Path Finding and Traffic Simulation with Zumo Robot

LE-GAO CHEN[†1] CHEN FENG[†1] TIPPORN LAOHAKANGVALVIT[†1]
PEERAYA SRIPIAN[†1] MIDORI SUGAYA[†1]

Abstract: Autonomous robots and path finding algorithms have been used to progress autonomous vehicle studies. Being able to determine the shortest path from one point to another and to execute the given path is the core of the future of autonomous transportation. In this study, we employed an Arduino-controlled Zumo robot to simulate the path finding and traffic based on Dijkstra's algorithm for finding the shortest path between two points on a given map while considering traffic according to the paths generated by the algorithm. In addition, we employed PID control and compass to enable line-tracing feature so that the robot can move along a predefined path. Combining those features, we succeeded in developing a simple simulation of path finding and traffic using Zumo robot.

Keywords: Path finding algorithm, Line tracing, PID control, Zumo robot

1. Introduction

With advancements in autonomous driving as well as IoT systems, it is not unimaginable for future vehicles to be able to drive from point A to point B taking the fastest route while considering the location and destination of other vehicles for accurate traffic predictions.

Path finding algorithms works in many different areas, most notably in applications such as Google Maps where a shortest path from point A to point B is required. Path finding algorithms can be used in any situations where the structure of the system is that of a graph.

There are currently many path finding algorithms all with different benefits as well as downsides and the use of them are widespread. Google Maps uses an algorithm called A*, which is an algorithm with the fastest average run time [1]. It is an algorithm that is based off the algorithm being used in this study called Dijkstra's.

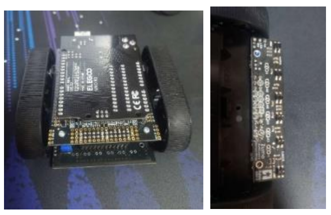
Our goal in this study is to simulate the shortest path finding algorithm while considering traffic from multiple vehicles and to project the algorithm onto a robot as a prototype. This paper presents our implementation for path finding and traffic simulation visualized on a Zumo robot, which was run in a physical map using the combination of our implemented simulation and the PID-controlled line tracing.

2. Simulation of Autonomous Vehicles

2.1 Hardware

To simulate an autonomous vehicle, we used an Arduino-controlled Zumo robot (Fig. 1a). The robot has various optional components, in which we used three components for our simulation as follows:

- Motors with caterpillar: Allows the robot to move
- Reflectance sensor array(Fig. 1b): Allows the robot to trace lines which simulate roads
- Compass: Allows the robot to know which direction to turn on crossroads.



(a) (b)

Fig. 1. (a) Zumo robot with Arduino board on top. (b) Reflectance sensor arrays (6 IR sensors) attached to the front of the Zumo robot.

2.2 Tracing Lines using PID controller

PID controller is a mechanism used for precise robot controls consisting of proportional (P), integral (I), and derivative (D) terms. This study only employed the proportional (P) part of PID because it is accurate enough for this use case. The proportional part takes the current error to tell how drastic the robot's actions need to be to fix the error: The bigger the error, the faster the motor runs to correct the error, and vice versa [2].

2.3 Turning

As we aim to simulate the real traffic, the robot should also turn according to cardinal directions: north, south, west, and east. We used a compass to implement this feature so that the robot can figure out and make a right turn at each crossroad. Particularly, at each crossroad, the compass measures the specific direction each road connected to the crossroad was at. Then, when the robot was fed the information of which road it should take next, it uses the compass to know which road it was pointed towards, then rotated till it finds the correct one.

3. Storage of the Map

3.1 Representation of map

A map was constructed based on the area near Toyosu campus of Shibaura Institute of Technology. It was represented with (1) black lines for roads and (2) squares for crossroads. This is for the robot to know when it is at a crossroad, and for the ease of navigating through the map.

3.2 Implementation of the map

Graphs is a type of data structure that consists of two main

†1 Shibaura Institute of Technology

components, vertices, and edges. It is mostly used to represent networks such as telephone networks, circuit networks, or for the purpose of this paper, maps. Vertices are points that are connected to other vertices. Edges are the connections between vertices. In this experiment, we assume that vertices are crossroads and edges are roads. The properties that a vertex needs are unique IDs, and locations connected to it. The properties edges needs are weights. Weights in this experiment represents physical distance plus the number of vehicles on it.

As shown in Fig. 2 as an example, if the road between A and B have the physical distance of 3, then the default value of the edge between A and B is 3. When there are two vehicles on this road, the value of the edge becomes 5, which is, 3 (physical distance) + 2 (number of vehicles on the road).

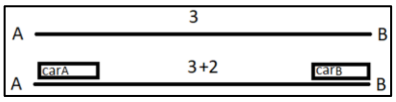


Fig. 2. Example of weight calculation with (upper) and without (lower) vehicles on the road.

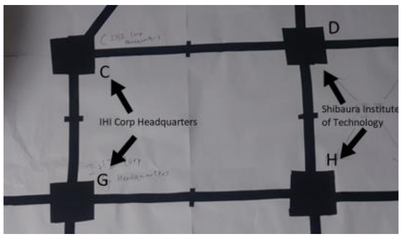


Fig. 3. This is a part of the map that I have constructed. Each vertex is assigned location and unique ID.

4. Path Finding Algorithm

There are many different shortest path algorithms such as Bellman-Ford algorithm, Floyd-Warshall, Dijkstra's shortest path, and more. In this study, we chose Dijkstra's algorithm because it is one of the most efficient path finding algorithms. It has a runtime complexity of O(E + VlogV), which means it is very efficient when scaling with the size of the map compared to most other algorithms [3]. Similar algorithms had been used by applications such as Google Maps.

Dijkstra's algorithm is a type of greedy algorithm that makes the best choice at every small stage. It uses a priority queue which the closest vertex where the edges it is connected to is not explored yet always have the highest priority. While exploring the edges, the distance of the other vertices is updated to their shortest possible values. If those other vertices' new values are shorter, then the previous vertex of the current vertex is altered. The algorithm ends when it reaches the destined vertex. The shortest path is simply produced by tracing back what vertex is the previous ones of the current ones until it reaches the starting point [3].

5. Traffic Simulation Algorithm

Due to the need to simulate an actual map, the algorithm stops when it reaches a specific destination instead of a specific vertex. Also, due to the need to simulate traffic, while backtracking for the results of the shortest path, the algorithm also adds 1 to the edges connected in the resulted path. This algorithm results in the cars that comes after selecting different paths with lower weights to avoid traffic.

6. Experiment and Discussion

The path finding and traffic simulation algorithms were implemented and then fed to the Arduino board for controlling the Zumo robot. Then, we can observe the simulation of the robot following the shortest path while considering virtual traffic.

As a result, the simulation can always find the shortest path while considering the traffic in case more than one vehicle was running on the same path.

However, the calculation for traffic representation in weights was not perfect because accurate traffic cannot simply be calculated by distance plus the number of vehicles on it. To create a simulation closer to reality, more variables must be included such as the type of road, weather, and more.

Moreover, this simulation has limitation that we have not included the weight calculation in case the vehicle reaches the destination or is not on the road, in which the weight is supposed to return to the initial weight.

The main challenges while conducting the experiments are the inconsistencies of the robot sensors. A main part of how our robot was able to follow the path given by our algorithm is to use the compass to know which road it is pointed to, but due to heavy influence of magnets, depending on where the map was placed, there had been extreme inconsistencies during testing and demonstrations.

7. Conclusion

In this study, we employed a Zumo robot as an autonomous vehicle for the goal to illustrate the shortest path and traffic simulation algorithms in real world. We implemented an altered version of Dijkstra's shortest path algorithm for the calculation of the shortest path while considering and adding traffic. For the robot's simulation, we implemented the proportional part of PID control as well as the compass for the navigation of map made of black lines and squares.

As a result, we were able to simulate a basic shortest path finding, taking simulated traffic into account and represent it in the physical world with Zumo robot.

This is still the first step in this study; however, we have encountered several limitations for more-complexed simulation comparing to a real-life situation which will be useful for our future study.

References

[1] Crovari, P. (2019, September 18). Google maps and graph theory. Impactscool Magazine. Retrieved September 20, 2022, from https://magazine.impactscool.com/en/speciali/google-maps-e-la-teoria-dei-grafi/

[2] Franklin, G. F., Powell, J. D., & Emami-Naeini, A. (2018). Feedback Control of Dynamic Systems (8th Edition) (What's New in Engineering). Pearson.

[3] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. Numerische Mathematik, 1(1), 269–271.