Robotics and Embedded Systems: Maze Runner: Hybrid Control

Jac Jones (jaj58)

Abstract

This task had 4 main goals to complete, firstly the robot had to explore a simple maze and record and store information about its design using a structure or map. Then whilst exploring it should be discovering certain nested areas which were areas that were shaded and when it finds these areas the robot should signal this. Next once the whole maze had been explored the robot should stop exploring and stop and lastly the robot should output a representation of the explored maze using the earlier created structure or map showing walls, positions of nest areas and how many nest areas there are in total.

Controller Design

Exploring the Maze

The task of exploring the maze was the easiest to implement, this is because I chose to use a simple left wall following algorithm that follows the left wall the whole way around the maze until all cells have been visited at least once. I had to do this whilst being sure that the robot would not collide with the wall since this could be problematic, to do this I kept track of if there was a wall in each adjoining cell to the current one the robot was in and kept this in my cells structure as a enum as shown below:

```
1 typedef enum {
2     unexplored,
3     explored,
4     wall
5 } exit options;
```

This allowed for me to create an array of exits with a size of 4 so I could keep track of the North, East, South and West exists for each cell, making it easier to decide where to go next from the current cell. I also kept track of a cell's lighting which Is used later in further implementation and if this cell has been visited yet or not. Keeping track of a cell being visited is important because it can be useful for when we reach a dead end and must revisit cells but do not want to count the visited cells as new cells, which could mess up how we decide when we have visited all cells of the maze and stop exploring it. Keeping track of the direction the Robot was facing was an important factor to consider as well since this could effect how we keep track of where we are and also if we are still following the left wall, to do this I initially counted the direction the robot was facing when it started as North and every time it would turn I would then use a function to overwrite it's current direction that it was now facing into a global variable of current_direction. I also kept track of which direction I was facing before this new direction by first storing current_direction into last_direction, I decided storing these could come in handy for future development however I did not utilise using these as much as I had possibly hoped. I decided not to implement a function to keep track of cell changes using the black stripes that would be marked on the floor of cells in the maze, this is due to not wanting to change my code too much from it's original base and also being unsure how accurate it

would be when I was happy with my current behaviour using a pre-defined distance and keeping track of where walls are currently from where the robot is.

Recognizing Nest areas

Keeping track of nest areas was something I didn't find too challenging due to the easy use of the light sensor that is equipped on the robot. After testing some values using a shaded and non-shaded area and deciding to take an average over a small time period it was successful at noticing when a cell was indeed a shaded cell and when it wasn't. I used a former mentioned cell struct which has a variable of lighting in the struct, which had a possible 2 values of normal (0) and shaded (1). This allowed for me to easily keep track of if a cell was a nest area, and when I filled out this information in my fill_cell function I also incremented my global variable of shaded_cells, which would make displaying the total amount of nest areas at the end of the exploration of the maze easy to do.

Outputting the data

This part of the implementation was possibly one of the harder ones to figure out, and I wish I could have spent longer researching and discovering a nicer way of outputting the data for the end user to see once the exploration of the maze was complete. I ended up opting for just displaying a simple list of all the cells I had explored and where each one had walls, exits and if they are shaded or not and in what order I visited these cells. However, I wanted to create an actual map of the maze using my generated structure, but this proved very complicated and I couldn't get a correct result due to the changing starting areas and the possibility of there being a maze with a different layout of walls. I am unsure if there was possibly a way of going about this which did not require these to be known and if there is that is something I would wish to implement if I was to repeat this task.

Behaviours and Control Strategies

Internal Maze Model

The Structure

I decided not to over complicate my internal maze structure when I first designed this code because I thought if I kept it simple to understand and see what was going on that would make it generally easier to keep the code clean and easy to follow. I decided upon using 3 main enums which all where then used inside a struct named cell, this would contain all information we would need for a cell, here is the mentioned struct:

```
1 typedef struct {
2     bool visited;
3     cell_lighting lighting;
4     exit_options exits [4];//stored NORTH[0], EAST[1], SOUTH[2], WEST[3]
5 } cell;
```

There is a total of 3 variables used in this struct, a bool which keeps track of if a cell has been visited, a cell_lighting variable which is a enum of 2 values either normal or shaded, and a exit_options variable which is a enum of 3 values either the default value of unexplored (0) , explored (1) and wall(3). This is was a great way of keeping track of our exits since we created an array of 4 so an exit for each direction and this meant we could decide which direction we could and would go next. I didn't need all these exit options in the end however they could still be utilized further if I had more time to work on the code and possibly help in the end goal of creating a proper map of the maze.

Updating the structure using behaviours

I used a couple different behaviours of the robot to keep my structure of the maze updated I preformed most of these behaviours within my fill_cell function. Firstly I would check to see if the cell I was currently in was already visited, this would make sure That I don't bother re reading information I already know into the structure and also did not accidently increase my cells_explored variable which is used to keep track of how much of the maze I have currently explored. I then used the robots IR Sensors to keep track of if a wall was in the way of a possible exit from a cell, after checking each possible exit direction and filling out this information into my cells data structure I then moved onto filling out the lighting variable of the cell. To do this I used the robots Variable Light Sensor that is on the front, I took an average of 3 values using this sensor spaced over 1-2 seconds and then checked this against a pre calculated value which I had determined in testing to be a good value to decide if the cell was indeed a shaded one or not. If a cell was determined to be a shaded cell the robot would fill this out into the cell information and play a little tune and print out that it had done so on the LDR screen. Once all the structure had been updated the robot would then carry on and access this filled out information to decide on where best to go next and what actions to take.

Sensors

What was found out

When I was testing and using the robot's sensors for the first time, I discovered that they are not always the most reliable things to trust and taking things like average readings is a good idea where possible. Firstly when using the IR Sensors there is many variables to consider that can modify their return value, things like the object that is in the way even when the distance to this object is the same can give different values due to it's colour, brightness/darkness and so on. Also when using the light sensor I found it's behaviour to be quite unpredictable and did a good amount of testing of values in different conditions and realised that it's going to be very hard to know for sure what sort of lighting we will have when navigating the maze since it can depend on the room's lighting as to how we react to finding nest areas.

How this was useful

Doing a lot of testing using the robot's sensors was useful in many ways, firstly I realised that taking averages for values where possible would be a good idea, and also testing in the atmosphere the robot would be in when navigating the actual maze would be ideal to get a better idea for the sort of values to expect and their range that could be seen. This would help to prevent the robot reacting to the sensor value read incorrectly and then causing incorrect behaviour. I also realised how unreliable using the downwards facing line sensors are and this impacted on my decision to not implement this feature, however it could have come in useful for making sure the robot does not lose track of being in a certain cell and incorrectly moving into the wrong cell or something similar which could effect the end resulting created maze structure map. I also could have used the IR Sensors to correct my position if needed after moving and turning within cells since the robot's motor's could be slightly incorrectly tuned and instead of turning exactly 90 degrees it might have turned 95 and so on, this could be corrected using the extra IR Sensors however I decided not to do this due to time constraints and also that when I started implementing this it started over complicating the code and ran slower and did not work as expected.

Conclusion

Overall I am quite happy with the end result of this controller, it can navigate the whole maze successfully and record the data it discovers in doing so in a internal data struct, it also discovers nest areas whilst doing it and then once it has explored the whole maze it knows to stop exploring and indicates some of the end data it retrieved. However there is room for improvement in outputting the data the robot retrieves and if there is anything I wish I had more time to develop it would be this part of the controller, by doing more research and testing to figure out an effective and working way of displaying all the data which would in return show a nice map of the maze explored including showing where walls are located, where all the nest areas are and where we started/finished. In conclusion I am happy with how this controller turned out but there are areas with room for improvement that I should take into consideration with further new projects similar or not so like this.