HOCHSCHULE
RAVENSBURG-WEINGARTEN
UNIVERSITY
OF APPLIED SCIENCES

Comparison path planning algorithms

Manuel Gnannt - 34946, IN
Florian Betz - 35653, IN

13.03.2023

Contents

1 Introduction

This paper explores the topic of path planning, with a focus on three specific algorithms A*, LaMCTS, and LaP3. Path planning is an important aspect of robotics and artificial intelligence, as it enables robots and other autonomous systems to navigate through an environment and reach a desired destination. The A* algorithm is a well-known and widely used method for pathfinding and graph traversal, which incorporates heuristic functions and predicted costs to find the most efficient path. LaMCTS and LaP3, on the other hand, are more recent algorithms that have been developed to address specific challenges in path planning, such as handling large or complex environments.

2 Evaluation Criteria

short description

2.1 Time Complexity

The time complexity of an algorithm is a measure of the execution time of the algorithm, which is calculated by summing the frequency of all statements in the algorithm. The statement frequency is the number of times a statement is executed and it is closely related to the size of the problem that the algorithm is solving. When the problem size is represented by $T(n)$, the statement frequency can be expressed as a function of $T(n)$ and the time complexity of the algorithm can also be expressed as a function of $T(n)$. [Sip13, p. 248 l. 22] Common scales for measuring time complexity include constant order $O(1)$, logarithmic order $O(log(n))$, linear order $O(n)$, linear logarithmic order $O(nlog(n))$, square order $O(n^2)$, cubic order $O(n^3)$, kth order $O(n^k)$ and exponential order $O(2^n)$.

2.2 Space Complexity

Space complexity is a metric that expresses the amount of temporary storage space utilized by an algorithm during its execution. It is represented as $O(f(n))$, where $f(n)$ is the function that describes the storage space required by the algorithm. [Sip13, p. 303] The amount of temporary storage space used during runtime can differ between different algorithms. Some algorithms only require a minimal amount of temporary space and remain constant regardless of the size of the problem and are called "in-situ" which means space-efficient. While other algorithms use multiple temporary work units that are related to the size of the problem, represented as n, and increases as n

increases. When n is large, more storage units will be utilized, resulting in a higher space complexity.

2.3 Search Efficiency

Global path planning algorithms create paths based on pre-existing information, while local path planning algorithms generate paths by gathering information about the environment via a sensor. Regardless of the algorithm, it must possess the capability to react quickly to environmental information, have low computational requirements and short search time, and have high search efficiency. Different algorithms exhibit varying levels of search efficiency.

2.4 Adaptability

There are various path planning algorithms available, and the degree of adaptability varies among them depending on the design of the algorithm principles and computational performance. The more versatile an algorithm is in adapting to different scenarios such as dense or sparse graphs, graphs with positive or negative edge weights, and shortest paths with single or multiple sources, the more effective it is.

cite

3 Path Planning Algorithms

short chapter description

3.1 A*

The A* algorithm, first introduced by Peter Hart and other researchers at the Stanford Research Institute in 1968, is a widely used method for pathfinding and graph traversal. [HNR68] It builds on Dijkstra's algorithm by incorporating heuristic functions and predicted costs, making it the most efficient direct search method for finding the shortest paths in a static road network and a popular heuristic for various other problems.[FBKT00]

The key component of the algorithm is the design of the valuation function, $f(n) = g(n) + h(n)$, where $g(n)$ is the actual cost from the initial node to node n, $h(n)$ is the

estimated cost from node n to the target node, and $f(n)$ is the estimated cost from the initial node via node n to the target node.

The steps of the A* algorithm are:

Step 1: Add the starting node to the priority queue.

Step 2: Select the node with the smallest F-value from the current priority queue and make it the current node.

Step 3: Mark it as visited and process its adjacent nodes.

Step 4: If the neighboring node has not been visited, add it to the queue, set the current node as its parent, and record its F, H, and G values. If the neighboring node has already been visited, check if the current node has a shorter path by comparing G values. If the current node has a smaller G value, update the parent node and G, H values of that node.

Step 5: Repeat steps 2 to 4 until the target node is marked or the priority queue is empty.

Step 6: When the path is found, trace back from the endpoint to the start node using the parent node.

The A* algorithm can reach a time complexity of $O(n)$.

3.2 Monte-Carlo Tree-Search

The leaves in the monte-carlo tree represent different states in a search space. The links to different children of a leaf represent different actions to get to a different state.

The tree now learns by performing the following steps:

Step1: Selection When in a given state, an appropriate action is selected based on a policy.

Step 2: Expansion When the selected leaf does not have any children, different actions are performed which lead to additional nodes.

Step 3: Simulation The reward of the newly created leaves is calculated and the best nodes are added to the tree.

Step 4: Backpropagation All nodes above the selected one are updated based on the reward they lead to.

3.3 LaMCTS

in a Latent space Monte-Carlo Tree-search, the search space is represented using a monte carlo tree. This is done by splitting the search space in a high-reward partition and a low-reward partition using k-keans. These partitions are represented by two child nodes in the monte-carlo tree. in a next step, samples are taked from around the cluster center and based on these samples, the now child-node is partitioned again in a high- and low-reward region. This results in a monte carlo tree with the left-mose leaf being the highest-reward region, the leaf left to the highest one being the second-highest and so on. [Lamcts image]

3.4 LaP3

write LaP3 algorithm

The Latent Space Partitions for Path Planning (LaP3) algorithm is an extensions of the LaMCTS. [YZC+21] It is a new path planning method which improves the function value estimation within each sub-region. This algorithm use a latent representation of the search space. LaP3 and Latent Space Monte Carlo Tree Search (LaMCTS) use the maximum, instead of the mean, as the node score to improve sample efficiency.

4 Evaluation

5 Conclusions and Further Discussion

Acronyms

LaP3 Latent Space Partitions for Path Planning

LaMCTS Latent Space Monte Carlo Tree Search

References

[FBKT00] Dieter Fox, Wolfram Burgard, Hannes Kruppa, and Sebastian Thrun. A probabilistic approach to collaborative multi-robot localization. *Autonomous Robots*, 8:325–344, 06 2000.

[HNR68] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[Sip13] Michael Sipser. Introduction to the theory of computation. pages 325–344, 2013.

[YZC+21] Kevin Yang, Tianjun Zhang, Chris Cummins, Brandon Cui, Benoit Steiner, Linnan Wang, Joseph E Gonzalez, Dan Klein, and Yuandong Tian. Learning space partitions for path planning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 378–391. Curran Associates, Inc., 2021.

Alphabetisch sortieren