E206 Lab 2: A* Search

Shivam Malpani and Nathan Sunbury

***Abstract*— This paper proposes a robot motion planner that implements a modified version of the A* graph search algorithm. The modification makes use of the robot's ability to move both forwards and backwards, resulting in paths that are an average of 15% shorter than those generated by the unmodified algorithm. The performance of the modified algorithm was measured by calculating the average length of the resulting trajectories and comparing it to those of an unmodified A* search. Across 20 trials, the unmodified A* algorithm generated paths with an average length of $15.9m$ while the modified A* algorithm generated paths with an average length of $13.5m$.**

I. INTRODUCTION

Many robotic applications require the robot to move through the world in some way. To avoid damage to bystanders, the surroundings, and the robot itself, the robot must be able to navigate its environment while avoiding obstacles. To do this, it needs to search through the environment to find a collision-free path to the goal. Often, there are many possible ways to reach the goal without collisions, so it is important for a robot to be able to quickly find an efficient path to the goal.

One common approach to solving this problem is by applying graph search techniques. Graph search works by first discretizing a space into a network of connected points (a graph), and then searching through that network to find a suitable path. Each point in the graph is known as a node, and each connection between points is known as an edge. A general approach to graph search is described in algorithm 1 below [1].

Algorithm 1: Generic Graph Search

```
add initial node to fringe;
while fringe not empty do
    pick node n from fringe via strategy;
    if node n contains the goal then
        return path from initial node to goal;
    else
        add children of node n to fringe;
    end
end
return failure;
```

The algorithm is initialized by adding the starting location as the initial node on the fringe. The fringe is a collection of nodes that represent all the possible "directions" to search along the graph. The next steps iterate as long as there are still areas of the graph left to explore, that is, while the fringe is not empty. The first iterative step is to choose the best node from the fringe using some strategy. This strategy for choosing nodes from the fringe is the main area where different graph search algorithms vary. The next step is to check if the chosen node contains the goal. If it does, then the algorithm was successful and returns the path from the initial node to the goal by tracing back the path it took to get there. If the selected node does not contain the goal, the children of the node are added to the fringe. These correspond to all the possible nodes that can be reached from the selected node. Each child node stores points back to its parent, effectively leaving a trail of breadcrumbs that can lead back to the initial node if the goal is found. If the goal has not been found by the time all the nodes on the fringe have been explored and there are no more nodes to add, this indicates that the algorithm found no valid paths through the environment and it returns a failure.

The purpose of this report is to develop an implementation of a graph search algorithm known as A* search that will allow a differential drive robot to navigate an environment with unknown obstacles.

II. METHOD

A. A* Search

A* search is an informed search algorithm, meaning that it uses information about the environment to make a better decision about which nodes to pick from the fringe. Specifically, A* takes into account the distance required to reach a given node from the starting position and the estimated distance from that node to the goal. This technique attempts to create a strategy that combines the strengths of uniform-cost search, which always looks for the shortest path available, and greedy search, which always tries to move towards the goal. Equation 1 below shows this mathematically.

$$f(n) = g(n) + h(n) \tag{1}$$

$g(n)$ is the cost from the starting node to the current node. This is calculated as the sum of all the edge costs along the path to the current node. $h(n)$ is an estimated cheapest cost from the current node to the goal. This is often calculated as the euclidean distance from the current point to the goal. The sum of these two costs is $f(n)$, is used as a metric to determine which node to pick from the fringe. Nodes with lower $f(n)$ costs are picked first.

B. Alteration for an Unknown Environment

Usually, graph search algorithms are applied to graphs where all the nodes and edges are known from the start. In a real-world environment, however, this is not always the case. To operate in an unknown environment, the robot must

build up a graph as it explores the environment. This requires two alterations to the graph search techniques described previously.

The first change has to do with how a node's children are determined. Because the parent is not connected to a larger graph, new nodes must be generated each time its children are added to the fringe. This is done by propagating the robot's motion forwards with slight perturbations to the direction. If a collision-free Dubins trajectory [2] can be generated between the parent node and the child node, the child node is added to the fringe. The cost of the edge connecting the child node to its parent is the length of the Dubins trajectory.

The second change has to do with the success condition. Because the robot's environment is unknown, it is not guaranteed that the position of any of the nodes generated will exactly correspond to the position of the goal. Thus, the success condition may never be satisfied, even if the robot comes extremely close to the goal. Instead of requiring a node to exactly match the goal position, the success condition checks whether a collision-free Dubins trajectory can be generated between the current node and the goal.

C. Modification to A* Search

One downside with using Dubins trajectories to connect nodes is that they sometimes require the robot to perform loops in order to turn around. An example of this is shown in figure 1.

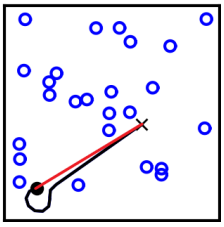

Fig. 1. Dubins trajectory (black) requiring the robot to loop around to change direction compared with the more direct trajectory (red) that could be followed if the robot drove backwards.

Because this A* search implementation is intended to be used with a differential drive robot, it can be extended to make use of the robot's ability to drive backwards. This is accomplished by creating a second search tree extending out behind the robot, and then choosing the tree that gives the shorter path. Driving backwards is not always necessary, but it can make a significant difference if the goal is behind the robot.

III. Experiments and Results

A. Experimental Setup

The performance of the modified A* search algorithm was evaluated by comparing the total length of the trajectories that it produced with those of the unmodified algorithm. The planned paths were then evaluated in an open source physics simulator and test environment using a simulated differential drive robot [3] [4]. The robot followed the trajectories generated by the A* algorithm using the proportional control algorithm developed in prior coursework [2].

B. Modified A* Search Results

To compare the performance of the modified and unmodified A* search algorithms, 20 trials were conducted with randomly-generated obstacles and a random goal location. Figure 2 shows an example of one of these trials. The robot began at $(-8, -8)$ facing the negative y-axis and drove towards a goal at $(-2.5, 2.5)$. The plot on the left of the figure shows the path that would be taken if the robot could only move forwards $(18.75m)$, while the plot on the right shows the path that the robot takes if allowed to move backwards $(15.75m)$.

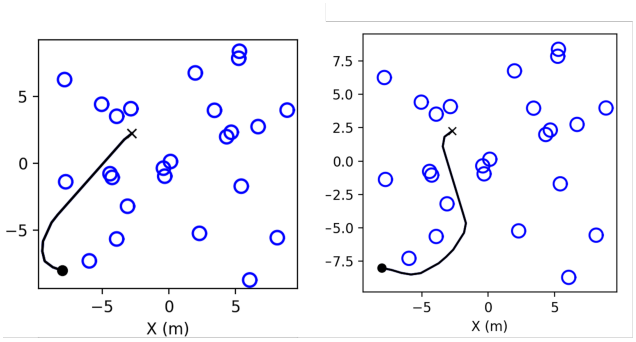

Fig. 2. Forward (left) vs backward (right) paths for one test

The average performance of each algorithm is shown in table I below.

A* Search	Average Path Length
Unmodified	15.9 m
Modified	13.5 m

TABLE I
Performance Comparison of Modified and Unmodified A* Search Algorithms

The backwards motion capability of the modified A* algorithm will not always be utilized, but over the course of many trials it yielded a 15% reduction in path length compared to the unmodified algorithm.

C. Trajectory Tracking

To ensure that the paths generated could actually be followed, the trajectories were passed to a previously developed trajectory tracking algorithm that guides a simulated robot [2]. Figure 3 shows the planned trajectory and the trajectory that the robot actually followed.

Table II below shows the RMS error of the robot's position and orientation as it tracked the trajectory.

IV. Conclusions

The modifications to the A* search algorithm described in this paper yielded an average path length that was 15% shorter than that of the unmodified algorithm. Additionally, the efficacy of these trajectories were validated in a simulated

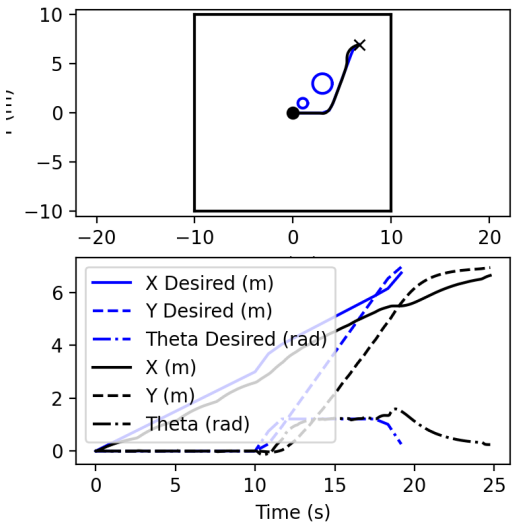

Fig. 3. Results of Trajectory Tracking with Simulated Robot

Variable	RMS Error
x	0.746 m
y	1.053 m
$\boldsymbol{\theta}$	0.439 rad

TABLE II

RMS ERROR OF (x, y, θ) OVER THE TRAJECTORY IN FIGURE 3

environment with unknown obstacles, indicating that this technique could be deployed to help a real-world robot navigate its environment. Future directions for this work could include developing an online version of the algorithm that would allow the robot to re-plan on the fly.

ACKNOWLEDGMENT

Thanks to Prof. C. Clark for his assistance with completing and understanding this lab and to Varun Singh for his assistance with debugging.

REFERENCES

[1] C. Clark, "E206 Lectures: Lecture 3B," *E206 Class Website*, [Online], Available: https://sites.google.com/g.hmc.edu/e206-motion-planning/lectures?authuser=0. [Accessed Feb. 20, 2020]

[2] S. Malpani and N. Sunbury, "Lab 1: Trajectory Tracking," *E206 Lab Reports*, Feb, 2021. [Online]. [Accessed Feb. 18, 2020]

[3] "Specifications," *Fetch Robotics*, [Online], Available: https://fetchrobotics.com/. [Accessed Feb. 22, 2020]

[4] "GYM," *Open AI*, [Online], Available: https://gym.openai.com/. [Accessed Feb. 22, 2020]