ISSN:0975-9646

Modern Machine Learning Approaches For Robotic Path Planning

Analytic research and comparison

Shreyas J
Dept. of Computer Science
Christ University,
Bangalore, India

Sandeep J
Dept. of Computer Science,
Christ University,
Bangalore, India

***Abstract*—Many years have passed since the first path planning algorithm was found. These algorithms have evolved a lot since then. Even the best of the path finding algorithms were incapable to give the ideal result which was needed to find the path. Especially in terrains where the algorithms couldn't overcome the obstacle because that situation wasn't predicted beforehand. The ideal path was found using trial and error methods after the unpredicted events led to the collision of the robot. This problem was resolved when the robots gained the ability to outperform itself in these situations by learning what went wrong on its own and making sure the exact same event doesn't occur again. These machine learning algorithms emerged in the past decade and were continuously improvised and were perfected to have a very high success rate. In this survey of the path finding algorithms, the timeline and the working of algorithms and how these algorithms have developed and improvised over time can be seen.**

Keywords*— path-planning; machine-learning; robotics; Artificial intelligence; high-efficiency

I. Introduction

Navigation of rovers can be defined as the amalgamation of the three fundamental competences which need to be achieved by the rover in real time.

- Self-localisation
- Path finding
- Map interpretation
- Actuation

Self-localization is a state where the robot makes complete sense of the environment around it and is familiar with what is around it in extensively.

The path finding subsystem is responsible for finding an optimal traversable path from the start point to the end point. This includes the steps taken in a particular direction using the entire sequence of actions and rewards.

The map-interpretation used to explain the algorithms in this paper have simple representations which may consist of either a single value or a single pair of values. Complex representations are also used which include high end graphical model and geometric maps of the environment

Actuation is where the robot reacts on the environment based on the above two processes. This involves the robot physically moving in certain directions that in turn modify the physical relationship between the robot and the environment in is in and represented in.

As you can see the above components of path finding are all co related to each other. In this paper, focus is on the finding subsystems.

II. Methods that are not Path-Planning

The intent of this paper is to examine the differences between the classical path planning algorithms and the algorithms that use machine learning in order to traverse through the environment to reach the goal state from the start. Path planning is sometimes confused with these machine learning algorithms. These two ways of path finding aren't the same. Therefore, the need to distinguish between what actual path planning is and how it's done and what are machine learning path planning algorithms and how they are done.

III. Path planning: ALGORITHMS

Path planning algorithms find the path from the start to the end autonomously and monotonously by finding a set sequence of actions and decisions based on the representation given. These sequences are decided by the algorithm based on the representation of the map present. These algorithms are usually designed for certain type of environments suitable for the algorithm to function. This is a reason why there are so many more path planning algorithms than machine learning algorithms.

A. Dijkstra's algorithm

Dijkstra's algorithm is the most basic path planning algorithm available. This was the very first of the path planning algorithms. The algorithm achieves the shortest path to the destination using a set of visited and unvisited nodes and values assigned to each node.

1. The node at which we are starting at as the initial node is called. The following nodes say Y will be the distance of that node from the initial node.

2. The first initialization is the tentative distance values of all the nodes to infinity except the initial node which will be initialized to 0.

3. The initial node is now marked as current and all the other nodes are added to a newly created set and name it as the unvisited node set.

4. For the current node, consider all the unvisited nodes around that node and calculate their tentative distance. Compare the newly calculated tentative distance to the current assigned value and assign the smaller value to the node. If the node isn't connected to the current node, leave the node as is.

5. When all the nodes around the current node are check, mark the current node as visited and remove it from the unvisited set.

6. If the destination node is marked as visited or if the destination node is not traversable, then stop.

7. If not, select he next unvisited node from the set of unvisited nodes and mark it as the current node and return to step 4.

B. A algorithm*

A* is an informed search algorithm, it is the most favorable path finding algorithm because it's very flexible and can find paths in a variety of different situations effortlessly. It is largely based on the Dijkstra's algorithm and uses a heuristic search to guide itself to find the shortest path. The secret to its high success rates is that it uses the best features of the Dijkstra's search i.e. it favors the vertices closest to the starting point and the best features from the greedy best-first-search i.e. favoring the vertices closest to the goal. In the usual terminology while using A* algorithm, g(n) is used to denote the exact cost of traversal from the current vertex to any other vertex n on the given representation, and h(n) to denote the heuristic estimated cost from the vertex n to the goal. While the A* search is run, it balances both these values by finding the lowest f(n) value with n being the value of the next vertex and $f(n) = g(n) + h(n)$.

The algorithm puts all the vertices present, in a list called the open list which lets the algorithm know that the vertices in that list aren't traversed yet. While traversing through the vertices that the algorithm chooses, the algorithm puts the vertices traversed in the closed list so that these vertices won't be traversed again. The vertices not traversed will remain in the open list.

Additionally, if the heuristic becomes very monotonous, a closed set of nodes already traversed before may be used in order to make the search for the goal vertex more efficient.

C. LPA Algorithm*

The Lifelong Planning a* algorithm has been developed with a specific type of path finding in mind. This kind of pathfinding has the start and the goal state as the same points but the actual path finding is performed by changing the representation of the path. If the number of changes is comparatively small, then it is more efficient to repair the existing search tree than to perform the search from the start again. If the given search tree is as the one required for this algorithm then, the Lifelong Planning A* can be implemented by reinserting the vertices into the open-list of all nodes with the modified edge costs or connectivity. These changes can be then sent to the algorithm through the search tree as always. The lifelong planning stores the search tree structure not with pointer but with the cost of the values from these vertices to the goal state. When the algorithm reaches the start node, the path is then returned.

D. D lite algorithm*

The Dynamic A* or the D* algorithm is very similar to the Lifelong Planning A* algorithm, except that the start node here is allowed to change in-between searches. Similar to the LPA* the edge costs are allowed to change here too. In this algorithm, the more traditional notion of explicitly updating the back pointers is used as opposed to the one used in lifelong planning A* which is the implicit search-tree. This algorithm is the modified version of the D* and was developed almost a decade later which again used the implicit search-tree representation. Since then algorithms use the word 'lite' to describe algorithms with the implied search-tree representation.

This algorithm is widely used for robot navigation which run with the help of sensors. Hence the algorithm selects the best path visible to the robot and the path it is going to traverse in the near future. But the goal state is always rooted for obvious reasons. Hence the changes that the algorithm makes can only affect the edge structure or the cost of the edges which is farther away from the goal. Therefore, the changes made only affect the outer branches of the search-tree and most of the search tree need not be modified. Like the A* the D* also takes up quite a lot of time to compute the initial path to traverse, but the subsequent searches are much faster. In a long run, the D* can reduce in magnitude when compared to the A*. This allows the representations to ink out larger pieces of the representation and this representation can be modeled at a higher resolution.

*E. Field D**

Graph search techniques such as Dijkstra A* and D* algorithms find the optimal path to the goal using graph representation which is two dimensional 4- or 8- connected structure. The movement is broken into horizontal and vertical movements i.e. are 90 degree turns whereas in 8-connected it is decomposed to 45 degree turns. In a uniform map, the path that occurs on a 4-connectred(8-connected) graph, there are usually many optimal paths having the same costs to traverse. These paths alternate between horizontal and vertical movements. While the former is globally suboptimal the latter is suboptimal locally. However, from the set of given optimal paths, finding the best w.r.t. the real world is computationally infeasible. Usually the common solution for this is to break ties by moving towards the goal but this technique fails if there is an obstacle in the path. The Field D* avoids the problem of tie-breaking by operating in a continuous environment that envelopes the 4-connected or 8-connected graph. The graph nodes present in this system are contained as discrete sampling over a continuous field for the distance cost to be calculated which is required to reach the goal. The Field D* operates similar to the D* lite algorithm, except that the calculation of the cost from the vertex to the goal for continuous points on a graph is done using a linear interpolation between these cost values of the edges two end nodes. This allows the paths planned to follow the trajectories in the continuous domain. This algorithm is assumed to be implemented in a 'lite' environment which was mentioned earlier. Here, the edges are not explicitly followed, although in an 8-connected structure the edges is used to determine how the relationship between the node values and the field values and how they were calculated.

IV. MACHINE LEARNING: ALGORITHMS

A. *Reinforcement Learning*

These algorithms are machine learning algorithms which teach the system to take appropriate actions based on the concept of rewards and punishments. This world in which this system exists is assumed as the Markov decision process or MDP. The reinforcement learning principle was derived from this system. Although this algorithm doesn't explicitly model the MDP, these still are in this system and are called as the model-free methods. When a particular action is performed are a particular state, the probability distribution created with the system is called a policy. The system is allowed to modify its policy w.r.t. the rewards and/or punishments it receives. Punishments are also called are negative rewards. Usually the policy is computed by using the sum of the immediate reward from a particular action and the discounted sum of all the rewards that are eventually received after the original action was chosen. The drawback of this algorithm is that the system is trained to perform like an expert in the path planning process. This can pose to be a threat because of the policy based learning system.

B. *Classical Q Learning*

The Classical Q-Learning algorithm (CQL) like the reinforced learning algorithm gets the results by using the concept of the reward system. This is done by taking into account the rewards that the system will receive in the future, that are because of the actions taken by the system previously. This makes the algorithm better as it takes into account the fact that some actions that are optimal locally are sub optimal globally. The future rewards that are taken into consideration are subtracted with the value γ to account the uncertainty of it actually happening. The Q in the Q-Learning comes from the representation of the sum of the reward to be received from performing a particular action. Taking an assumption that there is a Q-table present with all the possible Q values, the largest of these Q values at every step is chosen. These Q values are obtained by repeated interaction with the environment.

The CQL requires a memory of $(n \times m)$ to keep track of the Q-table. The drawback in this method is that for large values of n, the space complexity is high. This is revised in the IQL algorithm, were an attempt was made to reduce this space complexity.

C. *Improverd Q Learning*

In the IQL algorithm presented here, involves having n Boolean variables called Lock for n states to indicate whether Q (S, a) at state S due to action a need to be updated. The Lock variables are used to avoid unnecessary update of entries Q (S, a) in the Q-table and, thus, to save time complexity. Beside this, in IQL, there is a requirement of n-memories to store n-Lock variables associated with n states. Here, instead of the Q-table of $n \times m$ dimension, there is a requirement to store the best Q-value of a state because of any action and thus require n-memories for n best Q-values of n states. This is denoted by L_i.

In this path-planning algorithm, the environment is made up of states just like that of the CQL algorithm. A state can have four neighbors. Consequently, the next best action to make the next move for the robot is achieved by selecting the largest Q-value. This continues till the goal state is achieved.

In the initialization phase of the algorithm, there exists a lock variable at all states except the goal state which is set to zero. The immediate reward from any neighboring state to the goal state is set to a certain constant value and discounting factor for estimating the uncertainty is set to γ and the initial state of the algorithm are fixed up.

In the present updated update policy of the IQL Q-table, the lock state L is initialized to 1 and the Q-Table isn't modified unless the current state or the next state is the goal state.
In order, to reach the goal state, the robot usually has to traverse in the environment for a finitely large number of iterations. To avoid the unnecessary modification of the Q-table update, a small repeat-until loop between the two main phases of the program is added. This loop continues selecting an action and executing it (without updating the Q-table) until the robot reaches the goal. Once the robot reaches the goal, the first repeat-until loop exits, and the Q-table updating is initiated. The process of Q-table updating is continued until all the states are locked.

V. CONCLUSION

In this survey and analysis of all these algorithms gives an idea and an understanding of the flow of the creation and implementations of the algorithms and how these algorithms were modified and bettered as time passed and how they were modified for different situations and predicaments. Also, there an increase in the artificial intelligence in every algorithm as we go through the flow of the paper.

Despite these algorithms being intelligent, there were better ways to find paths, faster ways, ways with which we do not even need an algorithm. Ways which were intelligent enough to learn and become more intelligent on its own. These algorithms are better than the path planning algorithms as they hardly take up space on the computer. Even though they need to run for a set number of times to start off the learning process, they are the fastest way to find a path after this learning process is done. These machine learning concept uses the Q-learning way of defining the states and rewards to find this path.

This classical approach was further optimised and structured to outperform the previous version.

Now that there's a new way of path planning, i.e. by finding a dynamic self-customised path spontaneously, there is no need of having algorithms on the computer to find paths. The use of the IQL uses only a table of Q-values with which the path is found which uses only a quarter of the space a normal algorithm may use. These new ways of finding the paths not only save time and energy but also save a lot of space on on-board computer on robots which usually have limited data space and memory.

REFERENCES

[1] A D. Ferguson, M. Likhachev, A. Stentz, A guide to heuristic-based path planning, in: Proceedings of the Workshop on Planning under Uncertainty for Autonomous Systems at the International Conference on Automated Planning and Scheduling (ICAPS), 2005.

[2] Jiang, Bomin, et al. "Path planning for minimizing detection." IFAC Proceedings Volumes 47.3 (2014): 10200-10206.

[3] Masehian, Ellips, and Davoud Sedighizadeh. "Classic and heuristic approaches in robot motion planning-a chronological review." World Academy of Science, Engineering and Technology 23 (2007): 101-106.

[4] Koenig, Sven, and Maxim Likhachev. "D^* Lite." AAAI/IAAI. 2002.

[5] Mills-Tettey, G. Ayorkor, Anthony Stentz, and M. Bernardine Dias. "DD* Lite: Efficient Incremental Search with State Dominance." Proceedings of the National Conference on Artificial Intelligence. Vol. 21. No. 2. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[6] Konar, Amit, et al. "A deterministic improved Q-learning for path planning of a mobile robot." IEEE Transactions on Systems, Man, and Cybernetics: Systems 43.5 (2013): 1141-1153.

[7] Koenig, Sven, Maxim Likhachev, and David Furcy. "Lifelong planning A∗." Artificial Intelligence 155.1-2 (2004): 93-146.

[8] Lumelsky, Vladimir J. "A comparative study on the path length performance of maze-searching and robot motion planning algorithms." IEEE Transactions on Robotics and Automation 7.1 (1991): 57-66.

[9] Sariff, N., and Norlida Buniyamin. "An overview of autonomous mobile robot path planning algorithms." Research and Development, 2006. SCOReD 2006. 4th Student Conference on. IEEE, 2006.

[10] Zhu, D. J., and J-C. Latombe. "New heuristic algorithms for efficient hierarchical path planning." IEEE Transactions on Robotics and Automation 7.1 (1991): 9-20.

[11] Thrun, Sebastian, et al. "Stanley: The robot that won the DARPA Grand Challenge." Journal of field Robotics 23.9 (2006): 661-692.

[12] Massari, Mauro, et al. "Optimal path planning for planetary exploration rovers based on artificial vision system for environment reconstruction." Advanced Intelligent Mechatronics. Proceedings, 2005 IEEE/ASME International Conference on. IEEE, 2005.

[13] Ono, Masahiro, et al. "Risk-aware planetary rover operation: Autonomous terrain classification and path planning." Aerospace Conference, 2015 IEEE. IEEE, 2015.

[14] Dong, Shaoyang, Hehua Ju, and Hongxia Xu. "An improvement of D∗ lite algorithm for planetary rover mission planning." Mechatronics and Automation (ICMA), 2011 International Conference on. IEEE, 2011.

[15] Ishigami, Genya, Keiji Nagatani, and Kazuya Yoshida. "Path planning for planetary exploration rovers and its evaluation based on wheel slip dynamics." Robotics and Automation, 2007 IEEE International Conference on. IEEE, 2007.

[16] Sakuta, Mariko, Shogo Takanashi, and Takashi Kubota. "An image based path planning scheme for exploration rover." Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on. IEEE, 2011.

[17] Meyer, Jean-Arcady, and David Filliat. "Map-based navigation in mobile robots:: Ii. a review of map-learning and path-planning strategies." Cognitive Systems Research 4.4 (2003): 283-317.

[18] Cannon, Jarad, Kevin Rose, and Wheeler Ruml. "Real-Time Motion Planning with Dynamic Obstacles." SOCS. 2012.

[19] Otte, Michael W. "A survey of machine learning approaches to robotic path-planning." Cited on: pdfs.semanticscholar.org.

[20] Stentz, Anthony. "The focussed D^* algorithm for real-time replanning."IJCAI.Vol.95.1995.