7/06/16



Mechatronics

Assignment 4 Report:

iRobot Maze Run

Paul Dockar - 12059869

Dennis Kim - 12050466

Michael Smith - 11745167

Kimi Izzo - 11971278

Nick Rabey - 11988589

LEMONGRABS

# MemBER CONTRIBUTION

|  |  |  |  |
| --- | --- | --- | --- |
| **GROUP MEMBER** | **CONTRIBUTION** | **TASK** | **GROUP MEMBERS SIGNATURE** |
| **Paul Dockar** | 20% | * A\* (A Star) path finding programming * Report |  |
| **Dennis Kim** | 20% | * A\* path finding programming * Report |  |
| **Michael Smith** | 20% | * Writing song to EEPROM * iRobot angle and distance correction |  |
| **Kimi Izzo** | 20% | * Victim finding using grid memorisation method * Video |  |
| **Nick Rabey** | 20% | * Virtual wall, bump, cliff sensors * Report/presentation |  |

# Introduction

This report follows the maze run of team Lemongrab’s robot as detailed in assignment 3 of this course. The report will provide a high-level description of the algorithm through the use of flowcharts and state diagrams, and the challenges and solutions using the A\* (A star) pathfinding algorithm, which is widely used with maze finding robot.

Figure below is the maze for the assignment 3. The goal for this assignment was to program the iRobot (provided by UTS) to do certain tasks.

1. Play a song when starting the iRobot
2. Explore the maze
3. Find victims and when found, play a song. But when it finds the same victim, it shouldn’t play the song.
4. Return to the starting place when 2 victims (red circle in the figure) are found with the shortest path.

Where we had to utilise what we learnt in the labs and previous assignments to complete this task.

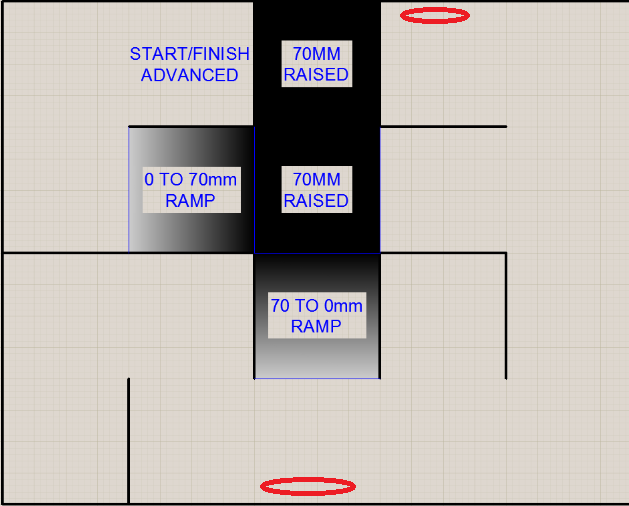


Figure 1. Maze layout

# Block diagram for hardware

The iRobot is a platform which is controlled by the DSX board through using PIC. The figure below will show the block diagram of the overall system for this assignment.

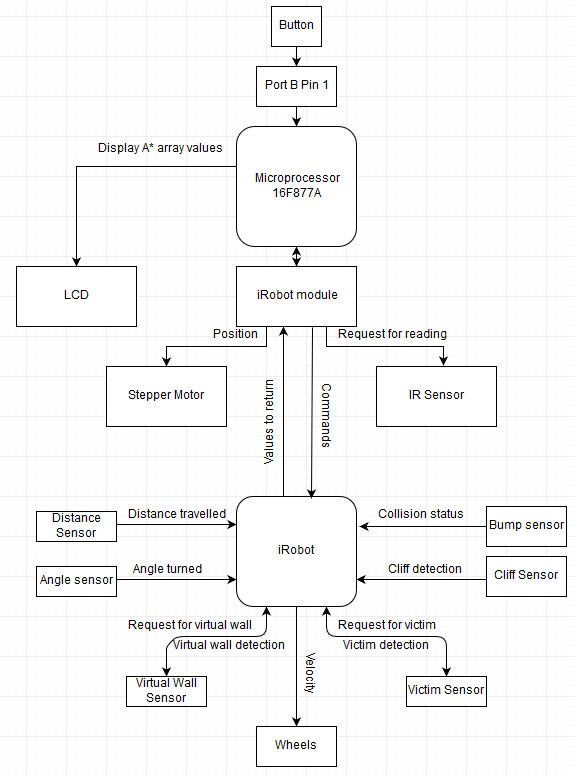


Figure 2. Block diagram of overall hardware

Figure 3 block diagram outlines the hardware design for the iRobot. DSX kit serial communication operates iRobot hardware as shown in the figure 3.

# Flowcharts

explore_flowChart.png

driveStraight_flowChart.png

aStar_flowChart.png

scanLocal_flowChart.png

# Issues and solutions

To come up with our decisions for our code, we had to brainstorm time to time and find a solution that is suitable for specific challenge.

**Challenge 1 - Choosing algorithm**We struggled to choose which algorithm to use for the assignment as there were many different methods we can use to complete the maze. It was hard to choose which method would be most efficient.

**Solution to challenge 1**With few days of brainstorming and finding different methods and that specific method works, then we decided to list our preferences and narrow our choices.

1. Hardcode vs update as we drive
2. Faster but risk vs slower but no risk

From this, we were able to narrow down to our decision to an algorithm that is used widely for maze solving robots, A\* algorithm.

**Challenge 2 - Advance maze: virtual wall detection and algorithm**  
As A\* algorithm goes to specific goal we have set in the grid with fastest solution (refer to page 6, A\* algorithm), when a virtual wall is blocking the path to the goal, it was going around to different path to get to the goal, yet if that path is also not available, it was losing its track.

**Solution to challenge 2**When we found out that enabling virtual wall was not allowing us to finish the maze, we had 2 choices.

1. Change our code to move into next goal when specific first goal cannot be reached and in our demonstration, use virtual wall.
2. Take the virtual wall out from the demonstration and focus on debugging other bugs that can be present

We chose to take number 2 as our solution as using number 1 is a huge change to our code as we have to set extra 3 goals and still causes the problem when a virtual wall is blocking more than 1 goal. Due to this, we decided an inefficient way of solving the problem which loses us a mark by not using virtual wall in the demonstration but was an efficient way of solving the problem in this situation.

**Challenge 3 – victim finder**

Struggle with playing song on same victim and integrating victim solution into a star, getting the robot to recognise the victim at a suitable distance away (it was picking it up from quite far)

**Solution to challenge 3**

* Didn’t need to integrate the victim finding function into a star
* We used only the force field IR packets, not the buoys because the buoys projected a signal from too far away
* We used flags and if statements to determine if the victim had already been found. Once the first victim had been found, the location would be logged as coordinates externally as separate variables to be compared to the map location, the first song would play, a flag would be set, and the first song played. The goal of map would’ve also have been updated. This flag would mean that once a victim had already been found, it would make a check to see if it was in the same set of coordinates, and if it wasn’t, then the second victim would’ve been found. At this point, another flag would’ve been set indicating that it was time to go home, the goal would’ve been home, and the robot was to travel along the shortest path.

**Challenge 4 – IR sensor**

After some time testing we found the IR sensor that we were using to be reading inaccurate measurements. This was impacting the quality of our corrections whilst implementing the wall follow and later A\* method.

**Solution to challenge 3**

We decided to try a different IR sensor belonging to another member of our team. From this we remade the linear equations for the distance conversion (from assignment 1) for both sensors and tested them to determine the best performing sensor.

The other IR sensor was found to be more accurate and we proceeded with that for the remainder of our project.

During our final presentation (and the moments leading up to it) however, the IR sensor became problematic, including detecting a dead-end (that was not there) on the return path during the demonstration, causing the robot to turn around.

**Challenge 5 – Compiling issues/debugging**

There were number of incidents where we were stuck on simple problems, where it wasn’t a problem such as IR sensor not working due to hardware problem or algorithm problem. There were incidents where we were getting our integers wrong in the code, which we weren’t able to find it as fast as expected, not allowing us to compile and not displaying an “error” sign for us to debug easily or algorithm failing to work.

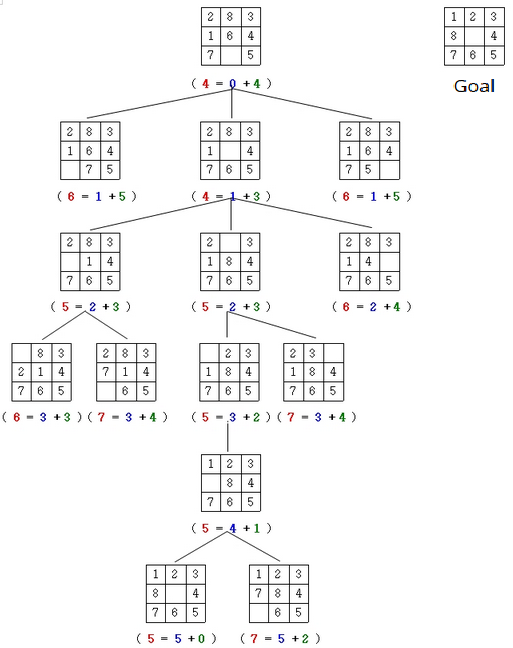
**Solution to challenge 5**

In these cases, we debugged a section of a code that had a performance error by mainly displaying values on the LCD. By doing so, we were able to see if that code was working or not and was also able to see what kind of values were displaying. By doing so, we were able to see values that were higher than what it was supposed to be due to using wrong integer value. From this, we were able to debug our integer problems easily.

# A\* Algorithm

A\* path planning algorithm is used in variety of different applications. Such as video game development to even autonomous cars.

It’s a method where it finds the shortest path among all possible path to the solution/goal with the smallest cost (shortest distance travel) as long as there isn’t a case where the goal or the node isn’t reachable. At individual iteration, A\* method chooses and estimates the cost to get to the goal node. For this, A\* uses a certain equation: **f(n) = g(n) + h(n)**Where g(n) is the iteration number and h(n) is the estimated cost.  
This example can be seen in the figure 3, where red number is f(n), Blue number is g(n)/iteration and Green number is h(n)/how many boxes you are away from obtaining same number order.



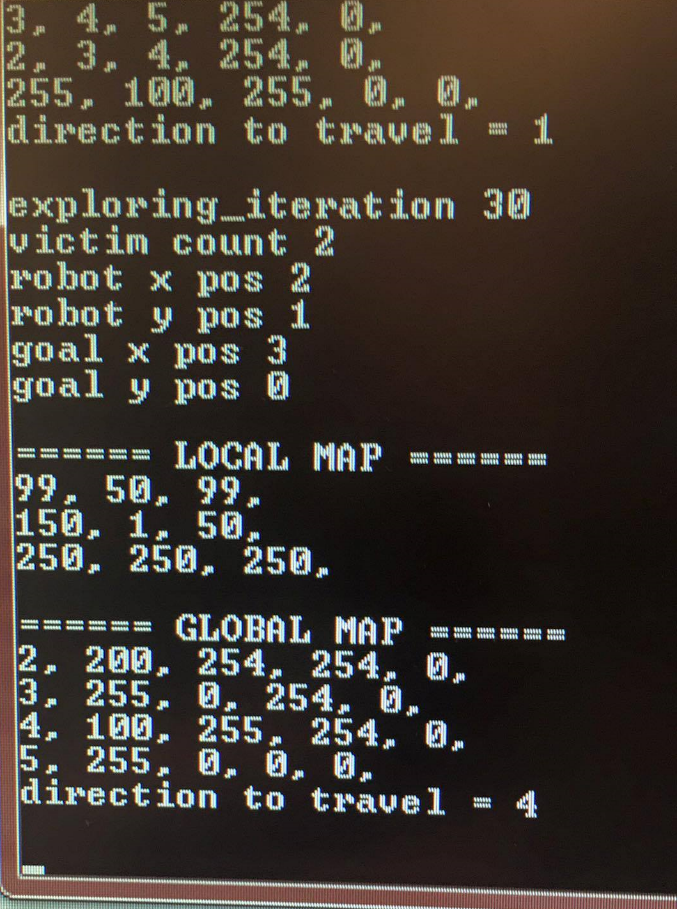


Figure 4. A\* simulation

The robot moves toward the node which has the least total value toward the goal unless there is certain condition that will not let the robot to get to the smaller value. In figure 4, global map, it is visible that the robot (100) will move to the node with value “4” this is due to wall (255) is blocking the path to the 3 and 2. Due to this, the robot will move toward the smallest cost and a path it can take, which is 4 as seen in the figure 4 “direction to travel = 4”.

# Corrections/flags

**Correction + flag/recovery (nick)**

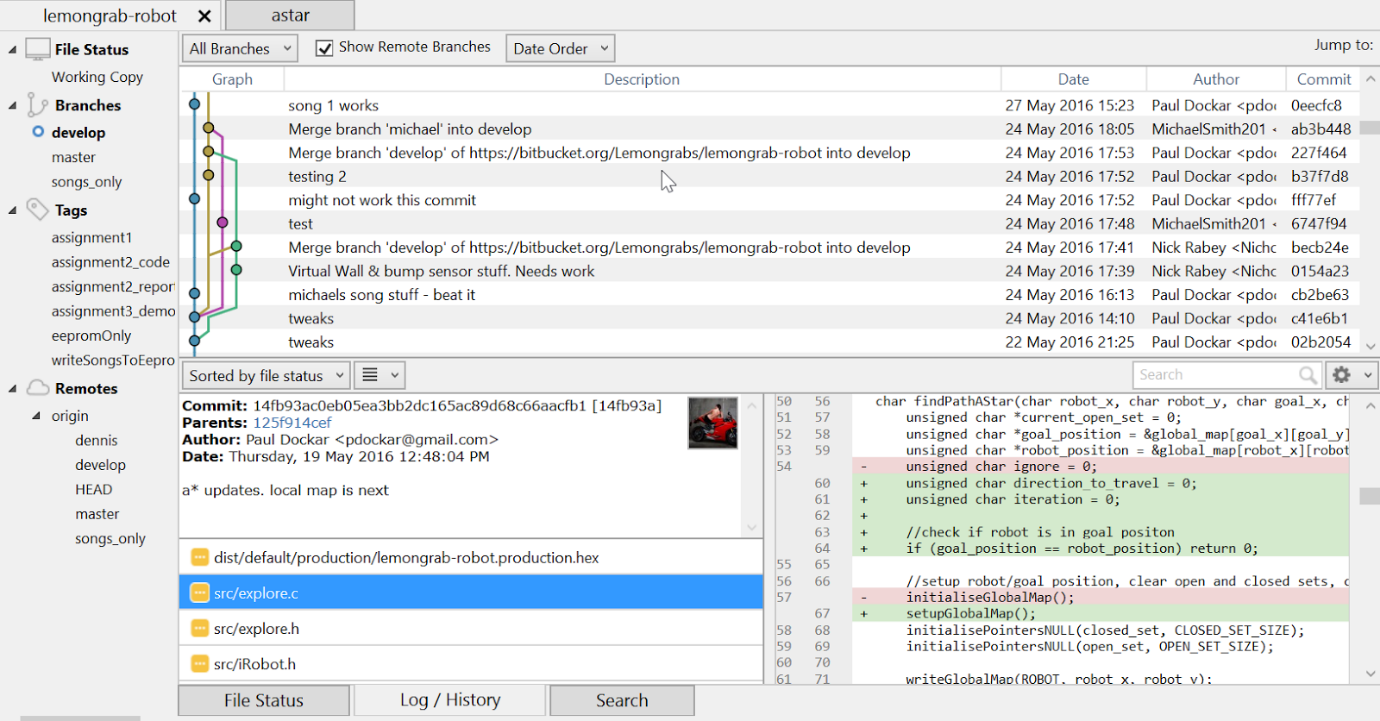
We used a wall follow method for corrections should the robot veer towards one wall. A variable named ‘manoeuvre’ was used to react. A switch case statement set the manoeuvre to either continue driving straight, slowly drift right, slowly drift left, and stop/reverse in response to the IR sensor detecting a wall close to the right, left, or directly in front.

Flags were used to good effect in our code. The bump, cliff and virtual wall sensors all raised flags that responded by updating their position on the global map.

We also used a looking left and looking right flag, depending on which wall the IR sensor was facing, and determined the appropriate manoeuver to avoid that wall.

# Appendices

**Version Control**



We used git (and SourceTree) for our version control with the repository hosted on Bitbucket. All the members in the group had their own branches to pull/push/fetch the code. Comments were left when committing the code for an update on what have been added or changed.