**Embedded system Design – Maze solver**

Design choices:

The first task we decided to implement was to make the navigation through the maze consistent. Knowing that the path was composed of only 90 degrees turns, we optimized the turning mechanisms to reflect this reality. We initially started with the idea of using only timers to control the turns, but we quickly realized it required a lot of micro tweaking, and it was dependent on the battery power level.

We then opted for an IR sensor driven only turn mechanism. This method was promising, be we realized that because some of the paths were close together, there was a lot of edge cases to manage to make sure the turn didn’t end prematurely by identifying another part of the maze as an ending signal.

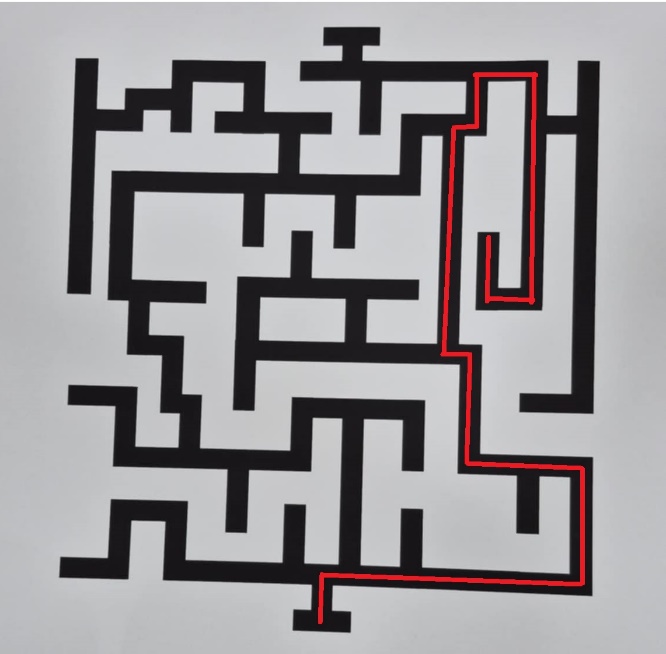
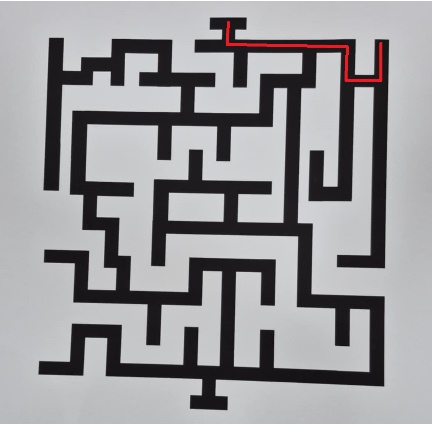
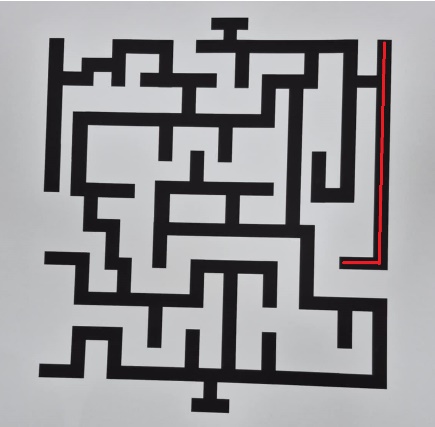
Finally, we opted for a hybrid solution using both timers, for minimal and maximum turn time, and IR sensor driven mechanism for the turns. The minimal turning time prevented the detection of perpendicular or nearby path to end the turn prematurely while the maximum time made it so it wouldn’t miss the turn if the sensors were slightly unaligned. The IR sensors were used for the rest of the functionalities like intersection detection, possible path identification, dead end handling and to readjust the line to the center of the robot when it got unaligned.

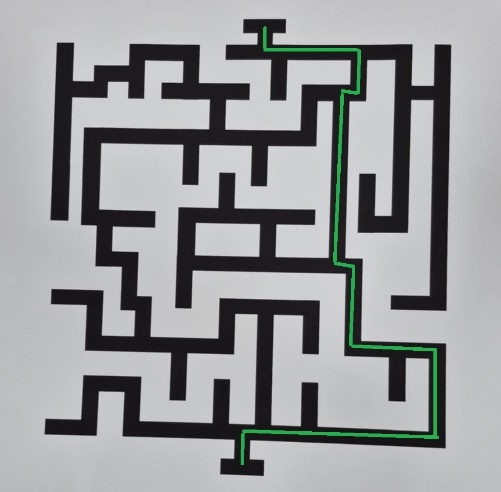
Phase 1 – algorithm for the maze escape:

For the maze escape algorithm, we decided to go with a simple *right Hand on the wall algorithm*. We chose this simple algorithm because we identified it would be a choice that would limit the amount of exploration the robot would have to do to solve the maze with our setup. These are the basic rules of the path selection of our algorithm:

* When reaching an intersection, identify the possible directions the robot can take (right, left or straight).
* The choice is made based on the idea that if someone was in the maze he would try to keep a hand on a wall on its right. This results in the following priority for turn choices:
  + Highest priority = right turn.
  + Then continue straight.
  + Lowest priority = turn left.
* Following this priority, we expected the robot first trajectory before hitting a dead-end to be the following:

Une image contenant art, Rectangle, motif, carré

Le contenu généré par l’IA peut être incorrect.At each intersection, the possible directions are added to an array. When reaching a dead-end, the robot will go back to the last intersection where it add another possible direction to explore, and based on the direction priority, choose the next place to explore. The explored but leading to dead-end intersections are removed from the array. Following this logic the next iterations of choices made by the algorithm should be:

 Once the robot reach the exit it will stop and the remaining directions inside the array should represent the optimal path the robot should take on the next run which should be this one :

Phase 2 - actual results :