying to achieve but also keep an eye on any new behaviour that might emerge, even if these be erroneous.

In this project will be add rules to the robots in order to elicit the robots moving the goal state. More rules will be added to determine how many and what type of simple rules are needed for emergence behaviour to be present.

It could be argued that the behaviour of these robots is not emergent and that the rules set is forcing it to complete the goal and this could be a fair assessment but this simple project will at least show the signs emergent behaviour even if it has been slightly pushed in that direction.

3.4 DECENTRALISED COLLECTIVE BEHAVIOUR

Some forms of swarm intelligence stem from a leader or single source; however, one could argue that this is not true swarm intelligence as the swarm is being led rather than acting as individuals to produce results. Decentralised collective behaviour is separating the process from a central location or authority and giving the power to the individual entities in the swarm. However, there will be a base computer which houses the pro-log logic, this is just because it is not cost effective and time effective to equip each robot with the capability to contain this internal logic - the separate threads of the pro-log script act as the individual brains of the robots. One could argue that in the case of ants, only very simple instructions are being processed on the individual level, and therefore this could be replicated with the robots; but, given the scope of this project, outsourcing it to a main computer is ultimately more feasible.

3.5 BDI MODEL

The BDI Model (Belief, desire, intention model) was another part of the background research that could have been implemented in this project. It “is a schema that calculates the actions of an agent based on its beliefs and its desires.” (Darryl Davis, 2012), what they want to do and where they are heading next. It is a very interesting model that could really closely replicate how animals make decisions based on factors like what the swarm is doing and its desires and how do these conflict with the individual.

This model, although very applicable and interesting, would be too complex for the scope of this project and was deemed unfit as a model for this project. The research in to this model has not been wasted as it has helped to construct the prolog and robot code.

3.6 COMPARISON OF TECHNOLOGIES

There are many technologies that are integrated together for this project and many more that could have been used. There are hardware components that build the robot, different languages used to command the robot and inform its logical processes, and even quite diverse technologies just to manage the project itself.

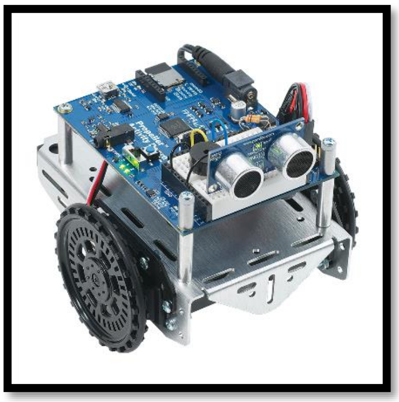
3.6.1 CONSTRAINTS

At this point in the report it should be mentioned that although there were robots and code samples that were already in place I could have forgone these and worked with my own robots or different langue’s. Because of this I drew up a selection matrix but the heaviest waiting is given to the ease of use. Ease of use will take in to account the pre-existing code samples and pre-built robots.

Below you will find an overview of the technologies used and then a selection matrix that takes into account the given field.

3.6.2 PROPELLER BOT

This is a microcontroller based on an activity board; it has basic I/O and some features such as PWM. It uses Propeller C as the default language which is very similar to C, but uses high level functions to perform some of the more day to day tasks. It comes with space on the top of the robot to add in extra components and header pins that make adding extras simple. Servos have been added for movement and each wheel is tracked using a ping sensor.



A wireless chip is also installed to send and receive data from the base computer. There will also be some sensors such as a photo resistor and ping sensor.

The propeller library also has built in function such as **Drive_speed()** or **ping_cm()** to help ease to processes of controlling the robot.

The propeller bot is the natural choice for this project due to its flexibility, cost and overall user-friendly nature. The department also has 5 of these ready to use so cost and ease of use score very high here.

Familiarity: 1

Complexity: 2

Cost: 5

Ease of Use: 5

3.6.3 ARDUINO

This is another micro controller that could have been used for the robots; it uses C++ for its default language and, much like the Propeller chip, it has a user friendly library to make more complex electronic tasks simpler to programme. Although this comes with the same connections and relative low complexity, these would have to be purchased and the extra complication is not such that this would be an effective idea.

Familiarity: 4

Complexity: 3

Cost: 0

Ease of Use: 2

3.6.4 C

The C language is a very flexible programming language that is an imperative type paradigm. It is widely used in everything from embedded systems to game engines and so documentation for it is plentiful and vast. Its fits perfectly into controlling the robots as it gives lower level commands with ease, but the libraries already exist to make some of the more rudimentary pieces of code quicker to implement.

Familiarity: 1

Complexity: 1

Available resources: 3

Ease of Use: 1

3.6.5 PYTHON

Python is a widely used high level language with a focus on code readability. Python can be object oriented and this makes it a tempting prospect, as building states for the robot as objects could be a great way to organise the logic in the designs and in the code; but, as a result of little experience with Python, coupled with the fact that pro-log offers the exact type of logical programming needed, means that this will not be the choice for this project.

Familiarity: 1

Complexity: 4

Available resources: 3

Ease of Use: 1

3.6.6 PRO-LOG

This is a logical declarative language that is commonly used “especially in the field of artificial intelligence” (Bramer, 2013). The idea behind Pro-log was to create a programming language that can answer questions and do a lot of the logic for you. Pro-log uses fact rules and simple data types in order to build up a bank of knowledge.

This knowledge bank can then be queried in order to ascertain logical responses; an example of this would be:

boy(dan)

which is equivalent to saying Dan is a boy. We can then query the data and ask

?- boy(dan)

Pro-log has now queried ‘is Dan a boy?’ to which we would be given the result: true.

This type of logical programming can be very useful for this project by building a base of states for the robot, for example “Roaming” or “Stopped” and then, depending on the sensor data received from the robot, a new state can be calculated by pro-log and sent back to the robot to simulate a new state.

As mentioned in the previous chapter there is a prolog communication script in place that I can use (appendix A-2) and for this reason the ease of use has a very high score it is most likely the choice to make.

Familiarity: 1

Complexity: 1

Available resources: 5

Ease of Use: 5

3.6.7 SELECTION MATRIX

	Pro-log	**Python**	**C**
Familiarity (x1)	1	1	1
Complexity (x2)	1	4	1
Available resources (x3)	5	3	3
Ease of Use (x4)	5	1	1
Total	38	19	13
	Arduino	Propeller bot	
Familiarity (x1)	4	1	
Complexity (x2)	3	2	
Cost (x3)	0	5	
Ease of Use (x4)	2	5	
Total	18	40	

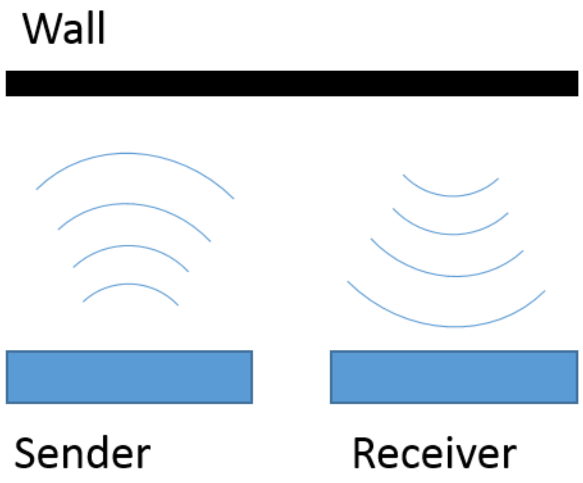
From this it is clear that the choices will be the propeller bot and prolog.

3.7 SENSORS AND LEDS

Aside from the languages and robots being used there is also sensors mounted to the robots and LEDs mounted to the rear. Below is an overview of those sensors and LEDs.

3.7.1 SONAR SENSOR

This component will be used in order to determine distance from an object. By sending a sonar ping from the sensor and timing the return time, if any, one can measure the distance the robot is from an object.



This simple formula can be used for this timing, and as speed of sound is a fixed speed (within reasonable tolerances) then this can be used to determine distance. Sound travels at 0.03448 cm/µs at room temperature.

$$v = d * t$$

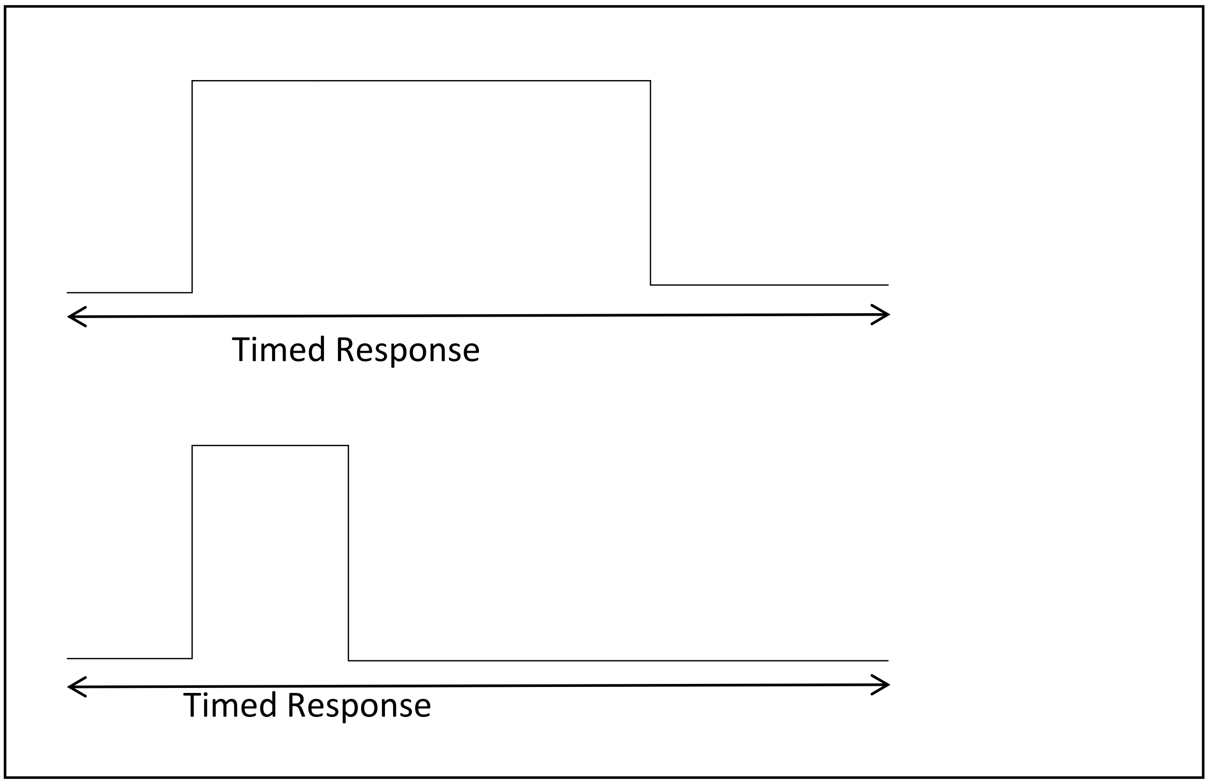
v = Speed, d = distance, t = time.

Because there is a ping sent and then received, this has to be taken into account in the calculation, so the final value becomes:

$$d = \frac{(s * t)}{2}$$

3.7.2 COLOUR SENSOR

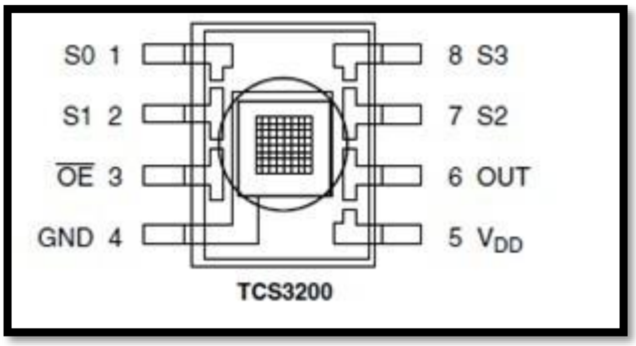
The colour sensor used is the TC3200. This sensor is mounted to a breakout board that contains all the necessary resistors and regulators. The sensor takes a reading the amount of light passing through it. It measures in green light red light and blue light. The basic principle is that you send a high signal and when the light is detected it returns a low signal on the same line, this gives you an amount of time which represents the intensity of that given light colour.



The way to select which colour you want is as follows; to select a colour a signal combination must be set, below is the char for these values.

S2	S3	Colour to Read
LOW	LOW	Red
LOW	HIGH	Blue
HIGH	HIGH	Green

Below is an overview of the sensor and the pins outputs/inputs.



There are a number of pins for this sensor and they are follows:

- SO – Ground Scaling – This is used in conjunction with the supply scaling to sync the voltage.
- S1 – Supply Scaling – This is used in conjunction with the ground scaling to sync the voltage.
- S2 – Selection Input – This is used to select red, green or blue from the sensor.
- S3 – Selection Input – This is used to select red, green or blue from the sensor.
- OE – Enable for active low
- GND - Ground
- VDD – Supply Voltage

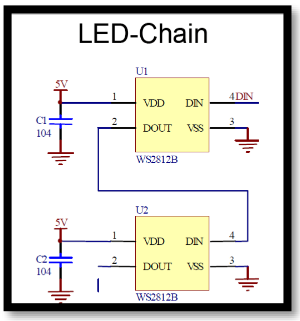
For a more detailed picture of the sensor please see http://www.mouser.com/catalog/specsheets/TCS3200-E11.pdf for the data sheet.

3.7.3 LED

The LEDs used in this project were the ws2812b. These are 3 colour led that a very bright for their small size.

(Adafruit, 2016)

They have 3 inputs and 3 outputs, a Vcc, Ground and data line. The outputs are matched to the inputs and are present so that many can be wired together to make a pattern or matrix.



(Adafruit, 2016)

In this project only one will be used per robots so there is no need for an out-put connection.

3.8 ALTERNATIVE SOLUTIONS

There are plenty of ways of addressing swarms and emergent behaviour and in this project there was an alternate route to follow.

The project could have been centred around a fixed amount of commands that grew or maybe even learnt in order to try and create new behaviours. This idea would still have fitted the brief and could have been a very interesting idea for this project but there are a few factors that limit the success of that project. Using machine learning is not something taught until the second semester and that would not leave much time to get into the main section of that idea. Another factor is that my research on this idea found it to be very complex and would be beyond the scope of a third year project. It is however worth stating that these two ideas could have or could be combined to create a project that not only increase rules to elicits emergence behaviour but also morphing and learning of those rules

CHAPTER 4 DESIGNS

This section covers all the designs of the project as well as the aims and thinking be hide them

4.1 ROBOT DESIGN

Each robot will be designed to have sensors that can give basic information about the surroundings. The behaviour of insects such as ants and termite is central to the core concepts of self-organisation and swarm logic as a whole; therefore, the design of these robots will reflect that in some capacity, using sonar sensors in place of antennas and a microcontroller as a simple insect brain. The parallels are tenuous here, as even a simple creature such as an ant still has a very complex system of nerves and neurons, but these robots will demonstrate what a swarm can achieve with simple instructions and simple design.

As mentioned before these robots has been pre-build and as such I have not designed the construction of the robot but rather how it will be used and what sensor I will use. I have, however, added a colour sensor to each robot.

4.2 PRO-LOG CODE STRUCTURE DESIGN

The idea behind the pro-log code is to create a socket that can join with the robot and stream data back and forth through that socket. This code will receive a package from the robot that will be structured to read as first what the data is that is being send and then its value.

There was some code given to me by Dr Davis that helped with the initial stages of prolog connection. Please find this in appendix B. This code, however helpful, was not fully understood by me and so to better understand prolog connections I created a peace of code to connect prolog to a c# console application, please find this in appendix C. This code is a simple TCP socket that is easy to construct in c# using the TCP listener class. I then worked through several iterations of code to connect send and receive a message in prolog. I was able to create a successful connection and understand the TCP connection in prolog.

The code will also house states to pass back to the robot, these will contain a unique name, a message to send to the robot and a response from the robot. A value between 0-255 will be send back to the robot and that will correspond to a function embed into the robot's brain.

This interaction back and forth might seem to be moving away from the core concepts of swarm intelligence as there is two entities communicating, but really this is just a way of removing the complex logic that brain has to do and placing it in a faster computer that can store more code.

The Central computer code can be broken up in to 4 main sections. Connection, Packet manager, Decision, Send.

4.2.1 CONNECTION CODE

The pseudo code is below. Please see appendix F-2 for actual code.

```
Set motor speed
Drive motors
Set Distance
While(Ping Sensor => Distance)
{
        Stop motors
        Ask central computer for more instructions.
}
```

4.2.2 PACKET MANAGER CODE

The pseudo code is below. Please see appendix F-2 for actual code.

```
Set motor speed
Drive motors
Set Distance
While(Ping Sensor => Distance)
{
        Stop motors
        Ask central computer for more instructions.
}
```

4.2.3 DECISION CODE

The pseudo code is below. Please see appendix F-2 for actual code.

```
Set motor speed
Drive motors
Set Distance
While(Ping Sensor => Distance)
{
    Stop motors
    Ask central computer for more instructions.
}
```

4.2.4 SEND CODE

The pseudo code is below. Please see appendix F-2 for actual code.

```
Set motor speed
Drive motors
Set Distance
While(Ping Sensor => Distance)
{
    Stop motors
    Ask central computer for more instructions.
}
```

4.3 ROBOT CODE STRUCTURE DESIGN

The robot will create a serial connection with the terminal and the Wi-Fi chip in order to communicate what is happening to the screen and connect to the pro-log socket. This is done with an inbuilt library called fdserial which creates a full duplex connection to the pins provided.

Below is an example of this connection.

pc_serial = fdserial_open(31, 30, 0, 115200); //

Enable serial port out to monitor

```
ext_serial = fdserial_open(6, 5, 0, 115200);  //
```

Enable serial port for Wi-Fi by creating an object **ext_serial** we can send data via commands such as **fdserial_rxReady**. By passing the **ext_serial** to this function we can check the buffer of that connection.

The code will run a while statement that check to see if a shutdown command is send, if not, the code will run through a switch. The input for this switch is the return value from pro-log and depending on the result the corresponding function is run to command the robot.

Below is an over view of the whole system in pseudo code

The next subsections detail the main functions of the robot's code.

3.4.1 RULE SELECTION

This is a switch case that waited on the central computer to send data to the robot, data is then picked up and a function is selected based on the value sent. We use the **fdserial_rxTime** function which waited for data to be placed into the buffer and then reads it. It takes the **ext_serial** object pointer as a parameter and in this way the function knows where to look for the revived data.

The pseudo code is below. Please see appendix F-1 for actual code.

```
While(Connection object < 0){} //Wait for connection

Return Value = Data sent from central computer.

While(Return Value != 48) //This will be a disconnection number

{

        Select function based on Return Value

}

Close Connection Object
```

3.4.2 VOID ROAM

This function provides the robots the ability to move around the enclosure. This will, in a sense, be the default behaviour for the robots, it will be the starting behaviour and the one that the robots returns to after detecting a wall or another robot.

The idea behind this function is to drive the motors, using the drive speed function, whilst checking the values on the ping sensor, when the sensor returns a value lower than a pre-defined distance the motors will stop and a transmission will be sent to the central computer for further instructions.

The pseudo code is below. Please see appendix F-2 for actual code.

```
Set motor speed
Drive motors
Set Distance
While(Ping Sensor => Distance)
{
    Stop motors
    Ask central computer for more instructions.
}
```

3.4.3 VOID WALL_CHECK

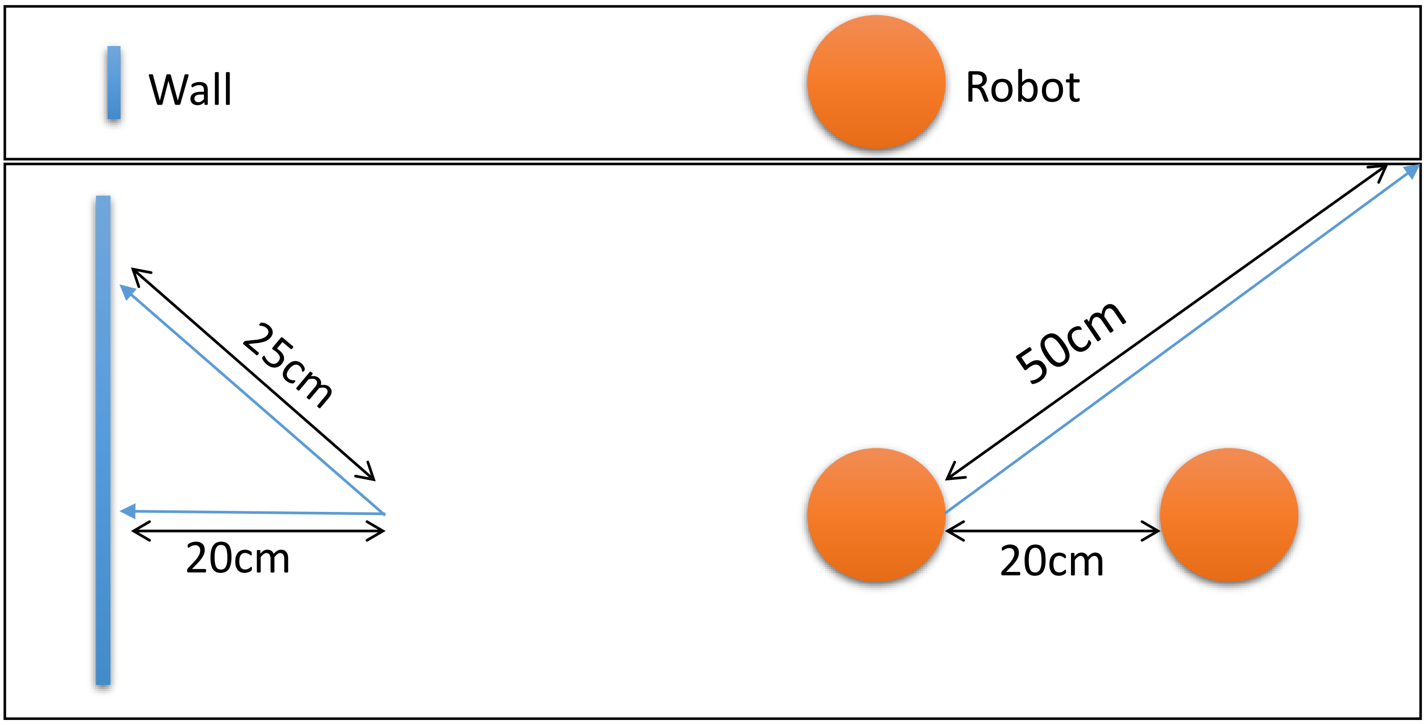
This function will be called after the robots has detected an object and will ascertain if that object is a wall or not. The idea is to check if there is any strong green light, this will quickly check if the robot is looking at a goal object (which has a green light). The ping sensor will then check to see the object has moved over the next second, this will give further indication that the object the robots is in front of is a wall (they seldom move).

The pseudo code is below. Please see appendix F-3 for actual code.

```
Check for green light
Save ping sensor data to a variable
Save another ping sensor reading to a variable
Compare the two readings
{
    Minus second reading from first
    (if compared reading is < -30 or > 30 )
    {
        Send to central computer object is a wall.
    }
    Else
    {
        Send to central computer object is not a wall.
    }
}
```

3.4.4 VOID ROBOT_CHECK

This function will be called after robots has detected an object and will ascertain if that object is another robot. There are two steps to this functions, checking for a red light and then turning 45 degrees and checking distance. All robots will have a red light on them, so if the colour sensors detect a red light then we can instantly determine that object in front of the robot is another robot, most of the time this should be the determining factor. Another check if this fails is the move the robots 45 degrees and check distance if it increased by a large amount then the robots will not be looking at another robot as it will have moved out of site.



The pseudo code is below. Please see appendix F-4 for actual code.

```
If(Colour is not red)
{
        Move 45 degrees
        Check distance
        If(distance is greater than 25 )
        { Send to central computer that object is not robot }
        Else { Send to central computer that object is robot }
}
Else { Send to central computer that object is robot }
```

3.4.5 VOID GOAL_CHECK

This will simply check if the light in front is green. As more rules are added this will also include turning the colour of the robots led green.

The pseudo code is below. Please see appendix F-5 for actual code.

```
If(light green)
{
    Send to central computer that object is goal, then call push function.
}
Else
{
    Send to central computer that object is goal, then call push function.
}
```

3.4.6 VOID COLOUR_CHECK

This is the function that check the colour. It takes in a parameter of a colour and a pin then returns the value of micros seconds it too to receive a response. This is a very simple function and is mostly a switch case, but it still saves time and make code more readable to use this function.

The pseudo code is below. Please see appendix F-6 for actual code.

```
Switch pin 7 and 9 to high or low depending on the colour selected.
Return Pulse_in(Pin selected,0)
```

4.4 RULES

Detailed below is that rules that will be implement to complete the objective of moving an object to a goal. There are 4 additional rules added and these are:

- Robots Collision Rule
- Robot Mazing Movement Rule
- Stigmergy Rule
- Colour Seeker Rule

The basic rule set contains only information to avoid Collison with a wall and to push anything with a green light. As more are added the time will be recorded and compared to addition of rules.

These next sections will detail the rules, including their aim and their pseudo code.

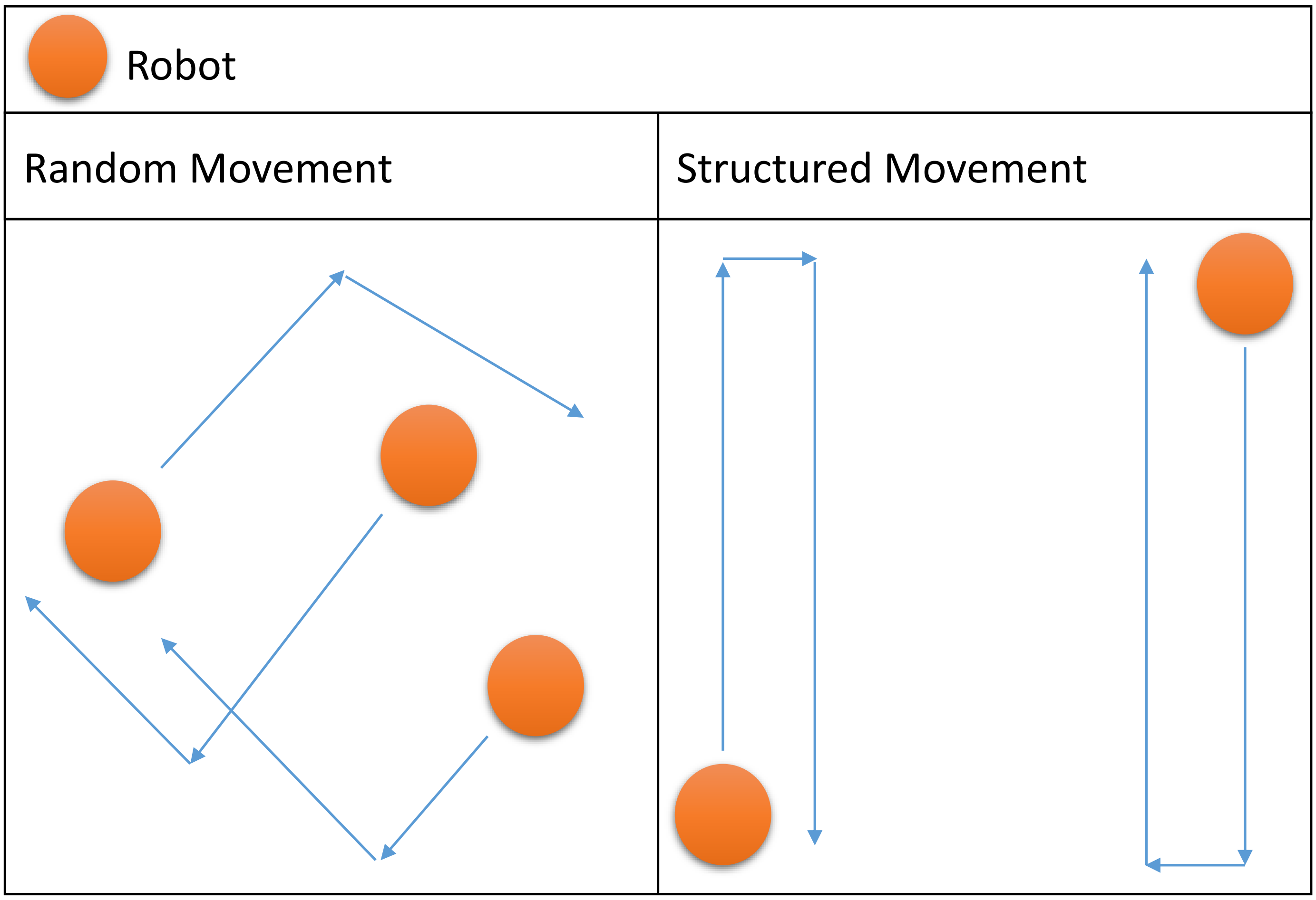
4.4.1 ROBOTS COLLISION RULE

This rule is to deal with when a robot detects another robot. This might in practice seem like a very similar rule to normal collision but this will stop the robots clustering as they move differently when they detect a robot. Spreading the robots out will help find the goal object quicker.

This rule has already been detailed in section 3.4.4 “Void Robots Check”, please see there for code.

4.4.2 ROBOT MAZING MOVEMENT RULE

This rule is to set the moment after object detection to something more regimented methodical. The aim is the have the robots move directly back after a detection and turn 90 degrees right, move forward a small amount and then turn another 90 degrees right. This will mean that the robots move in a structures manner that will always encompass the entire robot pen. The name mazing movement was given as it replicate a maze structure.



Below is the pseudo code.

```
Detect collision
Turn 90
Move
Turn 90
```

4.4.3 STIGMERGY RULE

In this project the idea of Stigmergy (changing the environment to affect the swarm's behaviour) is put across by incorporating the robot into the environment. When a robot starts to push the goal object it will turn it rear facing LED green. This will mean that other robots will see the green light and want to push the robot instead, this will have the effect of extending the goal object. The aim here is to make it more likely that other robots will come into contact with the goal object and faster push it to the finish.

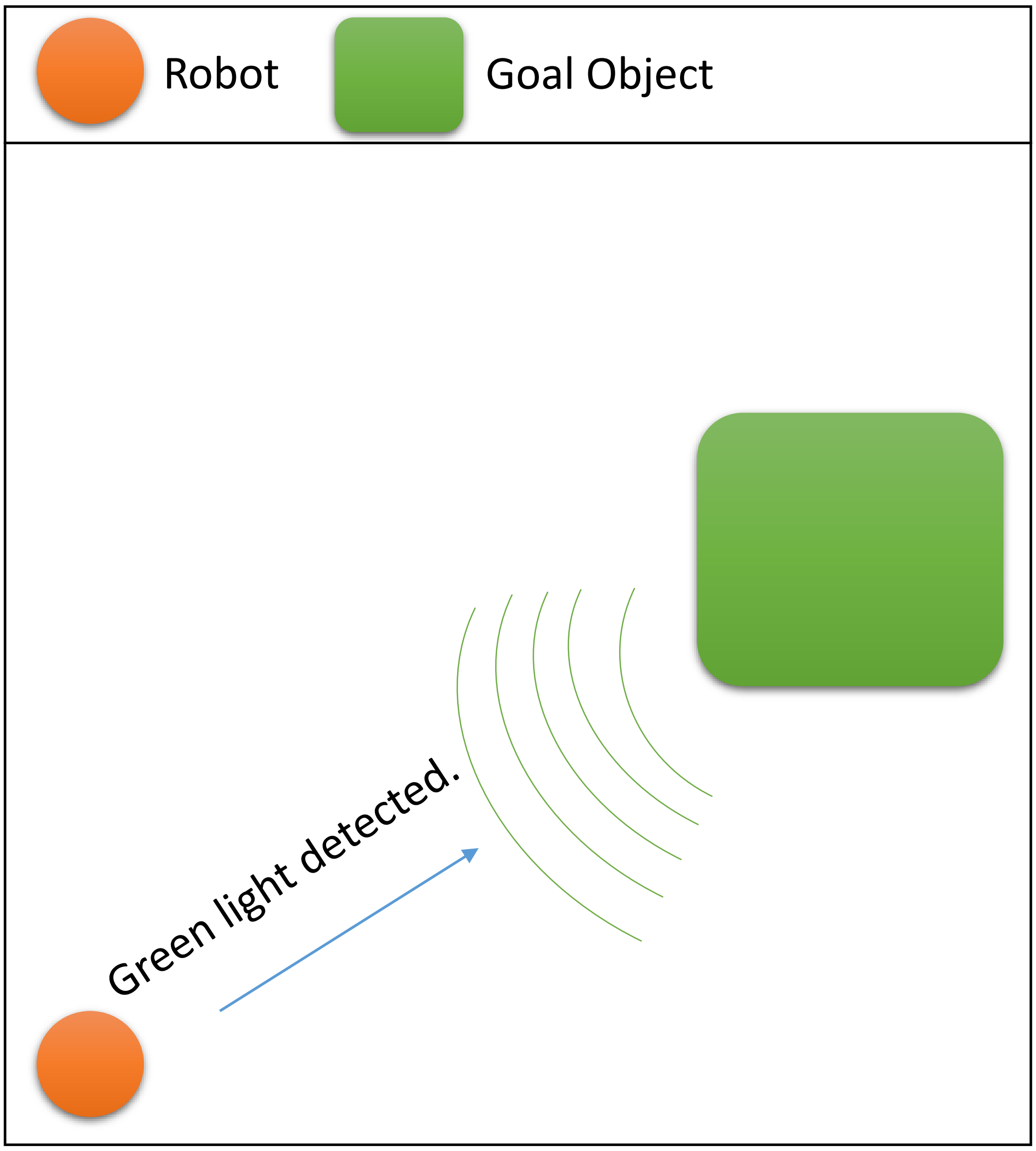
Below is the pseudo code.

```
Goal object confirmed, push function is called.

Push()

{

        Drive motors

        Turn rear facing LED green

}
```

4.4.4 COLOUR SEEKER RULE

This is the final rule that will be added, the aim is to have the robots seeks out areas of higher green colour. This will give a greater change of the robots finding the goal object.

The robots, upon dealing with a wall detection, will turn slowly and take reading of green light. If it picks up a hint of green light it keeps turning until the values get lower, if no proponent green light is found then the robot will rotate


Robot
Goal Object
Green light detected.

Below is the pseudo code.

```
Loop
{
    Rotate robot for 100ms
    Read green light
    If(Green light > 1000)
    {
        Rotate robot for 100ms
        Read green light
        while (Green light reading > last green light reading)
        {
            Rotate robot for 100ms
            Read green light
            Back to roaming
        }
    Else
    {
        Normal behaviour, back to roaming
    }
}
```

4.5 EXPERIMENTAL DESIGN

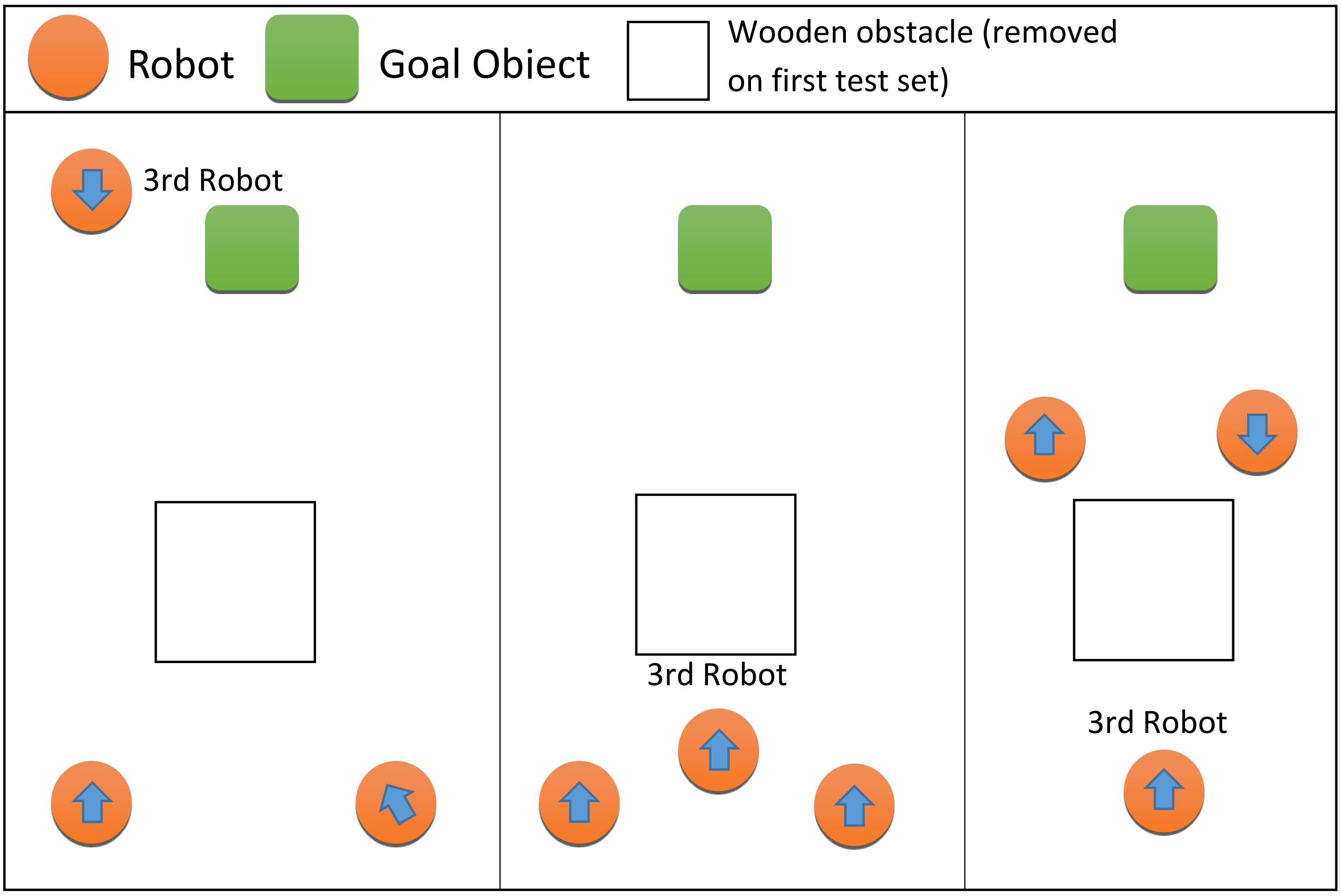
After the basic communication, wall collision logic and encounter of other robot's logic is working an experiment will be carried out to see what happens in prolonged use. Simple testing of these features will weed out the smaller bug in the software but having the robots interact with their surroundings will further test the capability, but, more so than this, might show some unexpected results, as is true for real swarm and their fluctuating behaviour.

Below is the test plan that was devised. This plan will run through a test with 2 robots and test with three robots for each rule. The tests will be run several times with the robots placed at different position each time. These position will stay constant across all the test however. An average will then be taken from the different positions to find the final value for time taken.

The first set of test will be carried out in an environment with only the gaol object, the entire set of test will then be re-run with an adjustment to the test, a small box in the middle to make it more difficult for the robots to navigate to the goal object. The second set of tests should take more time but one of the predicted outcomes is that there will be a non-linear difference between the first set of timings and the second set because more rules added the more they because greater than the sum of their parts.

	BASIC RULE SET
2 ROBOTS	
3 ROBOTS	
	Robot Collision Rule Set
2 ROBOTS	
3 ROBOTS	
	Robot Maze Movement Rule Set
2 ROBOTS	
3 ROBOTS	
	Stigmergy Rule Set
2 ROBOTS	
3 ROBOTS	
	Colour Seek Rule Set
2 ROBOTS	
3 ROBOTS	

Below is a representation of the starting positions for the tests.



4.6 TESTING DESIGN

Through this project tests will be run to ensure that all software is working correctly. This will be done after completion of specific sections of the project, sections that are either integral to the successes of this project or have complex code or behaviours associated with them.

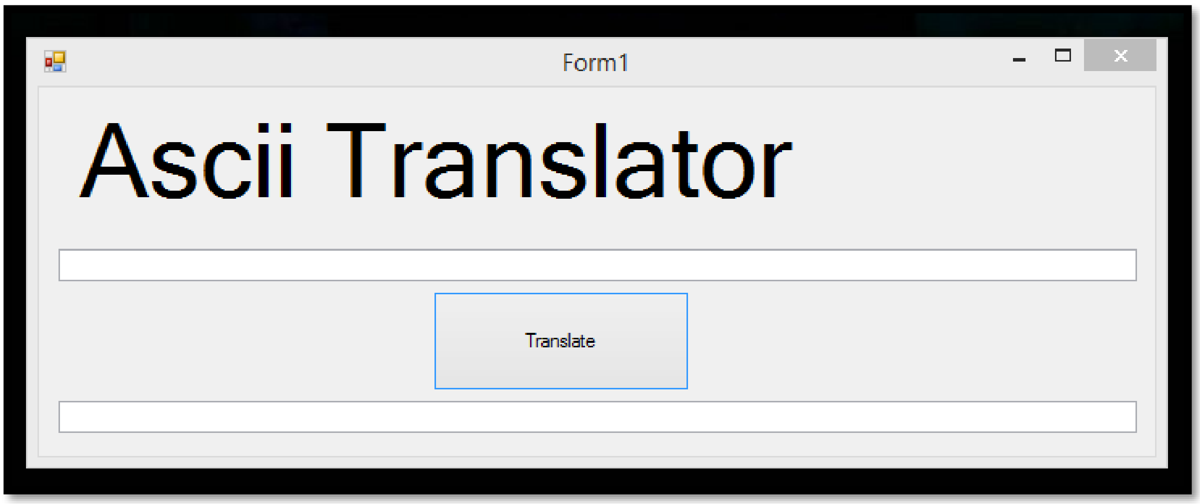
Each test, while being unique to that peace of code, will have a set structure that is followed. Each test will have four simple directives:

- Does it perform to the minimum expectation – This test is to examine if the tested item works, these tests will be at the bottom level and will only test that bare minimum functionality.
- Does it perform as expected – These tests are to check if it performs not just as required but as hoped for to make the project a success. The tests will challenge some of the more desired functionality of the item as well as document some side effects that might not be detrimental or even preferable.
- Does it have any negative side effects – Are the any unexpected results that negatively affect the core functionality.
- Does it have any areas of improvement – These tests will check to see if there are parts of the item that could benefit form more work or even a re-deign. An example of this might be timing the completion of a peace of code, this might not be necessary to project successes but still useful for quality. This is the only type of testing that might not be run on all sections, but it is hoped to be run on as many as feasible.

CHAPTER 5 SYSTEM IMPLEMENTATION

5.1 ASCI CONVERTER

While learning about socket connections in prolog there was a lot of byte code being sent over the socket, in order to query these packets and convert them into a form I could easily understand I make an asci converter. This took a byte code and translated it into asci text. This was a simple construction but saved a lot of time and trouble when used.



Please see appendix G for the code.

5.2 COLOUR SENSOR

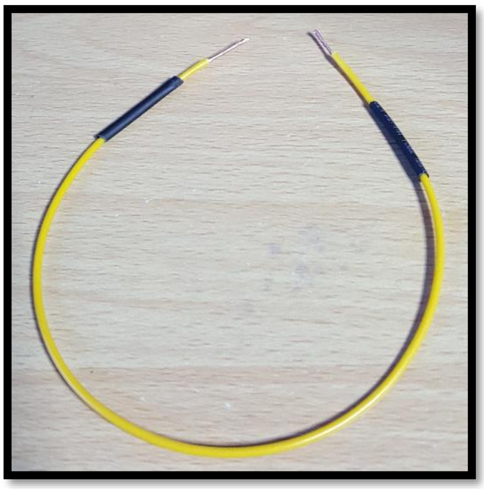
The colour sensors that are described in the previous section had to be wired together and I had to make a function for the sensor.

5.2.1 ATTACHMENT AND WIRING

Wiring these sensors was broken down in to 4 steps.

All picture taken form project workbench.

1. Prepare wire – striping both ends to reveal wire and then twisting to help keep the strands together. There was then a shrink sheath placed on each end of the wire.

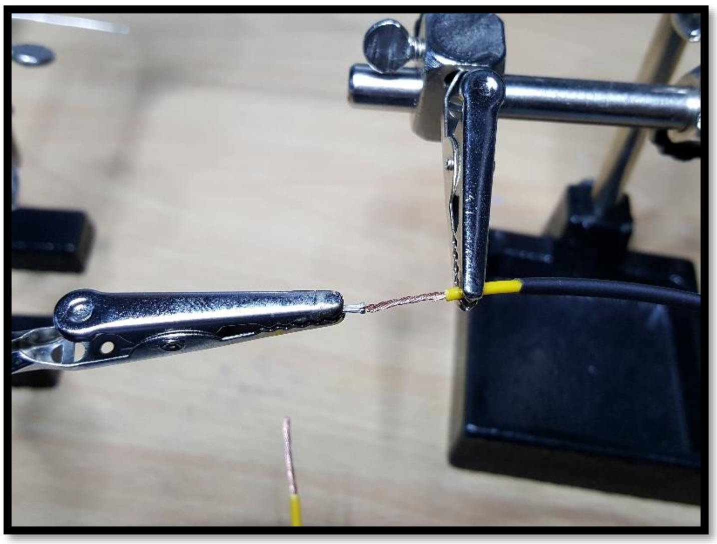


Prepared Wire

2. Solder male header – A male header that fits into a bread board was fitted to each end.

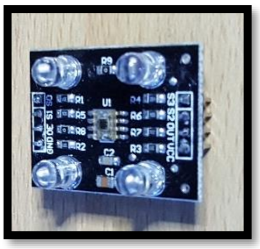
Male header

This not only mean I can plug them in to most beard boards (like the one located on top of the robot), but also it is easier to solder the other end to the sensor.



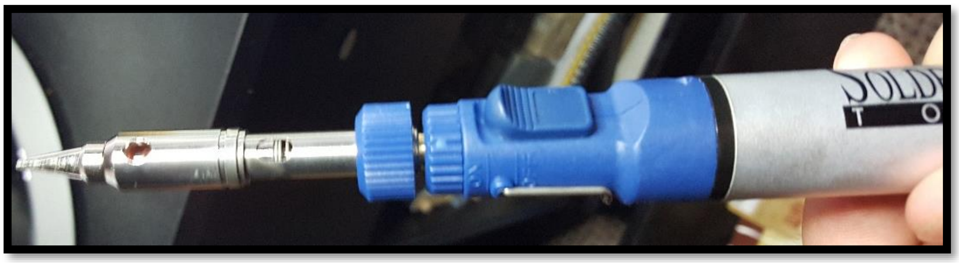
Male header ready for soldering

3. Solder to sensor – Solder one end of the wire to each of the pins on the sensor.



Colour sensor

4. Sheath – by applying heat from my gas powered soldering iron I can shrink the plastic sheath to help protect and secure the wire.



Soldering iron used.

5.2.2 CODE

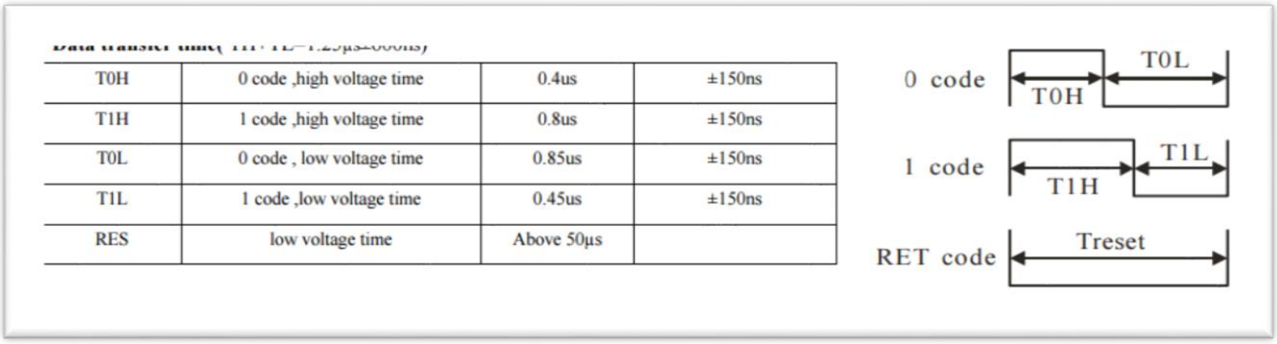
The code for the sensor was put into a function, this function takes an integer that will decide the colour that is to be read. The function will then set the correct pins to high or low in order to select the colour to read. Finally, the function will send a high pulse to the output pin, this “**pulse_in**” function will return the amount of time in micro seconds that the pin took to return to low, this value is returned as an integer. The function will then return that time.

Below is the pseudo code.

```
Int Check Colour ( int colour selection)

{

        Switch to determin colour

        Set pin 7 and 9 a combination of high and low depending on switch

        Return Pulse_in()

}
```

5.3 ROBOTS LEDS

These are the LEDs (part code WS2812B) that are attached to the rear of the robot. They are used to change colour when the robot finds a goal object. In order for these to work a driver need to be written for them, they work on a one data line idea that creates a command from ones and zeros, these are input by placing the data line high for a set amount of time then low for a set amount of time, varying the time will vary the input. Please see below for a chart of timing from the data sheet.


Data transfer time(TH+TL=1.25µs±600ns)

T0H	0 code ,high voltage time	0.4us	±150ns
T1H	1 code ,high voltage time	0.8us	±150ns
T0L	0 code , low voltage time	0.85us	±150ns
T1L	1 code ,low voltage time	0.45us	±150ns
RES	low voltage time	Above 50µs	

0 code
T0H
T0L
1 code
T1H
T1L
RET code
Treset

Initially created my own library for this task, taking the processer speed and calculating when to send data and how often, but after some limited successes I decided that this was not a good use of time and searched for a pre-built library.

Please note that all credit for this library goes to its creator David Betz.

Please see appendix D for WS2812B library.

5.4 GOAL LEDS

The goal object is a plastic box that contains an Arduino and a strip of three neopixels. The neo-pixels have an Arduino compatible library that was used, this is provided by Adafruit (Adafruit, 2016).

The library is simple to used, initialise and instantiate a neopixels object.

Adafruit_NeoPixel strip = Adafruit_NeoPixel(3, PIN, NEO_GRB + NEO_KHZ800);

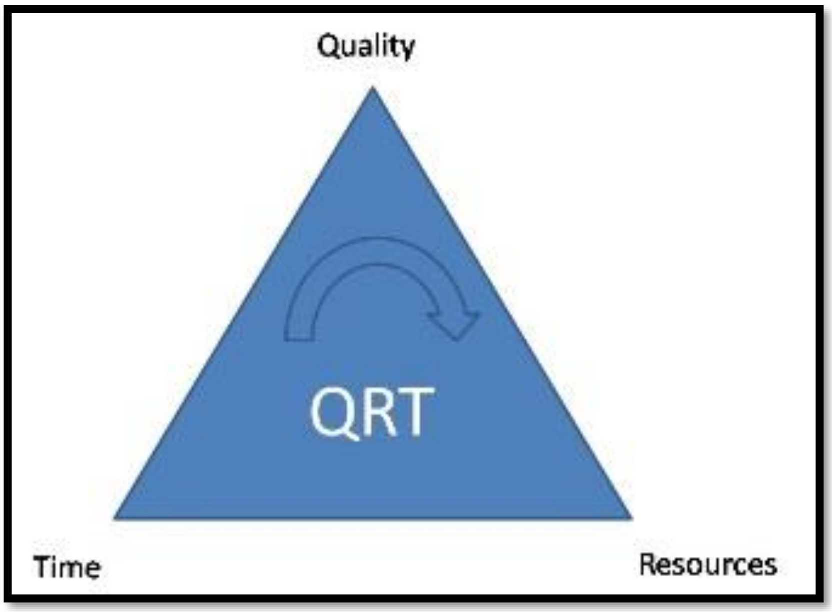
Then strip has to be started by using ***strip.begine().***

The strip object then can to be given the LED number and the colour, this is accomplished by used the ***strip.setPixelColor(0, 255, 0, 0).*** The first parameter is the LED number in the LED strip and the next 3 are the amount of red, blue and green to send to that LED. Finally, call the ***strip.show(),*** to turn on the LEDs.

The LED strip was affixed to the plastic container with an engineer's best friend, sticky tape.

CHAPTER 6 PROJECT MANAGEMENT CONSIDERATIONS

The basic structure for any project comes down to three key points: quality, time, and resources. These three considerations make up a good structure for any project. A project manager must find a good balance between these, making sure that the overall quality of the project and its outcome is high but understanding that this will come at the cost of time and money. In this project money is not necessarily a concern, however quality and time certainly are. The image below shows this balance.



(basic principles of effective project management, 2010)

This structure can also be found in *Project Management for Information Systems* (James Cadle, 2008, p. 209)

6.1 PROCESSES AND METHODOLOGY

In this project, an agile methodology was used in conjunction with a website called trello. Tasks were placed on the trello board and then designed, built and tested. There was also a physical reminder of what had to be done via the office whiteboard. Although this might seem a trivial means of project management, having a physical reminder of what work is outstanding and what has been achieved is a great motivational tool.

6.2 PROJECT SUCCESS AND FAILURE CRITERIA

From the results of a study, Shenhar argues that there are four main criteria for evaluating success; "Efficiency…Customer Impact/satisfaction…Business/Direct success…Future potential" (Shenhar, 1997) .

- Efficiency in a project can be measured by achieving the schedule and budget in the quickest and most cost saving way; this will be very important to project success as it ultimately means that the project is either on time or early, and on budget or cheaper.
- Customer impact/satisfaction is a common way of evaluating a project. It is in some ways the most tactile response to a project as the project should be receiving constant feedback from customers to see if it is fit for purpose.
- Business/Direct success is not always as directly relevant as yield or the project in terms of money. This can sometimes be defined as how many people it has educated or time saved on other projects, however it is still very important to measure this output of the project. All projects will have a purpose or goal to achieve and measuring the outcome of that goal is by definition measuring its success.

- Future potential is a harder one to measure. Mantel et al describe this dimension as “more difficult to assess” and “includes establishing a presence in a new market, developing a new technology, and such.” (Mantel, 2001, p. 239) This can be hard, as evaluating a project based on speculation or events that are to come is never an exact science. This is still important though, as this will measure where the project can be taken, if anywhere, and if this could be adapted to fit another similar project. This might not be critical to the client’s idea of success but a project can be more than just meeting a client’s needs.

On the other side of appraising projects, Harrison says that failure can be caused by “a lack of clear definition of organisation structure” (Harrison, 1981, p. 4) which leads to “delays and over-expenditure because of inadequate planning and good control systems.” (Harrison, 1981, p. 5). That is why in this project clear aim and objects have been state and the root to take to achieve them.

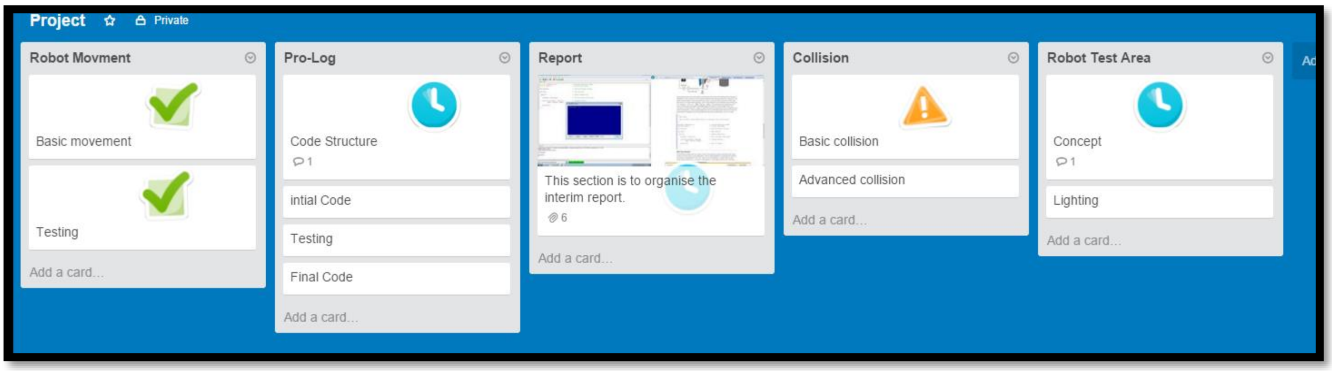
6.3 TIME MANAGEMENT

Moving on from the initial report, there were not only changes to the time plan but also to the design and concepts behind the project; for this reason, a new time plan has been drawn up for the final ‘sprint’ to the end that will be up to date with the current progress of the project. Please see appendix H for interim time plan.

6.4 PLANNING AND ORGANISATION

To plan and organise this project better, Trello will be used for the day to day running. Trello is an online pin board in which tasks can be placed and then maintained via the website. The cards that are placed are able to be updated, have notes added and even pictures. The cards also offer work to be marked off, alarms to be set when deadlines are due and can be removed with ease. Having a system that not only tracks all your data online (so can be accessed anywhere) but also offers many ways to maintain and update your task is key to the success of this project.

Below is an example of tasks laid out on the board and a task that has been updated many times to reflect changes in the project.



This section is to organise the interim report.

RISK ASSESSMENT 6.5

At the mid stage of the project, some of the risks that were previously stated have become irrelevant; for example, basic robot movement has been completed and is now no longer a risk, and so a new risk assessment has been drawn up to encounter early the possible pitfalls and traps still anticipated in the project, as well as new risks that might have arisen.

See the old risk assessment in appendix I and appendix J for the interim risk assessment.

CHAPTER 7 TESTING

The test that are listen in the next section were the core sections of hardware of code, these tests helped keep on top of errors that were occurring that could have help up the project in a bit way. Some of these tests on the sections passed first time but it was still worth recording this as a check list of what has been accomplished.

Date:	19-Nov-15				
Test Name and Description	Minimum Expectation	Normal Expectation	Side Effect	Areas of Improvement	Note's
Prolog Socket	No	No	N/A	No connection	Socket not connecting to computer in darels office.
Date:	19-Nov-15				
Test Name and Description	Minimum Expectation	Normal Expectation	Side Effect	Areas of Improvement	Note's
Prolog Socket	No	No	N/A	No connection	Server socket not connecting to robot.
Date:	26-Nov-15				
Test Name and Description	Minimum Expectation	Normal Expectation	Side Effect	Areas of Improvement	Note's
Prolog Socket	No	No	N/A	Connection made but no return from robot	Code to pick up response not complete, hanging on sending data.
Date:	26-Nov-15				
Test Name and Description	Minimum Expectation	Normal Expectation	Side Effect	Areas of Improvement	Note's
Prolog Socket	Yes	No	N/A	Data is being sent but not what was expected to be sent.	Check output using ascii converterw, check symbols being sent.
Date:	03-Dec-15				
Test Name and Description	Minimum Expectation	Normal Expectation	Side Effect	Areas of Improvement	Note's
Prolog Socket	Yes	Yes	N/A	Packet manager	
Date:	10-Dec-15				
Test Name and Description	Minimum Expectation	Normal Expectation	Side Effect	Areas of Improvement	Note's

Packet Manager	No	No	Does not unify and always returns one.	None	A variable might need to be set rather than trying to unify with the direct value.
Date:	10-Dec-15				
Test Name and Description	Minimum Expectation	Normal Expectation	Side Effect	Areas of Improvement	Note's
Packet Manager	Yes	Yes	None	None	None
Date:	26-Nov-15				
Test Name and Description	Minimum Expectation	Normal Expectation	Side Effect	Areas of Improvement	Note's
State selection	Yes	Yes			This function is in complete working order. All function can now be selected using the prolog commands on the central computer.
Date:	26-Nov-15				
Test Name and Description	Minimum Expectation	Normal Expectation	Side Effect	Areas of Improvement	Note's
Ping sensor	Yes	Yes			This uses an inbuilt funtion, so no surprise that it works first time.
Date:	24-Mar-16				
Test Name and Description	Minimum Expectation	Normal Expectation	Side Effect	Areas of Improvement	Note's
Colour sensor	No	No	None	No power to clean light LEDs	check driver
Date:	24-Mar-16				
Test Name and Description	Minimum Expectation	Normal Expectation	Side Effect	Areas of Improvement	Note's

Colour sensor	No	No	None	No power to clean light LEDs	Code is ok, check connection.
Date:	24-Mar-16				
Test Name and Description	Minimum Expectation	Normal Expectation	Side Effect	Areas of Improvement	Note's
Colour sensor	No	No	None	No return from output line.	check if code is sending active low or high.
Date:	24-Mar-16				
Test Name and Description	Minimum Expectation	Normal Expectation	Side Effect	Areas of Improvement	Note's
Colour sensor	Yes	Yes	None		
Date:	26-Nov-15				
Test Name and Description	Minimum Expectation	Normal Expectation	Side Effect	Areas of Improvement	Note's
Robot communication	No	No		data being send is not the same as being received.	This might be a prolog issues that is sending the wroung data, check packet manager.
Date:	26-Nov-15				
Test Name and Description	Minimum Expectation	Normal Expectation	Side Effect	Areas of Improvement	Note's
Robot communication	Yes	Yes			Prolog packet manager was always sending 1.

CHAPTER 8 EXPERIMENTS OUTCOME

This section will look at the data from the experiments that were run. From this data we will be able to see if there was a correlation between adding more rules and time taken to complete the task or even if there was not behaviour that emerged that complete the task.

During the first tests it was discovered that for this to get realistically timed results the goal object should be placed on a track so it can only move backwards, and so to guiding railed were placed either side to make its direction of movement forward or back.

8.1 DATA

Below is the raw data from the experiments.

	Basic Rule Set
2 Ro-bots	20.01
3 Ro-bots	11.52
	Robot Collision Rule Set
2 Ro-bots	11.59
3 Ro-bots	11.51
	Robot Maze Movement Rule Set
2 Ro-bots	11.57
3 Ro-bots	11.07
	Stigmergy Rule Set
2 Ro-bots	8.05
3 Ro-bots	6.01
	Colour Seek Rule Set
2 Ro-bots	8.18
3 Ro-bots	6.017

	Basic Rule Set
2 Ro-bots	9.13
3 Ro-bots	8.04
	Robot Collision Rule Set
2 Ro-bots	9.15
3 Ro-bots	8.01
	Robot Maze Movement Rule Set
2 Ro-bots	8.13
3 Ro-bots	8.04
	Stigmergy Rule Set
2 Ro-bots	5.46
3 Ro-bots	5.02
	Colour Seek Rule Set
2 Ro-bots	5.32
3 Ro-bots	4.59

	Basic Rule Set	Avg.
2 Ro-bots	11.01	13.38
3 Ro-bots	10.46	10.01
	Robot Collision Rule Set	
2 Ro-bots	11.24	10.66
3 Ro-bots	10.34	9.95
	Robot Maze Movement Rule Set	
2 Ro-bots	8.53	9.41
3 Ro-bots	7.47	8.86
	Stigmergy Rule Set	
2 Ro-bots	9.38	7.63
3 Ro-bots	7.02	6.02
	Colour Seek Rule Set	
2 Ro-bots	4.12	5.87
3 Ro-bots	4.03	4.88

	Basic Rule Set
2 Robots	15.56
3 Robots	16.07
	Robot Collision Rule Set
2 Robots	15.56
3 Robots	16.03
	Robot Maze Movement Rule Set
2 Robots	12.02
3 Robots	11.57
	Stigmergy Rule Set
2 Robots	11.3
3 Robots	11.02
	Colour Seek Rule Set
2 Robots	10.55
3 Robots	10.54

	Basic Rule Set
2 Robots	15.25
3 Robots	14.48
	Robot Collision Rule Set
2 Robots	15.35
3 Robots	14.26
	Robot Maze Movement Rule Set
2 Robots	14.03
3 Robots	16.34
	Stigmergy Rule Set
2 Robots	10.49
3 Robots	9.36
	Colour Seek Rule Set
2 Robots	9.34
3 Robots	9.01

	Basic Rule Set	
2 Robots	13.23	14.68
3 Robots	13.54	14.70
	Robot Collision Rule Set	
2 Robots	13.25	14.72
3 Robots	13.53	14.61
	Robot Maze Movement Rule Set	
2 Robots	9.04	11.70
3 Robots	8.42	12.11
	Stigmergy Rule Set	
2 Robots	7.45	9.75
3 Robots	9.05	9.81
	Colour Seek Rule Set	
2 Robots	6.09	8.66
3 Robots	6.09	8.55

8.2 ANALYSIS

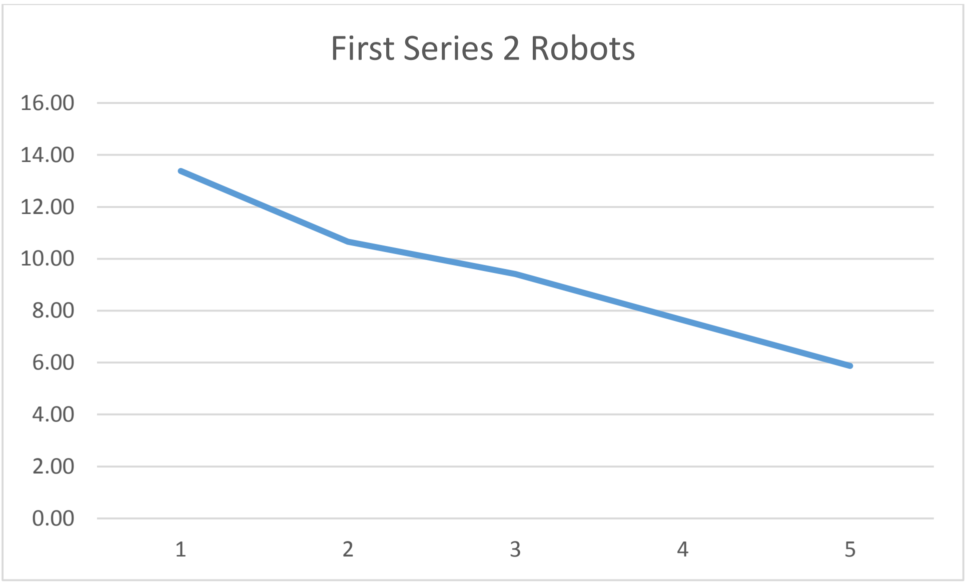
This data set is rather good and gives us a good picture of this project. This analysis can be broken in to a few sections to help narrow the focus.

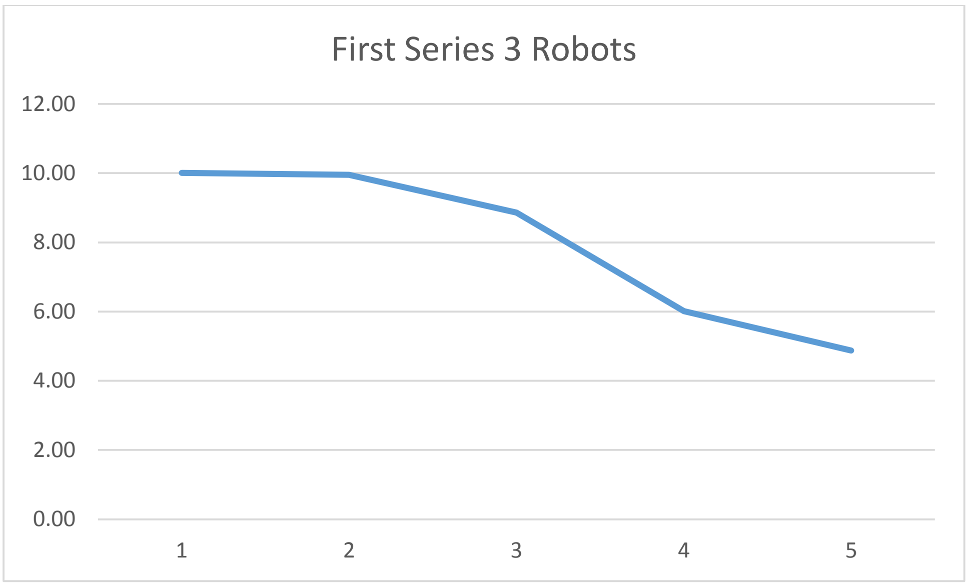
8.2.1 EMERGENT BEHAVIOUR

From this data we can see that all tests returned a results but some of these results are so long that they could almost be considered to have failed. The results that took a long time 13 mins plus could have some coincidently completed the task and it could have been more to do with time that any behaviour emerging, but never the less there is at least some result.

8.2.2 ADDING RULES

The graph below represents the data for the first series of tests.

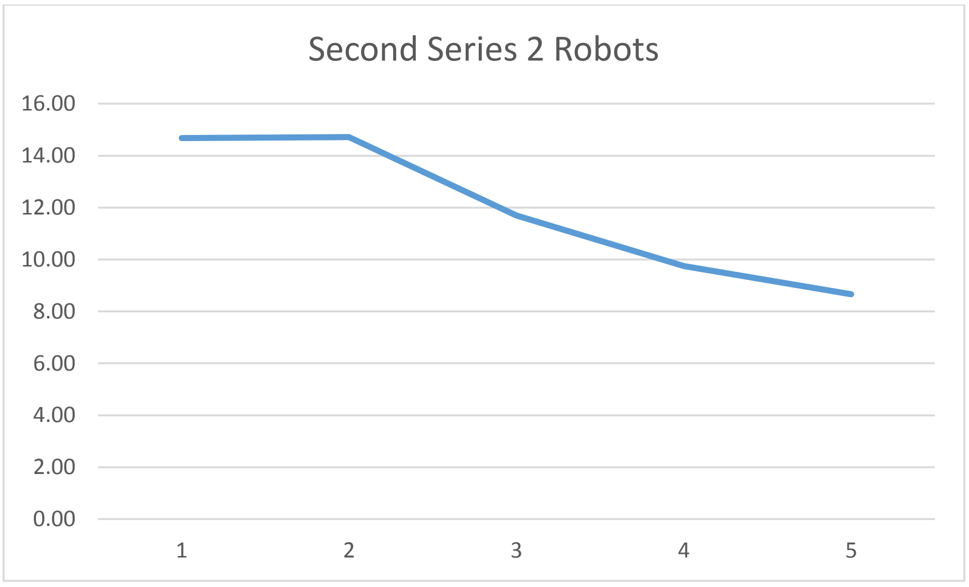


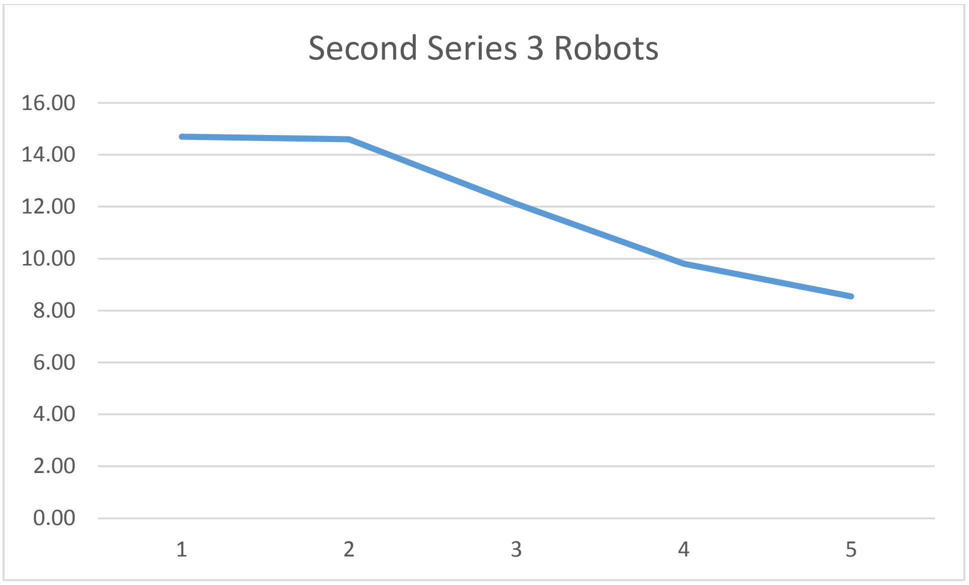


From these graphs we can see that large downward trend in the data as more rules are added. The second graph does not start to dip until the second rule is in place although this is not the case with the first graph. As the second rule deals with collisions with other robots it makes senses the first graph, which is 2 robots, is not as affected by a collision rule.

Both the first and second graph show that after the 3rd rule is added (Mazing movement) have a more significant drop. This might indicate to us that the moment rule is the most important in this experiment as this has the largest impact

The graph below represents the data for the second series of tests.

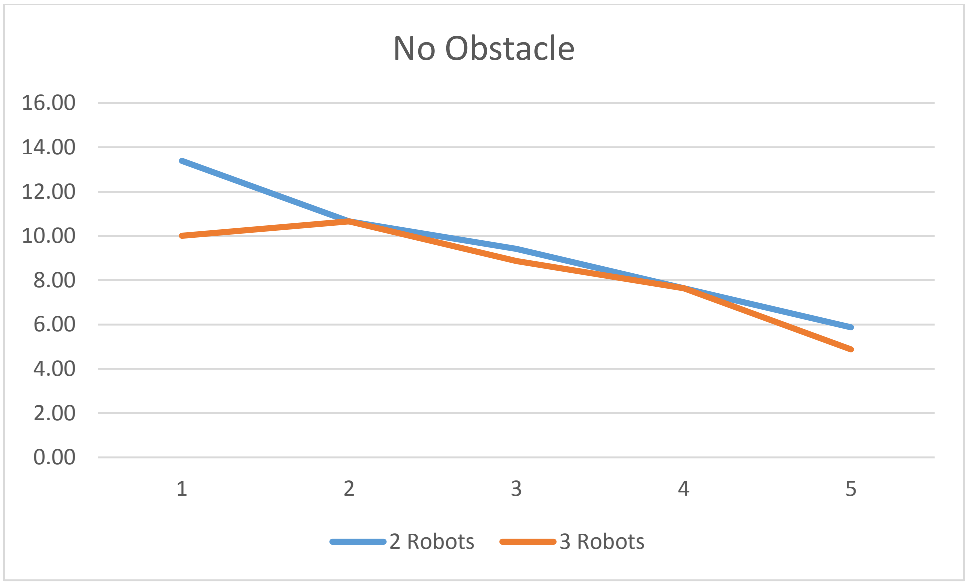




From both of these graphs we can see that the collision rule makes little to no difference and that the movement rule 3 and Stigmergy rule 4 states and maintains a significant downward trend. This could be because with a large obstacle in the middle of the robot's environment the robots are less likely to collide with each other and more likely to collide with the walls.

8.2.3 2 OR 3 ROBOTS

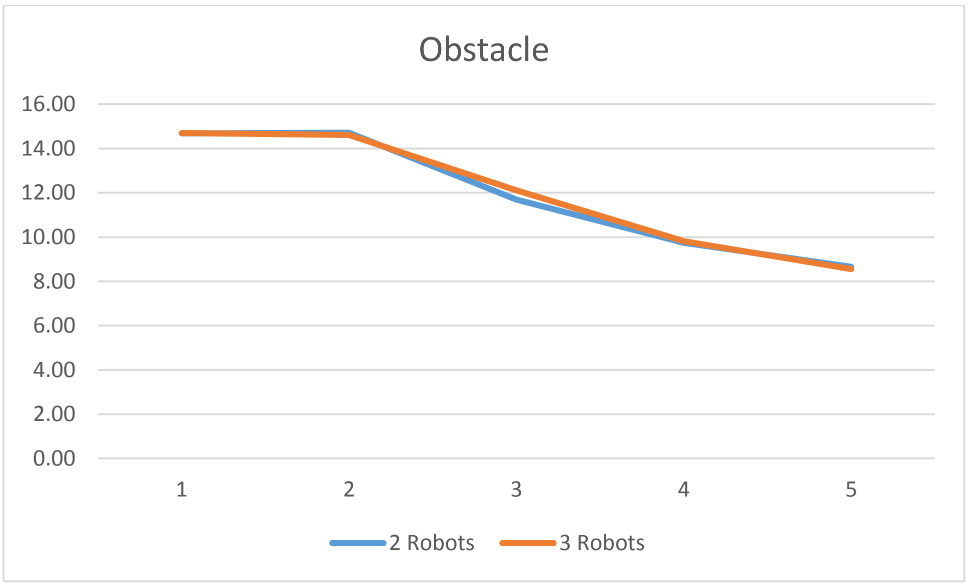
This next set of results compares the data form two or three robots; the data is form the first series.



From this graph it is clear that the time taken for 3 robots starts a lot lower than for 2 but, interestingly, this is evened out with the addition of the second rule (dealing with robot collision). After this rule the data starts to become closer together indicating that the rules are a larger factor here than the amount of robots, this is undoubtedly a pleasing result and does give so indication that we can produce goal orientated behaviour.

8.2.4 ADDING OBSTACLE

The last set of data to analyse is adding in an obstacle to the robot's environment.



The trend of this data is very similar to the previous set with the exception that all time is increased, this is somewhat predictable as there is a large box in the middle of the environment stopping the robots freely roaming.

One of the predication that was made in an earlier section was that the more complex environment will see a steeper trend and looking back on the previous graph there is some evidence for that indicating that more complex environment defines a larger significant to the rule sets. This is however not large difference between the graphs and certainly less than predicted.

8.3 OVERALL

Looking at the whole data set there is good evidence for some kind of behaviour emerging form basic rules and looking at the tread it gives a good indicator to which rules are important to this type of simple behaviour.

CHAPTER 9 EVALUATION

This section will address the project as a whole, looking at the achievements, development and further work.

9.1 PROJECT ACHIEVEMENTS

In this project there has been the main achievement of completing the stated goal and aims and objectives as well as running a project.

The first achievement is that of the project completion, this report has managed to detail data pertaining to emergent behaviour as well as complete all the aims and objectives. The data that was discovered could have more samples or slight changes but the data that has been presented indicates a successful project.

The second main achievement is that of completing the project its self. Although this was not as such a stated aim taking a project from its infancy to completion at this level is arguably not an easy feat. Therefor the upkeep, commitment, research and many other skills needed to complete this project, is a significant achievement in and of its self.

9.2 PERSONAL DEVELOPMENT

During this project I have gained many skills. Some in the form of technical achievements such as learning prolog or sharping my C skills and also have had to develop project skills such as work management and self-motivation.

These might seem like trivial skills but I have learned as the project progressed that they are important skills if a project is going to succeed. Being able to look at a trello board or white board to gain a sense of completion of the project or to organise myself, day by day really helped maintain a constant work flow.

9.3 FURTHER WORK

If this project was going to be continued, then there are a few areas that it could go down. It is important to note that these ideas are not just different angle of the same project but direct continuations of this work, there would be lots of solutions to list if we were to look at the swarming logic as a whole.

Please find below the further ideas for this project:

- Further development of rule sets – Taking the fact that not including the base rule set, there were four rules. A continuation could be to further try and reduce the time taken for the experiments by adding in more rules or different type of rule. The project could separate the rule in to categories and see which type of rules sets work the best and even combine them to fine the overall best rule. The project in its current state would also lend its self well to this as all the commination work and set up has been completed leave only development of new rules and a simple addition to the prolog script.
- Different goal objective – In this project a box was used as the goal objective, another route for this project could be to make this more dynamic, such as a moving object or another robot even. This would mean a re-test of the current rule set and then changes to be made accordingly. The real insight to that project could be observing which rule sets need to change and maybe trying to understand to correlation between those changes and the change of the goal objective.

CHAPTER 10 CONCLUSION

After some time reflecting in this project I can conclude that form a technical point of view it is entirely possible for robots to be given simple rules and new and predicted behaviours to appear. This is a great encouragement for the future of robots and what it could mean in the future, dreams like Nano robotics could be just round the corner with fields like this blooming.

The results that I good form this project have also given a real insight into the type of rules that need to be considered for simple behaviours. Further work in this field could yield some impressive results and maybe in the future this will be revisited.

Looking at this project from a personal point of view it has been a tough but rewarding processes that has taught me a lot about how to run a larger project and how to organise myself to get some results.

In finishing, this project has been a great successes and the learning and skills developed during this process have been invaluable for now and the foreseeable future.

CHAPTER 11 GLOSSARY

decentralised behaviour – Local entities of a component act on their own information to accomplish global goals.

collective behaviour – The actions of the sum of a swarm.

collective intelligence – The final outcome and ideas of a sum.

positive feedback – rewarding an animal to shape its actions.

negative feedback - reprimanding an animal to shape its actions.

antennation - An act of touching the antennae to another insect.

trophallaxis - Transfer of food or other fluids among members of a community through mouth-to-mouth (stomodeal) or anus-to-mouth (proctodeal) feeding.

Object Oriented – A programming paradigm in which code is separated in the object. The 'objects' form an almost 'Lego like' structure of code, with object of code being placed together to create a programme. Example, in a programme for a simple bank you might want a customer object, an account object and a bank object. These would contain the information for the customers, their account and the banks details respectively.

CHAPTER 12 APPENDIX

12.1 APPENDIX A

Objective 2 – Develop mapping algorithm

Developing a mapping algorithm will involve reading sonar sensor data to understand if a robot has found a wall and then using some greyscale or pattern recognition to understand which wall it is facing. Sensor data will be sent to the main computer; here, work will be done to understand its environment.

When the main computer understands the location of a robot, it will map it to a grid and from that be able to track the position of each robot. This position tracking will not be used to directly control the robots, as that is not in keeping with the Gene/Machine principle that states the gene is a simple set of instruction that the machine then follows; they do not directly interact. The sensor information will be used to better inform the instruction or “instincts” given back to the main computer. In this case, the robot will not implicitly understand its surroundings but rather deal with each instance of an object it finds and then makes decisions based upon its pre-programed behaviors.

12.2 APPENDIX A – 2

Objective 5 – Swarm to cooperate to move object

This is the main feature of the swarm: to move an object too heavy for one robot to a goal via self-organization. When the robot has isolated the object, it will then go about trying to push the object; the robot will line up with the object and start to push. If the robot can't move and therefore can't move the object, it will stop and try again at a set time. After a small amount of time, enough robots will have found the goal object and have begun pushing at the same time; this will move the object to the desired place in the testing area. This idea of the robots communicating via the change in the environment is called "Stigmergy" (Eric Bonabeau, 1999) and will be key in creating a swarm that reacts to its surroundings.

12.3 APPENDIX B

Code given by Dr Davis;

```
% Connection
robot_id(swarmV, '192.168.0.14', 2000).
robot_connect(Robot, Socket, Read, Write):-
        robot_id(Robot, Address, Port),
        tcp_socket(Socket),
        tcp_connect(Socket, Address:Port, Read, Write), !,
        clear_connect_buffer(Robot, Read, 7),
        !.
robot_connect(Robot, _Socket, _Read, _Write):-
        !,
        write('Failed to connect: '), writeln(Robot).

send_from_bot(Robot, Message):-
        connection(Robot, _S, R, _W),
        get_byte(R, Message), !,
        write('Robot sent: '), writeln(Message),
        !.

%       flush_output(R).
send_from_bot(Robot, Message):-
        write('** Message from Bot failed'),
        write(Robot),
        write(' Message: '), writeln(Message),
        !.

fdserial* pc_serial;
fdserial* ext_serial;

  pc_serial = fdserial_open(31, 30, 0, 115200); // Enable serial port out to monitor
  ext_serial = fdserial_open(6, 5, 0, 115200);  // Enable serial port for wifi
// fdserial_rxReady
// fdserial_txFlush

void initialise()
{
  fdserial_txFlush (ext_serial);
}
```

12.4 APPENDIX C

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Net.Sockets;
using System.IO;

namespace Prolog_Server
{
    class Program
    {
        static void Main(string[] args)
        {

            TcpListener server = new TcpListener(2000);

            server.Start();

            TcpClient clientSocket = server.AcceptTcpClient();

            NetworkStream strmServer = clientSocket.GetStream();

            StreamWriter strmOut = new StreamWriter(strmServer);

            StreamReader strmIn = new StreamReader(strmServer);

            while (true)
            {

                Console.WriteLine("Please press send...");

                Console.ReadLine();

                string strTest = "hello";

                strmOut.WriteLine(strTest);

                strmOut.Flush();

            }
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Net.Sockets;
using System.IO;


namespace PrologClient
{
    class Program
    {
        static void Main(string[] args)
        {

            TcpClient client = new TcpClient("127.0.0.1", 2000);

            NetworkStream strmClient = client.GetStream();

            StreamWriter strmOut = new StreamWriter(strmClient);

            StreamReader strmIn = new StreamReader(strmClient);

            string messageIn = strmIn.ReadToEnd().ToString();

            Console.ReadLine();

        }
    }
}
```

12.5 APPENDIX D

11.5 Appendix D

```
#include <propeller.h>
#include "ws2812.h"
#ifndef TRUE
#define TRUE    1
#define FALSE   0
#endif
// driver header structure
typedef struct {
    uint32_t    jmp_inst;
    uint32_t    resettix;
    uint32_t    bit0hi;
    uint32_t    bit0lo;
    uint32_t    bit1hi;
    uint32_t    bit1lo;
    uint32_t    swaprg;
} ws2812_hdr;
int ws2812_init(ws2812_t *state)
{
    return ws_init(state, 50, 350, 800, 700, 600, TYPE_GRB);
}

int ws2812b_init(ws2812_t *state)
{
    return ws_init(state, 50, 350, 900, 900, 350, TYPE_GRB);
}
// -- usreset is reset timing (us)
// -- ns0h is 0-bit high timing (ns)
// -- ns0l is 0-bit low timing (ns)
// -- ns1h is 1-bit high timing (ns)
// -- ns1l is 1-bit low timing (ns)
// -- type is TYPE_GRB for ws2812 or ws2812b
int ws_init(ws2812_t *state, int usreset, int ns0h, int ns0l, int ns1h, int ns1l, int
type)
{
    extern uint32_t binary_ws2812_driver_dat_start[];
    ws2812_hdr *hdr = (ws2812_hdr *)binary_ws2812_driver_dat_start;
    uint32_t ustix;

    ustix = CLKFREQ / 1000000;          // ticks in 1us

    hdr->resettix = ustix * usreset;
    hdr->bit0hi   = ustix * ns0h / 1000;
    hdr->bit0lo   = ustix * ns0l / 1000;
    hdr->bit1hi   = ustix * ns1h / 1000;
    hdr->bit1lo   = ustix * ns1l / 1000;
    hdr->swaprg   = (type == TYPE_GRB);

    state->command = 0;
    state->cog = cognew(hdr, &state->command);

    return state->cog;
}
void ws2812_close(ws2812_t *state)
{
    if (state->cog >= 0) {
        cogstop(state->cog);
        state->cog = -1;
```

```c
    }
}
void ws2812_refresh(ws2812_t *state, int pin, uint32_t *colors, int count)
{
    uint32_t cmd;
    cmd =  pin
        | ((count - 1) << 8)
        | ((uint32_t)colors << 16);
    while (state->command)
        ;
    state->command = cmd;
}
uint32_t ws2812_wheel(int pos)
{
    uint32_t color;

// Creates color from 0 to 255 position input
// -- colors transition r-g-b back to r

    // red range
    if (pos < 85)
        color = COLOR(255-pos*3, pos*3, 0);

    // green range
    else if (pos < 170) {
        pos -= 85;
        color = COLOR(0, 255-pos*3, pos*3);
    }

    // blue range
    else {
        pos -= 170;
        color = COLOR(pos*3, 0, 255-pos*3);
    }

    return color;
}
uint32_t ws2812_wheel_dim(int pos, int brightness)
{
    uint32_t color;

// Creates color from 0 to 255 position input
// -- colors transition r-g-b back to r

    // red range
    if (pos < 85)
        color = COLORX(255-pos*3, pos*3, 0, brightness);

    // green range
    else if (pos < 170) {
        pos -= 85;
        color = COLORX(0, 255-pos*3, pos*3, brightness);
    }

    // blue range
    else {
        pos -= 170;
        color = COLORX(pos*3, 0, 255-pos*3, brightness);
    }
    return color;

}
```

12.6 APPENDIX E

12.6.2 APPENDIX E-1

```
robot_connect(Robot, Socket, Read, Write):-

        robot_id(Robot, Address, Port),

        tcp_socket(Socket),

        tcp_connect(Socket, Address:Port, Read, Write), !,

        clear_connect_buffer(Robot, Read, 7),

        !.

robot_connect(Robot, _Socket, _Read, _Write):-

        !,

        write('Failed to connect: '), writeln(Robot).call_bot(swarmIV,shut_down).




clear_connect_buffer(Robot, _Read, 0):-

        write(Robot), writeln('  connected'),

        !.

clear_connect_buffer(Robot, Read, N):-

        get_byte(Read, Message2),

        write('Robot sent: '), writeln(Message2),

        M is N-1,

        clear_connect_buffer(Robot, Read, M).
```

12.6.3 APPENDIX E-2

```
%This is a packet tool to check that the stream is in sync.

packet(Robot,Value):-

        send_from_bot(Robot,Message),

        send_from_bot(Robot,Message),

        send_from_bot(Robot,Message),

        send_from_bot(Robot,Value).
```

12.6.4 APPENDIX E-3

```
% This is to after a collision is detected. This wil check what type of

% object that robot has detected.

collision(Robot):-

            call_robot(Robot,wall_check),

            packet(Robot,Value),

            writeln(Value),

            check(Value),

            call_robot(Robot,roam).




collision(Robot):-

            call_robot(Robot,robot_check),

            packet(Robot,Value),

            writeln(Value),

            check(Value),


            call_robot(Robot,roam).




collision(Robot):-

            call_robot(Robot,goal_check),

            packet(Robot,Value),

            writeln(Value),

            check(Value)

            call_robot(Robot,pus

h).

check(1).
```

12.6.5 APPENDIX E-4

```
send_to_bot(Robot, Message):-

            connection(Robot, _S, _R, W),

            put_byte(W, Message), !,

            flush_output(W).

send_to_bot(Robot, Message):-

            writeln('** Message to Bot failed:'-Robot-':Code:'-Message),

            !.


send_from_bot(Robot, Message):-

            connection(Robot, _S, R, _W),

            get_byte(R, Message), !,

            write('Robot sent: '), writeln(Message),

            !.


%           flush_output(R).

send_from_bot(Robot, Message):-

            write('** Message from Bot failed'),

            write(Robot),

            write(' Message: '), writeln(Message),
```

12.7 APPENDIX F

12.7.1 APPENDIX F-1

```
while ((fdserial_rxCheck(ext_serial)) < 0) {}; // needs connect to go
       dprint(pc_serial, "Ready.");
       // int fdserial_rxTime (fdserial *term, int ms)
       returnval = fdserial_rxTime(ext_serial, 50);
       fdserial_txFlush(ext_serial);

       // loop until receive dioconnect   ie ‘0’
       while ((returnval != 48)) // '0'
       {

              switch (returnval)
              {

              case 65:
                     halt();
                     break;

              case 67:
                     roam();
                     break;

              case 68:
                     sendBack();
                     break;

              case 71:
                     wall_check();
                     break;

              case 72:
                     robot_check();
                     break;

              case 73:
                     goal_check();
                     break;

              case 91:
                     push();
                     break;

              default:             default_behaviour();
              }
              //dprint(pc_serial,returnval);

              returnval = (fdserial_rxTime(ext_serial, 50));

       }
```

12.7.2 APPENDIX F-2

```
void roam()
{

       drive_setRampStep(10);                         // 10 ticks/sec / 20 ms

       drive_ramp(64, 64);                          // Forward 1 RPS

       // While disatance greater than or equal
       // to 20 cm, wait 5 ms & recheck.
       while (ping_cm(8) >= Gap) pause(5);           // Wait until object in range

       drive_ramp(0, 0);                              // Then stop

       fdserial_txChar(ext_serial, '-');
       fdserial_txChar(ext_serial, '-');
       fdserial_txChar(ext_serial, '-');
       fdserial_txChar(ext_serial, 81);
       fdserial_txFlush(ext_serial);


}
```

12.7.3 APPENDIX F-3

```
void wall_check()
{
	//set Green
	if (return_colour(2) > 700)
	{
		//set Red
		low(7);
		low(9);
		if (pulse_in(3, 0) > 700)
		{
			int pinger;
			pinger = ping_cm(8);
			pause(500);
			int pingerCheck;
			pingerCheck = ping_cm(8);
			int final = pinger - pingerCheck;
			dprint(pc_serial, "Final Value = %d\n", final);

			if ((final > 30) || (final < -30))
			{
				dprint(pc_serial, "Not Wall, Moving Object\n");

				//roamingMovment();

				fdserial_txChar(ext_serial, '-');
				fdserial_txChar(ext_serial, '-');
				fdserial_txChar(ext_serial, '-');
				fdserial_txChar(ext_serial, 0);
				fdserial_txFlush(ext_serial);

			}






			else
			{

				roamingMovment();
				//Seek();
	   //MazingMovment();

				fdserial_txChar(ext_serial, '-');
				fdserial_txChar(ext_serial, '-');
				fdserial_txChar(ext_serial, '-');
				fdserial_txChar(ext_serial, 1);
				fdserial_txFlush(ext_serial);

			}
		}
		else
		{
			dprint(pc_serial, "Red Light Deteced\n");
			fdserial_txChar(ext_serial, '-');
			fdserial_txChar(ext_serial, '-');
			fdserial_txChar(ext_serial, '-');
			fdserial_txChar(ext_serial, 0);
			fdserial_txFlush(ext_serial);
		}
```

```
        }
        else
        {
                dprint(pc_serial, "Green Light Deteced\n");
                fdserial_txChar(ext_serial, '-');
                fdserial_txChar(ext_serial, '-');
                fdserial_txChar(ext_serial, '-');
                fdserial_txChar(ext_serial, 0);
                fdserial_txFlush(ext_serial);
        }

}
```

12.7.4 APPENDIX F-4

```c
void robot_check()
{
	dprint(pc_serial, "RobotCheck\n");

	//set Red
	if (return_colour(3) < 500)
	{

		drive_speed(-32, 32);
		pause(1000);
		drive_speed(0, 0);
		if (ping_cm(8) > 25)
		{
			drive_speed(32, -32);
			pause(1000);
			drive_speed(0, 0);
			fdserial_txChar(ext_serial, '-');
			fdserial_txChar(ext_serial, '-');
			fdserial_txChar(ext_serial, '-');
			fdserial_txChar(ext_serial, 1);
			fdserial_txFlush(ext_serial);
		}
		else
		{

			drive_speed(32, -32);
			pause(1000);
			drive_speed(0, 0);
			fdserial_txChar(ext_serial, '-');
			fdserial_txChar(ext_serial, '-');
			fdserial_txChar(ext_serial, '-');
			fdserial_txChar(ext_serial, 0);
			fdserial_txFlush(ext_serial);

		}
	}
	else
	{

		fdserial_txChar(ext_serial, '-');
		fdserial_txChar(ext_serial, '-');
		fdserial_txChar(ext_serial, '-');
		fdserial_txChar(ext_serial, 0);
		fdserial_txFlush(ext_serial);

	}

}
```

12.7.5 APPENDIX F-5

```c
void goal_check()
{
	if(return_colour(2) < 600)
	{
		LightGreen();
		ws2812_refresh(&driver, LED_PIN, ledColors, LED_COUNT);
      push();
		fdserial_txChar(ext_serial, '-');
		fdserial_txChar(ext_serial, '-');
```

```
            fdserial_txChar(ext_serial, '-');
            fdserial_txChar(ext_serial, 1);
            fdserial_txFlush(ext_serial);
        }
        else
        {

            ws2812_refresh(&driver, LED_PIN, ledColors, LED_COUNT);

            fdserial_txChar(ext_serial, '-');
            fdserial_txChar(ext_serial, '-');
            fdserial_txChar(ext_serial, '-');
            fdserial_txChar(ext_serial, 0);
            fdserial_txFlush(ext_serial);

        }

    }
```

12.7.6 APPENDIX F-6

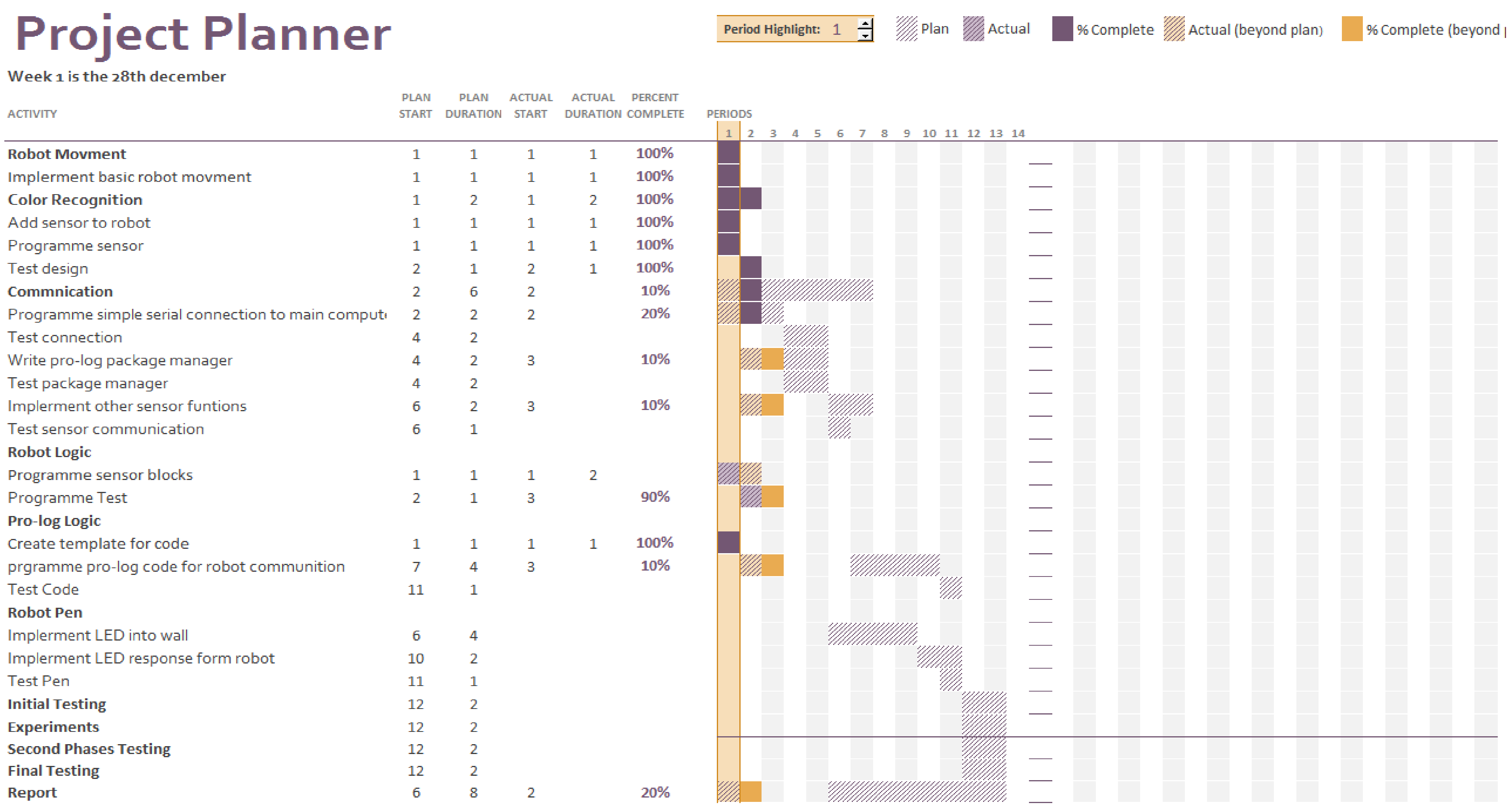
```
int return_colour(int Colour)
{

	switch(Colour){
	case 1 :
		low(7);
		high(9);
		return pulse_in(3, 0);
	case 2 :
		high(7);
		high(9);
		return pulse_in(3, 0);
	case 3 :
		low(7);
		low(9);
		return pulse_in(3, 0);
	}

}
```

12.8 APPENDIX G

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Ascii
{
    class Translate
    {

        public List<char> outputChars = new List<char>();

        public List<Exception> errors = new List<Exception>();

        public void ToAscii(string Translate)
        {

            outputChars.Clear();
            errors.Clear();

            string inputText = Translate;

            string[] inputWords = inputText.Split(',');

            foreach (string code in inputWords)
            {

                try
                {

                    int asciiCode = Int32.Parse(code);

                    if(asciiCode >= 0 && asciiCode <= 128)
                    {

                        char c = (char)asciiCode;

                        outputChars.Add(c);

                    }
                    else
                    {
                        throw new ArgumentException("Parameter must be between 0 and
128", "Input Text");
                    }
                }
                catch (Exception e)
                {
                    errors.Add(e);
                }

            }

        }

    }
}
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Ascii
{
    public partial class Form1 : Form
    {

        Translate translator = new Translate();

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }

        private void btnTranslate_Click(object sender, EventArgs e)
        {

            txbxOutput.Clear();

            translator.ToAscii(txbxInput.Text);
            foreach (char item in translator.outputChars)
            {

                txbxOutput.AppendText(item + " ");

            }

        }
    }
}
```

12.9 APPENDIX H



Project Planner

Period Highlight: 1

▨ Plan ▨ Actual ■ % Complete ▨ Actual (beyond plan) ■ % Complete (beyond

Week 1 is the 28th december

ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Robot Movment	1	1	1	1	100%
Implerment basic robot movment	1	1	1	1	100%
Color Recognition	1	2	1	2	100%
Add sensor to robot	1	1	1	1	100%
Programme sensor	1	1	1	1	100%
Test design	2	1	2	1	100%
Commnication	2	6	2		10%
Programme simple serial connection to main comput	2	2	2		20%
Test connection	4	2			
Write pro-log package manager	4	2	3		10%
Test package manager	4	2			
Implerment other sensor funtions	6	2	3		10%
Test sensor communication	6	1			
Robot Logic					
Programme sensor blocks	1	1	1	2	
Programme Test	2	1	3		90%
Pro-log Logic					
Create template for code	1	1	1	1	100%
prgramme pro-log code for robot communition	7	4	3		10%
Test Code	11	1			
Robot Pen					
Implerment LED into wall	6	4			
Implerment LED response form robot	10	2			
Test Pen	11	1			
Initial Testing	12	2			
Experiments	12	2			
Second Phases Testing	12	2			
Final Testing	12	2			
Report	6	8	2		20%

12.10 APPENDIX I

Risk	Severity (L/M/H)	Likelihood (L/M/H)	Significance (Sev. x Like.)	How to Avoid	How to Recover
Data loss	H	M	HM	Keep Backups	Reinstate from backups
Loss of backups	H	L	HL	Multiple Backups	Use alternate
Can't interface Prolog with Python	H	M	HM	Commit enough time to properly research and learn the Prolog Python interface.	Drop Prolog commands and implement stricter Python scripts
Camera is too powerful for robot board	L	M	LM	Check data sheet in proper time	Change Camera
Camera will not interface with propeller board	M	M	MM	Check data sheet in proper time	Change Camera or use inferred LED sensor.
Can't interface Python with OpenCV	H	H	HH	Commit enough time to properly research and learn the *OpenCV* interface.	use inferred LED sensor instead of a complex camera pattern recognition
Robot wireless communication will not work in lab	L	L	LL	Check internet security with university and attempt a connection	Use other connection device, (XBEE, Bluetooth)
Arduino C does not cross over to Propeller C	L	L	LL	Run test program written in C that runs on Arduino at the early stages of the project	Change robots to Arduino boards
Robot not set in test area (Human Danger)	L	L	LL	Only run robots in test area and close door out of lab	Analyze any damage and fix.
Robot damaged due to incorrect voltage supplied	L	L	LL	Check data sheet in proper time	Analyze any damage and fix.

12.11 APPENDIX J

Risk	Severity (L/M/H)	Likelihood (L/M/H)	Significance (Sev. x Like.)	How to Avoid	How to Recover
Data loss	H	M	HM	Keep Backups	Reinstate from backups
Loss of backups	H	L	HL	Multiple Backups	Use alternate
Can’t create prolog socket	H	L	HL	Commit enough time creating socket. Seek help earlier rather than later.	Drop Prolog complex command and start with simple commands.
Camera is too powerful for robot board	L	M	LM	Check data sheet in proper time	Change Camera
No resources to create LED Wall.	M	M	MM	Check early. And reaches source from home.	Drop this section of the project and revert back to using a grid system.
Packet manager too large for simple communication.	H	H	HH	Test with simple packet and build up.	Re-think packet system using a simpler code with less transfer.
Robot wireless communication will not work in lab	L	L	LL	Check internet security with university and attempt a connection	Use other connection device, (XBEE, Bluetooth)
Report not kept up to date.	H	L	HL	Keep track of day to day producing on trello board.	Backtrack through old code on SVN to start to build the picture back up.
Robot not set in test area (Human Danger)	L	L	LL	Only run robots in test area and close door out of lab	Analyze any damage and fix.

CHAPTER 13 BIBLIOGRAPHY

Adafruit. (2016, April 01). *https://www.adafruit.com/*. Retrieved from https://www.adafruit.com/: https://www.adafruit.com/

basic principles of effective project management. (2010, January 19). Retrieved from http://www.haukeborow.org: http://www.haukeborow.org/2010/01/7-basic-principles-of-effective-project-management/

BBC (Director). (2005). *Blue Planet* [Motion Picture].

Bramer, M. (2013). *Logic Programming with Prolog.* London: Springer.

Bratko, I. (2001). *Prolog Programming for artificial intelligence.* Harlow: Person Education Limited.

Changyun Wei, K. V. (n.d.). An Agent-Based Cognitive Robot Architecture. *An Agent-Based Cognitive Robot Architecture*.

Darkins, R. (1976). *The Selfish Gene.* Oxford: Oxford Press.

Darryl Davis, J. G. (2012). robo-CAMAL: A BDI Motivational Robot. *robo-CAMAL: A BDI Motivational Robot*, 1-30.

Dawkins, R. (1976). *The Selfish Gene* (30th Anniversary edition ed.). Oxford: Oxford University Press.

Eric Bonabeau, M. D. (1999). *Swarm Intelligence.* Oxford: Oxford press.

Harrison. (1981). *Advanced Project Management.* Hants: Gower publishing Company Limited.

James Cadle, D. Y. (2008). *Project Management for Information Systems.* Harlow: Pearson Education Limited.

James Kennedy, R. C. (2001). *Swarm intelligence.* Indianapolis: Morgan Kuafmann publishers INC.

Mantel, M. (2001). *Project management in practice.* New York: John Wiley and Son.

Michael Beetz, L. M. (2010). *A Cognitive Robot Abstract Machine.* Munchen: IEEE.

Moritz Tenorth, M. B. (2009). KNOWROB — Knowledge Processing for Autonomous Personal Robots. *KNOWROB — Knowledge Processing for Autonomous Personal Robots*.

Pascal Haazebroek, S. v. (2011). A computational model of perception and action for cognitive robotics. *A computational model of perception and action for cognitive robotics*.

Ruijiao Li, B. L.-M. (2015). Cognitive assisted living ambient system: a survey. *Cognitive assisted living ambient system: a survey*.

Sertac Karapinar, S. S. (2015). Cognitive Robots Learning Failure Contexts. *Cognitive Robots Learning Failure Contexts*.

Shenhar, A. (1997). Mapping the dimensions of Project success". *Project Management Journal*, 5-13.

ZDZISŁAW KOWALCZUK, M. C. (2011). INTELLIGENT DECISION–MAKING SYSTEM FOR AUTONOMOUS ROBOTS. *INTELLIGENT DECISION–MAKING SYSTEM FOR AUTONOMOUS ROBOTS*.