# Maze navigating car

Contents

[Maze navigating car 1](#_Toc117785893)

[Analysis and research 2](#_Toc117785894)

[Initial Problem identification 2](#_Toc117785895)

[Identification of stakeholders 2](#_Toc117785896)

[Computational Methods 3](#_Toc117785897)

[Design 4](#_Toc117785898)

[Main Car Menu (GUI) 4](#_Toc117785899)

[Development 6](#_Toc117785900)

[Evaluation 6](#_Toc117785901)

## Analysis and research

### Initial 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 |

## Design

### Main Car Menu (GUI)

For this project one of the success criteria is to have a functional menu that can be used to interact with the basic abilities of the car. There are different ways to design and produce a menu, each of which have their drawbacks and benefits.

|  |  |  |
| --- | --- | --- |
| **Design** | **Benefits** | **Drawbacks** |
| All the elements on one page | * Simple to program, will save a lot of time * Can be easily changed if necessary | * Cluttered and hard to read as a user * Difficult to use when testing iteratively as navigation won’t be easy |
| “Drop down” style menu | * Can break down and chunk out the different aspects of the car * Easier to navigate as each command has its respective location | * Potentially difficult to program * Difficult to change individual aspects of the menu |
| Speech driven Menu | * Has much more to offer in terms of versatility. Just add a voice line and the action. * There is already code to identify speech patterns | * Difficult to integrate whilst being time efficient. * Voice commands will need to be exact, otherwise they won’t recognise, meaning less viable in a work environment where the car will be alone. |

#### Menu Style 1

This menu style will be developed from all elements of the robot being accessed on a single page. This means this menu will be the simplicist to code but may be tedious to use for longer lists of commands. This will be perfect for early development of the car’s movement and algorithms but will not be sufficient once more commands are introduced.

The different aspects of the car that will need to be included in this early model are:

|  |  |
| --- | --- |
| **Command** | **Function** |
| Move Forward | Simply runs the motors to move forwards |
| Turn Right 90 | Tests the turning capability of the car by turning 90 clockwise |
| Turn Left 90 | Tests the turning capability of the car by turning 90 anti-clockwise |
| Identify Line | Will identify if there is a black line underneath the car. |
| Move Forward until Line | Moves the car forward until a black line is underneath the car. |

Once the car can consistently execute “Move Forward until Line” this menu can be developed further iterations of Style 2, as this one is very limited for several commands.

#### Menu Style 2

This version of the menu will be much more developed. This means the version should be developed to be much more user friendly and easier to navigate, as well as editable and well documented. This version should be seen to become the final version of the menu, however if time is available then further iterations may be developed.

The main aim of this menu is to neatly divide the commands into different “drop down” sections with the commands sorted into each depending on their function. This will allow the overall development experience and user experience to be better as commands can be navigated to much easier.

The commands that will potentially be included in this menu are:

|  |  |  |
| --- | --- | --- |
| **Sub List** | **Command** | **Function** |
| **Movement** | Simple Movement | Just moves forward and back to test that the motors are functioning |
|  | Rotate Increment | Rotates at increments of 90 degrees to test motor rotation. |
|  | Square Test | Runs a test combining 90 degree turns and simple movement |
| **Sensors** | Sense Colour | Will sense if a particular colour is present underneath the sensor. |
|  | Sense object | Will sense if there is an object at a distance and returns what distance that is. |
|  | Sense Square | Runs the main algorithm |
| **Quick Test** |  |  |
| **Run** | Run | Runs the Main algorithm of the car (Search and move) |

## Development

## Evaluation