D* Lite Algorithm in Path Planning of Autonomous Robot

Jose Hosea 13516027
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13516027@std.stei.itb.ac.id

Abstract — Robotics is an interdisciplinary branch of engineering and science that includes mechanical engineering, electrical engineering, computer science, and others. Robotics deals with the design, construction, operation, and use of robots, as well as computer systems for their control, sensory feedback, and information processing.

One of the branch of robotics which involve path planning (or navigation problem) is auto-robot engineering. One must assess the skill of understanding how the robot move, how it must perceive its environment in the ideal way so it can move efficiently, both in time to process its environment, and move with minimum cost.

This paper will focus on application of D* algorithm, a variance of A* algorithm in auto-robot path planning, and how its derivation, D* Lite, is advantageous in terms as stated above.

***Keywords* — Auto-Robot, D*, Path planning, Robot Navigation.**

I. Introduction

The term path planning was developed in many fields, such as robotics, artificial intelligence or control theory. That is why each scientist uses own definition of this term. In robotics, path planning concerns with problem as how to move a robot from one point to another point. With the advances in robotics path planning also includes many complications such as uncertainties, multiple robots, or dynamics. In artificial intelligence, path planning means a search for a sequence of logical actions that transform an initial robot state into a desired goal state. Such planning may include many decision-theoretic ideas such as Markov decision processes, imperfect state information, learning methods or game-theoretic equilibrium. In the control theory, path planning deals with issues of stability, feedback, and optimality. As it can be seen, path planning of a mobile robot is a wide problem and there exist many methods and approaches to it.

Generally in robotics, path planning is focused on designing algorithms that generate useful motions by processing simple or more complicated geometric models. This paper is focused on such algorithms. Path planning addresses the automation of mechanical systems that have sensors, actuators, and computation capabilities. Path planning is defined as a fundamental needs in robotics that are described by algorithms that convert high-level specification of task from humans into low-level description of how to move. This is described by a Piano Mover's Problem. The task is defined with a precise model of house and a piano as input to an algorithm. The algorithm must determine how to move the piano from one room to another without hitting anything. Robot's path planning is defined in similar way. However, path planning usually ignores dynamics and other constraints and focuses primarily on the translations and rotations of controlled object – robot. Recent research in this area considers also other aspects such as uncertainties, differential constraints, optimality, etc.

For goal-directed path planning, a robot has to move from the given starting position towards the goal position with the capability to avoid obstacles. The robot should also be able to traverse in both known and unknown environments. Besides, time to traverse from starting point to goal point is also an important aspect emphasized by researches. It is important for robot to plan a shortest path from starting point to goal point. Moreover, therobot must be able to re-plan its path quickly if there is a new obstacle in front or nearby. Some of the existing goal-direct navigation algorithms are such as A* Algorithm, D* Algorithm, and D* Lite Algorithm.

II. Theories

2.1 Robotics

2.1.1 Robots

A **robot** is a machine—especially one programmable by a computer— capable of carrying out a complex series of actions automatically. Robots can be guided by an external control device or the control may be embedded within. Robots may be constructed to take on human form but most robots are machines designed to perform a task with no regard to how they look.

Robots have replaced humans in performing repetitive and dangerous tasks which humans prefer not to do, or are unable to do because of size limitations, or which take place in extreme environments such as outer space or the bottom of the sea. There are concerns about the increasing use of robots and their role in society. Robots are blamed for rising technological unemployment as they replace workers in increasing numbers of functions. The use of robots in military combat raises ethical

concerns. The possibilities of robot autonomy and potential repercussions have been addressed in fiction and may be a realistic concern in the future.

There are a lot different types of robots that can be made, but the basic similarities of their construction is the most important part in creating a robot, with the rest of the process after it are left to engineers' creativities and goals.

They are used in many different environments and for many different uses, although being very diverse in application and form they all share three basic similarities when it comes to their construction:

1. Robots all have some kind of **mechanical construction**, a frame, form or shape designed to achieve a particular task. For example, a robot designed to travel across heavy dirt or mud, might use caterpillar tracks. The mechanical aspect is mostly the creator's solution to completing the assigned task and dealing with the physics of the environment around it. Form follows function.

2. Robots have **electrical components** which power and control the machinery. For example, the robot with caterpillar tracks would need some kind of power to move the tracker treads. That power comes in the form of electricity, which will have to travel through a wire and originate from a battery, a basic electrical circuit. Even petrol powered machines that get their power mainly from petrol still require an electric current to start the combustion process which is why most petrol powered machines like cars, have batteries. The electrical aspect of robots is used for movement (through motors), sensing (where electrical signals are used to measure things like heat, sound, position, and energy status) and operation (robots need some level of electrical energy supplied to their motors and sensors in order to activate and perform basic operations)

3. All robots contain some level of **computer programming** code. A program is how a robot decides when or how to do something. In the caterpillar track example, a robot that needs to move across a muddy road may have the correct mechanical construction and receive the correct amount of power from its battery, but would not go anywhere without a program telling it to move. Programs are the core essence of a robot, it could have excellent mechanical and electrical construction, but if its program is poorly constructed its performance will be very poor (or it may not perform at all). There are three different types of robotic programs: remote control, artificial intelligence and hybrid. A robot with remote control programing has a preexisting set of commands that it will only perform if and when it receives a signal from a control source, typically a human being with a remote control. It is perhaps more appropriate to view devices controlled primarily by human commands as falling in the discipline of automation rather than robotics. Robots that use artificial intelligence interact with their environment on their own without a control source, and can determine reactions to objects and problems they encounter using their preexisting programming. Hybrid is a form of programming that incorporates both AI and RC functions.

2.1.2 Autonomous Robots

An autonomous robot performs behaviors or tasks with a high degree of autonomy, which is particularly desirable in fields such as spaceflight, household maintenance (such as cleaning), waste water treatment and delivering goods and services.

Some modern factory robots are "autonomous" within the strict confines of their direct environment. It may not be that every degree of freedom exists in their surrounding environment, but the factory robot's workplace is challenging and can often contain chaotic, unpredicted variables. The exact orientation and position of the next object of work and (in the more advanced factories) even the type of object and the required task must be determined. This can vary unpredictably (at least from the robot's point of view).

One important area of robotics research is to enable the robot to cope with its environment whether this be on land, underwater, in the air, underground, or in space.

A fully autonomous robot can:

- Gain information about the environment.
- Work for an extended period without human intervention.
- Move either all or part of itself throughout its operating environment without human assistance.
- Avoid situations that are harmful to people, property, or itself unless those are part of its design specifications.

An autonomous robot may also learn or gain new knowledge like adjusting for new methods of accomplishing its tasks or adapting to changing surroundings. Like other machines, autonomous robots still require regular maintenance.

2.2 Path planning

2.2.1 Definitions

Path planning (also known as the navigation problem or the piano mover's problem) is a term used in robotics for the process of breaking down a desired movement task into discrete motions that satisfy movement constraints and possibly optimize some aspect of the movement.

For example, consider navigating a mobile robot inside a building to a distant waypoint. It should execute this task while avoiding walls and not falling down stairs. A path planning algorithm would take a description of these tasks as input, and produce the speed and turning commands sent to the robot's wheels. Path planning algorithms might address robots with a larger number of joints (e.g., industrial manipulators), more complex tasks (e.g. manipulation of objects), different constraints (e.g., a car that can only drive forward), and

uncertainty (e.g. imperfect models of the environment or robot).

Path planning has several robotics applications, such as autonomy, automation, and robot design in CAD software, as well as applications in other fields, such as animating digital characters, video game artificial intelligence, architectural design, robotic surgery, and the study of biological molecules.

A motion planner is said to be complete if the planner in finite time either produces a solution or correctly reports that there is none. Most complete algorithms are geometry-based. The performance of a complete planner is assessed by its computational complexity.

Resolution completeness is the property that the planner is guaranteed to find a path if the resolution of an underlying grid is fine enough. Most resolution complete planners are grid-based or interval-based. The computational complexity of resolution complete planners is dependent on the number of points in the underlying grid, which is O(1/hd), where h is the resolution (the length of one side of a grid cell) and d is the configuration space dimension.

Probabilistic completeness is the property that as more "work" is performed, the probability that the planner fails to find a path, if one exists, asymptotically approaches zero. Several sample-based methods are probabilistically complete. The performance of a probabilistically complete planner is measured by the rate of convergence.

Incomplete planners do not always produce a feasible path when one exists. Sometimes incomplete planners do work well in practice.

2.2 D Algorithm*

2.2.1 Definitions

D*(read as D star) is an incremental search algorithm to solve the same assumption-based path planning problems, including planning with the freespace assumption, where a robot has to navigate to given goal coordinates in unknown terrain. It makes assumptions about the unknown part of the terrain (for example: that it contains no obstacles) and finds a shortest path from its current coordinates to the goal coordinates under these assumptions. The robot then follows the path. When it observes new map information (such as previously unknown obstacles), it adds the information to its map and, if necessary, replans a new shortest path from its current coordinates to the given goal coordinates. It repeats the process until it reaches the goal coordinates or determines that the goal coordinates cannot be reached. When traversing unknown terrain, new obstacles may be discovered frequently, so this replanning needs to be fast. Incremental (heuristic) search algorithms speed up searches for sequences of similar search problems by using experience with the previous problems to speed up the search for the current one. Assuming the goal coordinates do not change, this search algorithm is more efficient than repeated A* searches.

D* and its variants have been widely used for mobile robot and autonomous vehicle navigation. Current systems are typically based on D* Lite rather than the original D* or Focussed D*. In fact, even Stentz's lab uses D* Lite rather than D* in some implementations. Such navigation systems include a prototype system tested on the Mars rovers Opportunity and Spirit and the navigation system of the winning entry in the DARPA Urban Challenge, both developed at Carnegie Mellon University.

The original D* was introduced by Anthony Stentz in 1994. The name D* comes from the term "Dynamic A*", because the algorithm behaves like A* except that the arc costs can change as the algorithm runs.

A* Algorithm (Lester, 2009) is an early developed popular graph search algorithm which finds the shortest path from a given initial start node to the goal node. A* Algorithm uses distance plus path cost function to determine the shortest path to the goal node. In A* Algorithm, node or square notation is used rather than coordinate because a map is divided into small grids or squares and nodes represent the center point of each grid D* Algorithm, which is also known as Stentz algorithm or Dynamic A* Algorithm, is developed by Anthony Stentz in 1994. It is better than A* Algorithm because it could be used in partially or completely unknown and also dynamic environment. Dynamic environment means there might contain moving obstacles. A* Algorithm is a simple algorithm which could be only used in static environment. But D* Algorithm is able to repair or update the map of the dynamic environment and it can re-plan quickly whenever it detects there is a new obstacle or an obstacle is removed on the way to goal node (Stentz, 1994).

Like Dijkstra's algorithm and A*, D* maintains a list of nodes to be evaluated, known as the "OPEN list". Nodes are marked as having one of several states:

- NEW, meaning it has never been placed on the OPEN list
- OPEN, meaning it is currently on the OPEN list
- CLOSED, meaning it is no longer on the OPEN list
- RAISE, indicating its cost is higher than the last time it was on the OPEN list
- LOWER, indicating its cost is lower than the last time it was on the OPEN list

For D*, it is important to distinguish between current and minimum costs. The former is only important at the time of collection and the latter is critical because it sorts the OPEN list. The function which returns the minimum cost is always the lowest cost to the current point since it is the first entry of the OPEN list.

2.2.1 D* Lite

D* Lite Algorithm is a reverse or backward searching method and it is able to re-plan from current position when there is a new obstacle blocking the path. It determines the same paths as D* Algorithm and moves the mobile robot the same way but it is algorithmically different from D* Algorithm. D* Lite Algorithm is developed by Koenig and Likhachev (Koenig & Likhachev, 2002; Likhachev & Koenig, 2002; Koenig &

Likhachev, 2005) based on Lifelong Planning A* (Koenig et al., 2004) Algorithm in 2002. It has been widely used for mobile robot navigation in unknown environment. Similarly, it divides the environment into grids and path finding and robot's movement are from grid to grid.

The pseudo-code is given below:

```
procedure CalculateKey(s)
{01"} return [min(g(s), rhs(s)) + h(s_start, s) + k_m ; min(g(s), rhs(s))];

procedure Initialize()
{02"} U = ∅;
{03"} k_m = 0;
{04"} for all s ∈ S rhs(s) = g(s) = ∞;
{05"} rhs(s_goal) = 0;
{06"} U.Insert(s_goal, [h(s_start, s_goal); 0]);

procedure UpdateVertex(u)
{07"} if (g(u) ≠ rhs(u) AND u ∈ U) U.Update(u, CalculateKey(u));
{08"} else if (g(u) ≠ rhs(u) AND u ∉ U) U.Insert(u, CalculateKey(u));
{09"} else if (g(u) = rhs(u) AND u ∈ U) U.Remove(u);

procedure ComputeShortestPath()
{10"} while (U.TopKey() < CalculateKey(s_start) OR rhs(s_start) > g(s_start))
{11"}   u = U.Top();
{12"}   k_old = U.TopKey();
{13"}   k_new = CalculateKey(u));
{14"}   if(k_old < k_new)
{15"}     U.Update(u, k_new);
{16"}   else if (g(u) > rhs(u))
{17"}     g(u) = rhs(u);
{18"}     U.Remove(u);
{19"}     for all s ∈ Pred(u)
{20"}       if (s ≠ s_goal) rhs(s) = min(rhs(s), c(s, u) + g(u));
{21"}       UpdateVertex(s);
{22"}   else
{23"}     g_old = g(u);
{24"}     g(u) = ∞;
{25"}     for all s ∈ Pred(u) ∪ {u}
{26"}       if (rhs(s) = c(s, u) + g_old)
{27"}         if (s ≠ s_goal) rhs(s) = min_{s'∈Succ(s)}(c(s, s') + g(s'));
{28"}       UpdateVertex(s);

procedure Main()
{29"} s_last = s_start;
{30"} Initialize();
{31"} ComputeShortestPath();
{32"} while (s_start ≠ s_goal)
{33"}   /* if (rhs(s_start) = ∞) then there is no known path */
{34"}   s_start = arg min_{s'∈Succ(s_start)}(c(s_start, s') + g(s'));
{35"}   Move to s_start;
{36"}   Scan graph for changed edge costs;
{37"}   if any edge costs changed
{38"}     k_m = k_m + h(s_last, s_start);
{39"}     s_last = s_start;
{40"}     for all directed edges (u, v) with changed edge costs
{41"}       c_old = c(u, v);
{42"}       Update the edge cost c(u, v);
{43"}       if (c_old > c(u, v))
{44"}         if (u ≠ s_goal) rhs(u) = min(rhs(u), c(u, v) + g(v));
{45"}       else if (rhs(u) = c_old + g(v))
{46"}         if (u ≠ s_goal) rhs(u) = min_{s'∈Succ(u)}(c(u, s') + g(s'));
{47"}       UpdateVertex(u);
{48"}     ComputeShortestPath();
```

Figure 1. D* Lite Algorithm
Source : D* Lite by Sven Koenig and Maxim Likhachev.

III. D* LITE IN PATH PLANNING

3.1 Overview

Consider a goal-directed robot-navigation task in unknown terrain, where the robot always observes which of its eight adjacent cells are traversable and then moves with cost one to one of them. The robot starts at the start cell and has to move to the goal cell. It always computes a shortest path from its current cell to the goal cell under the assumption that cells with unknown blockage status are traversable. It then follows this path until it reaches the goal cell, in which case it stops successfully, or it observes an untraversable cell, in which case it recomputes a shortest path from its current cell to the goal cell.

Figure 2 shows the goal distances of all traversable cells and the shortest paths from its current cell to the goal cell both before and after the robot has moved along the path and discovered the first blocked cell it did not know about. Cells whose goal distances have changed are shaded gray. The goal distances are important because one can easily determine a shortest path from its current cell of the robot to the goal cell by greedily decreasing the goal distances once the goal distances have been computed. Notice that the number of cells with changed goal distances is small and most of the changed goal distances are irrelevant for recalculating a shortest path from its current cell to the goal cell.

Thus, one can efficiently recalculate a shortest path from its current cell to the goal cell by recalculating only those goal distances that have changed (or have not been calculated before) *and* are relevant for recalculating the shortest path. This is what D* Lite does. The challenge is to identify these cells efficiently.

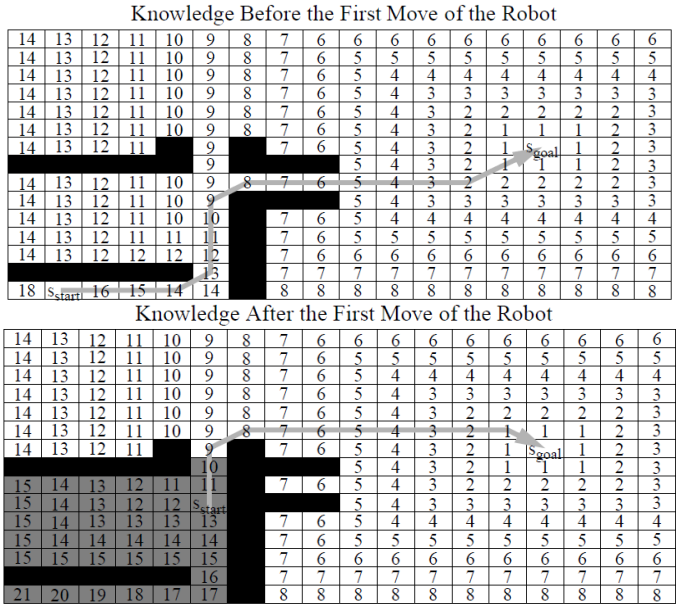

Figure 2. An Example of Path planning with D * Lite
Source : D* Lite by Sven Koenig and Maxim Likhachev.

3.2 Execution

The experiment is to simulates the dynamic replanning ability of the path planning algorithm by allowing the user to add obstacles by clicking on cells in the grid. The robot is the red circle, while the goal cell is green. The robot has a visibility range, shown by the thin black lines, and added obstacles are not taken into account by the robot until they are visible and change to darker grey. Pressing space bar makes the robot observe new obstacles, replan if necessary, and advances on the current best path until the goal is reached.

The experiment here merely showing how the D* Lite algorithm works in a grid-based environment(with the assumption that the robot holds sufficient processing power) and its replanning capabilities when the robot 'see' the manually-added obstacle. The experiment will be simulated by a visualization program.

Figure 3 shows the initial state of the robot, its knowledge about the environment, and the pre-defined goal cell.

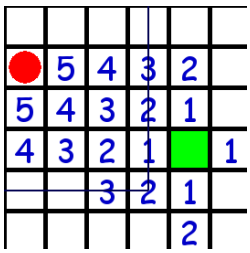

Figure 3. Initial Position of Robot (red circle), Its Vision Range (thin line), and the Goal (green cell)

The first step of the experiment is done by pressing the space bar once. This step simulates what the robot does when there is no obstacle known.

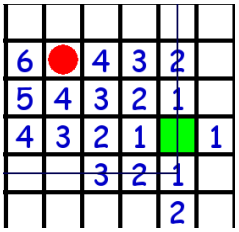

Figure 4. Step Taken by The Robot

Figure 4 shows what happened after space bar is pressed. As seen in it, when there is no obstacle 'seen' by the robot, it chooses a path, move to the next minimum-cost cell and mark the last cell as a costly path.

The second step is to press space bar repeatedly until the robot reaches the goal cell.

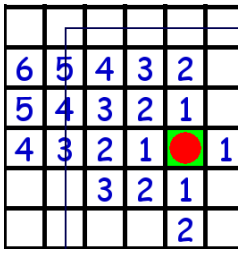

Figure 5. Robot reaches the Goal

As seen in Figure 5, this step shows that the robot can reach its goal, given its current knowledge on the environment. The robot is proven to be able to plan a shortest path and consistently follows it.

Next, the program is restarted, and then an obstacle is added by clicking a cell, marking it grey. After the space bar is pressed, the robot 'sees' the obstacle and mark it as untraversable. See Figure 6.

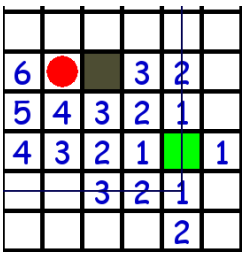

Figure 6. The Robot Marks The Obstacle

In the next step, the program simulates how the robot responds, if the current known paths are untraversable because of obstacles.

As seen in Figure 7, the robot will expand its current known paths, by expanding and updating its cost graph.

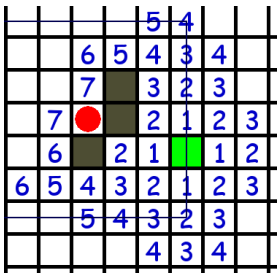

Figure 7. Expanded Graph in the Map

For the next step, the program simulates how the robot responds the fact that no paths can reach the goal.

As seen in Figure 8, the robot calculates paths a few times before it decides that there is no path that leads to the goal cell.

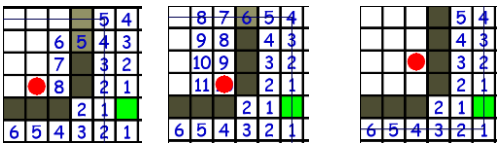

Figure 8. No Path to the Goal

This step shows that D* Lite algorithm is a complete algorithm, which can observe in a finite amount of time(or iterations) that there is no path available to its destination. This step also shows that D* Lite does not depends on probabilistic event. It determines the path based on its knowledge of the environment.

As we can see in the experiment, D* Lite algorithm is a viable option for auto-robot path planning. It dynamically updates the path when obstacle is found, and it is a complete algorithm in terms of deciding whether there exists path to the destination or not.

IV. Conclusion

D* Lite algorithm is a fast replanning method for robot navigation in unknown terrain that implements the same navigation strategies as Dynamic A* (D*). The algorithm search from the goal vertex towards the current vertex of the robot, use heuristics to focus the search, and use a way to minimize having to reorder the priority queue.

D* Lite builds on our LPA*, a strong similarity to A*, is efficient (since it does not expand any vertices whose values were already equal to their respective goal distances) and has been extended in a number of ways. It is algorithmically different from D*, easier to understand and extend, yet at least as efficient as D*.

D* Lite can provide auto-robot an efficient way to plan its path and reach its goal, by determining, dynamically as it makes its move, the shortest path available, and replans when the path is not traversable because of obstacles.

V. Acknowledgment

A special note of thanks to Dr. Ir. Rinaldi Munir, MT., the most inspiring lecturer at Bandung Institute of Technology for this interesting assignment that has broaden my knowledge on appliance of D* algorithm in robotics.

I would also like to thank ScienceDirect for the readings and other sources of my inspirations that helped me get this paper done.

References

[1] Anonymous, 2000, The Interplay between Mathematics and Robotics, National Science Foundation Arlington, Virginia.
[2] Munir, R., 2007, Diktat Kuliah: Strategi Algoritma, Departemen Teknik Informatika, Institut Teknologi Bandung.
[3] https://www.sokanu.com/careers/robotics-engineer/
[4] https://robotics.stackexchange.com/questions/9337/suitable-d-star-variant-is-for-non-holonomic-motion-planning-of-mobile-robots
[5] Duchoň, F., Babinec, A., Kajan, M., Beňo, P., Florek, M., Fico, T., Jurišica, L., 2014, Path planning with modified A star algorithm for a mobile robot, Modelling of Mechanical and Mechatronic Systems MMaMS.
[6] Ferguson, D., Stentz, A., The Field D* Algorithm for Improved Path planning and Replanning in Uniform and Non-Uniform Cost Environments, Carnegie Mellon University, Pittsburgh.
[7] Guruji, A.K., Agarwal, H., Parsediya, D.K., 2016, Time-Efficient A* Algorithm for Robot Path planning, 3rd International Conference on Innovations in Automation and Mechatronics Engineering, ICIAME 2016.
[8] Koenig, S., Likhachev, M., 2002, D* Lite, American Association for Artificial Intelligence.

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 14 Mei 2018

Jose Hosea 13516027