Dynamic Maze Solver: Autonomously Scene generation and Navigating thymio

Marzio Lunghi Jorge Espinar

June 1, 2023

Abstract

This project shows the development and implementation of an autonomous navigation thymio to navigate a maze that is randomly generated and spawned in Coppelia using the API, where the thymio navigates from a start point (a node of a graph) to an end point (another node of the graph). Differents path planning algorithms are implemented to get the optimal path.

1 Introduction

We solve (navigate from start to end) a maze using a thymio. It necessitates the incorporation of several strategies from the domains of robotics (ros2 and Coppelia) and software engineering. This labyrinth is generated and spawned in Coppelia by using ZMQ Remote API. We used a variety of path-planning algorithms to traverse it, each of which is capable of calculating the fastest route from one node (position in coppelia) to another. This document discusses the maze creation, path planning methods, maze spawning, maze navigation, results, and mistakes we made throughout this implementation.

2 Methodology

The maze creation is done by using a depth-first search algorithm, which is a well-known method for generating mazes, it generates the maze being sure that there is always one path from the start to the end. The maze is then translated into physical walls to spawn it in the CoppeliaSim simulation environment. This translation creates a faithful representation for the thymio to navigate while taking into account the maze's size and complexity. Finally, we use path planning to create the thymio's optimal path from one end of the maze to the other. This path planning method takes into account the maze's layout. The thymio is then directed along this path, and the maze is successfully navigated.

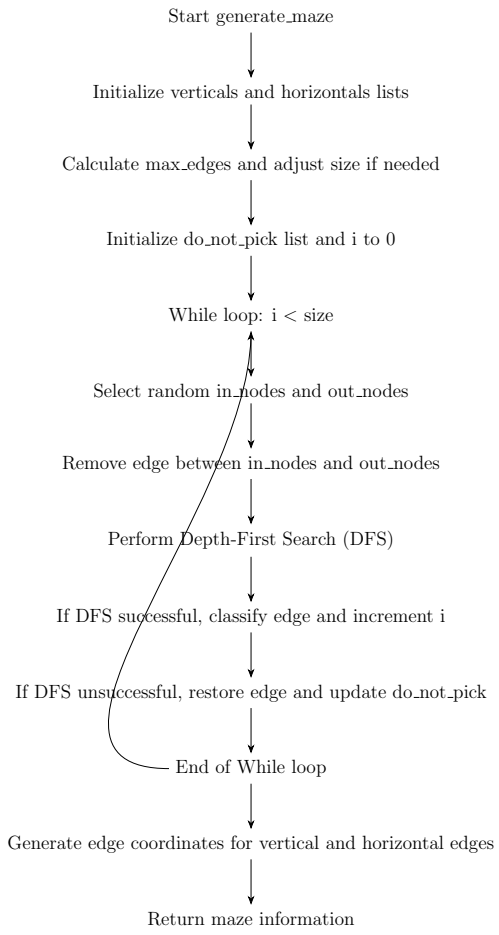

Figure 1: Flowchart of the generate_maze Algorithm

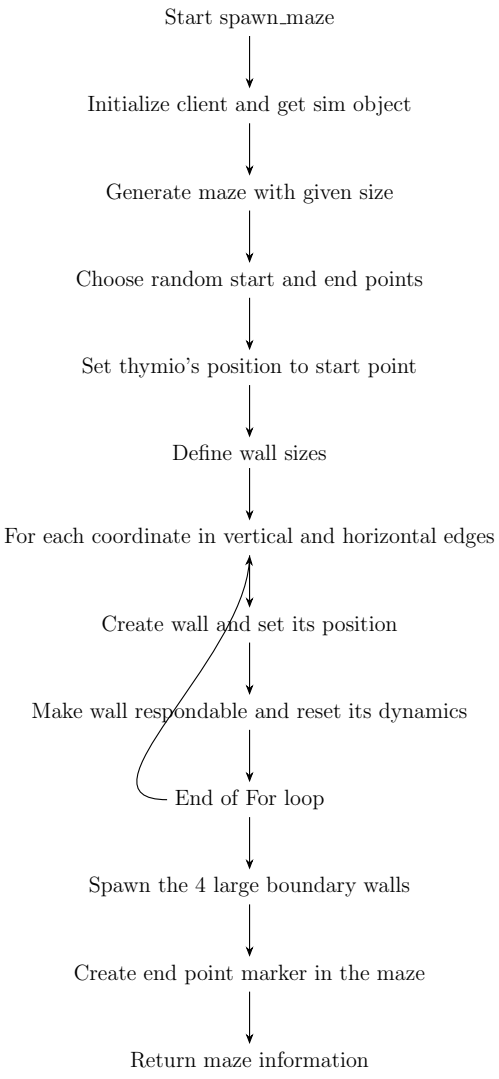

Figure 2: Flowchart of the spawn_maze Function

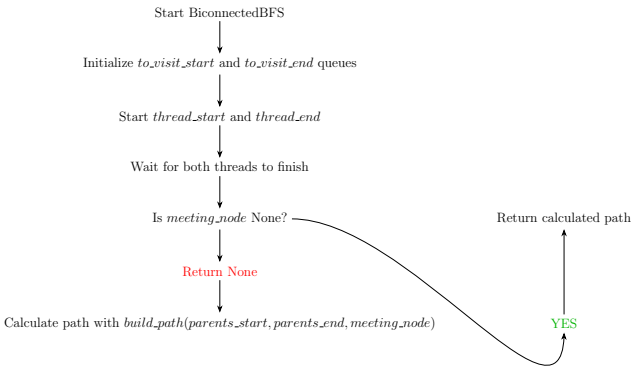

Figure 3: Flowchart of the BiconnectedBFS Algorithm

3 Explanation of the algorithms

In the previous section, we explained how our algorithms work. In this section, we will explain the final algorithms in more detail.

3.1 Maze Generation and Spawn

A graph containing nodes and edges must be created to generate a maze. Using a depth-first search technique, we create the maze and spawned it into the CoppeliaSim environment.

The maze that the thymio must traverse in the CoppeliaSim is physically represented by obstacles (walls). The beginning and ending positions of the maze are chosen at random.

3.2 Path Planning

The goal of our path planning technique is to determine the most direct route from the maze's entrance to its exit. We developed appropriate path planning algorithms that takes into account the layout of the maze.

Concretely we implemented: BFS, DFS, BiconnectedBFS similar to RTT Biconnect) and a Dijkstra.

3.3 Maze Navigation

Without using the path planning we implemented follow-right-wall algorithm and a Runtime BFS algorithm without previous knowledge of the maze.

With the path planning we used the BFS biconnected algorithm to navigate the maze. When we move the thymio, we get its position from Coppelia and send it to its next position in an automatic way.

4 Results

Our project had encouraging outcomes. The produced mazes were successfully navigated by the thymio and also they where different for every run of coppelia. Also, we made work our

path planning algorithm by successfully mapping coppelia to a graph and finding the shortest path between two nodes.

Our thymio was always able to complete the maze, independently of the maze's size or complexity in a reasonable amount of time depending of the algorithm, and the path planning algorithm was able to find the shortest path from the maze's entrance to its exit.

This is an example of a maze generated by our algorithm:

This is an example of the maze being spawned in coppelia:

This is an example of the thymio navigating the maze with the follow-right-wall algorithm:

This is an example of the thymio navigating the maze with the path planning algorithm:

This is an example of the thymio navigating the maze with the marzio real time algorithm:

5 Errors

The implementation of the project encountered a number of problems despite the thymio final effective navigation. The absence of collision detection with the maze walls was one of the key problems. The precision of the robot's movement and the efficiency of the path planning algorithm might both be impacted by the thymio's periodic collisions with the maze's walls.

The project's usage of out-of-date APIs was another problem. Even while these APIs were still usable, using them may cause issues later on since they might be altered or eliminated, which would perhaps cause our implementation's functionality to be broken. Later we discovered that have to use ZMQ Remote API instead of the legacy Remote API.

We also had many problems until we got the maze generation. We had to change the algorithm many times until we got it right. We also had to change the way we were spawning the maze in coppelia, because we were spawning the maze in the wrong way. We also had to change the way we were mapping the maze to a graph, because we were mapping it in the wrong way. We also had to change the way we were doing the path planning, because we were doing it in the wrong way. We also had to change the way we were navigating the maze, because we were navigating it in the wrong way. blablabla

6 Conclusion

In conclusion, our experiment proved the thymio's capacity to independently navigate a dynamic maze. The overall findings were encouraging despite a lot of implementation difficulties.