Robotics: Final Task Report

Author: Gabriel Vanca

25 May 2016

I am a member of Pair 7 and together with Eduard Ursinschi I have to work to find the best solution to solve the Maze task for the Robotics Module.

We have started by developing two separate pieces of code which we have tested both through the simulator provided and using the actual robots in the lab classes. We have finally pieced together the code, taking phase 1 (the mapping phase from my code) and phase 2 (shortest path phase) from Eduard's code.

Phase 1 is based on Tremaux's Algorithm (a variant of a depth-first search basically) that requires keeping a map with the number of steps that we have gone over each cell. When the robot arrives at an unvisited junction, it picks a random unmarked direction if this is available (if not, it goes back). When arrived at an already visited junction, the robot goes back if it's on a new path, or it picks a new direction (preferably unvisited) if it is on an old path (a return path). Because no cell is marked as an end cell, the algorithm makes the robot return to the starting cell by default.

For cell mapping, what we do is that we go through every single cell on the map, we rotate the robot around and we scan the cell using the PING))) sensor.

Between phase 1 and 2, all that we do is rotating the robot, we pause the robot, activated the LED light and computing the shortest path from the starting cell to the final cell. The way we do that is basically by using a type of breath first search algorithm adapted to work on a matrix – specifically: Lee's algorithm. We compute the path with the shortest number of cell and we store it in an array. This does not ensure to provide the best time, so we plan to replace the BFS with a version of Dijkstra's algorithm (or maybe Floyd-Warshall's Algorithm if it proves to be easier to implement – we have to keep in mind that the map only has 16 + 1 cells) that would also take into consideration the number of turns the robot takes. That is because taking a turn (rotating by 90 or 180 degrees) might take the same time as going over a cell. We plan to use this better implementation for the completion.

For phase 2, we use drive_goto() to follow the stored path and to arrive in a very good time from the start cell to the end cell. We also plan to replace this by using drive_speed() and maybe even dead reckoning for the competition. We have written the code but this failed during testing so we used the drive_goto() variant as this was easier to work with while we figure out how to solve it more efficiently.

It is worth mentioning that we are also using the left and right IR sensors to make sure that the robot remains steady on his path and does not deviate due to imprecise turns (a 90 degrees turn has proved to be actually somewhere between 85 and 95 degrees; multiple turns will create undesired/unexpected results).