UNIVERSITÁ DI VERONA
COMPUTER ENGINEERING FOR ROBOTICS AND SMART INDUSTRY

January 6, 2024

Mobile robotics Course Project: A* implementation using potential field as heuristic

by

Francesco Dal Santo VR496940

Abstract:

This report presents the implementation of the A* pathfinding algorithm within a known environment, utilizing the potential field as the A* heuristic.
The entire experimentation process was conducted in a simulated setting.

The simulation environment allows for a comprehensive analysis of the A* algorithm's performance in a controlled and observable setting.

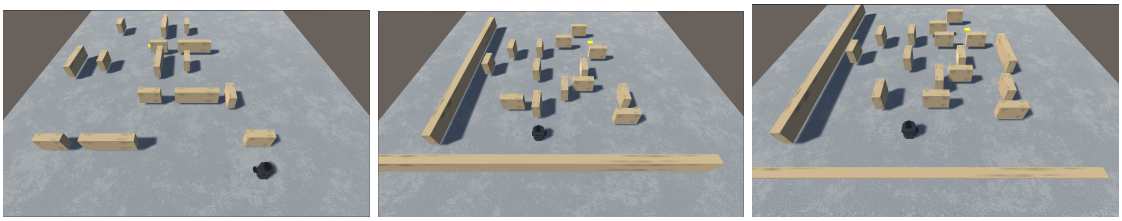
The report details the implementation process together with an analysis of the outcomes of various test cases.

1 Problem

This study addresses an assessment of the performance of the A* pathfinding algorithm, augmented with the potential field heuristic.

2 Environment

We use a Unity environment having a Turtlebot which needs to reach a goal.
There are no physical boundaries to the map, however the algorithm will automatically take the furthers corners out of all the objects and make them the map's limits.



(a) First use case (b) Second use case (c) Third use case

Figure 1: Three Cases

Since the algorithm works in a discrete environment I chose to discretise the Unity continuous environment in squares having dimensions equal to the Turtlebot (0,22x0,16).

3 Theory

3.1 A* algorithm

The A* algorithm is used to find the shortest path from a starting point to a goal point on a graph considering both the cost of reaching a particular node and a cost based on a chosen heuristic.

3.1.1 Initialization

- Initialize open and closed lists. The open list contains nodes to be evaluated, and the closed list contains nodes that have already been evaluated.
- Assign a cost of 0 to the starting node and calculate the heuristic (estimated cost to the goal) for that node.

3.1.2 Main loop

While the open list is not empty:

- Select the node with the lowest total cost (sum of actual cost and heuristic) from the open list
- Move the selected node from the open list to the closed list

3.1.3

For each neighboring node not in the closed list:

- Calculate the cost to reach that node from the current node
- If the node is not in the open list, add it with the calculated cost and heuristic to the open list
- If the node is already in the open list and the new cost is lower, update the cost and heuristic values

3.1.4 Termination

When the goal node is reached or the open list is empty, the algorithm terminates

3.1.5 Path Reconstruction

If the goal node is reached, reconstruct the path from the starting node to the goal node using the information stored in each node

3.2 Path planning

The Potential Field method for path planning simulates attractive forces toward the goal and repulsive forces from obstacles, creating a vector field that guides the robot through the environment

4 Code considerations

The details of the code are in its comments, it's worth to mention some things though.

The map dimension is dynamic and it's based on the extremes of the environment. The starting position, as well as the goal and obstacle position (center,scales and rotation) needs to be inputed manually.

The attractive and repulsive forces used for heuristic are considered without any direction for simplicity, moreover the formulas to derive them were taken from [1].

5 Workflow

It'll follow a brief explanation on the workflow:

- Unity environment is created, choosing a starting position, a goal, and placing the obstacles.
- This data is then manually computed in the code which is gonna discretize the space and it's gonna give in output the path in the continuous space (x,y) coordinates and a video showing the developing on the algorithm's choices.
- We then launch the code for the motion of the robot, which is gonna move the robot coordinate by coordinate till reaching the goal.

6 Code

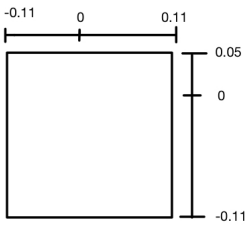
6.1 NODE.PY

This class contains the definition of the nodes, each node has the following attributes:

- parent: assigned depending on the path chosen
- x and y coordinates: known
- gcost: travel cost
- hcost: sum of the norms of attractive and repulsive forces
- fcost: sum of g and h costs
- flag: can be start,goal,walkable,obstacle

6.2 DISCRETE_TRANSFORM.PY

This file contains all functions needed for converting the map in the discrete domain.



- **assign_value_vertical:** Assigns the discrete number to a integer coordinate on the y axis.
  Everything is mapped to have 0.05 units in front of the frame and 0.11 units on the back to maintain the robot size as sampling size.

- **assign_value_horizontal:** Assigns the discrete number to a integer coordinate on the x axis.
  Everything is mapped to have 0.11 units in front of the frame and 0.11 units on the back to maintain the robot size as sampling size.

- **update_obstacles:** Takes in input a tuple of obstacle, calculates its corners and assigns to a continuous obstacles the nodes which it'll occupy

- **update_start_and_goal:** Assigns nodes coordinate for start and goal positions given the corresponding tuple.

6.3 A_STAR.PY

Now I'll go through the main features and function of the code, for more details check the commented code in the Github folder

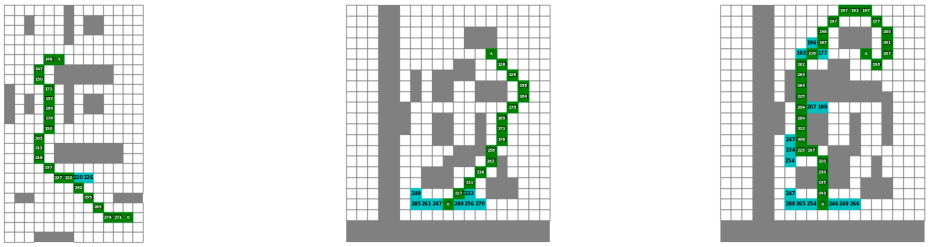
- Definition of start and goal
- Definition of obstacles
- Discretization of the environment
- Offset_compensation: transform all the negative numbers to 0 by compensating the offset
- Definition of map as empty node.
  Map's size is based on maximum values of the objects in the environment
- Constant definition: the ones used for attractive and repulsive forces were chosen through trial and error, so may be not the optimal ones
- Map is filled based on the discrete coordinates
- Standard A* algorithm with potential field heuristic

Here is gonna be explained what is different from the standard A* algorithm:

- The heuristic cost is calculated as the sum of the attractive and repulsive forces.
  Attractive force: generated by the goal, it is a simple linear distance formula
  Repulsive force: the chosen range of action was 3 discrete units; if a node has more repulsive forces at the same time on it then the norm of those will be summed resulting in a stronger repulsive force.
- The diagonal path is chosen only if the nodes where the robot needs to pass through are free, this is for avoiding collision in the simulation environment.

7 Results

It'll follow the visual representation of the path chosen by the algorithm:



(a) First use case (b) Second use case (c) Third use case

Figure 2: Three Cases

The green nodes are the ones chosen for the final path, the blue ones are the ones which were explored but didn't result in a good path

8 Conclusions

The results are not disappointing, since it finds a not too long path. It is to be considered if this improved the standard A* algorithm in which in my opinion it does not.

Considering for example the 3rd case, the robot avoids going on the right because the repulsive force is too strong, resulting in a longer path on the left.
Moreover the concentration of obstacles force the robot to go over the obstacle making a useless turn.

In the end according to my opinion this heuristic does not perform better than the standard A* algorithm in terms of the shortest path.

9 Future works

It would be needed an analysis in terms of the following:

- Dynamic environment with moving obstacles

- Numbers of explored nodes

References

[1] Sabudin E. N, Omar. R, and Che Ku Melor C. K. A. N. H.
potential field methods and their inherent approaches for path planning.
Journal of Robotics, 2022.