Assignment Report of *Maze Runner: Hybrid Controller For AllCode Formula - Buggie Robot*

Adrian Tukendorf [adt24@aber.ac.uk]

Table of Contents

1. What I managed to complete

I managed to include in my hybrid controller all 4 tasks described as:

- Exploring the maze and recording detailed information about walls, nested ares inside robot structures.
- While exploring robot is looking for darkened places in the maze called nest areas and can signalize their presence by emitting short sound signal.
- Robot can recognise when all maze is explored and finish exploration.
- At the end robot displays maze representation, showing the maze layout, position of nested areas and number of them as a digit.

  I had to test small bits of code separately in the maze which I had to build in my house from empty cardboard boxes and it seemed to work, but completed code wasn't tested in the maze located in the computer science laboratory.

2. Controller Design

The design of my controller is based on the state machine presented on the very helpful Lecture 12 - "Finite State Machines and Multitasking". This makes it possible to drive through the maze, explore it, avoid obstacles and model maze representation inside robot structure at the same time. The state of the robot is changing depending on the tasks it performs or sensor data. There are 6 states of the robot:

1 – initialisation of robot data structures setting every boolean to false, setting current position, direction, resetting cell count and turning LEDs on.
2 – scanning cells and at the same time avoiding obstacles
3 – driving through the maze using left wall technique
4 – driving straight until new cell entered
5 – displaying maze representation gathered during exploration on LCD screen
6 – finish dance and light show

How does my controller records all data about the maze inside the robot structures?
I decided to use a 2D array maze[5][5] where two "keys" are coordinates X, Y of the maze. This array holds data structure Cell with three attributes:

• bool explored	-indicate if cell was explored before
• bool nested	-indicate if cell was recorder as nested
• array bool walls[4]	-indicate if wall on specific direction is present or not.

How do I keep track of the robot current direction according to the maze?
This seemed very complicated at the beginning, but I found a solution using the modulo operator. I got inspired by using this operation to wrapping around arrays index to do circular arrays but in this case, the goal was to achieve a direction always not greater than 3. For example, robot heading is (direction=3) which means 270 degrees and it is turning right (direction++). direction = ((direction + 1) % 4 + 4) % 4 and instead of direction=4, direction is equal to 0 which corresponds to 0 degrees.

Four possible starting points?
After deeper analysis of the maze, I realised that there is one starting point. This is because coordinates 0,2 and current direction 1 is valid for four of starting points. Maze representation is just rotated on the visual LCD representation. This is because localisation of starting points and robot heading is relative to the maze not to the real world.

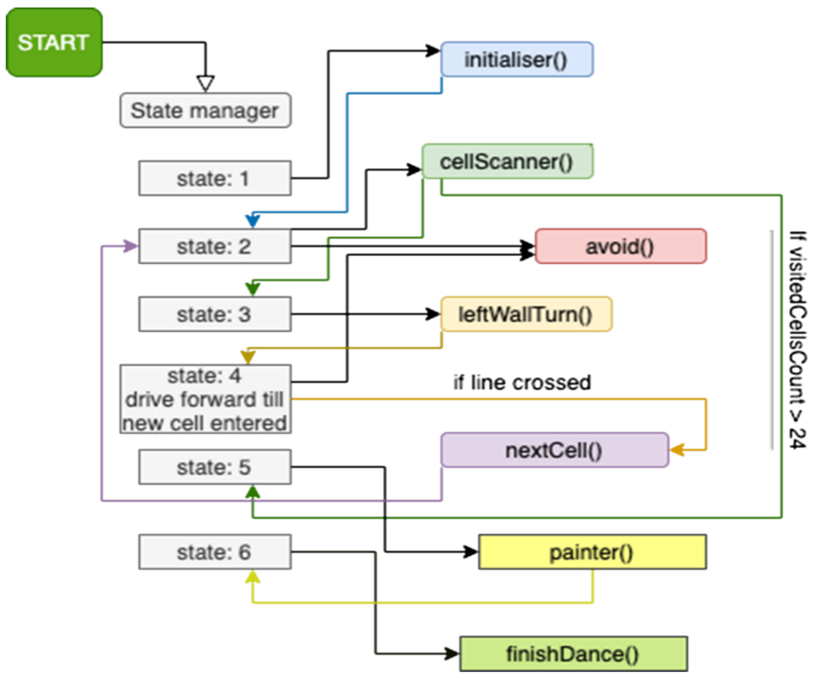


Diagram of Hybrid Controller v1.0

3. Control Strategies

The main control strategy in my controller is wall-following as a deliberative behaviour. I use left wall following strategy to explore every cell of the maze, using previously recorded information about cell the robot can identify cell and if it was visited in the past it won't be counted as a new cell. Using a deliberative and reactive behaviours combination the robot is able to avoid obstacles during exploration and detect floor lines which indicates it entering the next cell. The robot keeps track of its current position using coordinates X and Y and updating them after entering the next cell.

- cellScanner() checks if all cells have been explored, uses sensors on every side of the robot to detect presence of walls and saves it in struct Cell, checks light level and if its lower than certain value marks the cell as nested, if cell was not visited before then it is marked as visited and count of visited cells is increased.

- leftWallRule() its implementation of left hand rule, if cell nested turn around because all nested cells are dead-ends, check if turn left possible – turn left and update direction, check if forward possible – exit if-else statement go to state 4, check if turn right possible – turn right and update direction, if none of test statements were valid that means robot is in dead-end and have to turn around and update direction.

- Avoid() check if obstacle on the front – go back, check if obstacle on the right – turn left, check if obstacle on the left – turn right.

  State 4 implements driving forward and looking for the floor line using sensors underneath the robot, if floor line is detected then go to nextCell() – update current position in the maze accordingly to current direction of the robot. If current cell wasn't explored before flag as explored and increment visited cells count. Go to state 2 - cellScanner() and avoid()

4. Internal Maze Model

Internal maze model is stored in 5×5 array (maze [5][5]) of Cell structures

```
typedef struct {
    bool explored;
    bool nested;
    bool walls[4];
} Cell;
```

- Indicate if cell was explored
- Indicate if cell is nested
- Indicate if wall on specific direction is present or not by using Boolean true or false statements

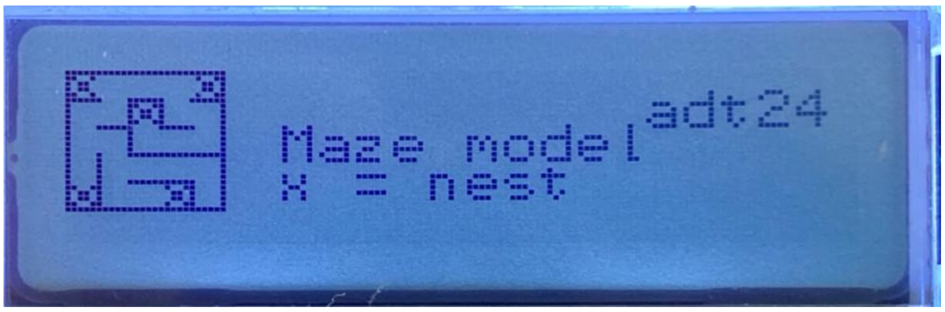
The representation is updated after entering the next cell of the maze considering the current heading of the robot. If the robot's direction is 0 which means moving straight ahead looking from maze perspective, a sensor from the left side of the robot corresponds to the left wall of the maze in internal maze representation. If the sensor is detecting a wall by checking if a value from the sensor is higher than the wall threshold value, created true or false Boolean is assigned to wall attribute in internal maze representation. It's a very simple and efficient way to update the representation of the maze.

```
if (direction == 0) { //If direction of the robot is 0 degrees then:
    maze[Pos_X][Pos_Y].walls[0] = FA_ReadIR(IR_FRONT) > WALL_DISTANCE;
    maze[Pos_X][Pos_Y].walls[1] = FA_ReadIR(IR_RIGHT) > WALL_DISTANCE;
    maze[Pos_X][Pos_Y].walls[2] = FA_ReadIR(IR_REAR) > WALL_DISTANCE;
    maze[Pos_X][Pos_Y].walls[3] = FA_ReadIR(IR_LEFT) > WALL_DISTANCE;
} else if (direction == 1) {
    maze[Pos_X][Pos_Y].walls[0] = FA_ReadIR(IR_LEFT) > WALL_DISTANCE;
    maze[Pos_X][Pos_Y].walls[1] = FA_ReadIR(IR_FRONT) > WALL_DISTANCE;
    maze[Pos_X][Pos_Y].walls[2] = FA_ReadIR(IR_RIGHT) > WALL_DISTANCE;
    maze[Pos_X][Pos_Y].walls[3] = FA_ReadIR(IR_REAR) > WALL_DISTANCE;
} else if (direction == 2) {
    maze[Pos_X][Pos_Y].walls[0] = FA_ReadIR(IR_REAR) > WALL_DISTANCE;
    maze[Pos_X][Pos_Y].walls[1] = FA_ReadIR(IR_LEFT) > WALL_DISTANCE;
    maze[Pos_X][Pos_Y].walls[2] = FA_ReadIR(IR_FRONT) > WALL_DISTANCE;
    maze[Pos_X][Pos_Y].walls[3] = FA_ReadIR(IR_RIGHT) > WALL_DISTANCE;
} else if (direction == 3) {
    maze[Pos_X][Pos_Y].walls[0] = FA_ReadIR(IR_RIGHT) > WALL_DISTANCE;
    maze[Pos_X][Pos_Y].walls[1] = FA_ReadIR(IR_REAR) > WALL_DISTANCE;
    maze[Pos_X][Pos_Y].walls[2] = FA_ReadIR(IR_LEFT) > WALL_DISTANCE;
    maze[Pos_X][Pos_Y].walls[3] = FA_ReadIR(IR_FRONT) > WALL_DISTANCE;
}
```

I approached similar solution to update if cell is nested or no. Additionally increasing count of nested cells inside the maze when light level is lower than specified earlier nested area light level.

```
if (FA_ReadLight() < NESTED_AREA) {
    maze[Pos_X][Pos_Y].nested = true;
    nestedCellsCount++;
```

After finishing exploration the robot is supposed to display representation on build-in LCD screen, I used *FA_LCDLine()* function to draw every single maze cell. This is an example result. (My university nick name normally is replaced by number of nested cells inside the maze.)



5. Sensors

There is huge variability between sensors readings on different materials, it is caused by an amount of reflected infrared light from specific materials. Dark materials are likely to absorb light much more efficiently than brighter materials. The sensor's reading shows false distance if we compare two kinds of materials in the same distance from the sensor. The sensor reading with darker material shows that it is further than a brighter one even if the distance maintains the same. This is why in my robot controller I used constant variables instead of using a specified integer value in every statement; because it's easier to adjust one value instead of editing every threshold value in every single if-statement checking sensors reading.

The variability is present also between two sensors and the same distance, readings from them aren't identical, but the difference wasn't major enough to implement an additional solution to solve this issue. That's why I am using the same threshold values for different sensors. Therefore, my code will be tested on a different robot. I had an idea to sample sensors readings as the first step after robot activation inside the maze to create threshold values automatically, but I experienced a lot of issues with this solution, so it is not implemented, unfortunately.

IR sensors are not perfect to calculate distance from the objects to the robot, but they are more than enough to detect the presence of an object in a close distance to the robot. This makes possible detecting obstacles and avoiding them by using not complicated if-statements, by comparing sensor value with a variable holding a value from the sensor when an obstacle is close to the robot. If if-statement is valid then the robot is manoeuvring to avoid the detected obstacle.

6. Problems

The biggest problem I have experienced during work on this project was a lack of test in the real maze, it was really challenging to program the robot without access to the laboratory. To overcome this issue, I created 15cmx15cm cell size, cardboard 3x3 maze to test the most crucial behaviours of the robot. Constant values probably need adjusting to improve robot's behaviours. If it would be possible, I wish I could test my robot in the real maze in the future.

I also had to overcome challenges with stopping functions like turning a certain degree or driving forward a specific length and at the same time looking for floor line. I did it using if-statement which was executed after the line was detected and motor speed was set to 0.

I came across another problem when I was developing function printing maze representation of the maze on the screen. As there was no maze to drive and later represent on the screen I've decided to hard-code example maze from worksheet 6 inside a program using other function to help me with debugging my printing algorithm. As a default, function hard-coding the maze layout should be commented out of the program. I left it here in case it might be helpful in the assessment and it shows my debugging process.

7. Possible Improvements

If I would be given more time, I would like to improve the navigation system to use more complex exploration techniques than the left wall technique. For example, to avoid exploring the same cell a couple of times by using search of the shortest path from explored cell to unexplored closest cell, and an option to visit every nested cell using the most efficient path. I think my controller needs improvements in correcting position after moving from one cell to another because I had some problems in my house but I don't know if issue is in my homemade test maze or my source code, unfortunately I wasn't able to test it properly in real maze located in the laboratory. I think my controller would be more efficient and polished, but I couldn't really help the ongoing situation with the virus.