|  |
| --- |
|  |

|  |
| --- |
| Maze Mapping and Efficient Traversal using a Micromouse |
| An attempt at designing and programming an autonomous, mobile robot that can navigate the shortest path through a maze, efficiently and unassisted. |
| Jasper Cashmore |
| BSc (Hons) Computer Science  Staffordshire University  A project submitted in partial fulfilment of the award of the degree of BSc (Hons) Computer Science from Staffordshire University |
| Supervised by Rachel Cornes |
| May 2015 |



Table of Contents

[1. Research and Analysis 3](#_Toc418160778)

[1.1. MoSCoW Requirements 3](#_Toc418160779)

[1.2. Problem Specification - The Maze 3](#_Toc418160780)

[1.2.1. Outline 3](#_Toc418160781)

[1.2.2. Types of maze 3](#_Toc418160782)

[1.2.3. Maze MoSCoW Requirements 3](#_Toc418160783)

[1.2.4. Maze Terminology 3](#_Toc418160784)

[1.2.5. Suitable and Unsuitable Mazes 3](#_Toc418160785)

[1.2.6. Potential materials 3](#_Toc418160786)

[1.3. Solution – The Robot 3](#_Toc418160787)

[1.3.1. The Micromouse 3](#_Toc418160788)

[1.3.2. Birmingham City University Robotics Club 3](#_Toc418160789)

[1.3.3. Non-functional MoSCoW Requirements 3](#_Toc418160790)

[1.3.4. Functional MoSCoW Requirements 3](#_Toc418160791)

[1.3.5. Hardware Requirements 3](#_Toc418160792)

[1.3.6. Software Requirements - Language 3](#_Toc418160793)

[1.3.7. Obtaining the Robot 3](#_Toc418160794)

[1.4. Mapping and Pathfinding 3](#_Toc418160795)

[1.4.1. Perspective Experiment 3](#_Toc418160796)

[1.4.2. Maze Mapping and Analysis 3](#_Toc418160797)

[1.4.3. Pathfinding and Shortest Path Calculation 3](#_Toc418160798)

[1.5. Motor Control 3](#_Toc418160799)

[1.5.1. Centring 3](#_Toc418160800)

[1.5.2. Proportional-integral-derivative Motor Control 3](#_Toc418160801)

[1.6. Program Development 3](#_Toc418160802)

[1.6.1. Memory Management 3](#_Toc418160803)

[1.6.2. Memory Types 3](#_Toc418160804)

[1.6.3. Memory Leaks 3](#_Toc418160805)

[1.7. Debugging 3](#_Toc418160806)

[1.8. Testing 3](#_Toc418160807)

[1.8.1. Test Driven Development 3](#_Toc418160808)

[1.9. Project Tools 3](#_Toc418160809)

[1.9.1. Microsoft Windows 8.1 3](#_Toc418160810)

[1.9.2. Microsoft Office 2013 3](#_Toc418160811)

[1.9.3. Atmel Studio 6.2 3](#_Toc418160812)

[1.9.4. Sublime Text 3 3](#_Toc418160813)

[1.9.5. MinGW + GCC 3](#_Toc418160814)

[1.10. Methodology 3](#_Toc418160815)

[1.10.1. Traditional Methodologies 3](#_Toc418160816)

[1.10.2. Agile Methodologies 3](#_Toc418160817)

[1.10.3. Conclusion 3](#_Toc418160818)

[1.11. Version Control and Project Management 3](#_Toc418160819)

[1.12. Issue Tracking 3](#_Toc418160820)

[2. Design 3](#_Toc418160821)

[2.1. Physical Robot Design 3](#_Toc418160822)

[2.1.1. Soldering Components 3](#_Toc418160823)

[2.2. Maze Design 3](#_Toc418160824)

[2.3. Program Design 3](#_Toc418160825)

[2.4. Design with Development 3](#_Toc418160826)

[2.4.1. Development using Method Stubs 3](#_Toc418160827)

[2.4.2. Data Dictionary 3](#_Toc418160828)

[3. Testing 3](#_Toc418160829)

[3.1. Initial testing 3](#_Toc418160830)

[3.2. Unit Tests 3](#_Toc418160831)

[4. Critical Evaluation 3](#_Toc418160832)

[4.1. Summary 3](#_Toc418160833)

[4.2. Component Assembly 3](#_Toc418160834)

[4.2.1. Micrometal Gearmotors 3](#_Toc418160835)

[4.2.2. Digital Sensors 3](#_Toc418160836)

[4.2.3. Analog Sensors 3](#_Toc418160837)

[4.2.4. Decision to fall-back to a line maze 3](#_Toc418160838)

[4.2.5. Calibration 3](#_Toc418160839)

[4.3. Lack of standard library 3](#_Toc418160840)

[4.4. Memory Leaks 3](#_Toc418160841)

[4.5. Reflection 3](#_Toc418160842)

[5. Deliverables 3](#_Toc418160843)

[5.1. Deliverables 3](#_Toc418160844)

[6. Appendix 3](#_Toc418160845)

[6.1. Email Correspondence with (Harrison, P) 3](#_Toc418160846)

[6.2. Gantt Chart 3](#_Toc418160847)

[6.3. Pre-built Candidates and Assembly Kits 3](#_Toc418160848)

[6.4. Build or Buy Decision Matrix 3](#_Toc418160849)

[6.5. Parts List 3](#_Toc418160850)

[6.6. Digital Sensors’ Attachment 3](#_Toc418160851)

[6.7. Analog Sensors’ Attachment 3](#_Toc418160852)

[6.8. Cardboard Maze Design 3](#_Toc418160853)

[6.9. Component Testing 3](#_Toc418160854)

[6.10. Distance Trials Based on Time Taken 3](#_Toc418160855)

[6.11. Micromouse Competition 3](#_Toc418160856)

[6.12. Git Terminology 3](#_Toc418160857)

[6.13. Line Maze Design 3](#_Toc418160858)

[6.14. Lined Maze Implementation 3](#_Toc418160859)

[6.15. Expected STL Constructs That were not available 3](#_Toc418160860)

[6.16. Analog PID Tuning 3](#_Toc418160861)

[6.17. Logbook 3](#_Toc418160862)

Introduction and Background

Hobby robots that are design to solve mazes, both lined, and physically walled, has been a common pursuit of robot enthusiasts since the early 70s. However, solutions that not only solve mazes but solve them by traversing the shortest path, 100% of the time, are few. In addition, those that solve this problem this in addition to solving looped mazes are even fewer. The inspiration for this project is the challenge of solving both these problems.

The project aims to acquire a small, single-board robot and develop software that cases this robot to traverse the shortest path of a maze from start to finish. The project is divided into multiple phases, generally relative to their order of completion, but not necessarily, largely due to the project strategy.

Research and analysis is the first stage. Exact requirements and specifications will be formed for both the maze, and the robot. Hardware and software requirements will be evaluated, then attaining of the robot investigated. The entirety of the project’s development strategy, methodology and plan will be assessed and constructed. Finally, the development of the application that the robot will run will be analysed, scrutinised and concluded in full, referencing sources where necessary.

The second stage is design. During this stage, the robot’s design from software to hardware, if necessary, will be completed. This may include the possibility of component assembly in addition to the core application development. Both hardware and software unit tests will be created and the physical design of the maze must also be done during this phase.

In conjunction with development, implementation will entail assembling the robot, if required, building the maze and developing an application to meet the robot and maze requirements. This process will carry out instructions as documented in the design, during the same timeframe as the development, adapting to change if necessary, as described by the agile model in section 1.10.3. Exact software development techniques and strategy will be documented, prior to actual implementation beginning.

Development will be test driven and implementation should continue until all unit tests pass, signifying core functionality being met. The program may then be advanced in terms of efficiency, and extra functionality if time management permits, as planned in the project plan found in section 6.2 of the appendix.

Finally, project evaluation will be completed, analysing areas where the project was strong, and where it was weak, describing processes that could have been done differently, problems that arose and how they were dealt with.

By the end of the project plan, a fully implemented artefact and written report should have been developed and documented in accordance with BCS code of conduct and ethical expectations.

# Research and Analysis

## MoSCoW Requirements

The MoSCoW method (Clegg & Barker, 1994), is a technique commonly applied in software development to summarise requirement priorities. It breaks requirements down into those that MUST be satisfied, those that SHOULD be satisfied, COULD be satisfied and WOULD be satisfied given alternate circumstances.

‘Must’ are requirements that are vital to the project’s success and their completion of which is necessary. ‘Should’ are the requirements that should be attempted to complete and are typically critical but alternative solutions may be used as a compromise if not possible. ‘Could’ are requirements that can be sought if spare time and resources are available. Generally, these can be considered advancements and extensions of the core requirements. Their completion is not fundamental to the completion or success of the project. ‘Would’ are requirements that would otherwise be fulfilled, given the need, resources or differing use-case. These requirements are also known as “WON’T” as their completion has been decidedly objected to, i.e. “Will Not” implement stated requirements.

MoSCoW requirements will be used to describe the requirements of both the maze, and the robot.

## Problem Specification - The Maze

### Outline

Given a physical set of objects that form a maze-like structure, the robot needs to get from one position in the maze to another via the shortest route possible. The robot should not touch the maze’s walls, nor should it receive information about the maze’s structure from any external source, relying solely upon its own sensors and mechanisms to learn by itself. The robot’s goal is to travel the shortest path, but not necessarily travel it in the shortest time.

### Types of maze

There are two types of maze that would prove suitable for this project. These are a flat, lined maze or a physically walled maze. The former would be less complex but would be easier to build, whilst the latter would introduce significant complexity and provide an all-round difficult challenge but would likely be more costly and time-consuming. To plan and design for both would double all effort required, therefore only one of these options will be strived towards.

The decision made is that the physically walled maze will be attempted and all efforts will be focussed on this challenge. However, the lined maze should serve as a compromise should it be required.

Therefore, careful action should be taken to ensure analysis and design be as closely compatible as possible with both types of maze to make the fall-back to a lined maze a relatively straightforward process if necessary.

### Maze MoSCoW Requirements

#### Must

* Have all walls angled on 180° sides. i.e. No diagonal walls.
* Every wall’s length must be rationally divisible by the same number, e.g. a wall could be 10cm, 20cm, 30cm or 40cm long but not any other value that isn’t perfectly divisible by 10.
* Be possible to travel from the start point to the finish.
* Height of the maze must be greater than the height of the highest sensor from the ground.

#### Should

* Be modifiable to allow restructuring.
* Be closed, i.e. have no exit to the outside of the maze.
* The thickness of the path through the maze should always be the minimum length of a wall (10cm for example).
* Have a black surface as the base, to ensure interference with the IR sensors is minimized.

#### Could

* Have a start point and an end point that are identifiable by a sensor.

#### Would

* Meet all official Micromouse competition requirements. This is not possible due to lack of resources

### Maze Terminology

Throughout this document, the terms in Figure 1.1 below will be used to describe properties of the maze.

|  |  |
| --- | --- |
| Wall | A wall that is built of card that the robot can detect but cannot pass through |
| Non-wall | A side of a tile that is not a wall |
| Tile | A square section of the maze that has four sides: North, East, South and West. Any of these sides can be a wall or a non-wall |
| Dead-end | A tile whereby three of its four sides are walls |
| Loop | A collection of one or more joined sides that does not attach to the maze perimeter at either end, forming a cycle the robot could conceptually travel through infinitely |
| Start | The beginning point of the maze |
| Finish | The end point of the maze |
| Path | A journey from one tile to another |
| Route | Another term for path |

Figure 1.1

### Suitable and Unsuitable Mazes

Figure 1.2 below illustrates all possible valid and invalid maze structures. The solution to the problem need only attempt at solving valid mazes.

|  |  |
| --- | --- |
| **Valid Maze**  Start and Finish are accessible inside the maze | **Valid Maze**  Multiple routes to the finish with dead-ends |
| **Valid Maze**  Loops and many routes to finish | **Invalid Maze**  Has openings |
| **Invalid Maze**  No possible route to the finish | **Invalid Maze**  Varying thickness of the corridors |

Figure 1.2

### Potential materials

|  |  |  |
| --- | --- | --- |
| **Material** | **Advantages** | **Disadvantages** |
| LEGO® / LEGO® Duplo | Easy to setup and readjust | Expensive for the amount that would be needed |
| Cardboard | Cheap | Would require time spent to craft the maze |
| MDF/fibreboard | Good surface for reflecting infrared | Requires time spent crafting, costly |

Figure 1.3

## Solution – The Robot

### The Micromouse

Research into this project’s field revealed existing solutions to the problem in the form of single-board, minimal design robots known as “Micromice” designed for the purposes of competitions. These competitions officially began in the 1970’s and are commonly held in the UK, the US and various parts of Asia. See appendix item 6.11 for a photo of a typical competition.

The aim of a Micromouse competition is to reach the centre of a looped maze in the fastest time. The start and finish points are known, but the maze’s structure is not made aware to the entries. However, a short two minute period is available for the Micromice to explore and form virtual maps of the maze before their entry actually begins.

Some factors which play a part in an entry’s chance of winning are: the algorithm it uses to map the maze, the algorithm used to solve the maze, its hardware, such as motors, wheels and total weight. The Path finding algorithms used that will result in profitable routes are commonly known and widely available for the public, however (Willardson, 2001) states that the most efficient algorithms are kept secret.

The robot for this project will fit the criteria of a Micromouse, and thus, it will be known as such. However, the maze specification will not be the same as official Micromouse competition rules due to the monetary limitations. It is not a requirement, but if this project does produce a deliverable worthy of being entered into a Micromouse competition, it is possible it will be submitted for entry.

### Birmingham City University Robotics Club

During Micromouse research, (Harrison, 2005) of micromouseonline.com was contacted, outlining the goals of the project asking for advice on building or obtaining a robot and an invitation was made to visit Birmingham City University Robotics Club – see 2.4.

Attending the Robotics Club in February gave much clearer insight into steps needs to achieve the project’s goal. There was an opportunity to talk with Harrison and several other field experts in person about the project, providing knowledge of a suitable course of action to take, including a defined set of requirements the robot must have to achieve the goal.

Specifically, (Harrison, 2005) stated an adaption for the Pololu 3pi robot would likely be the best choice to meet this project’s goals. He also stated that using a pre-built robot to solve a looped maze may prove a more difficult challenge than predicted and that there is a lack of documentation about adapting a pre-built robot to solve a walled maze. He suggested contacting Pololu directly and seeking their advice as they would know this problem better.

### Non-functional MoSCoW Requirements

#### Must

* Run from a reliable, affordable source of power
* Be portable

#### Should

* Be efficient in reuse of code
* Be reliable

#### Could

* Be extensible for future modifications

#### Would

* Comply with official Micromouse competition rules

### Functional MoSCoW Requirements

#### Must

* Travel the shortest path
* Be able to move forwards whilst simultaneously performing other operations
* Be able to turn up to 180° whilst simultaneously performing other operations
* Receive data regarding the distance from the front and to the sides of the robot and be able to use this data in software
* Travel the route of the maze from start to finish without requiring manual assistance

#### Should

* Travel the shortest path within reasonable time, relative to the maze’s size
* Require a way to know if it has reached the finish point

#### Could

* Have varying levels of speed
* Be able to move backwards

#### Would

* Relay data to another device via Bluetooth
* Be able to travel diagonally if required – e.g. ‘staircase’ shaped sections

### Hardware Requirements

#### Microcontroller

The microcontroller is the core component and serves as the central hub for all the input and output connections. All listed components will connect to the microcontroller in some way. It will provide the CPU, memory, IO and will be the unit that receives instructions from its inputs, processes these and translates them into other instructions for the outputs to receive and obey. In the case of a Micromouse, when a button press signal is fed in, it may process a particular program in its memory and dictate to the motors what they should be currently doing.

Factors that will establish which microcontroller to purchase include the following:

* Flash memory capacity
* RAM capacity
* EEPROM capacity
* Clock speed
* Cost

Research shows that the majority of hobby robots use either ARM or Atmel-made architecture such as 8-bit AVR microcontrollers, the same as Arduino uses. The ARM/Atmel community is large, meaning advice and support are easy to obtain. The costs are much lower than Intel solutions and the nature of these companies is focussing on mass microcontroller production for embedded systems. It is like the most appropriate robot candidates will use an Atmel-made processor.

#### Sensors

##### Range Finding

Also known as distance sensors, these will determine how far away an obstacle is from the sensor. In a Micromouse, they will be useful for knowing how close the unit is to crashing, as well as calculating the positions of walls in the maze. Measuring exact distance requires more computation than simply checking for existence and power consumption may be greater. Distance sensors are therefore expectedly more costly than their counterpart. (E.R. Davies: Theory, 1990) Describes range finding sensors and existence sensors.

Infrared reflective intensity sensors are sensors that work by emitting IR to a surface and using phototransistors, a type of transistor that generates an electrical current from received light, to calculate the distance between the surface and the sensor.

A flaw with this type of sensor is that surfaces may have different degrees of reflection which can cause miscalculated distances if various materials were to be used for the maze’s walls. However, as the design of the maze, including the materials it will be made from, will be decided in the design phase, it will be possible to design the maze’s walls using a single material, hereby eliminating this problem. All IR sensors also suffer when used in conjunction with other IR ones as confusion is caused by detectors receiving beams from different emitters.

IR triangulation, another type of range finding, also emits a concentrated IR beam and receives the reflection. However, the difference is that here a type of position sensitive detector is used to extract the angle at which the IR beam is received. When calibrated correctly, this triangulation process allows the sensor to determine how far away the object is based on the degree of the angle.

Problems with this sensor stem from the complexity added by interpretation of the received beam. Unless directed straight forward at a perfect angle, the beam is unlikely to linearly travel directly back to the received after it bounces off the first obstacle. Even if it does bounce straight back, it is difficult to know which specific wall in the maze it bounced back from. Although possible, the added time consumption by researching and implementing this type of sensor is large. The cost is also significantly higher than other types of sensor.

Ultra-sonic sensors use sonar to detect the distance away from an obstacle. Similar to IR reflection, an ultra-sonic sensor emits a semi-focussed wave of ultrasound and based on the intensity of the sound reflected back into the ultra-sonic detector, and the time taken to receive the reflection, whether an object exists or not can be determined, as well as its distance from the emitter.

Unfortunately, these sensors also have complications caused by multiple emitters producing confusion between the detectors. The power required to emit the type of ultrasound required is large and are therefore more costly than even IR triangulation type sensors.

Existence sensors are designed to simply detect if an object is detected or not. They’re generally much cheaper than range detectors as their functionality is a lot simpler. There are two potential types of existence sensor available.

The first type, is infrared reflective. Similar to IR reflective intensity, these work in the same fashion but need to be calibrated for a set distance at which to detect objects. If an object is too close or too far away from the calibrated distance, then a false reading may occur. If calibrated correctly, however, these sensors would serve the simple purpose of navigating a path through the maze at an efficient cost.

The second existence sensor, is an IR interrupt sensor. The concept behind these is to constantly transmit a beam and as long as it is received, it can be safely assumed that no obstacle is directly between the transmitter and the detector. Applying this concept to a Micromouse, some have been designed to position these sensors above the walls of the maze to determine if a wall is immediately next to the Micromouse.

#### Battery

A source of energy is required to provide power to the motors and microcontroller. A suitable battery will be required to power these components long enough to demonstrate the Micromouse’s functionality. 10-20 minutes is the standard time required for any typical Micromouse competition.

(Auyeung, 2005) Notes that extra power supplied to the motors is not likely to incur any problems. However, it is risky to do this with the microcontroller as it is not as flexible. He states:

“If a logic component is rated at 5V, you cannot feed it 6V or 4V”.

To reduce risk of damaging any components in this way, a voltage restrictor must be used to limit extra energy from being fed into the logic unit.

To calculate the battery specifications required, several factors need to be considered:

* The length of time the unit needs to run for
* The cumulative amount of energy consumption required by the components
* The rate of power delivery required by the components

Nickel Cadmium (NiCd) cells are a type of rechargeable battery and have a terminal discharge voltage of 1.2v. NiCds offer between 500 and 1000 recharges.

Nickel-Metal Hydride (Ni-MH) batteries are another form of rechargeable battery, numerically superior to NiCds. Although both share the same 1.2 voltage production, Ni-MHs are generally twice the capacity of an NiCd and have the extra functionality of being able to be recharged at any point without significantly degrading the quality of the battery, unlike NiCds which suffer from memory effect damage if recharged without being fully discharged.

Lithium-ion (Li-ion) batteries provide many of the similar specifications as Ni-MHs except Li-ion cells have much faster charge and discharge rates and a greater memory-effect efficiency than Ni-MHs meaning they can be recharged at any point without any noticeable battery degradation.

#### Motors

One type of motor is a stepper motor. Stepper motors work by running current through four coils in a specific order to produce an electromagnetic field. Unlike regular motors that turns smoothly, these motors turn in incremental steps and are more than capable of powering the Micromouse’s wheels. They are typically the motor of choice for the majority of Micromouse, however, they can be costly depending on how many are required.

DC motors are another motor type. These are cheaper than stepper motors and simpler to integrate into the Micromouse but require feedback sensors in order to control the Micromouse’s speed. The rotation of the wheels needs to be measured in order to analyse the effect the DC motor’s output has.

#### Motor Control

Control of the motor is done by an H-bridge. An H-bridge operates by manipulating four switches through which control power to the motor. By changing the states of said switches, it is possible to reverse the polarity of the motor or stop it completely. Using this mechanism it is possible to control the rotation of the wheels.

To move forward or backward, the H-bridge will need to be configured so that opposing switches are enabled or disabled in relation to whether moving forward or backward is required. This will need to be setup the same for both H-bridges to ensure synchronisation when both motors apply their torque.

To rotate, the H-bridge for each motor will need to apply the opposite rotation to the other. For example, to rotate the Micromouse clockwise, the S1 and S3 switch on one H-bridge will need to be enabled whilst the S2 and S4 must be the enabled switches on the H-bridge for the other wheel. The process of applying alternate movements to each wheel will cause the Micromouse to rotate.

#### Encoders

Using rotary/shaft encoders it is possible to determine when a full rotation of a wheel completes. The most basic type of encoding is where an LED is shone through a hole in the wheel with a detector on the opposite side recognising each individual received beam as a full rotation of the wheel.

The readings from these encoders can then be translated into meaningful information such as speed or distance metrics. Once calibrated correctly, it will be possible to calculate and link number of rotations with corresponding data using algorithms formed from multiple trials.

For example, if it is known the Micromouse travelled 10cm and completed 5 full wheel rotations during this time, the rotations per cm will be equals to 10cm/5, which is 2cm per rotation. As rotations are linear with distance, this calculation can then be scaled up or down. The robot later measures the wheels to have completed 7.5 rotations which it can translate into know it has travelled 7.5 \* 2cm = 15cm.

The algorithm’s accuracy will depend on how many trial tests are performed, linking rotations to distance.

#### Input Buttons

At least two buttons will be required on the Micromouse. The first will be used to activate the mapping mode, telling the Micromouse to load the program used for generating a virtual map of the maze that it stores. The second button will load the Micromouse into pathing mode, telling it to travel to the finish location if placed on the start positon.

### Software Requirements - Language

The language and engine used for writing and compiling embedded code must suit the following requirements:

* Suitable for 2KB RAM – i.e. no overly large libraries that include unnecessary components
* Has memory management operations to ensure memory efficiency
* Free or included compiler with the robot chosen
* Versatile enough to provide the minimum necessary functions to fulfil the requirements whilst still abstracting complexity through interfaces

#### C

C provides low-level functionality and is arguably the best choice due to its general-purpose usage design and flexibility. It would give the greatest amount of freedom and control down to detailed memory management making it the primary choice if memory is a significant problem.

However, it lacks in higher level concepts such as object orientation and to achieve simple functionality such as string handling and manipulation, the implementation would need to be written from the ground-up, increasing work-load significantly. Containers such as lists, stacks and queues would need to be built manually, although, this may serve as an advantage due to added control over their implementation and memory efficiency.

#### C++

C++ tackles this issue by providing a large standard library with pre-built types, constructs and features such as function/operator overloading, templates, strings and type casting without compromising on explicit code flexibility. It provides object orientation which will increase code management by allowing the possibility to abstract code complexity through the use of class objects. These features make C++ a favourable choice due to the implementation of vital structures being pre-built, meaning more time can be spent on fulfilling this project’s requirements.

The main drawback of choosing C++ over C would be the increase in memory usage due to the size of the standard library and its components. The addition of features the STL provides may not be worth the memory cost it takes to use it to its full potential.

#### ASM

Assembly is versatile and has the ability to construct any desired function, given enough effort and time, due to its almost direct interaction with the hardware. If used correctly, this would be the best choice for memory efficiency. (Holland, 2004) Notes that the drawback of using an assembly language is significantly increased development time and code complexity. If the task were simpler, assembly might be a suitable candidate.

#### BASIC

PICAXE Micromice can only be programmed in BASIC. This could be a limitation for flexibility, expansion and direct memory management, but will provide enough functionality to implement pathfinding algorithms.

#### Conclusion

Whilst complete control of language choice is ideal, it is likely the implementation will need to be done using a language dictated by the chosen robot’s architecture. This is due to the robot control libraries being implemented in typically one language. In addition, the added support provided by using the recommended language for a particular robot will be of great use.

Therefore, the choice of language will be done subsequent to the robot’s architecture being known.

### Obtaining the Robot

#### Building or Buying

One of the foreseen questions is should the Micromouse be designed and built, analysing each component, purchasing them and assembling them? Or should a pre-built Micromouse be purchased to allow transitioning directly into the software development phase? As this is possibly the most important decision of the project, significant research and analysis needs to be dedicated to considering both options in-depth.

A decision matrix was created to rate both options quantitatively. Each point can have a maximum score of 50 and a minimum of -50. The cost of these weightings are determined based on how much impact each factor has on the overall decision.

#### Buying a Micromouse

If a pre-built Micromouse is to be bought, many suitable devices must be investigated and assessed, assigning weights which place some above or below other choices. Factors which form and affect this rank are as follows:

* Price - including shipping costs
* Assembly state
* Hardware capabilities - including clock speed, memory capacity, types of sensor
* How many of the robot’s core requirements are met

The results of this process are documented in the table below with higher ranking candidates being placed first leading to the lowest ranked at the bottom

#### Building a Micromouse

Building the robot from several components is the optimal solution for meeting the requirements and having complete control over the design. The robot would be built to meet the project’s goals making it a more favourable solution. However the opportunity cost of doing this is high as more time would need to be dedicated to this process that could otherwise be spent working on other resource-dependant areas of the project such as implementing sensor code and pathing algorithms.

A possibility would be to seek assistance for this process from an electronics engineer who could assist in assembling a kit from the candidates analysed. For this step to occur, a more in-depth analysis of components would need to be performed to affirm sure knowledge of the Micromouse’s assembly and assembly requirements. Potential project goals will need to be limited and cut short if the Micromouse needs to be built without assistance as this will hinder the resources and time available for higher-level programming and implementation.

#### Conclusion

The final decision, after weighing up all candidates as shown in appendix sections 6.3 and 6.4, and receiving advice from Micromouse expert (Harrison, 2005) during the visit to BCU, described in appendix item 6.1, is to purchase the Pololu 3pi. As the 3pi is built for line sensing, extra components will need to be purchased and added, adapting the 3pi to be suitable for a physically walled maze. Should unforeseen issues arise regarding adapting the 3pi to function in a walled maze, the added benefit of choosing this robot is the possibility of a fall-back to a line maze.

#### Adapting the Pololu 3pi

The Pololu 3pi includes the following features:

* Atmel ATmega328P microcontroller – 32KB Flash, 2KB RAM, 1KB EEPROM
* Five bottom facing reflective IR sensors for line detection
* Sound Buzzer
* Top facing LED array, two underside LEDs
* LCD display
* Ships with Atmel Studio – a Visual Studio fork designed for embedded AVR development
* Includes both C and C++ libraries for interaction with the robot and its I/O
* Requires 4 AAA batteries

Hardware requirements that were not met were analysed, and affordable retail was sought. The complete list of items ordered from Pololu are as follows, with links to each provided in section 6.56.4 of the appendix.

* Pololu m3pi Robot with mbed Socket
* Pololu USB AVR Programmer
* 2 x Sharp Analog Distance Sensor 2-15cm
* 3 x Pololu Carrier with Sharp Digital Distance Sensor
* 2 x 30:1 Micro Metal Gearmotor with Extended Motor Shaft
* Magnetic Encoder Pair Kit for Micro Metal Gearmotors
* Pololu Wheel 42x19mm Pair

## Mapping and Pathfinding

The difficulty of solving a maze depends upon several factors - the most important is perspective. Solving a maze from a top-down, external perspective where the maze’s entirety can be viewed is considerably less complex than solving from a first-person perspective. For example, having somebody solve a relatively small maze drawn on paper can be completed in a matter of seconds, however, putting somebody in a physical maze of the same design would take a much longer time for them to solve.

### Perspective Experiment

(Ferrari, et al., 2001) Conducted an experiment to show how external references and note-taking can aid maze-solving from a first-person perspective. This two-person experiment involved one person (A) seeing a maze and the other person (B) stating how they want to traverse the maze. For example, person B would state they want to travel forward one square and person A would respond with a ‘Yes’ or ‘No’ depending on if the move was viable or whether there was a wall blocking the path. The time taken to complete the maze successfully was far greater than one person having both the maze data and control of the movement.

(Crowe, et al., 2000)’s research using cognitive pattern studies suggest that when humans solve a maze from such a top-down view that they sub-consciously perform several operations based on observations to quickly find a solution. In essence this occurs as a brute-force method where the solver will quickly run through as many routes as possible until they find one that gets them to the end, and in a larger drawn maze may become more evident that this is how a human will generally solve a maze.

In order for an entity to know its way through a maze, the only way is for it to already have a knowledge of the maze’s design. Therefore, it can be concluded that for the Micromouse to be able to calculate the shortest path through the maze, it must already know exactly how the maze is structured. As one of the maze’s conditions for this project is that it must be customable, this maze data must be dynamically generated by the robot. Therefore, the robot’s implementation will be split into two phases, mapping and solving.

### Maze Mapping and Analysis

Mapping the maze requires visiting every tile to be able to form a complete virtual map of the maze. Once the map is attained and stored in a suitable format, it is later possible to perform path finding algorithms on the map.

#### Left Side or Right Side Mapping

A well-known approach to finding the exit of a maze is to stick to the left or right side and just follow it until the exit is reached. In a maze with no exits to the outside, this would be a suitable way to map out the entirety of a maze. However, (Ferrari, et al., 2001) state that we can only do this under the following two conditions:

1. “When the maze is flat, and has both the entrance and exit placed along its perimeter
2. When the maze is flat, and the entrance and exit are points arbitrarily chosen anywhere in the maze, where the latter doesn’t contain any loops. That is, it doesn’t contain multiple paths that connect any two points”

The maze specifications state that the start and end points are allowed to be placed anywhere and that loops are also valid. This means that the left/right side rule will not be of use here. (Ferrari, et al., 2001) Continue:

“When you cannot apply the rule previously stated, you rely on two strategies:

1. Executing random turns
2. Tracking your route”

As the first approach is random, it could potentially take a long time depending on the complexity or nature of the maze. Tracking previously visited tiles would yield more fruitful results and the robot will have the capability to do this with its own memory and storage system.

#### Stack Based Mapping

Splitting the maze into a grid with each tile being a square of the grid of the same size, it is possible for the robot to track which tiles it has been to and which ones it has not. With the use of a stack, it becomes possible to track visited parts of a maze, revisit locations which connect to unexplored tiles and traverse through the entirety of the maze without missing any tile.

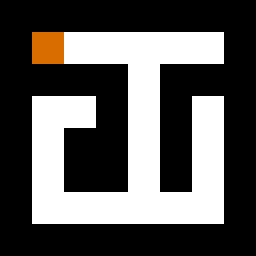


Figure 1.4

Figure 1.4 above shows is the maze with the robot inserted at a dead-end, denoted by an orange block. During the mapping process it will not matter where the robot begins, as its job is to visit every part of the maze so start and finish points are irrelevant here.



Figure 1.5

It starts by moving in any direction. This could be prioritised by moving forward, left, right in respective order depending on which is blocked by a wall. In this case, it moves forward, memorising each tile it has visited (green squares), until it reaches a point where it is surrounded by more than one non-wall.



Figure 1.6

It marks this tile in its memory (denoted by the blue square in this example) and then repeats its movement directive by checking if the tile directly in front of it is blocked first. It isn’t, so it continues moving forward until all its sensors report walls. This means the robot has hit a dead end so it must trace its route back to the last blue tile it visited.



Figure 1.7

To make the process of journeying back to the last decision tile (blue) easier it can make a note of its directives as it executes them. In this case, it travelled *forward*, *forward* from the decision tile which means to get back to this tile it must travel *back*, *back*.



Figure 1.8

The robot now travels in the next direction that it hasn’t visited. Again, in the event that there is a choice between multiple paths it can prioritise directions. Alternatively, it can simply choose randomly between the free paths available as it will make no difference to the efficiency of the mapping process.



Figure 1.9

Using this process of marking choice tiles as it traverses the maze allows it to know there are still unexplored tiles meaning that whilst these blue tiles still exist that the mapping process is incomplete.

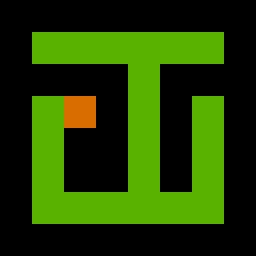


Figure 1.10

This process continues, shown in Figure 1.10 above, until it has no more choice tiles to return to, meaning the mapping stage is complete and the robot now has a memorised version of the map stored, ready for processing and analysis.

#### The Complication of Loops

When there is a possibility of a looped maze, complexity increases significantly. Simply sticking to the left or right strategies will still yield a route to the finish, provided the start and finish points at the edges of the maze. However, as the goal is to map the entire maze in order to be able to calculate the fastest route, a robot following this strategy will never visit areas of the maze unconnected to the edges, which are likely key to obtaining the shortest path.

During navigation, it is possible for the robot to get stuck in an endless loop as it has no method of determining its position within the maze. There are two requirements to combat this problem. The first, is to ensure the maze is exactly proportional, as described in the maze specification. Doing this allows each tile to be assigned a specific pair of co-ordinates, denoting its position in the maze. Second, is to make use of the encoders to allow the robot to determine the distance it has travelled. Fulfilling both these requirements will provide a mechanism through which the robot can determine its position within the maze.

#### Graph Theory

Graph theory is a concept of representing connected data using edges and vertices, of which typical graph algorithms can be applied. Once the mapping process is complete, the Micromouse will have a virtual model of the maze stored in its memory as a graph. With this data it can calculate the fastest route from one point to any other and translate the route into a series of actions it must perform in order to traverse the path to the finish.

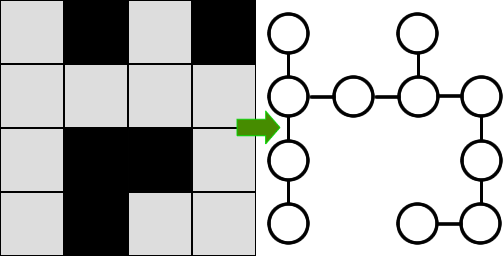


Figure 1.11

To be able to represent each tile as a graph node, a data dictionary must be formed in regards to the properties of a tile. Each tile will store the following:

* Its position in the maze (X, Y co-ordinates)
* Connections - other tiles it is next to, including the cost to each of these connections
* Whether it has been visited

Other properties may be required depending on the path finding algorithm used. Graph theory will be used to represent the maze and will allow standard graph theory techniques and algorithms to be applied to the scenario. (Millington, 2009) Explains that this will entail translating the maze's structure and information in real space into a logical, digital form that can be read and stored by the computer. The process for this is defined as follows:

* Each tile in the maze is represented as a graph node
* When the Micromouse travels from one node to another, it adds the nodes as bidirectional connections to each other
* When every tile has been visited, the maze is now fully mapped and can be virtually envisioned by the Micromouse in the form of an undirected graph

Then path finding and navigation algorithms can be applied to the graph to find the fastest route through the maze between any given tiles. - (Kühl, 1996)

#### Mapping Process Described with Structured English

To summarise the mapping process, the exact procedure that will be used to map the maze will be as follows:

Get Tile Information

1. Read left, right and forward digital existence sensors
2. Translate these readings into logical variables (true/false) denoting the presence of walls
3. Read left and right analog sensors
4. Translate these readings into numeric variables denoting the distance from the left wall and right wall in the tile ahead

Mapping

1. Get tile information for the current tile, store this as the starting tile
2. Push the starting tile to the mapping stack
3. Current tile = tile at the top of the stack
4. If the current tile has unvisited connections, travel to one of these connections, otherwise, pop it from the stack and go to step 3
5. Get tile information
6. If the tile hasn’t been visited before, push it to the stack, otherwise, pop it from the stack and travel back to the previous tile, then go to step 3
7. Repeat until stack is empty, which means all tiles have been visited

### Pathfinding and Shortest Path Calculation

There are a number of algorithms related to maze navigation. The most documented algorithms will be investigated and classified as applicable or inapplicable in relation to the problem of mapping or solving the maze. All algorithms researched below are either products of graph theory research, or if not, are fully functional with graph theory regardless.

#### General Graph Algorithms

Depth-first search, or DFS, employs the strategy of traversing as deep as possible into the graph, then recoiling back up and repeating the traversal in a different route. This would be an optimal solution for the grid above as B is as far away from A as it can possibly be and the search will discover the destination in the first traversal. However, this algorithm could be inefficient if B were closer to A and the branches searched first didn’t connect to the destination.

DFS is a search algorithm which would be useful if the goal was locating the finish point in the shortest amount of time. However, as the goal is to traverse the shortest path to the finish, DFS will not be applicable as searching is not a priority.

Breadth-first search (BFS) achieves the opposite in that it explores all nodes connected to it first, then all nodes connected to each of its children, in turn. This is efficient if the destination is close to the starting position, but unlike depth-first search, it can be costly if the destination is located far away at the deeper ends of the branches.

#### Weighted Algorithms

Improving from a ‘blind-search’ strategy are cost-based algorithms which take edge cost into account. For example, if we wish to travel from position A to position B via the quickest route, and we have two routes: One which involves travelling up a hill and the other which does not, both of which have the same number of tiles. A cost based algorithm such as Dijkstra will reveal the latter route to be quicker whereas a blind-search one such as DFS or BFS may find the former as the quickest route. However, the maze will not include direct, tile-to-tile costs, therefore Dijkstra would serve no more efficient than a BFS - (Mishra & Bande, 2008)

To find an algorithm that would suit this maze’s scenario, an algorithm that performs optimally on non-weighted graphs is needed. For this, informed-search methods will be analysed that make use of a heuristic function.

The computational complexity for both depth first search and breadth first search is dependent on the structure of the graph and how deep the nodes reach, not only how many nodes there are.

Similar to DFS, BFS will not be applicable for this project’s needs, as it is a search algorithm.

Flood fill is an algorithm that works well in grids where nodes can be assigned weights based on their heuristic distance from the target. It could be implemented using a stack, pushing the least costly connected nodes until a dead-end is reached. Then it pops nodes off the stack until it returns to a node with unvisited connections.

Flood fill is well suited to discovering the shortest path between two points, as it takes node distances into account when considering multiple paths of navigation.

##### Heuristics

In algorithms and problem solving in general, (Weise, 2009) states that a heuristic is a means of guiding decision making in a certain direction by determining which choices are more favourable using predefined information or information collected from previous experience in the current problem.

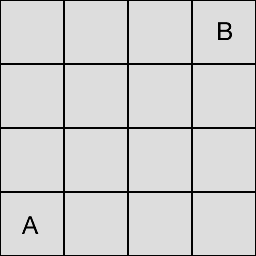


Figure 1.12

In path finding, a heuristic can be used based on whether the target destination’s position is known. For instance, in Figure 1.12, if an entity starts at A, and its target destination is B, it cannot decide which direction to start searching first without prior knowledge of where B is on the grid. A heuristic translating a known target destination into meaningful data that a path finding algorithm can use.

There are several variations of heuristics used for path finding, each with their own merit depending on the type of movement and steering behaviour possible. These are:

|  |  |
| --- | --- |
| **Heuristic** | **Description** |
| Manhattan | Distance between two positions using right-angle edges. I.e. staircase style pathing |
| Euclidean | Distance between two positions using a single straight line |
| Octile | Distance between two positions via 45° diagonal edges only |
| Chebyshev | Distance between two positions via any angle which is a multiple of 45°. Essentially combining Manhattan with Octile |

To make use of these heuristics, an algorithm that takes heuristic cost into account is required, such as Best First Search which is the basis for the most popular path finding algorithms in AI such as A\* and B\* as noted by (Pearl, 1984).

Best First Search works by keeping two lists of nodes, the open list and the closed list. Each node has its own calculated heuristic cost as determined by the heuristic implemented. If the open list is not empty, it removes the node from it that has the best heuristic cost. That is, the node which has the least number of steps to the target node if the heuristic path were possible. It then moves this node into the closed list and finds the nodes it is connected to.

For each of these connected nodes, it is added to the open list if it has never been inspected before, or if it has been and its heuristic cost is better than the version stored, then update the stored version with the best-case cost and set its parent node to be the one that was just added to the closed list. Put simply, the algorithm will ‘follow’ the heuristic path as best possible.

To clarify the process, the algorithm can be described using structured English:

**Figure 1.13**

Professor Edsger Wybe Dijkstra founded an algorithm which could find the shortest path in a weighted, uninformed graph. Similar to flood fill, this algorithm’s search frontier expands by visiting the next node via the edge that has the least distance cost from the start node. Once the target node is found, the algorithm halts and the path left will be the shortest route from start to finish.

The difference between Djikstra’s algorithm and flood fill is that Dijkstra stops the frontier’s expansion at a set depth, whereas flood fill will continue to inefficiently expand the frontier in any and all directions simultaneously.

(Patel, 1997) Describes A\* as a type of Best First Search in that it is an informed search and uses the same process. It differs by performing a more advanced node cost evaluation taking elements from Best First and Djikstra. It uses a ‘g’ cost, which is the distance taken to get from the starting node to the current node being inspected, an ‘h’ cost which is the heuristic distance from the current node to the target node. These are combines to form a total ‘f’ cost. By then using the process documented for best first search, the frontier expands outwards towards the direction of the target following a given heuristic, recalculating g costs as it iterates. Revaluated nodes with better g costs, that is nodes with lower g costs, are updated to always store the g cost of the shortest connection, the same as Best First.

#### Conclusion

As the core project goal is solving the maze via the quickest route, it is established that a shortest path algorithm that does not yield the shortest path 100% of the time is not a valid solution, regardless of how quickly it discovers a semi-efficient path. This discards all algorithms apart from Djikstra and A\*. A\* will be chosen due to Djikstra’s inefficiency by potentially requiring large amounts of memory resources, whereby A\* can use the Manhattan heuristic to hasten the shortest path calculation, decreasing memory usage and calculation time.

## Motor Control

### Centring

A goal of the Micromouse is to avoid contact with the walls of the maze. Without being able to do this, the Micromouse may get stuck and will require very finely tuned calibration. To achieve centring, a PID system will be used to control the speeds of the two motors, ensuring the Micromouse is constantly adjusting the motor speeds based on sensor data, resulting in constantly adjusting motor speeds, pertaining to the Micromouse forever verging on a centralised position. See appendix item 6.16 for a sample diagram of how the PID control loop will be used.

### Proportional-integral-derivative Motor Control

A PID controller is a system designed to calculate a change value based on present, current and predicted values, typically these are errors and the controller’s goal is to minimise them.

In short, the proportional is a means of minifying errors proportional to their difference with the target, which makes it adept correcting large error deviances. The integral handles small error fluctuations accumulated over time based on previous data, useful for fixing undesired offsets and noise. The derivative addresses the rate of change by factoring in how frequent errors are in relation to their change.

(Auyeung, 2005) Shows how to apply PID control theory to a Micromouse. The Micromouse will read in data from the two diagonally cross-facing analog distance sensors to determine its position on the current tile. As it is a possibility that one or both walls the sensors are directed at are not present on a single tile, there will be periods when the Micromouse blindly move until a wall is detected on either side. If one wall is present it can use previous data along with the one analog sensor’s reading to estimate its tile alignment.

#### Proportional

The difference between the current input and the target value is known as the proportional. Because the PID controller will be a centring system implementation, this value will be the distance from the centre of the tile. The target value will be 0 as this represent no skew to either left or right side. Distances with a left-side majority will be negative and values to the right will be positive.

The proportional will use the current input of the analog sensors. As two sensors will provide two separate values, they will need to be combined into a single value. The goal is positioning the Micromouse centrally between both walls, therefore, it must adjust its position until both sensors equate to the same distance values. Thusly, the single proportional value will be 0 minus the right sensor value added to the left sensor value. This will provide a single number that gives information of the Micromouse’s positioning between both walls. This can be shown as:

Where is the degree of error at one point in time, . Represents an adjustment value that may or may not be required to compensate for offsets.

If the Micromouse is positioned thusly:

The final proportional value will be (0 – 70) + 30 = -40. If the mouse was to the right side instead, the value would be (0 – 30) + 70, equating to 40. Perfect centring will result in the proportional value being (0 - 50) + 50 = 0.

Figure 1.14

30

70

#### Derivative

The derivative is calculated as the difference between the current input and the previous input, i.e. the rate of change. It is calculated as:

Where represents the difference between the previous value of error and the current value of error at the current point in time, .

E.g. The Micromouse is at -30 at 5 seconds, then -40 at 6 seconds. Therefore the derivative value is:

Which shows that the current rate of change is 10 units per second.

#### Integral

The integral is the summation of inputs at every reading multiplied by its difference between its time of reading and the previous value’s time of reading. This provides a response to errors accumulated over time. It is achieved thusly:

Whereby taking the integral of the cumulative sum of all error values over time is multiplied by the integral gain, .

E.g. The Micromouse is at 20 at 1 second, 30 at 3s and 40 at 5s:

#### Tuning

The, and constants represent tuning parameters relative to the degree of offset from the useful data. They serve as a means of filtering predicted noise or can be used to scale the PID control algorithm. The value of these constants will need to be set and adjusted using trial and error during debugging and testing.

#### Pseudocode

The following pseudocode shows how the PID control loop will work:



The output given by this function be equal to the difference of error between the current alignment and perfect central alignment at zero. This value can then be used to attempt to correct the error by adjusting either motor, relative to a set speed and its difference from the opposing motor. As this PID control is a looped feedback mechanism, each iteration will improve upon the last, resulting it a robot that continually attempts to centralise itself. Overshooting and noise will occur proportionate to the speed of the robot, as travelling at higher speeds is contingent with destabilisation.

## Program Development

### Memory Management

Due to the typical limitations of a microcontroller, memory will be an aspect of implementation that will need to be considered thoroughly. Careful planning will be required to ensure the robot's functionality does not exceed any of these memory boundaries.

The ATmega328P microcontroller specification shows that the memory capacities are 32KB of flash memory, 2KB of SRAM and 1KB of EEPROM. Research showed some memory usages of typical data structures in the AVR architecture:

|  |  |
| --- | --- |
| Char | 1 byte |
| Int | 2 bytes |
| Long | 4 bytes |
| Float | 4 bytes |
| Bool | 1 byte |

### Memory Types

The EEPROM, Electronically Erasable Programmable Read Only Memory, is a type of non-volatile memory that allows for storage of data regardless of being powered. Any persistent data required by the Micromouse will be stored here. This may include calibration levels, the shortest path data, and possibly the entire mapped maze data itself, should the capacity be large enough.

The SRAM, Static Random-Access Memory, is the most valuable of the three storage options as this is where the majority of temporary data will be stored during the robot's use. With only 2000 bytes capacity, the size of data and data structures will need to be minimized to use this memory as efficiently as possible.

The 32KB of flash memory provides considerable space to store predefined data such as pre-built strings, numeric values and other constructs. Any hard-coded data will be stored here to ease the load on SRAM.

### Memory Leaks

Memory leaks occur when memory is dynamically allocated (using malloc in C or new in C++) and the reference to the memory block gets lost. This makes it impossible to later deallocate this memory after its purpose has been served as there is no way to know where the memory block is located.

Depending on the need for dynamic memory allocation, memory leaks may or may not be a factor to consider and care should be taken to ensure these do not occur. The exact memory expectations and requirements will be analysed in full during the design phase.

A possible circumvention for memory leaks would be to avoid the use of manual memory management and instead use C++ standard library constructs, including smart pointers and pre-written containers such as the Stack, Queue, List and Vector implementations. (Smith, 2011) Lists six methods of avoiding memory leaks, therefore, if pointers are required, these guidelines should be followed.

## Debugging

There are several debugging possibilities for the Micromouse. For Atmel microcontrollers, there is a simulator debugger built into Atmel Studio. This provides a virtual simulation of the program running on any Atmel controller, also allowing for reading virtual register data and providing inputs, as if those signals were actually sent. Whilst this is a useful feature, the simulator can only assist so much. Debugging real world data such as sensor and encoder inputs would be difficult to implement software side and would not be worth the time or effort, simply for virtual debugging.

A second option would be to establish a serial connection between the Micromouse and the computer, then direct instructions from the computer to the Micromouse, allowing for options such as real-time data logging and debugging. Whilst this is favourable, a serial connection may require separate components and may be difficult to install and use.

A final option would be debugging using an LCD, should the Micromouse have one. This would allow for simple printing of desired information without disturbing the process. It would allow for real data to be used in debugging without the sacrifice of extra time and effort spent on establishing a sufficient debugging system. As this is the quickest, easiest and least expensive option, this is likely the debugging method that will be used.

## Testing

### Test Driven Development

TDD is an iterative cycle that is used to qualify functionality. Test cases are written before implementation begins and tested at frequent intervals during development. Initial failure of the tests are expected, as the tests should happen just prior to the beginning of development. Code is produced around the goal of passing the relevant test-case and is continually developed until the test-case passes, indicating the function has satisfied its requirements. Subsequently, the code can then be refined and improved in ways such as memory efficiency, complexity reduction and readability, so long as it still passes the test.

This strategy of development ensures a function's goal is made clear, serving as an aid to development. When these tests are automated, it can drastically reduce effort from manual testing.

TDD will be used for this project. However, testing will still require some manual effort due to the nature of testing an embedded system requiring manual input. Simulation test-cases may also be drawn up for scenarios whereby testing can be performed without the need of the robot. Examples of this will include the mapping and pathing algorithms.

## Project Tools

All versions of the stated tools below are either available free for public use, provided for by Staffordshire University via the Microsoft’s online software distribution platform for academia, Dreamspark, or were purchased prior to or during this project. The development tools are subject to change based on the needs of the robot.

### Microsoft Windows 8.1

Windows 8.1 is the latest publicly released stable variant and is also the version in use at the majority of desktop computers for student use at the university. It is also the version currently installed on the home desktop being used for this project. This makes this operating system the obvious candidate due to ease of accessibility and will be the sole OS used for project development.

### Microsoft Office 2013

This version of Office is widely supported and documented and is also currently installed on both the home desktop and the university computers. Therefore this office suite was chosen over alternatives because of ease of accessibility.

### Atmel Studio 6.2

Atmel Studio is an integrated development environment tailored for embedded systems. It is a Microsoft Visual Studio fork, stripping away desktop and web development components and replacing them with tools that support Atmel architecture, such as AVR simulation and memory logger.

### Sublime Text 3

This powerful text editor features an integrated command line and is specifically designed for software development usage, but also functions as a standard text editor. It includes many features to ease development such as various hotkeys, custom build-scripts, a third-party package manager and syntax highlighting for every popular programming or mark-up language. This will serve as a secondary development environment for testing.

### MinGW + GCC

MinGW (Minimalist GNU for Windows) brings many software tools and packages commonly available with any standard Linux distribution. The primary reason this is required is for GCC and G++ which are C/C++ compilers. Although VS2013 comes bundled with its own C++ compiler (Visual C++), the standard, most supported and most documented compiler is GCC/G++.

## Methodology

The two most popular styles of software methodology can be categorised by the terms ‘traditional’ and ‘agile’. To form the methodology for the project several development strategies from each will be researched and a methodology will be decided upon based on its suitability for this project. Should no specific methodology deem itself fully applicable for the project a ‘hybrid’ methodology may be produced that takes relevant aspects from several strategies.

### Traditional Methodologies

Traditional methodologies follow an underlying principle of a rigid development structure in which development follow a predetermined strategy in separate stages. Most traditional approaches treat each stage’s completion as a prerequisite of moving on to the next. (Awad, 2005) Describes the following traditional methodologies.

#### Waterfall

The waterfall model has an organised progression between each stage of the development process where a set of requirements must be met at each stage in order to progress to the next. The stages are typically ordered Analysis, Design, Implementation, Testing, Deployment and Maintenance in that respective order. A variation of this process includes returning to previous stages if problems occur that were not foreseen and then working back ‘down the waterfall’ stage by stage.

#### Unified Process

There are several methodologies based on a Unified Software Development Process. This Unified Process is an iterative and incremental development technique which involves the four stages:

1. Inception – A short period in which preliminary investigation is done into the project’s cause, scope and use-case and project schedules and cost estimates are drawn up.
2. Elaboration – Product requirements, risk analysis, concept of design and more finely detailed plans are determined and assessed in this phase. Anything that is a prerequisite of the construction will occur here.
3. Construction – The actual implementation of the system. A functional system should be available by the end of this process.
4. Transition – The system is released to its users and further refining of the system can be done as a result of the feedback from the users.

#### Spiral Model

A spiral model is a risk-based development strategy which identifies these potential risks and attempts to minimise and reduce them. Each cycle of the spiral process involves producing prototypes to ease the identification and analysis of potential risks as well as allowing the product’s end-user to steer the development in the desired direction.

There are four primary stages of a spiral model:

1. Target setting – Define goals for the current stage.
2. Risk assessment and reduction – Risks are identified, assessed and an attempt at reducing them is made.
3. Development – This can be the creation and advancement of a prototype that is built-up over the process of the spiral to finally resemble the end product.
4. Planning – Evaluation of the project is made and the next spiral stage is planned out.

### Agile Methodologies

Agile methodologies are commonly known to be ‘evolutionary’ and ‘adaptive’ in that they respond to changes throughout the product development and are flexible with their approaches to particular stages. They often rely upon an early product implementation delivery which is built on continuously. The Agile manifesto (Agile, 2001) states that it values:

* Individuals and interactions over processes and tools
* Working software over comprehensive documentation
* Customer collaboration over contract negotiation
* Responding to change over following a plan

“That is, while there is value in the items on  
the right, we value the items on the left more.”

#### Extreme Programming

This strategy is adaptive to user requirements in that it employs short development cycles and continuous feedback to incrementally develop the product in an evolutionary manner. This grants the benefit of being able to direct and steer the project to a more accurate degree as opposed to an approach that uses a specific implementation phase which imposes a greater risk upon the product exactly matching user requirements. Extreme Programming follows roughly a dozen practices to ensure its effectiveness which include strict rules such as limiting work hours to 40 per week and always involve a real, live user on the team who is available full-time to answer questions - (Beck, 2000)

#### Scrum

(Awad, 2005) Describes scrum as most often used amongst teams of developers and bases its core principles upon frequent communication and discussion. Another of its fundamental principles is that the user’s requirements can change which could cause unforeseen challenges at any point. It is also known to be an iterative, incremental process and has specific intervals at which meetings must occur and ‘sprints’ (30 days of adapting to environmental variables and adapting the software) end. Because of the way Scrum is structured it is favourable for development teams and long or continuous projects, thus, making it ill-suited for this project.

#### Rolling Wave Planning

Rolling Wave Planning introduces the concept of adaptive planning whereby a ‘wave’ signifies readjusting and refocussing the plan based on project clarity and understanding. I.e. As progress is made, ensure the plan is also adapted to suit the current goals. (Larman, 2003) Notes that rolling wave planning is a strong strategy of adapting to variables that can change in the project and integrates neatly into an agile methodology.

Each wave will occur every two weeks starting from the beginning of the implementation phase up until three weeks prior to the end of the implementation phase. This is a similar concept to sprints used in Scrum. However, in rolling wave planning specific focus is given towards reassessing project goals, potential and the plan to obtain these.

### Conclusion

The project’s development will be carried out using an agile approach due to the project’s nature being unpredictable and subject to large change. E.g. The possibility that a component breaks and having to fall back to a compromise. Obtaining the robot may have significant impact on the time left to carry out phases which having the robot is a prerequisite to.

Agile was also chosen to get a semi-functional prototype rolled out as soon as possible to avoid potential issues with the robot’s core functionality cleared up as opposed to them occurring in the later stages of the project when time constraints are limiting. Then the robot can be further developed and stretch goals can be worked towards.

Rolling Wave Planning will be used to reanalyse and reassess the project’s progress, goals and strategy in several waves. These are outlined in the project plan in appendix item 6.2.

It was decided that traditional methodologies would not be suitable for this project due to their rigid structure and opposition to change.

## Version Control and Project Management

Version control is a document organisation system designed to aid managing dynamically changing files. It works by keeping a base copy of a file when it is first uploaded. Then if the file is changed and the change is validated with the version control system, the change is saved separately and associated with the base file whilst storing metadata about the change, such as the time is was made. This way, when a user requests the latest revision of the file from the VCS, the VCS applies the changes to the file in the order they validated and gives the updated file to the user. The location in which the VCS server holds the files is the central location for the latest version of files and is known as the repository.

This serves many purposes such as easing multi-user collaboration as changes can be applied from multiple sources. It serves as a means of documenting file changes and allows users to see who made certain changes to documents and to examine the specific changes individually. It also serves as a natural backup system as the VCS host must store its own copy of the documents.

Two currently well-known VCS’ are Git and SVN, both of which are commonly used for personal and commercial purposes. For this project, Git will be used as GitHub, a popular, free VCS server, provides up to five private repositories for students. If not for this, the repository is made publically available for anybody to view, as the primary goal of GitHub is to provide free VCS server access to open-source software only. Otherwise, a server that provides free, private, VCS hosting would need to be sought. Common Git terminology can be found in appendix section 6.12.

## Issue Tracking

Issue tracking will mean using a designated system designed for hosting and tracking ‘issues’ or problems with the implementation or project as a whole. This acts as a platform where anybody currently working on a software project can submit an issue form/bug report which is then stored in the system and made publically viewable. An issue tracking system such as this has several useful purposes.

The first is that it allows developers to document bugs as they are discovered instead of relying on dedicated testers to fulfil the role. It has the potential to track certain information about a bug such as causes, suggested fixes, current work-arounds and more.

Integrating issue tracking into this project will streamline bug documentation into a managed system whereby specific issues can be assigned levels of priority and severity. It will also ease the process of project management and likely improve project structure and organisation as it will also be used to document inefficiencies or areas for improvement/restructure. GitHub Issue tracking will be used as it integrates into the Git VCS used.

# Design

## Physical Robot Design

This section describes how the components will be assembled and attached.

### Soldering Components

The first phase of implementation is assembling the components with the 3pi robot. This requires physical electronics work to be done. Therefore, a soldering kit, copper wire, wire cutters and a micro-toolkit that includes a set of screwdrivers and a pair of pliers will need to be purchased.

Once all components are acquired, assembly will be carried out thusly:

#### Gearboxes and Encoder Attachment

1. Disassembly of the 3pi's wheels
2. Disassembly and desoldering of the 3pi's standard gearboxes
3. Attachment of the 42x19mm wheels
4. Attachment of the 30:1 gearboxes with extended motor shaft
5. Attachment of the magnetic encoders to the gearboxes
6. Soldering the gearboxes to the 3pi

Once these steps are completed, the encoders' functionality should be tested via software using the USB programmer to ensure they are functioning correctly.

#### Digital Sensors

This part of the assembly will require soldering the digital sensors to the left, right and front of the 3pi's expansion board. They should then be wired up accordingly:

* Digital sensor -> 3pi
* GND -> GND (Ground / Negative current)
* VIN -> VCC (Position current)
* OUT -> PC0 (Digital I/O - Left sensor only)
* OUT -> PC1 (Digital I/O - Right sensor only)
* OUT -> AV6 (Digital/Analog I/O - Forward sensor only)

To make 3 positive and negative current connections possible, the 3pi's VCC and GND pins will be forwarded to an extension portion of the PCB, allowing for up to 16 extra connections for each VCC and GND. Placement of the digital sensors, including their wiring, is shown in appendix 6.6. Once attached, the sensors will then be tested to ensure their connections are successful.

The placement and connections required are shown in appendix section 6.6.

#### Analog Sensors

At this point, the two analog distance sensors must be attached. This will be done as follows:

1. Soldering of wires to the analog sensors' pins
2. Soldering of these wires to the GND, VCC and AV5/AV6 pins on the expansion board
3. Sensors placed correctly, then kept in place using sellotape

These sensors will be tested to ensure they are working correctly.

The placement of the analog sensors, including their wiring, is shown in appendix section 6.7.

## Maze Design

The cardboard maze will be based on a piece of reinforced card. The size of this will vary depending on the size and movement style of the robot so this will be decided at a later stage, once exact measurements can be deduced. This base will be raised above the surface by 1-2cm. It will then be split into a grid and slits will be cut into each section of the grid. The walls will be made of thinner card that can be slotted into the base to form the pathways of the maze. A graphical example of this is shown in section 6.8 of the appendix.

As the 3pi is approximately 9cm in diameter, the minimum size of a tile should be 10cm, but 1.5~cm extra space on either side should be added as a security, should the robot be slightly cramped in a 9x9cm tile. This brings the possible size of a tile to 13cm. A maze with 10x10 tiles for example, will require a 140cm^2 surface, considering 10cm extra to account for wall widths, stands for the base of the maze and any requirements that may have been unaccounted for.

## Program Design

Figure 15 below shows the activity flow of the basic program logic.

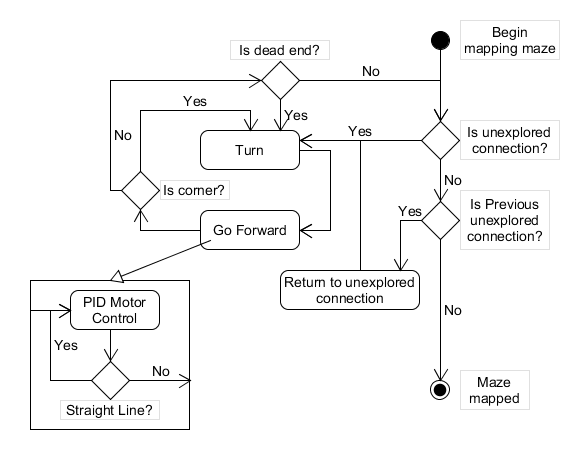


Figure 15

Figure 1.6 below shows the two finite state machines that will be used to track the robot’s current states.

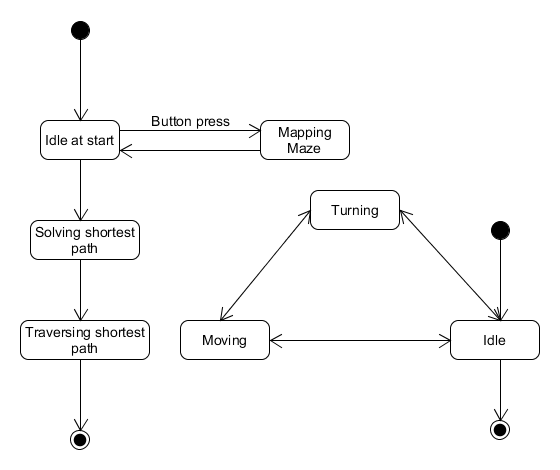


Figure 16

## Design with Development

The majority of program design will be done using method stubs, described in the next section. This suits the agile project methodology, as implementation will be done in conjunction with design, by the use of prototyping methods, then building up the full method functions gradually over time.

### Development using Method Stubs

Code development will make use of method stubs to wireframe the entire system before any meaningful code is written. Method stubs in this sense mean empty methods that may exist in the call stack but execute no code. Benefits from developing this way include demonstrating a clear program structure early on, aiding in knowledge of what methods are required, what each method's particular purpose is and being able to track progress easier.

In addition, method stubs lend themselves to test driven development as they promote initial requirement failure, followed by developing sub-systems to pass their respective test-case.

### Data Dictionary

Some initial enumerators and data types that will be required are:

Direction - Cardinal directions of North, South, East and West. Will be used for tracking and storing information about tile connections and which way the robot is facing, irrespective of its orientation.

Turn - Left, Right and Back. Will be used for denoting a robot's turn action if the cardinal direction is not sufficient

State - As denoted in the finite state machine, this can either be Mapping, Solving or Idle

Tile start - This will be the very first tile encountered in the maze. Upon beginning the mapping process, the robot should turn 360deg and collect information about its current tile surroundings, then store this data permanently for the session.

Tile finish - The end tile, denoted by a large, solid shape with black electrical tape that the Micromouse can detect using the 3pi's 5 bottom facing reflective sensors.

Int speed - A global variable that controls the speed of the robot. Depending on the effects of varying this value, this may need be set as a constant.

The initial method stubs will be as follows:

int main() - Entry point

void init() - Initialise variables, constructs, sensors etc

int goStraight(int speed) - Micromouse will go straight forward until an intersection or dead-end is met, returns the number of tiles travelled forward

void turn(int facing, int speed) - Where 0 = No turn, 1 = Turn left, 1 = Turn right and 2 = Turn around

void faceDirection(int direction) - 0 = South, 1 = West, 2 = North, 3 = East

void travelTo(int x, int y) - A\* algorithm here, only possible once mapping is complete

void mapMaze();

A stack implementation will also be required. Depending on memory usage relative to the 3pi's available SRAM (2000 bytes), C++'s Stack implementation included in the standard library may suffice, along with any other required structures or containers, which will be founded as development progresses.

# Testing

## Initial testing

A small testing phase will occur whereby all the required operations of the 3pi are tested with varying instructions to ensure the entire assembly was successful and that the implementation is ready to continue to the software development. These tests are unrelated to the project's actual TDD test-cases, however, they still follow typical test-case standards. The test-cases can be found in section 6.9 of the appendix, along with their actual results which were added subsequently.

## Unit Tests

Due to issues expressed in the critical evaluation, full unit tests for every component of the program were not completed. However, algorithms for both the mapping and the A\* calculation were implemented prior to implementation beginning. They were tested using GCC without the C++ STL, ensuring that the proof of concept was valid, making implementation of these components in the embedded program easy to integrate at a later stage.

# Critical Evaluation

## Summary

During implementation, a number of problems came to light, some expected and others not. This section will detail the problems that occurred during the project, the compromises that followed and processes formed in hindsight of how the problems should have been approached.

## Component Assembly

### Micrometal Gearmotors

Whilst attempting to desolder the 3pi's standard gearboxes, it was found that the solder used on the 3pi would not melt. This could have been due to the 3pi solder being lead-free, which has a higher melting point than lead based solder. It may also have been coated with an anti-corrosion spray which also renders soldering irons less effective. A solution to this issue was pursued by means of purchasing a higher temperature soldering iron, 40 watts as opposed to the 30 watt iron used prior. However, this was to no avail as the solder still did not melt.

Without encoders, it would prove difficult for the Micromouse to know the distance it travelled. Two compromises were formed. The first, would be to base distances on time taken. Similar to the calculation of multiplying rotations per cm by the number of rotations completed to retrieve the total distance travelled, substituting wheel rotations with a time delta between the start of a maze segment and the end of a maze segment would give the total time taken to travel n tiles. Trials may still be done to test roughly how long it takes to travel n number of tiles and the time taken for each will be noted. The results of the tests will result in an equation that can be used to find the most likely number of tiles travelled, given a number of milliseconds.

### Digital Sensors

The front-facing sensor was soldered in and wired up first, as shown in appendix item 6.6. A test was performed immediately after to make sure the soldering was successful and the wiring was correct. The test was successful so assembly of the next sensor was attempted. Whilst the soldering and wiring was the same as the previous sensor's, the test failed and a signal could not be received from the sensor. This could have been due to bad soldering so resoldering was done, but with no success. This also occurred with the third sensor for no discernible reason. After multiple failed attempts to get the two side sensors working, it was decided these too had to be sacrificed.

Lacking side facing digital binary sensors meant it would not be possible to know the exact location of a wall or a non-wall without turning the Micromouse to the left, then to the right at every tile, tripling the workload of the front sensor.

### Analog Sensors

Issues occurred with the analog sensors when soldering wires to the sensor's sockets. Pololu states:

"Because of the narrow pitch, it helps to have a fine-tipped soldering iron, and you should be very careful not to accidentally short two pins together with your wires or any added solder"

However, despite the warnings, this still occurred, rendering one analog sensor useless.

### Decision to fall-back to a line maze

Despite multiple assembly attempts, the forward facing digital sensor was the only sensor that functioned correctly. Because of this, the decision to fallback to a line maze was deemed necessary. This was of detriment to the project as the majority of time and effort was focussed on achieving a robot to solve a walled maze. Regardless, using a lined maze would not detract from achieving the core goal of traversing a maze via the shortest path and the majority of design could still be applied.

The mapping, pathing and PID control algorithms remained the same. The most significant impact was the issue of having no method to track distance travelled, due to lack of encoders. However, an alternative version was implemented that used time to measure distance. The essential concept of relating wheel rotations to centimetres travelled still applied, but using milliseconds instead of wheel rotations. The drawback of this system is the robot was not accurate with its distance travelled 100% of the time, and occasionally made incorrect calculations due to the fluctuation in time taken being marginally unpredictable. Six test trials were completed to provide enough data to allow the robot to know the distance the majority of times. These trials are shown in section 0 of the appendix.

Building the line maze was found to be easier than attempting to build the physically walled maze. A 1200x900mm magnetic whiteboard was used as the base and the measurements of the tiles were adjusted to 10cm^2 with 10cm extra on each side to compensate for robot travelling along lines on the outer edges of the maze. This brought the total number of tiles to 10x7. A maze design was formed in Microsoft Paint, working on a 21x15 pixel canvas (10 \* 2 + 1) \* (7 \* 2 + 1) to be able to fully illustrate what the maze should look like when finished. A grid was made using black electrical tape, and then an X-ACTO knife was used to cut out unwanted lines, leaving the lines left to form the maze designed. The maze design is shown in section 0 and the final maze implementation shown in section 6.14 of the appendix.

### Calibration

Using the underside reflective IR sensors required manual calibration by means of rotating the robot to either side until all five sensors had exposure to both the white surface of the whiteboard and the black colour of the electrical tape. By instructive the sensors to be set to calibrate during this time, maximum and minimum readings are stored for each individual sensor. This calibration allows the sensors to contrast future readings with the maximum and minimum values to evaluate a reading's measurement between the two.

One problem was that the robot required recalibration each time it was powered on, due to the maximum and minimum values being lost when powered off. To solve this, the minimum and maximum values were stored in EEPROM, so they were not lost when the robot lost power. They were then read in during the initialisation and used to calibrate the sensors. A disadvantage of this system was that calibrations were not flexible and were based on the last manual calibration. This brought about miscalibration issues.

Two problems occurred as a result of either miscalibration, or real time variables that affected the sensor readings. One such variable was lighting. Depending on the levels of light in the room, the reflective sensors reported differing values. This was due to the reflective surface of the whiteboard rebounding light into the sensor's retrievers. Early in the day, when the sun was on the window side, the sensors read lower inputs when attempting to detect a black line, then at night, functioned correctly when the room's lights were on.

Miscalibrations brought about an issue in the PID control loop which made the robot occasionally fail to recognise a line, causing it to veer off-course or fail to make turns successfully. The likelihood of this occurring was minimized by adjusting the tolerance and range of values used to classify the IR readings.

## Lack of standard library

Whilst preliminary research showed that the Pololu 3pi was capable of using C++ with Atmel's forked GCC compiler, it did not find that the standard library was unavailable due to memory issues and portability to the AVR platform. Without the C++ STL, workload increased, as typical constructs that prior, were taken for granted, had to be self-written. A short list of constructs that were expected to be available and their implemented alternative can be found in section 6.15 of the appendix. For example, templates were replaced with constructs tailored to specific types, such as NodeList instead of just List<T>. A stack implementation was built from the ground-up, built for node types, to serve as the mapping stack used to backtrack the mapping route.

An added benefit from this lack of STL inclusion, was the increased availability of memory. As constructs had to be implemented manually, this meant that only the minimum functionality was implemented to achieve the goal.

## Memory Leaks

Despite prior knowledge of the possibility of memory leaks, these still occurred in the mapping algorithm. The six techniques (Smith, 2011) lists to minimize memory leaks were stepped through, one at a time. Whilst code appeared to be adhering to correct memory allocations and address tracking, the robot displayed signs of increased memory loss as the duration of the mapping continued.

Fortunately, the majority of memory leaks were eventually found, through a whole day dedicated to finding and correcting allocating the memory address pointers. Construct sizes were also decreased significantly by replacing all integers with 8 bit signed/unsigned variants. That is, int -> int8\_t / uint8\_t.

## Reflection

Numerous problems arose throughout the project, causing strategy and plan to be constantly changing. Fortunately, the adaptive project management style chosen gave way to these changes and the rolling wave planning used allowed critical issues to be assessed, and the project’s direction moved to avoid these problems halting the complete development of the solution.

With hindsight, it became apparent that the concept of assembling a robot with as many components as there were, was severely underestimated. Knowledge of component assembly, including deciphering component specifications, soldering and wiring had to be acquired in the duration of the project. This process took approximately two months in total, from knowing which components were required, up until the component assembly was attempted.

More in-depth component assembly analysis should have been carried out, highlighting key stages of hardware implementation that could cause critical problems, and developing solutions to these problems. Either by means of seeking expertise or forming full contingency plans.

The majority of focus during this project was wrongly directed at the robot’s hardware, which meant other areas that required the same amount of attention fell behind. Exact unit tests were unable to be carried out, development design patterns were not fully assessed and calibration issues were not able to be fixed. These areas lacked due to lack of fully-functional contingency plan, including a secondary time management plan that should have served as the resolve, should the robot’s hardware requirements not have been met by a particular date.

Despite misuse of time, a functioning robot that attained completion of the primary goal was still developed successfully, albeit to the detriment of the project report. This could have been avoided by sacrificing the project’s main goal and instead put effort into demonstrating proof of concept.

One unrelated minor oversight, was lack of documented and fully commented code. This was again, due to time limitations.

# 

# Deliverables

## Deliverables

The final deliverables for this project are as follows:

* A robot that meets all requirements, bar obstacle avoidance related goals
* A maze that meets all requirements, bar walled maze related goals

A video showing the Micromouse running the program implemented for this project is linked to in appendix item 6.14.

The robot’s use is simply turning it on using the power button, positioning it on a desired starting location, then pressing the B button to map the maze. If the maze is mapped successfully, the robot will have returned to the starting location and will await the next B button press. Once this occurs, it will calculate the shortest path based on information gathered during the mapping phase, then proceed to traverse this path.

# Appendix

## Email Correspondence with (Harrison, P)

**It is noted that Peter Harrison gave permission to publish these emails for academic purposes in person during the visit to Birmingham City University.**

**----------------------------------**

On 19 Jan 2015, at 18:09, Jasper Cashmore <jaspercashmore@gmail.com> wrote:

From: Jasper Cashmore <jaspercashmore@gmail.com>

Subject: Need a Micromouse

Message Body:

Unsure if messages sent via this contact form are still checked or not, but here goes anyway.

I'm in my final year at university studying Computer Science and I'm looking to produce a robot that can solve a maze using my own algorithm. The primary focus is the software and algorithm, reading data from sensors and translating it into useful information the robot can use to decide its movements.

However, I'm really no electrical engineer and have next to no experience with building a robot or any type of hardware at all. Hence, I'm looking for a good way to obtain a simple Micromouse for use in my project. My budget is around £100. It doesn't matter how fast the Micromouse is, so long as it's relatively small.

Wondering if you could help me out or point me in the right direction?

Thanks!

-----------------------------

19 January 2015 at 18:37

From: Peter Harrison <peter.harrison@helicron.net>

Hi

Well, that is more complicated than you might think. The actual Micromouse contest has a number of particular requirements that are hard to satisfy.

However, if the assignment is about software, I might suggest the Pololu 3Pi robot (http://www.hobbytronics.co.uk/robotics/robot-kits/3pi-robot)

This is pretty well ready to go and the software is free. Programming is done in C/C++.

The robot is not suited to the full Micromouse contest as-is. It could be modified but you may find it more convenient to create a line-maze solver (http://youtu.be/9H3fiWP0lE4).

The maze is much easier to build and the challenges are similar. There is also a fair bit of information on this already available for the Pololu robot.

Can I ask where you are studying?

-----------------------------

19 January 2015 at 23:36

From: Jasper Cashmore <jaspercashmore@gmail.com>

Thanks for the quick reply. I'm studying at Staffordshire University.

I have considered the line maze, but wish to proceed with the physically walled maze for now then fall back to the line maze if I run into issues.

I'm not particularly bothered about satisfying all official Micromouse competition's rules, but the goal of the Micromouse is more or less the same.

My primary candidate is the PICAXE PICone, but is quite costly and still requires assembly that is beyond my experience. Again, I have no electronics experience. That said, I'm willing to attempt building a Micromouse myself, but I'd be worried about causing permanent damage to costly components due to my inexperience with soldering and such.

My current plan is a bit wishful, not sure if you'll know anybody that could help me or not. I'm hoping I'll be able to find somebody who's willing to build a Micromouse (such as this one) for me, and I'll pay the cost of components plus extra for assembly and shipping. This would put me significantly over my budget but I'm willing to do so for a quality Micromouse.

Alternatively, just buying a used one from somebody is a suitable option for me. Old ones, prototypes, sluggish ones that never made it to competition use. All of these will probably suit my needs fine. I imagine there's a fair few sitting around in draws from 10+ year old competitions. Unfortunately, I'm just in no easy position to find out how to go about getting my hands on one.

---------------------------

20 January 2015 at 01:14

From: Peter Harrison <peter.harrison@helicron.net>

I think you need to find out a bit more before parting with any cash.

Also, your goals are not entirely clear to me.

The PICone can be made to do the job but it is a fair bit of work.

Are you able to get down to Birmingham City University on Monday, Feb 2nd?

We run a club there from about 5:30 pm and you will be able to talk to me and a couple of other builders - including one who has built and used the PICone.

------------------------

20 January 2015 at 14:47

From: Jasper Cashmore <jaspercashmore@gmail.com>

Definitely, I'm still researching but the deadline is the end of April and I'm just hoping I don't end up caught in an electronics nightmare where I can't get anything to work.

I may be able to make it there, but if I can't I assume it's a club that runs every week/month?

------------------------

20 January 2015 at 17:30

From: Peter Harrison <peter.harrison@helicron.net>

We meet first Monday each month. You can find dates and details at www.micromouseonline.com

Give some thought to modifying the Pololu 3Pi

It and the PICone are probably the most ready platforms out there if you want to start as close to ready as possible.

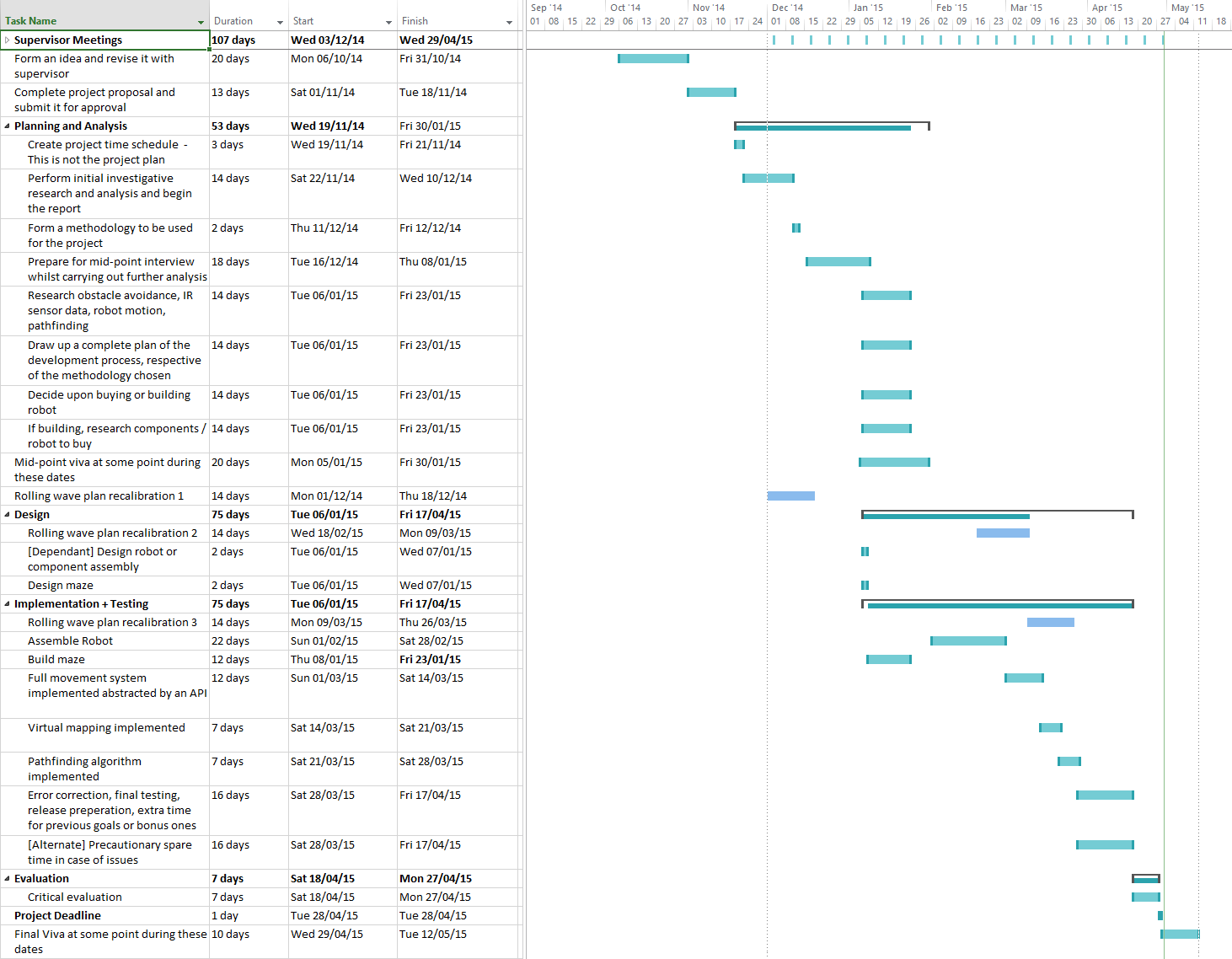
The Pololu robot is going to give you more transferrable skills and offer better chances of support.

If you don’t use a standard Micromouse maze then this may help: https://www.pololu.com/docs/0J26/all

In late march there is a contest at Aston University that will feature wall followers. The only constraint is the drive train I think. Can’t find a link now though.

## 

## Gantt Chart



## Pre-built Candidates and Assembly Kits

| **Image** | **Name** | **Is Built** | **Price** | **Notable Specs / Features** | **Other** | **Seller** |
| --- | --- | --- | --- | --- | --- | --- |
| Pololu 3pi Robot Kit | Pololu 3pi | Pre-built | £70 with free shipping | * C/C++-programmable Atmel ATmega328P microcontroller * 32KB flash memory * 2 KB RAM * 1 KB EEPROM * Capable of speeds exceeding 3 feet per second * Separate expansion board available | * Designed for line-following * Does not meet the necessary requirements for a maze with walls, but could be modified to achieve this | [Pololu](http://www.pololu.com/product/975)  [Hobbytronics](http://www.hobbytronics.co.uk/3pi-robot) |
| http://www.robotshop.com/media/files/images/arduino-2wd-robot-us-large.jpg | Arduino Robot | Pre-built | £170~ | * ATmega32u4 microcontroller * 5 Digital I/O Pins * 32KB flash memory * 5 Programmable buttons | * Highly modular and easily extensible * Solid platform, official documentation and support * Very expensive | [Arduino](http://arduino.cc/en/Main/Robot) |
| http://www.picaxe.com/Thumbnail.ashx?image=~/Site_Resources/Media/Site_1/kit110/kit110.jpg&h=600&w=800&mode=Absolute&k=0e00573d1a61a7119e640e40734c0d09a9f60d8adaf07e6e1eadfc81b6f263dc | PICAXE PICone | Assembly Kit | £90 including UK shipping cost | * PICAXE-28X2 64MHz microcontroller * 4KB flash memory * 1280 bytes of RAM | * Designed specifically for Micromouse competitions, making it a suitable choice * BASIC only, no support for other languages or frameworks * Expensive | [PICAXE](http://www.picaxe.com/Hardware/Robot-Kits/PICAXE-PICone-Micromouse/) |
| http://ecx.images-amazon.com/images/I/81NIouhWlwL.jpg | MiniQ 2WD | Assembly Kit | £60 including shipping cost from the US | * Atmel ATmega328P microcontroller * 32KB flash memory, 2 KB RAM and 1 KB EEPROM * 5 bottom reflective sensors * 2 forward facing IR sensors * Two motors with encoders to fully control the speed of movement. * Speed: 0 – 78cm/s | * Fulfils the requirements * Easily Extensible * Fairly cheap | [Banana Robotics](https://www.bananarobotics.com/shop/DFRobot-2WD-MiniQ-Robot)  [Amazon](http://www.amazon.co.uk/MiniQ-2WD-Complete-Kit/dp/B00E68HY88/ref=sr_1_cc_1?s=aps&ie=UTF8&qid=1417972564&sr=1-1-catcorr&keywords=4WD+MiniQ) |
| http://www.picaxe.com/Thumbnail.ashx?image=~/Site_Resources/Media/Site_1/bot120/BOT120.jpg&h=600&w=800&mode=Absolute&k=2c044d78a8e5c71d4623dd3169dd9da000e9ff7b1efcb317174519cb7c7c104c | PICAXE-20X2 Microbot | Assembly Kit | £48 including UK shipping cost | * PICAXE-20X2 microcontroller * 4KB flash memory * 256 bytes of RAM * Bumper module * Line follower module | * Is a barebones robot but extra components are easy to attach * Cheap * BASIC only, no support for other languages or frameworks | [PICAXE](http://www.picaxe.com/Hardware/Robot-Kits/PICAXE-20X2-Microbot/) |
| http://www.robotstorehk.com/micromouse/images/RJ_linetracer.gif | RS-Cruiser | Pre-built | £87 including shipping cost from China | * Atmega8 microcontroller * 6 bottom reflective IR sensors * 6 side IR sensors * Speed: 0 - 50cm/s * 6 LEDs * 2 programmable buttons * 8Kb flash memory * 1kb ram * 5 bottom reflective sensors * Free C compiler(WinAVR) | * Small website and company that is based in China – could be complications * Limited information about it | [robotstorehk](http://www.robotstorehk.com/micromouse/RS-CRUISER.HTML) |
| iRobot Roomba Vacuum | iRobot Create | Pre-built | £100 including shipping cost from the US | * Caster, left, and right wheel drop sensors * Left and right bumper * Wall sensor * Left, front left, front right and right cliff sensors * Omnidirectional IR receiver | * No longer available to purchase (was available at the start of the project) * Lacks much internal processing and computing power, would need this attached * Expensive | [iRobot (page removed)](http://store.irobot.com/education-research-robots/irobot-create-programmable-robot) |

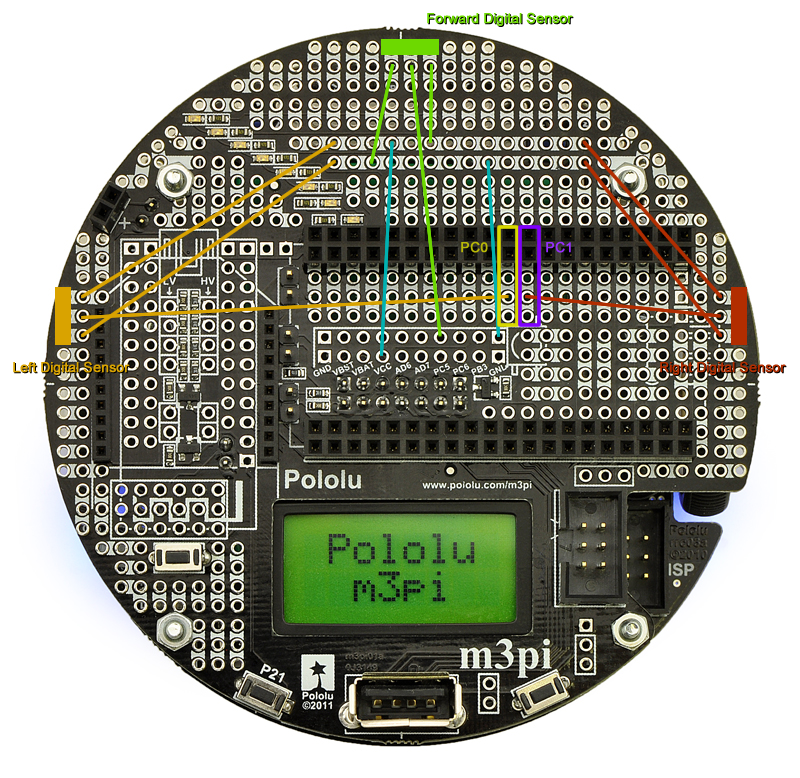
## Build or Buy Decision Matrix

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Build** | **Score** | **Buy** | **Score** |
| Advantages | Complete flexibility and control over the design and creation of the robot and no more | +45 | Saves significant time that could be spent on development | +55 |
| Only need to buy the exact components required for the project | +10 | May find a solution that is designed for the exact or similar purposes | +45 |
| Likely to be cheaper than a pre-built solution | +30 | Will be tested by previous consumers | +40 |
| A huge amount of options, ultimately deciding what the final robot will be capable of | +40 |
| Disadvantages | As the separate parts may not have been tested together before, problems could occur when assembling the robot | -20 | May come with unnecessary parts | -5 |
| Lots of time and research needed to find all the best components on a limited budget and time spent assembling them | -35 | Likely to be the more expensive | -30 |
| A very limited number of choices of pre-built robots | -25 |
| Total |  | 70 |  | 75 |

## Parts List

* Main Robot: Pololu m3pi Robot with mbed Socket <https://www.pololu.com/product/2151>
* USB Programmer: Pololu USB AVR Programmer <https://www.pololu.com/product/1300>
* Distance Sensors: 2 x Sharp GP2Y0A51SK0F Analog Distance Sensor 2-15cm <https://www.pololu.com/product/2450>
* Digital Binary Sensors: 3 x Pololu Carrier with Sharp GP2Y0D810Z0F Digital Distance Sensor [10cm https://www.pololu.com/product/1134](10cm%20https:/www.pololu.com/product/1134)
* Gearmotors: 2 x 30:1 Micro Metal Gearmotor with Extended Motor Shaft [https://www.pololu.com/product/2202](https://www.pololu.com/product/2202%20)
* Encoders: Magnetic Encoder Pair Kit for Micro Metal Gearmotors <https://www.pololu.com/product/2598>
* Pololu Wheel 42x19mm Pair: <https://www.pololu.com/product/1090>

## Digital Sensors’ Attachment



## Analog Sensors’ Attachment



## Cardboard Maze Design



## Component Testing

| **Component** | **Description** | **Test Statement** | **Expected Output** | **Actual Output** |
| --- | --- | --- | --- | --- |
| Left wheel | Testing that the left wheel can rotate forward on its axis | set\_motors(30, 0) | Left wheel turns forwards | Left wheel turned forwards |
| Left wheel | Testing that the left wheel can rotate backward on its axis | set\_motors(-30, 0) | Left wheel turns backwards | Left wheel turned backwards |
| Right wheel | Testing that the right wheel can rotate forward on its axis | set\_motors(0, 30) | Right wheel turns forwards | Right wheel turned forwards |
| Right wheel | Testing that the right wheel can rotate backward on its axis | set\_motors(0, -30) | Right wheel turns backwards | Right wheel turned backwards |
| Forward digital sensor | Testing the forward digital sensor can detect objects | is\_digital\_input\_high(IO\_D0) | -1 | -1 |
| Left digital sensor | Testing the left digital sensor can detect objects | is\_digital\_input\_high(IO\_D1) | -1 | 0 |
| Right digital sensor | Testing the right digital sensor can detect objects | is\_digital\_input\_high(IO\_C5) | -1 | 0 |
| Left analog sensor | Testing the left analog sensor can detect objects from range | analog\_read(6) | 0 - 650 | N/A |
| Right analog sensor | Testing the right analog sensor can detect objects from range | analog\_read(7) | 0 - 650 | N/A |
| Left encoder | Testing that wheel rotations are being counter correctly by the left encoder | getCountsM1() | 0 – 50 | N/A |
| Right encoder | Testing that wheel rotations are being counter correctly by the right encoder | getCountsM2() | 0 - 50 | N/A |

## Distance Trials Based on Time Taken

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Num of Tiles Travelled | Time Recorded (ms) | | | | | | Avg |
| 1 | 660 | 680 | 650 | 756 | 621 | 632 | 666.5 |
| 2 | 1211 | 1294 | 1253 | 1284 | 1206 | 1210 | 1243.0 |
| 3 | 1733 | 1745 | 1680 | 1600 | 1592 | 1789 | 1689.8 |
| 4 | 2243 | 2373 | 2349 | 2240 | 2315 | 2311 | 2305.2 |

Sample size: 24

Intercept: -243.75

Slope: 779.25

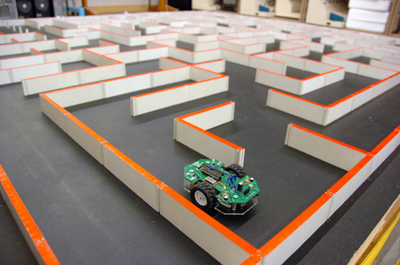
Regression line equation: y = 779.25x - 243.75

x = (y + 243.75) / 779.25

Manually calibrated ranges:

|  |  |
| --- | --- |
| **1 Tile** | 50 – 850 |
| **2 Tiles** | 851 – 1300 |
| **3 Tiles** | 1301 – 1850 |
| **4 Tiles** | 1850 - 2700 |

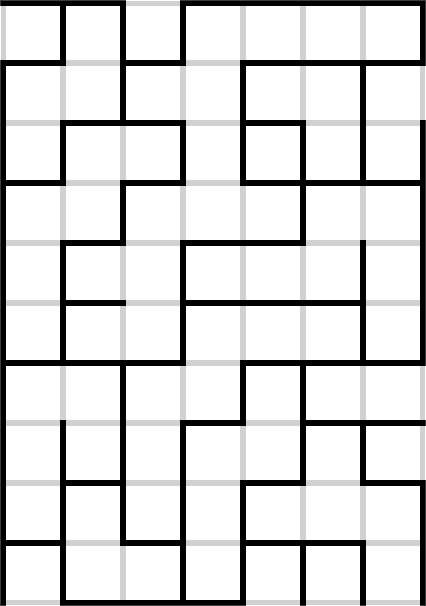
## Micromouse Competition



## Git Terminology

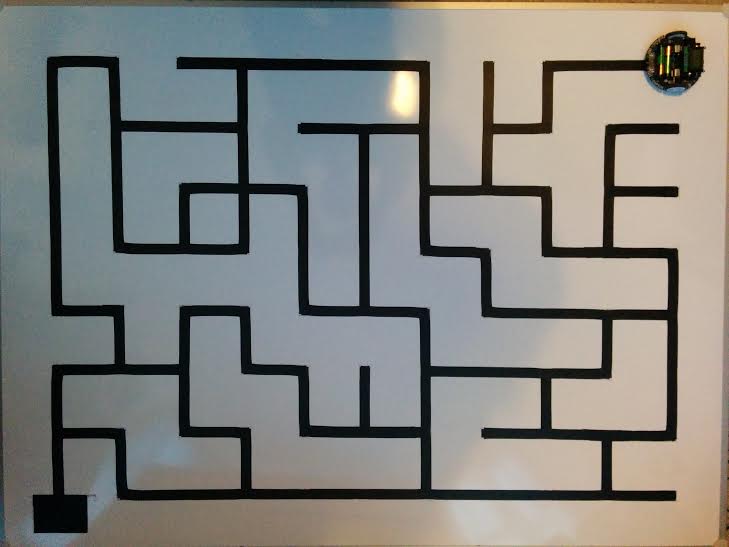
|  |  |
| --- | --- |
| **Term** | **Definition** |
| Branch | A stand-alone part of the repository. The main branch is known as the ‘master’ branch and copies of this can be made and are separate branches. Changes in one branch will not affect another. |
| Clone | A local copy of the repository, also known as the ‘working copy’. Changes can be ‘pushed’ from the working copy to the repository or ‘pulled’ from the repository to the working copy. |
| Commit | A stand-alone change. These contain the specific file changes made, who they were made by, when they were made and other similar metadata. |
| Merge | Finds how a branch differs from another, then integrates the changes from one branch into the other. |
| Fetch | Retrieve the latest file revisions from the repository without merging them into the working copy. |
| Pull | Retrieve the latest file revisions from the repository and merge them into the working copy. |
| Push | Merge files in the working copy into the server’s repository. |

## Line Maze Design



## Lined Maze Implementation

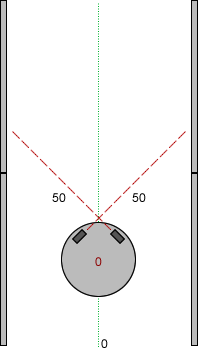
The robot is shown in the top right. A video also supplements this project with a recording of the robot mapping and solving the maze here: <https://www.youtube.com/watch?v=ONaXBkXQgc0>. This video is also attached electronically.



## Expected STL Constructs That were not available

* Templates – Built structures to suit particular types instead
* Vectors / Lists – Built a node list type from the ground-up
* Stack – Implemented a dynamic memory based stack implementation for nodes
* Classes + class scoping – Used structs with only public scoping
* Smart pointers – Resorted to using regular pointers with care taken to avoid memory leaks

## Analog PID Tuning



## Logbook

Project Supervisor: Rachel Cornes

Regular meetings took place at 12PM Wednesdays and ranged from 5 to 15 minutes. These were recorded in the form of short, informal notes, as shown below. Dates not listed are due to one of either party not being able to attend or term breaks.

|  |  |
| --- | --- |
| Date | Notes |
| 05/11/14 | Introductory session, explanation of Rachel’s role, refine project ideas for next meeting. |
| 19/11/14 | Suggestion of project ideas to Rachel, helped pick the most suitable idea. Rachel suggested seeking David Hodgkiss’ advice; the Computer Science award leader. Get idea on projector by next meeting. |
| 26/11/14 | Initial projector forms done, Rachel gave advice on how to refine the content by quantifying statements, using more formal writing and being clearer in the objectives section. Need to fill out risk assessment and submit proposal on projector for next meeting. Include statement of attendance of GradEx. |
| 03/12/14 | Ethics section not done properly, Rachel showed how the fast-track ethics form would be applicable for the project, get this done for next meeting. Start preparing for mid-point viva, contact second assessor and find out what they require seeing in the viva. |
| 10/12/14 | Rachel gave advice regarding the mid-point viva stating items I need to have prepared such as the project plan, literature review and possibly a presentation based on what the second assessor requires. |
| 17/12/14 | Rachel reviewed the initial research and analysis document and suggested  changing pros/cons to advantages/disadvantages as this is more formal. More focus on citing/referencing works and describing subject area. Use Harvard referencing. |
| 04/02/15 | Talked about how the viva went, much clearer plan established regarding the upcoming design phase. |
| 18/02/15 | Discussed my visit to Birmingham City University to talk with Micromouse expert Peter Harrison. Also discussed seeking help from a third-party with a part of the implementation. Rachel suggested talking to David Hodgkiss about this matter. |
| 25/02/15 | N/A – No progress this week due to work for other modules. Discussed what makes content design or analysis. |
| 04/03/15 | Problems with getting working components, Rachel explained how issues with the project are naturally ok, so long as they’re documented and evaluated properly later on. |
| 11/03/15 | Explained that the robot and all components are now fully acquired and assembly can begin. Described worries about not being able to complete assembly in time. Rachel mentioned how it’s still possible to achieve the majority of marks so long as proof of concept is demonstrated and documented adequately. |
| 18/03/15 | Updated Rachel on project progress – brought in robot showing basic functions such as the LEDs, buzzer and motors |
| 25/03/15 | Demonstrated robot running a small line maze, without loops, just to show functionality and proof of concept regarding navigation and mapping |
| 01/04/15 | Brought in sensors and talked about assembly of components and how this will be done |
| 20/05/15 | By this point, problems had occurred with the walled maze and the project fell back to the lined maze. Talked about finalisations and how the goals were still being achieved, regardless of the problems. |

Bibliography

Agile, 2001. *Manifesto for Agile Software Development.* [Online]   
Available at: http://agilemanifesto.org/  
[Accessed January 2015].

Auyeung, T., 2005. *Micromouse Handbook,* s.l.: s.n.

Awad, M. A., 2005. *A Comparison between Agile and Traditional Software Development Methodologies,* s.l.: s.n.

Beck, K., 2000. *Extreme Programming Explained.* s.l.:Addison-Wesley Professional.

Bell Telephone Labs. Inc, 1961. *An Algorithm for Path Connections and Its Applications.* s.l.:IEEE.

Clegg, D. & Barker, R., 1994. *Fast-track - a RAD approach.* s.l.:Addison-Wesley.

Crowe, D. A., Averbeck, B. B. & Chafee, M. V., 2000. Mental Maze Solving. *Journal of Cognitive Neuroscience,* 12(5), pp. 813-827.

E.R. Davies: Theory, A. P., 1990. *Computer and Machine Vision.* 4th ed. s.l.:Elsevier.

Ferrari, M., Ferrari, G. & Hempel, R., 2001. *Building Robots with LEGO Mindstorms.* s.l.:Syngress.

Github, 2015. *GitHub Issues.* [Online]   
Available at: https://github.com/

Harrison, P., 2005. *Micromouse Online.* [Online]   
Available at: http://www.micromouseonline.com/  
[Accessed January 2015].

Holland, J., 2004. *Designing Autonomous Mobile Robots.* Oxford: Elsevier.

Kühl, D., 1996. *Design Patterns for the Implementation of Graph Algorithms,* Berlin: Berlin Technical University.

Larman, C., 2003. *Agile and Iterative Development: A Manager's Guide.* s.l.:Addison Wesley.

Millington, I., 2009. *Artificial Intelligence for Games.* 2nd ed. s.l.:CRC Press.

Mishra, S. & Bande, P., 2008. *Maze Solving Algorithms for Micro Mouse,* s.l.: IEEE.

Patel, A., 1997. *Red Blob Games : Pathfinding.* [Online]   
Available at: http://www.redblobgames.com/pathfinding/  
[Accessed 4 January 2015].

Pearl, J., 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving,* s.l.: Addison-Wesley.

R. S. (., 1999. *Micromouse and Accessories.* [Online]   
Available at: http://www.robotstorehk.com/micromouse/micromouse.html  
[Accessed January 2015].

Sadik, A. M. J. et al., 2010. *A Comprehensive and Comparative Study of Maze-Solving Techniques by Implementing Graph Theory,* Dhaka, Bangladesh: Islamic University of Technology.

Sharma, M., 2009. *Algorithms for Micro-mouse,* s.l.: Kaizen Robeonics.

Smith, J., 2011. *StackOverflow.* [Online]   
Available at: http://stackoverflow.com/questions/6261201/how-to-find-memory-leak-in-a-c-code-project/8417851#8417851  
[Accessed 02 2015].

Weise, T., 2009. *Global Optimization Algorithms - Theory and Application,* s.l.: s.n.

Willardson, D. M., 2001. *Analysis of Micromouse Maze Solving Algorithms,* s.l.: Portland State University.