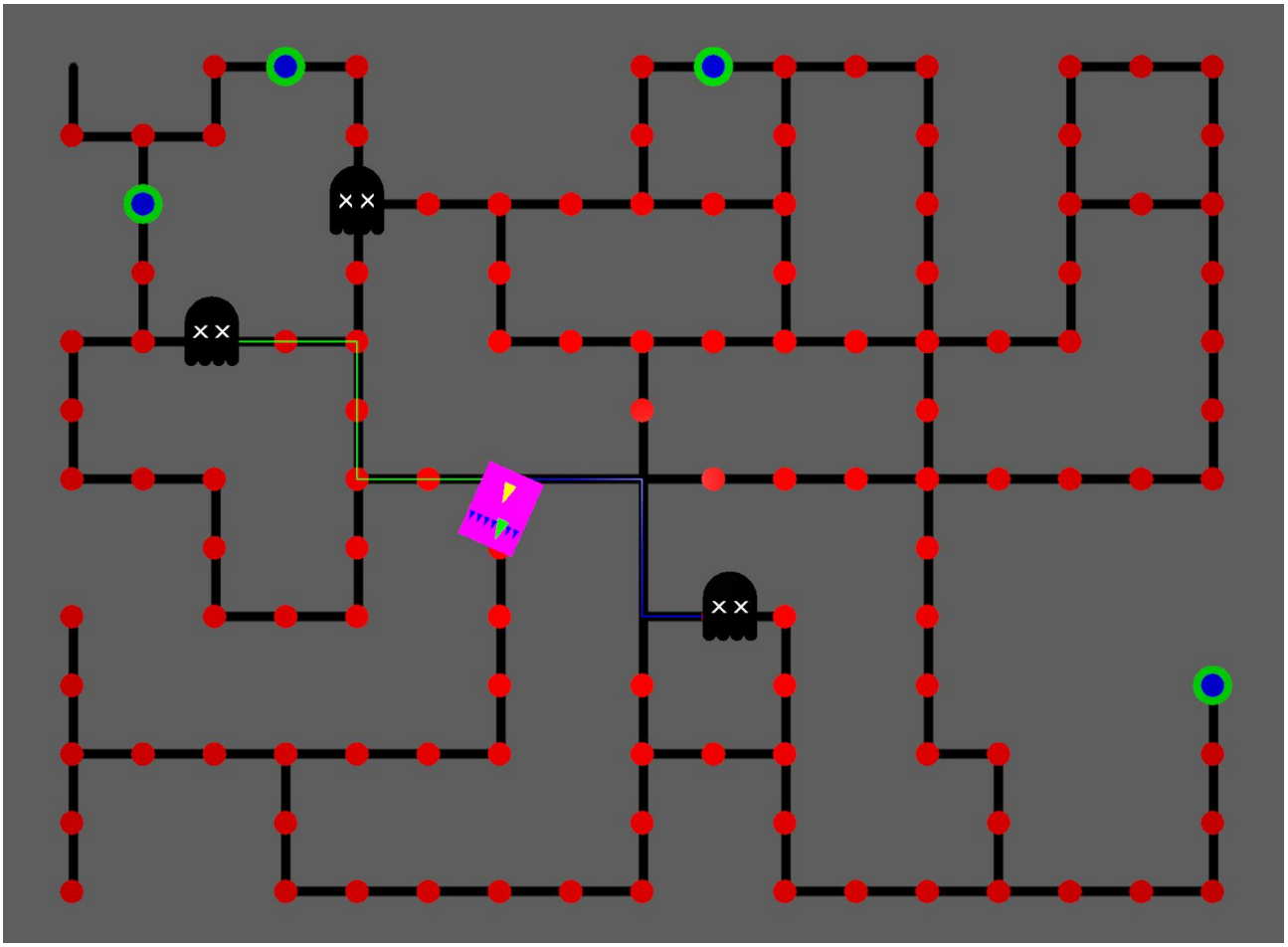
COMPSYS 301: Software Design

Completed by:

Name:	Kenny Zheng	Carrie Ng	David Huang	Jesse Zeng
UPI:	wzhe307	cng822	zhua687	jzen349



Abstract

This project aims to design and develop a software solution in C/C++ for a robot to navigate a maze and complete two levels. The group was split into pairs with each pair focusing on one algorithm, A* and BFS. The algorithms were tested on four different maps for performance evaluations. Our level one findings (I.e. map one to four) showed A* took 236s, 180s, 62s and 167s, which resulted in total distance travelled of 304, 229, 81 and 207 respectively. Likewise, BFS took 244s, 217s, 164s and 182s, which resulted in total distance travelled of 325, 273, 293 and 250 respectively. For level two, A* took 70s, 78s, 140s and 82s, which resulted in total distance travelled of 89, 90, 185 and 99 respectively. Similarly, BFS took 68s, 72s, 54s and 66s, which resulted in a total distance travelled of 83, 87, 80 and 85 respectively. From these results, we chose A* for level one and BFS for level two. Team collaboration was the key factor that resolved problems with line detection, intersection handling and robot movement. Light detection, robot movement and memory capacity are areas that would require changes to implement the code onto a physical robot.

Introduction

The software project for this course aimed to create a line following robot. Due to COVID-19 restrictions, we were tasked to implement a virtual version of the robot using C/C++ code. There are two levels of difficulty for the robot to complete: Level 1 requires the robot to eat all food particles displayed on the map while Level 2 requires the robot to calculate its shortest path. A base code was provided to us, and the two functions we had to modify were *VirtualCarUpdate()* and *VirtualCarInit().*

During the implementation process, we encountered several problems, such as the robot having difficulty determining the intersection correctly. The two algorithms we chose were A* for Level 1 and Breadth-First Search (BFS) for Level 2. This decision was made through a series of testing with different maps. Some factors include both A* and BFS had only a slight difference in completion time (700 - 1500ns) and had considered the robot's distance travelled with each algorithm.

Project Partition and Justification

Our team spent the majority of the time working in pairs, and the work was evenly distributed between the two pairs. The decision to work in pairs was due to making sure everyone had something to work on together and discuss any points that may need any clarification from another team member in the implementation process. This is due to the level 3 lockdown restrictions we had in place. As we could not meet physically, we had to compromise and meet regularly through Discord using voice chat and screen sharing. (Refer to Appendix Three for a work breakdown structure)

Before deciding which algorithm to implement, we researched several different algorithms. As a result, we discovered that A* and Dijsktra are more desirable. We discovered that A* essentially does the similar calculations to Dijsktra and all the weights for Dijsktra equal to 1. Therefore, we replaced Dijsktra with BFS. Both pairs worked on implementing these algorithms while we were attempting to find a more efficient way of implementing level 1 or level 2 first. Our pairs for the project were David and Jesse, who was in charge of BFS while Kenny and Carrie implemented the A* algorithm.

Group meetings were held regularly to keep everyone up to date with their respective progress and discuss any points that were needed to be addressed in the project. After every meeting, meeting minutes will be added for the group to view. Each pair had their own scheduled time to work on their respective parts together. This was held more frequently as the algorithm was best worked on together to avoid any syntax errors or conflict in code.

We discovered it was more effective to come together as a group when we discuss if our algorithm works as intended as it is easier to spot any errors the others could've overlooked. However, there were trade-offs as everyone had different versions of the code. It was imposed as a challenge as this means the other team person sometimes could not work on it by themselves due to any progress or changes between each meeting which may impact the current shared versions of the code. This enabled us to have efficient teamwork as we were able to simplify each other's workload by sharing our code and combining any similar sections of code. As a result, it caused us to have a better understanding of each other's code and made it easier to debug.

Algorithm modifications

For the algorithm implementation, our main focus was to integrate the line sensing algorithm with our shortest path algorithm. This is to ensure that we can sense the path correctly and to instruct the robot to turn to the correct orientation once it detects a turn. The two functions we were allowed to modify were *VirtualCarUpdate()* and *VirtualCarInit().* The *VirtualCarInit()* initialises the robot and *VirtualCarUpdate()* calculates and determines what the robot's next move is. *VirtualCarUpdate()* contributes to the movement of the robot based on the states of the sensors. A more detailed explanation of the algorithm illustrated through a flowchart is located in Appendix Two.

Key Findings

Please refer to Appendix One for the algorithm performance graphs.

For total time taken to complete level one (I.e. map one to four) , A* took 236s, 180s, 62s and 167s which resulted in total distance travelled of 304, 229, 81 and 207 respectively. On the other hand, BFS took 244s, 217s, 164s and 182s, which resulted in total distance travelled of 325, 273, 293 and 250 respectively. From figure one and three, A* achieved better results than BFS, where it had less completion time and travelled less distance for all the maps.

For total time taken to complete level two (I.e. map one to four), A* took 70s, 78s, 140s and 82s which resulted in total distance travelled of 89,90, 185 and 99 respectively. Likewise, BFS took 68s, 72s, 54s and 66s, which resulted in distance travelled of 83, 87, 80 and 85 respectively. From figure two and four, BFS wins over A* in terms of least cells travelled and lower total time.

In comparison between the two algorithms, A* and BFS have similar results for map one, two and four. However, Map three displayed the most significant difference for the time taken, and the distance travelled.

We thoroughly investigated the source that was causing the difference in time and distance between the algorithms. We discovered that the time to run each algorithm was roughly around 700ns during idle and around 1500ns when the algorithm is active. We concluded algorithms did not play a significant role in the time reduction and later discovered the method on how the robot searches for the next food played a more substantial role. For example, the BFS algorithm searches for the next food in a 2D array vertically. In contrast, the A* searches for the next food horizontally and additionally checks for food nearby at intersections.

In terms of interesting key findings, we discovered our algorithms could complete both levels with small alterations in the code. For example, for level one, the algorithm can travel to places it has never been before, therefore leave no spaces untravelled. As for level two, the robot can use the algorithm to search for the closest food.

Other small key findings showed A* required more storage space than BFS. A* had a total size of 600MB while A* required 247MB. Additionally, During the testing phase of our algorithms, we noticed the 'cells-travelled' code we used to measure distance, resulted in a significant increase in function time (i.e. Total time to run code). In some rare occurrences, the addition of the functions caused the robot to turn incorrectly at a specific intersection.

Project Hurdles and Roadblocks

In the process of designing our solution, we encountered many problems that took time to solve. These problems included but were not limited to the project specifications & restrictions and our shortest path algorithms' development.

At the beginning of our design process, we focused on level 1 of the project to have a functional line-following robot. We first forced the provided line-following code to turn left only or right only as desired for testing. Then we discussed any possible algorithms that could transverse the map and eat all the food. It was decided that the car should first follow along any path forward first, secondly turn right only and then thirdly check the intersection for any unvisited locations surrounding it. The roadblock appeared when we had trouble identifying the type of intersection. At the time, we tried to use the current car position to estimate the car's possible paths. However, we could not identify a turn from an intersection, nor were we able to find a way to determine if a forward path was present.

While stuck on the previous problem, we moved onto developing our two shortest path algorithms BFS and A* instead. After the shortest path calculation execution, we had trouble finding how to make the car follow the calculated path. Originally we created a secondary array filled with '1' where only the path would be '0'. The car's position was indexed in the map and secondary array then compared to determine where the car can move and how to turn. However, we found out later that the car position cannot be used during execution except for logging visited cells and at an intersection for decision making.

This led us back to our previous roadblock, where we needed to find a way to determine the type of intersection or turn correctly. As we could not use the car's position before confirming the intersection, only the sensors could be used as the determining factor. But the state of the sensors can not solely determine an intersection or not. For example: when the car is facing south, and the sensors [0] [1] [2] [3] are all on, this could be detecting either an intersection with a right turn and forward path or a right turn to the west direction. We solved this problem by moving the car a bit forward when a possible intersection was detected and recording any changes in the sensor states. If the sensors were all turned off, it would be a turn, and if only the side sensors turned off, it would be an intersection with a forward path. However, we saw the line-following would not process the turn with no sensors on. To fix this, "tiltsum" was set to a fixed value to force the turn.

Future Work

For any perceived avenues for future work could be the implementation of our code onto a physical robot. This would require modification of our code to make it compatible and have expected results when receiving signals and performing movements.

In terms of line detection, we would need to look into how the sensors detect the line in our code and how it could be compatible with the data being received by the robot's light sensors. Factors such as the difference in every light sensor and filters to process the input signals would be required before our code's conditions can correctly interpret what the light sensors are receiving.

In terms of robot movement, the setVirtualCarSpeed function responsible for the virtual robot's linear and angular speed would need to be expanded to generate signals transmitted to the physical robot's motors to manoeuvre as expected based on the line detection.

Another consideration is the amount of memory the PSoC board has. The code must be scaled down, so it is possible to program the board on the physical robot while keeping the line detection, robot movement handling, and algorithm.

Conclusion

In conclusion, we had decided to use A* for level one and BFS for level two. The algorithm we used for each level was determined by the time it took, and the total distance travelled to complete four different maps on each level during our testing phase.

Throughout the processes in research, design, testing, analysis, and evaluation. There have been many hurdles and roadblocks that we had to overcome to push for completion. We have learned a lot during our journey in working collaboratively as a team and making decisions that paved our next steps for future works.

For future work, we need to consider modifications to our code in areas that deal with light detection, robot movement, and memory for the code to program a physical robot and perform as expected, similar to the simulations.

Appendix One

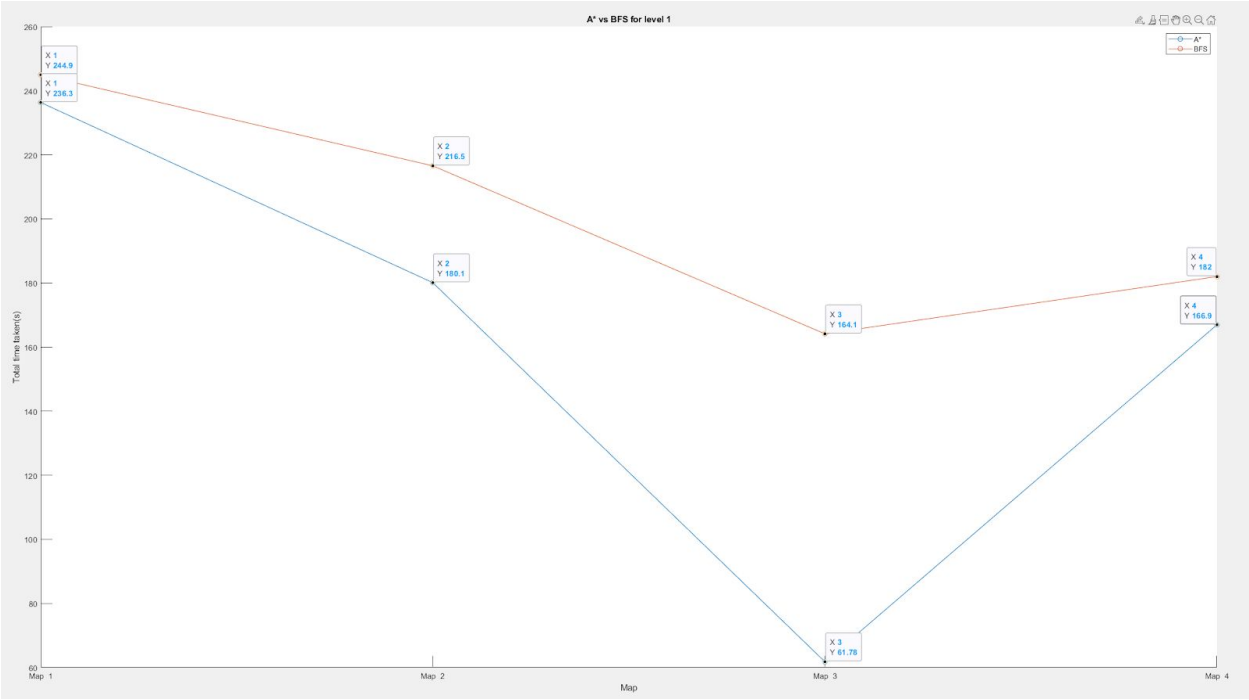


Figure 1: A vs BFS for total time taken (level 1)*

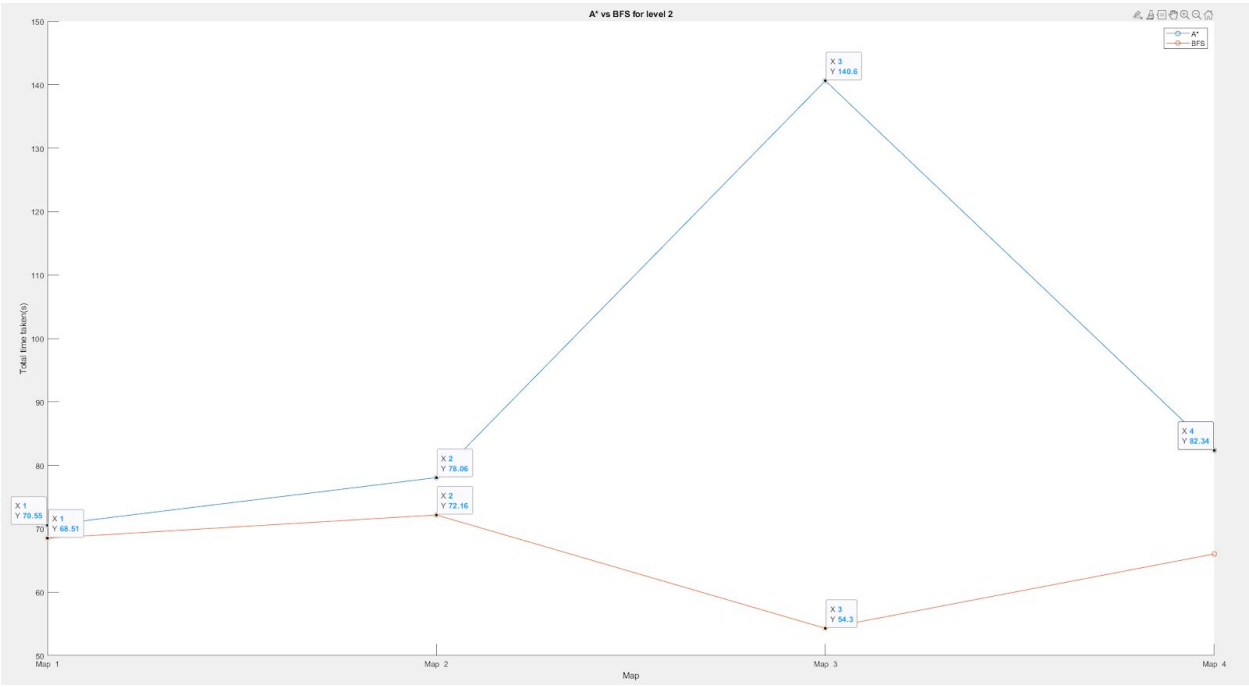


Figure 2: A vs BFS for total time taken (level 2)*

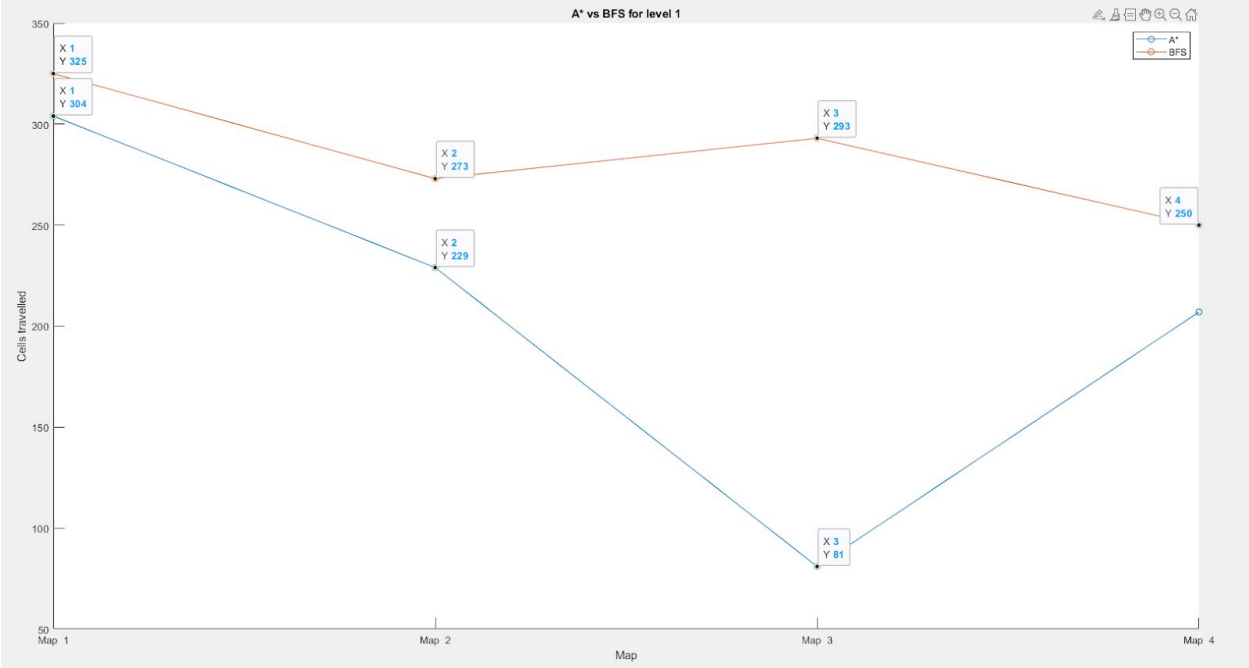


Figure 3: A vs BFS for cells travelled (level 1)*

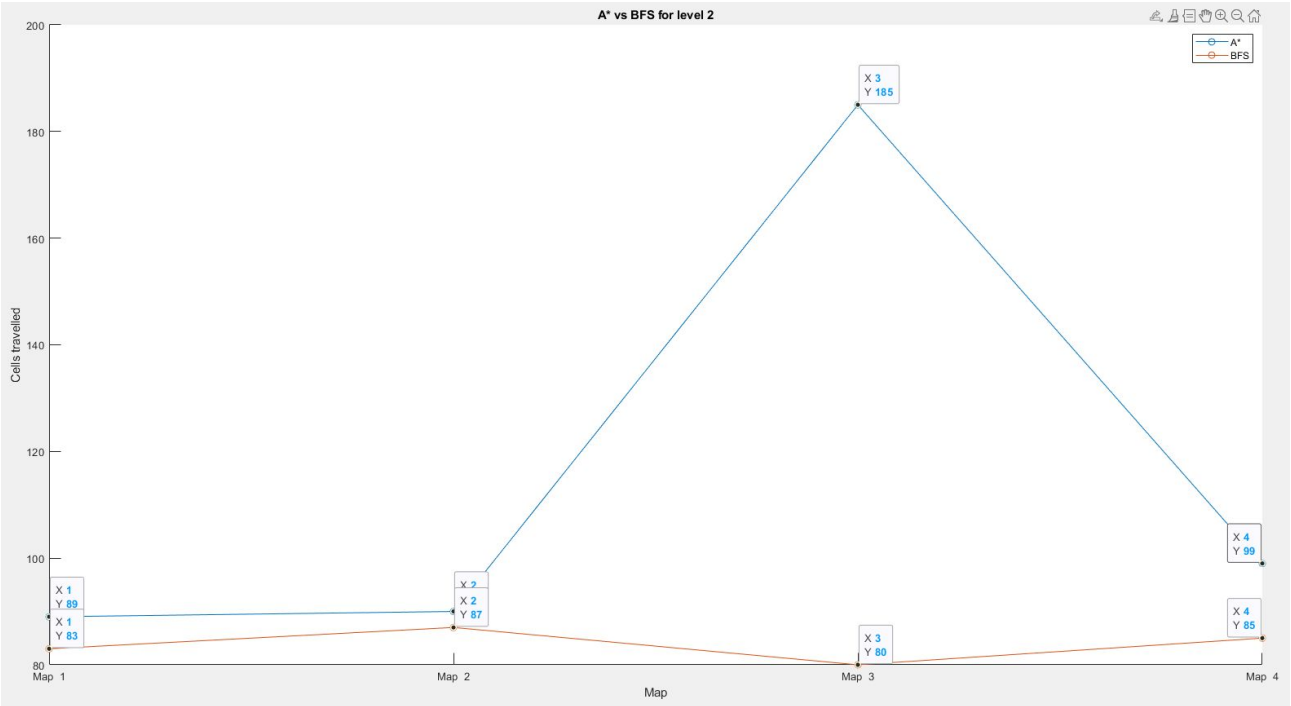


Figure 4: A vs BFS for cells travelled (level 2)*

Appendix Two

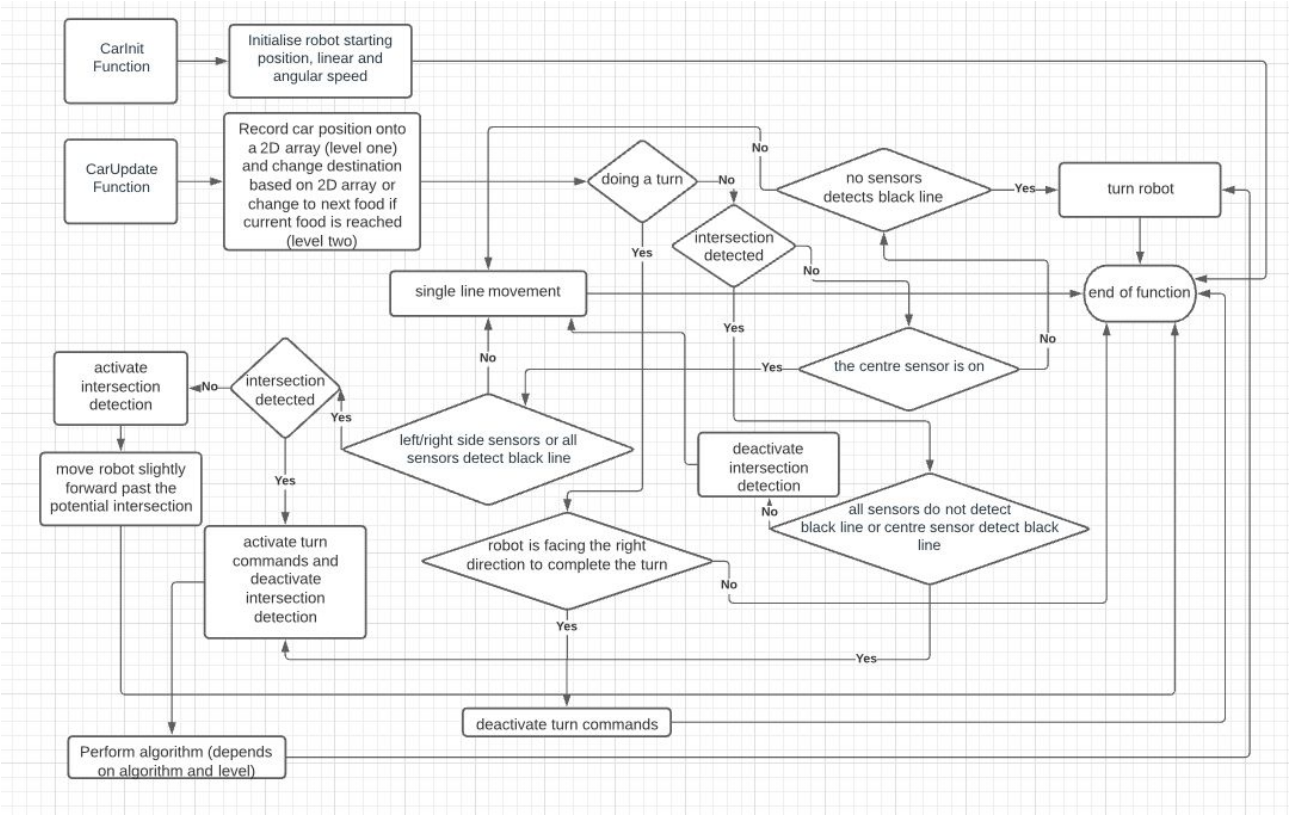


Figure 5: Flow chart of our sensing, intersection handling and robot movement

Appendix Three

Task	Members	Time
Planning	All	3 hr
Research	All	2 hr
Initial level one	All	10 hr
BFS implementation	David, Jesse	15 hr
A* implementation	Carrie, Kenny	15 hr
Shortest Path following	All	6 hr
Intersection detection	All	8 hr
Turning function	All	10 hr
Distance travelled implementation	All	1 hr
Timing implementation	All	1 hr
Testing	All	3 hr
Report	All	6 hr
Total Time	Every student	~ 80 hr

Figure 6: Work Distribution