Contents

[Analysis 2](#_Toc118227756)

[Problem identification 2](#_Toc118227757)

[Identification of stakeholders 2](#_Toc118227758)

[Computational Methods 3](#_Toc118227759)

[Research 4](#_Toc118227760)

[Existing similar solution 4](#_Toc118227761)

[Features of the proposed solution 6](#_Toc118227762)

[Requirements 7](#_Toc118227763)

[Software requirements 7](#_Toc118227764)

[Hardware Requirements 7](#_Toc118227765)

[Stakeholder requirements 7](#_Toc118227766)

[Success Criteria 7](#_Toc118227767)

# Analysis

## Problem identification

Robots that can navigate small or restricted areas are invaluable in modern day engineering. They allow project managers and developers to fully map out an area, potentially for development or removal of rubble. Other applications of self-learning algorithms apply to many other sectors of the world, such as car development with calculating the path of least resistance for airflow or calculating the shortest distance between stations in a city. The nature of self-learning algorithms means they can adapt and evolve in a large array of ideas.

To continue this trend, I will hopefully be exploring a small part of this in self driving cars. I will specifically be using a car to navigate a maze using several sensors to simulate the ability for cars to detect a path to follow. I will be exploring this by creating a small vehicle and attaching either a distance sensor or a colours sensor to follow a specific path, further expanding the path and even adding different routes.

### Identification of stakeholders

|  |  |  |  |
| --- | --- | --- | --- |
| **Stake Holder** | **Description** | **Needs** | **Appropriate** |
| Myself | I will be directly working on the project. So, I will be responsible for the development, testing and main design of the system. | I need the final system to run efficiently and to be stand alone. This will prevent any repairs that need to be done. | The final system will be appropriate if the car can traverse areas by itself whilst being simple and easy to manage any problems. |
| Engineer | An engineer will be using this project to navigate and survey small spaces. | Needs the system to be robust and reliable so it will work under many conditions. | The final system will be appropriate as it will be able to work alone whilst under many different conditions. |
| Apprentice | An apprentice will be learning about the system, so it is assumed there is no prior knowledge of the system | Needs the system to be simplistic and easy to learn, whilst also being applicable in the field so it will mimic real life scenarios. | This solution will be appropriate to apprentices as it will be able to teach them how to use the car whilst also being complex enough to mimic real life. |

### Computational Methods

|  |  |  |
| --- | --- | --- |
| **Feature** | **Relevance to the system** | **Computational Method** |
| Calculations | * How long has the car travelled for * The shortest route from A to B | * The car must track how long it has travelled for to get an accurate map of the area * The car will find the shortest route from A to B, then follow that route |
| Sorting | * Needs to sort the data | * Finding the shortest route by comparing different routes in a list. |
| Comparing data | * Data must be compared to find which data point is better for the application. | * Routes must be compared to each other to find optimal route * Comparing to previous sets of data for quicker search times |
| Storing data | * Different types and pieces of data must be stored to run the code efficiently and output the correct result. | * The car needs to store the different decisions it makes * Needs to be able to store the route to compare future routes |
| Searching data | * Searches for the minimum route in a list | * The algorithm will search through the different routes to find the most efficient one * The previous routs will be searched to match to current route. |
| Decomposition | * The system must be decomposed to efficiently execute a route | * The algorithms will be split up * The different menus will be split up * The graph will be broken down into different lengths |
| Abstraction | * Removes the details to efficiently model the problem | * The decisions will be vertices on a graph |

## Research

### Existing similar solution

#### .1. AI in LEGO EV3 Maze-Driving Robot by Tony—K

Goal of solution:

To navigate a maze, finding the optimal path, then executing the optimal path.

Description of the Solution

Multiple sensors are attached to a Lego EV3 car that can navigate a maze. The maze is first navigated inefficiently to crudely map out the maze and store the different directions it took. The directions are then converted into simplistic versions, reducing three instructions to one where possible. When placed back at the entrance, the car will navigate the optimal route back to the end.

Features of the solution:

* Uses the left-hand wall following method
* Stores its path and eliminates dead ends
* Finds combinations in the movements in the maze and replaces them with simplified instructions
* Uses three different sensors as ev3 cannot manage more than one type of sensor
* Uses a 2-wheel chassis to move through the maze
* Uses a colour sensor to detect when the maze is finished

Positives and Negatives of the solution:

|  |  |
| --- | --- |
| Positives | Negatives |
| Efficient solution for solving the maze | Turning takes a large amount of space |
| Simple to reproduce for end users | Uses a bespoke program that is hard to learn |
| Reduces the movements for optimal path | Will be inefficient for larger mazes |
|  | Does not allow for multiple ends |
|  | Does not allow for loops in the maze |

A sample of code from the solution:

Graphical user interface, application, table, Excel

Description automatically generated

#### .2. Design of a maze solving robot using Lego MINDSTORMS by B.J.S van Putten

Goal of solutions:

To use LEGO Mindstorms RCX 2.0 to navigate a maze made of black on white line patterns.

Description of the solution:

Uses the Mindstorm RCX 2.0 unit to explore the capabilities and restrictions of Lego hardware by developing a maze solving robot. The program designed will be able to efficiently navigate different maze scenarios and optimise a route through the maze, pushing the limits of the computing power of the RCX 2.0 unit. This identified the different restrictions of the unit and delivered solutions to the problems.

Features of the Solution:

* Uses a programming language call NQC (Not Quite C)
* Uses the follow the wall method to navigate the maze
* Uses two motors set up to be fully rotational to move
* Uses a light sensor to understand where the maze is
* Simple program outline to follow
* Efficient for the capabilities of the RCX 2.0 unit
* Optimises the information gathered in the first run

Positives and Negatives of the solution:

|  |  |
| --- | --- |
| Positives | Negatives |
| Simple Algorithm to follow and adapt | Unable to accurately detect dead ends and 90 degree turns |
| Once the track is solved, can solve it efficiently | Battery of the robot can reduce accuracy |
|  | For much larger mazes, the RCX will run of memory |
|  | Cannot be easily reproduced |

Diagram

Description automatically generatedImages of the solution:

Right:   
A visualisation of the solution’s algorithm:

If (on black line)  
{   
Forward (left motor);   
Off (right motor);   
}

Else (on white)  
{   
Forward (right motor);   
Off (left motor);   
}

### Features of the proposed solution

#### Initial Concept of my solution considering my research

My initial concept will be using the EV3 unit to follow a maze comprised of black lines on white background.

There are two different algorithms that exist that can navigate from A to B in a maze:

|  |  |  |
| --- | --- | --- |
|  | Dijkstra’s | Floyd’s |
| Description | A simple algorithm that will compute the shortest distance from a single vertex to any other vertex | A complex, multistage algorithm that will compute the shortest distance from any vertex to any other vertex |
| Positives | Simple to code the algorithm  Much more efficient to run | Simple to implement using arrays  Returns the shortest Hamiltonian cycle |
| Negatives | Cannot be used when multiple goals must be achieved | Uses a large amount of memory |
| Efficiency |  |  |
| Suitability | Is suited towards smaller mazes with only a single goal | Is suited towards larger mazes where multiple goals must be achieved |

Due to its much more efficient nature, and not needing to compute the distance between many goals, I will be using Dijkstra’s algorithm to find the shortest path through the maze. This is different to the previous solutions, as they both use the left or right-hand wall hugging method before simplifying their results. My solution will be more durable than the other solutions, which have a 20% success rate (from solution 2).

I have chosen to use a single colour sensor to navigate the maze for now. This is because it will be much more efficient to build and modify the maze compared to physical walls. This also allows me to calculate the result of Dijkstra’s algorithm beforehand to ensure the algorithm is working as intended. Also, this method is much more cost effective than other methods such as panels to develop the algorithm and work out any flaws.

Furthermore, I will be using EV3DEV software and OS to run my code. This allows me to use Bluetooth to communicate with the car and upload my program with ease. This means I am using MicroPython to program and control the car, as I am familiar with python and will be able to develop my code using Visual Studio Code.

The user will be able to interact with the car using a GUI interface located on the LCD display of the EV3 unit.

#### Limitations of my proposed solution

The main limitation in my proposed solution is the necessity of a set up maze. This means that the car will not be very applicable to the open world in its finished state as of the previously stated specifications. This will result in the product potentially not meeting the system requirements. One solution to this will be that the colour sensor could be swapped out for an ultrasonic or IR sensor.

Further limitations of this design include the inability for the end user to change any of the code or perform any maintenance whilst in the field without any extra equipment, which means the car will not be stand-alone.

## Requirements

### Software requirements

**Brickman OS** – This is the OS that is loaded onto an SD card so the EV3 can understand and interpret the code, as well as interfacing with the motors.

**Visual Studio Code** – Used to edit and load the code onto the car

**EV3Dev extension** – This is the extension that communicates with the car to transfer the code to the car

### Hardware Requirements

**EV3 Brick and base set** – This will be what the car is built from, so will be needed to reproduce the final product.

**Micro SD Card** – The Brickman OS will need to be installed onto some medium for the brick to understand. The EV3 unit has a built-in micro-SD card reader which can be booted from.

**A Maze or path to follow and solve** – The car must have something to follow and solve to work as intended.

**A computer with Bluetooth or USB** – This allows for the program to be loaded onto the car

### Stakeholder requirements

#### Design

|  |  |
| --- | --- |
| Requirement | Explanation |
| Simple Menu | The menu must be simple so that the users can easily understand how to navigate and use it |
| Fully independent | The car being fully independent, past pressing the “run” button, means that it will allow operation to be hassle free |
| Simple Instructions with screenshots | This allows users to understand each function of in the menu |

#### Functionality

|  |  |
| --- | --- |
| Requirement | Explanation |
| Use of the EV3 buttons to interact with the software | The software on the EV3 unit will require that buttons of the unit be pressed to begin operation. This allows the user to navigate the different sub-menus and run different subroutines |
| Easy setup from box to final product | The setup process must be easy for users to recommend and continue to use the product in mass and to deploy remotely |
| The ability to remote into the EV3 unit | The ability to receive information and see the EV3 screen remotely will allow for the users to actively see what is happening with the car if they wish |
| The ability to change the speed of the motors | As the car will navigate different areas, some users may wish that the car moves slower or faster than the pre-set speed, so be able to change the speed will be useful |
| Different menus for stats and settings of the car | The user may wish to see the different values that the EV3 unit has stored or is using to allow for more in-depth debugging or logging |

#### Hardware and Software

|  |  |
| --- | --- |
| Requirement | Explanation |
| A computer with USB or Bluetooth | This will allow the user to load the code and OS onto the EV3 brick and interact with it once the code is loaded |
| Brickman OS image | This is the OS that will be used to understand and process the code that will move the car |
| Micro SD Card | This will hold the Brickman OS image as well as any code loaded onto the car |
| Visual Studio Code with EV3DEV | This will be needed to load code from the computer onto the car. However, this can be avoided by creating a premade OS image of Brickman that includes all the code which can be loaded directly onto the car’s storage |

## Success Criteria

|  |  |
| --- | --- |
| Criteria | Evidence of completion |
|  |  |
|  |  |
|  |  |
|  |  |