CS26020 Assignment Report

Maze Runner - Hybrid Control

Sam Matthews - sam82

1 Controller Design

The overall design is a simple reactive system made up of a series of FSM's and some miscellaneous functions. The main behaviours are avoiding obstacles, keeping straight, detecting lines, updating the maze model and checking if the maze has been completed. Each of these behaviours are called in the main loop until the maze has been explored completely. Each behaviour is a series of simple if-else statements that monitor the robot's environment.

The robot explores the maze by following the left wall. This is done by turning right at every front-facing wall and turning left when there is no left wall. In the case where both conditions are true, the robot will turn left as the gap is prioritised due to the nature of wall following to solve mazes. The left turning movement is integrated into the line reading to more effectively detect a change in position. When the robot passes a line and moves to the next square, a timer is started. When the timer reaches a certain point and if there is no wall to the left, the robot turns to the left. Encoder data is also taken into account as the robot can sometimes take longer due to course correcting behaviour.

The robot keeps itself straight by turning three degrees away from a wall when it is too close. This used to also change motor speeds to account for performance differences, but this was later found to be unhelpful and simply exaggerated the issue when it got stuck and made too many adjustments. The line reading simply keeps track of whether the robot is on a line or not. If the readings are low and the robot is not on a line that means that it is beginning to pass over a line and vice versa for coming off of a line.

Once a line has been fully traversed this means that the robot is in a new square in the grid. When this happens, the location of the robot as well as any location information is updated. Once this process has occurred for every square, the task is complete. The robot will then do a victory dance and traverse the maze until it has found the nest.

2 Internal Maze Model

The maze is stored as a 4x4 array of structs containing details about all four sides of the square and if it has been visited. A zero means that a side is a wall and a one means that it leads to another square. The location of the robot is represented using coordinates referring to the corresponding array index and the location is updated every time a line is traversed.

At the start of the program, the position of the robot is assumed to be 1,0 as specified in the assignment brief. To cope with the robot going too far in any direction according to the internal model, the location is updated using a circular array strategy meaning that the layout of the maze is not strictly the same as a visual interpretation of the internal model. This is not entirely necessary for this assignment as the starting position is guaranteed but it is convienient to be able to start anywhere in any position.

3 Implementation

I have been able to implement all of the basic features and functionality of the assignment and have currently not implemented any flair features outside of basic nest finding. The robot finds the nest by simply exploring the maze as opposed the utilising the model. Most of what I have done is based on the suggestions given

in the worksheet, however, some aspects I thought to be unnecessary such as dealing with dead-ends as a separate case as opposed to detecting a wall and turning twice.

The robot does not drive for set amounts of distances but simply drives ahead continuously and focuses on only turning right unless there is a gap on the left. The success of the robot is fairly consistent from different starting positions, though there are some strange behaviours like turning left too early. However, because of the reactive avoidance behaviour, these sorts of issues do not impact the performance.

4 Problems Encountered

There were various issues regarding avoidance and corrective behaviour. One problem involved keeping the robot straight in response to the imbalance in motor performance. The problem encountered is that sometimes the robot would not move away from the wall enough and be stuck trying to fix the course. This resulted in issues involving timers running out prematurely and keeping the CPU busy to the point where lined were not being read. This was corrected by making the left and right sensors less sensitive, making more dramatic corrections and disallowing course correction while reading lines.

Another problem that was fairly major was knowing when to turn left. Originally, the robot would start the procedure to turn left when there was no wall to the left. However, this did not work when there was another immediate left turn due to there being no wall before or after the ideal activation point. The system was changed to activate when a line had been passed as that is a concrete signifier that a potential left turn is approaching.

I originally had conditions for making large adjustments by moving backwards and turning left or right. These ended up not helping later on and actually made keeping the robot straight harder. The robot would get stuck in corners, would not update its orientation correctly and would sometimes turn unnecessarily.

There was also an issue regarding the internal model and correctly updating the robot's position. I originally wanted to update the entire model to deal with not starting at the origin by shifting the rows/columns along once. However, this never seemed to work after extensive testing so that idea was replaced with using a circular array implementation.

5 Potential Improvements

Given more time, I would implement the suggested bonus features such as finding the nest using the maze model and drawing the maze. I would also like to implement some better behaviours for speed correction, angle correction as well as special cases like occasionally getting stuck on corners and edges. I've intentionally made the implementation very simple and have not needed to use many of the IR sensors. Use of these sensors could possibly make turns smother using curves as opposed to just turning 90 degrees.

I did attempt to navigate back to the nest by leaving markers in each cell after discovering the nest and simply backtracking, however, this has not been implemented. Ideally I would like to not use backtracking and instead plan an efficient route using the maze model.