The 2010 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 18-22, 2010, Taipei, Taiwan

The D++ Algorithm: Real-Time and Collision-Free Path-Planning for Mobile Robot

Pi-Ying Cheng, Pin-Jyun Chen

Abstract—**This paper proposed an improved algorithm that we call "D++" which can be applied to real-time and collision-free path -planning to solve some problems of common methods at present. D++ algorithm combines the Dijkstra's algorithm with sensor-based method so that D++ algorithm can deal with problems of unknown, large, complex, or dynamic environment, and need only local environmental information initially to find a shortest path. In the later of this article, we demonstrated some examples to show that D++ algorithm is very efficient for path-planning and also very practicable on real mobile robot system.**

I. Introduction

In past few years, researches on mobile robot were paid more and more attention, and widely applied to the industry, hospitals, offices, home-care and the military. This is because of the excellent moving-ability of mobile robot so that it can help challenged people, explores an unknown and danger environment, and drive cars automatically…ect. Path-planning is one of important technology for mobile robot.

With the rapid development of theories and methods in artificial intelligence, path-planning has been successfully applied to video game, navigator, autonomous car, and mobile robot. According to the acquired information of environment, path-planning can be divided into two types: (1) Global path-planning: means that environmental information is static and known in advance. So robot can use path-planning methods to find a shortest or optimal path from a start to a goal. (2) Local path-planning: means that environmental information is unknown or known about some parts. The shapes, sizes, and locations of obstacles and other objects must be obtained through observing later, and then make a decision immediately. Therefore, environmental information of local path-planning can change in any time. At present, research of path-planning on static environment has obtained many achievements, and it is paid more attention on those problems which environment is uncertain and variable.

Generally, for static problem, the searching algorithm is a one-time computation to find a shortest path. The most famous algorithms are Dijkstra's algorithm [1] and A* algorithm [2]. Dijkstra's algorithm, conceived by Edsger Dijkstra in 1959, is a graph search algorithm which can find the shortest path. However, Dijkstra's algorithm explores a great deal of nodes in searching space, so Dijkstra's algorithm is considered that isn't efficient enough. In 1972, Peter Hart, who proposed A* algorithm, used a heuristic estimate so that it will deal with better nodes in searching space first, and ignore some obviously worse nodes. Therefore, A* algorithm can find the path more quickly than Dijkstra's algorithm, and still acquires a very similar solution.

However, for dynamic environment, algorithms with one-time computation, such as Dijkstra's and A* algorithms, are quite inefficient. Because when the environment changes (for example, new obstacle appears), Dijkstra's or A* algorithms must run a complete search again from present location to the goal. And this needs to spend a lot of computing time. Another problem is that while only parts of environmental information are known, then Dijkstra's and A* algorithms will have very poor responses, even can't begin working.

In 1994, Anthony Stentz proposed D* algorithm [3][4] which is mainly to resolve the problem of Dijkstra's and A* algorithm that can't handle dynamic environments. In 2005, Sven Koenig proposed D*-Lite algorithm [6] which is easier to be understood and has better efficiency than D* algorithm. D*-Lite algorithm doesn't base on D* algorithm, but is built on LPA* algorithm [5] which was also proposed by Sven Koenig. However, D* and D*-Lite algorithms still have to run a complete search from start to goal at first so that there will be also a waiting time before finishing first search, especially for the large map.

In the past two decades, artificial potential fields (APF) which had been proposed by Khatib [7], is widely used in path-planning for manipulator and mobile robot. At first, APF was designed for manipulator to avoid collision when grabbing objects. However, later it was discovered that APF is also good at handling the path-planning problem of mobile robots, and can create a very smooth trajectory. Although the theory of APF is very simple, however, its problems are also very obvious. The most one is the local-minimum problem. When the resultant of repulsive and attractive force on robot is zero, robot will hover at this place and stop moving. There were many methods proposed to solve this problem. For example, adding a random disturbance force robot to move when robot is in the situation of local minimum, or combining

Manuscript received March 10, 2010. This work was supported in part by the Department of Mechanical Engineering of National Chiao Tung University.

P. Y. Cheng, is with the Department of Mechanical Engineering of National Chiao Tung University, 1001 University Road, Hsinchu, Taiwan 300, ROC (e-mail: pycheng@cc.nctu.edu.tw).

P. J. Chen, is with the Department of Mechanical Engineering of National Chiao Tung University, 1001 University Road, Hsinchu, Taiwan 300, ROC (e-mail: rexandimmj@hotmail.com).

APF with other artificial intelligent methods helps robot to leave the region of local minimum. These methods reduces the probability that robot goes into a local solution. However, when the size and sharp of obstacles are large and complex, it is still difficult to solve this problem completely. Besides, there are other problems in artificial potential fields, such as oscillate in narrow channel, or can't pass through small-size doors. There are more detailed descriptions in [8][9].

The other artificial intelligence methods, such as genetic algorithms [10], neural networks [11], and fuzzy [12][13]…etc., have a great deal of achievements in real-time, collision-free, and dynamic problem. However, for complex terrain or maze-type map, these methods still become relatively inefficient, and often result in unsatisfying solutions.

For mobile robot, there are some issues needed to be solved in reality. (1) Generally it is best that robot can travel in a changeless environment, and obtain global environmental information prior. However, in most of situations, it is difficult to acquire all of the environmental information every time before path-planning. Collecting information and building model for complex and large environment are not an easy job. (2) Even if robot already has acquired all environmental information before path-planning, however, environmental information may change, for example, pedestrians or moving objects. Therefore, robot must have the ability to deal with unknown and dynamic environment in real-time. Otherwise it will easily lead to accidents and disasters.

II. D++ Algorithm

D++ algorithm is an improved algorithm which combines Dijkstra's algorithm with sensor-based path-planning method [15]. Traditionally, the Dijkstra's algorithm belongs to global path-planning method with one-time computation, and can't solve the problem of unknown and uncertain environment. Besides, for large map, Dijkstra's algorithm needs to spend a lot of computing time before making any response. In order to overcome this situation, we add the idea of "detective range" into Dijkstra's algorithm so that robot can only deal with local environmental information in a cycle time. The detective range is similar to an area observed by sensor. Robot only needs to search a waypoint which is nearest to goal at present in every cycle, and decide how to move next. By keeping searching waypoints and next steps, robot will gradually approach goal until robot reaches it. The simple conception of D++ algorithm is shown in Figure 1.

In Figure 1, because the searching space of each loop is only within a detective range, the computing time will be very short if the detective range is small. Thus, robot could have high response in real-time system. Besides, by observing the information of detective range, robot can ignore and avoid many vain paths or local solutions as long as the detective range is large enough.

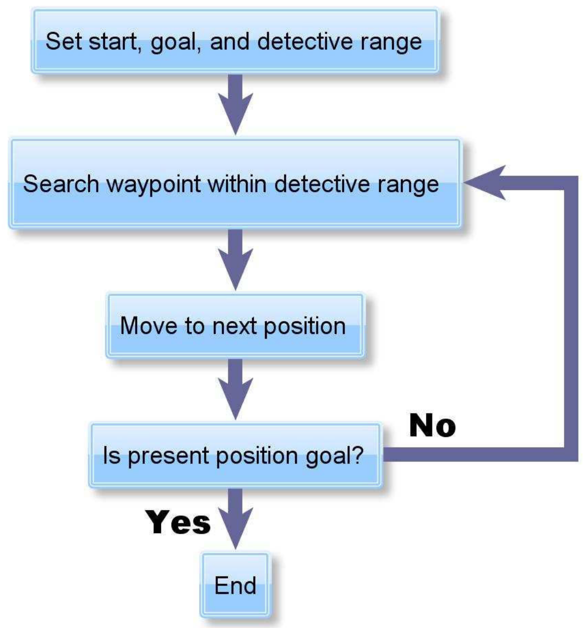
Set start, goal, and detective range
Search waypoint within detective range
Move to next position
Is present position goal?
No
Yes
End

Fig. 1. The simple conception of D++ algorithm

The Dijkstra's and A* algorithms usually use "Open" and "Close" lists to record the information of nodes, such as the father-node, cost, and that nodes are visited or not. Since we only use Dijkstra's algorithm within detective range, and will clear the contents of "Open" and "Close" lists before next cycle. Thus, these two lists couldn't record that nodes are visited or not for whole map in D++ algorithm. This will make robot visit old nodes again and again, and result in a local solution. In order to avoid this situation, D++ algorithm follows the mode of D* algorithm to record the status of nodes additionally. When the searching is beginning, status of all nodes are "New". Nodes will be changed their status to "Old" if they have been detected or visited once. Therefore, we add a "Select" list in D++ algorithm. Before the searching of every cycle is beginning, the "Select" list will be also cleared like Open and Close lists. In one searching cycle, nodes which are discovered by Dijkstra's algorithm will be checked that they are "New" or "Old". If node is "New", it will be put into the "Select" list, or it will be ignored. After all nodes in detective range are discovered and checked, a waypoint which is nearest to goal in "Select" list will be picked. Then, by tracing the father-node from waypoint to present location of robot, robot can decide how to move next.

Generally, when the searching space in one cycle reaches the size of detective range, we will end this cycle and pick the waypoint from "Select" list. However, if robot is into a dead end, such as a "U" trap, it is very possible that all of the nodes in detective range are "Old". This will result in that there is no node in "Select" list to be picked. This will causes that robot can't decide how to move next, then gets stuck. Therefore, in order to avoid this, D++ algorithm will keep expanding searching space which is over detective range until there is a

"New" node found. In real situation, because the "Old" nodes have been visited and checked, robot can record the information of "Old" nodes into its memory. Therefore, even if robot is in an area where all nodes are "Old" and is much larger than detective range, robot still can run the search in its "brain" and need not to move around to detect environment. In this way, robot won't fall into a local solution, and be sure to find a path to goal.

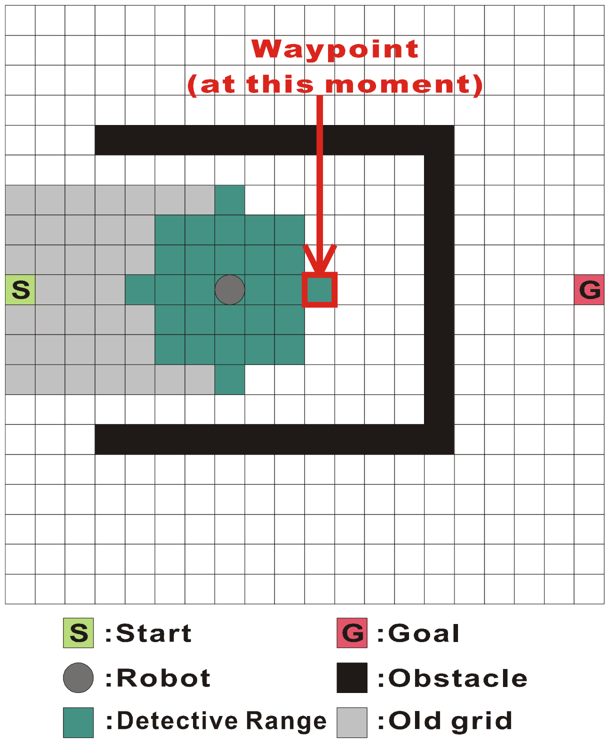

Fig. 2. The legends of D++ algorithm in this paper

By rearranging the idea and contentions described above, we detail D++ algorithm step by step as Figure 3. The waypoint is the node that has minimum distance to goal in "Select" List. The formula of calculating distance between waypoint and goal is described as following:

$$\text{distance} = \sqrt{\left(x_W - x_G\right)^2 + \left(y_W - y_G\right)^2} \quad (1)$$

where x_w means the position of waypoint in x axis, x_G means the position of "Goal" in x axis, y_w means the position of waypoint in y axis, and y_G means the position of "Goal" in y axis.

Then we demonstrated a simple example of D++ algorithm in figure 3. We set the detective range for one grid. In terms of the above steps, figure 4(a) presents the step 1 which the primary work is to initialize the process. Figure 4(b) shows that robot had finished the first cycle (step 2-12), then it picked the upper right grid as waypoint from "Select" list (step 9). In this case, robot moves one grid per cycle. And because the detective range is equal to only one grid in this case, the waypoint is also the robot's next position. Then the robot's position in Figure 4(c) was not the "Goal" (step 12), so robot continued to next cycle (step 2-11). Figure 4(c) shows that robot had found the "Goal" within detective range, so the process jumped from step 4 to 9. Then robot picked the waypoint (step 9), and moved robot to next position again (step 11). In Figure 3(d), because robot had reached the "Goal" (step 12), then we could end the whole searching process. From the case in Figure 4, it shows that D++ algorithm will be similar to the greedy best-first method if the detective range is quite small. This will cause that path may not be shortest, even path will be so long to waste a lot of time and energy. To avoid this, it is necessary to increase the size of detective range appropriately. We will compare the results with different size of detective range in next section.

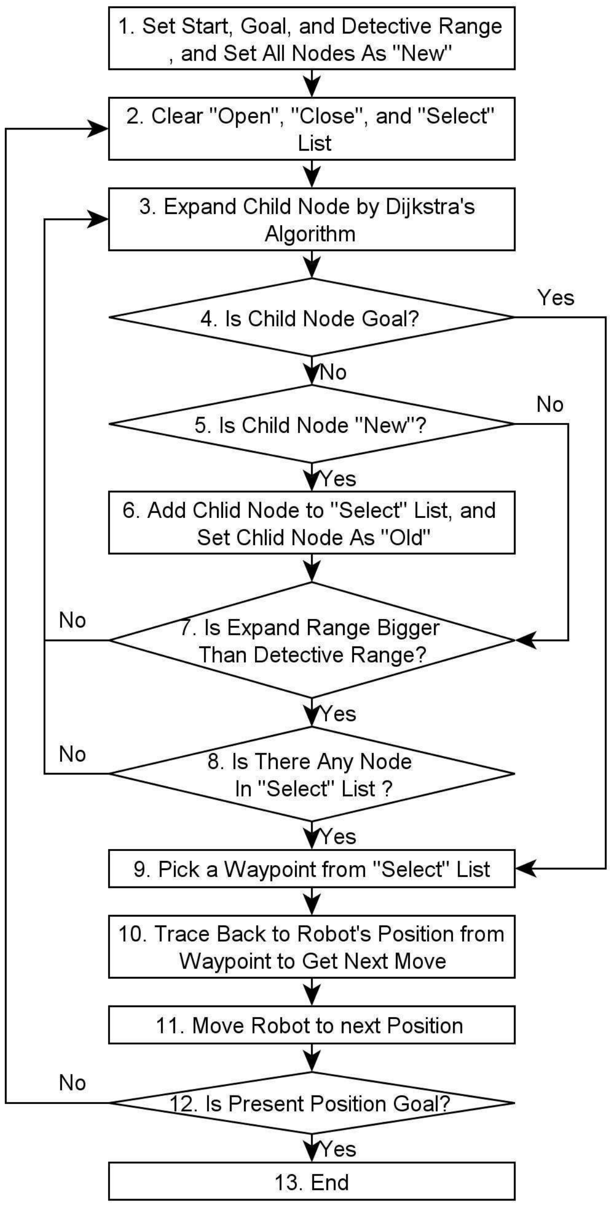
1. Set Start, Goal, and Detective Range , and Set All Nodes As "New"
2. Clear "Open", "Close", and "Select" List
3. Expand Child Node by Dijkstra's Algorithm
4. Is Child Node Goal?
Yes
No
5. Is Child Node "New"?
No
Yes
6. Add Chlid Node to "Select" List, and Set Chlid Node As "Old"
No
7. Is Expand Range Bigger Than Detective Range?
Yes
No
8. Is There Any Node In "Select" List ?
Yes
9. Pick a Waypoint from "Select" List
10. Trace Back to Robot's Position from Waypoint to Get Next Move
11. Move Robot to next Position
No
12. Is Present Position Goal?
Yes
13. End

Fig. 3. The flowchart of D++ algorithm

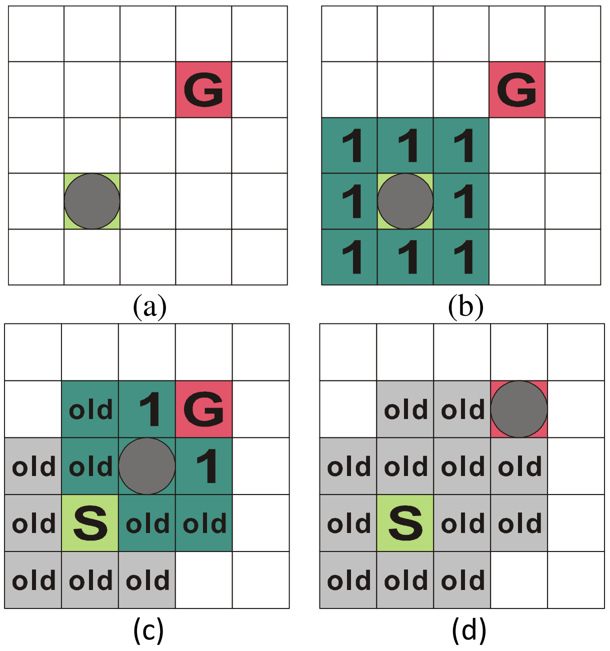

Fig. 4. A simple example of D++ algorithm with the detective range of one grid

III. Simulation Studies

In this section, we demonstrate some simulations to verify the performance and efficiency of D++ algorithm. At first, we ran the simulation for static problem, and compared the results with different sizes of detective range. Then we ran the simulation with a dynamic environment to verify the capability of D++ algorithm for real-time and collision-free problem. In order to provide a reference, we listed the specifications of our computer's hardware and software in Table I.

TABLE I
Specification of computer

Item	Specification
CPU	Intel Core 2 Duo T6600 2.2GHz
RAM	DDR2-800 2GB
VGA	ATI Radeon HD4650 512 MB
OS	Microsoft Windows XP
IDE	SharpDevelop

A. *Static problem*

Static problem means that all objects and obstacles in environment are fixed and changeless. If environmental information is acquired prior, Dijkstra's algorithm can be sure of finding a shortest path from start to goal. When the detective range contains whole map, D++ algorithm will be equal to Dijkstra's algorithm in one cycle time so that D++ algorithm can also promise to find a shortest path. However, in this situation, it will spend a lot of computing time in every cycle, and have very low response. Oppositely, if the detective range is small, D++ algorithm will be similar to greedy best-first method so that it is very possible to result in a long and inefficient path. So it is better to set detective range as wide as possible if cycle time is small enough to approximate a real-time system.

We adopted the case which had been tested by Stentz [4] (shown in Figure 5) to demonstrate the results with different detective range. In Figure 6, the detective range was set for three grids and the cycle time was set for twenty milli-seconds. And in Figure 7, the detective range was set for ten grids and the cycle time was also set for twenty milli-seconds. The results of search are shown individually in Figure 6 and 7.

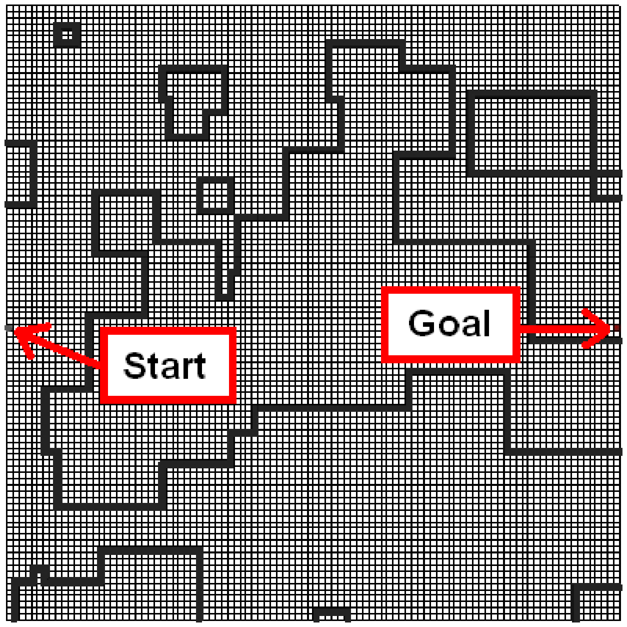

Fig. 5. The terrain which had been discussed by Stentz [4]

According to the result in Figure 6, it is quite obvious that path was long and passed through many unnecessary places. The primary reason is that the place where is near the start has two directions to choose. The detective range with three grids is not wide enough to make robot do correct decision. This affected the final result seriously. Then, in Figure 7, although only detection range was increased from three to ten grids, we obtained a much better solution than the one in Figure 6. On the other hand, the robot system still has high response because the computer hardware can sustain the load of computation in real-time. Thus, there is no pause or lag between the whole processes. We compared the result in Figure 7 with the one which was presented by Stentz, and we can see two very similar paths which were acquired by D++ and D* algorithms respectively. However, D++ algorithm used less searching space, and didn't need to run one complete search at first.

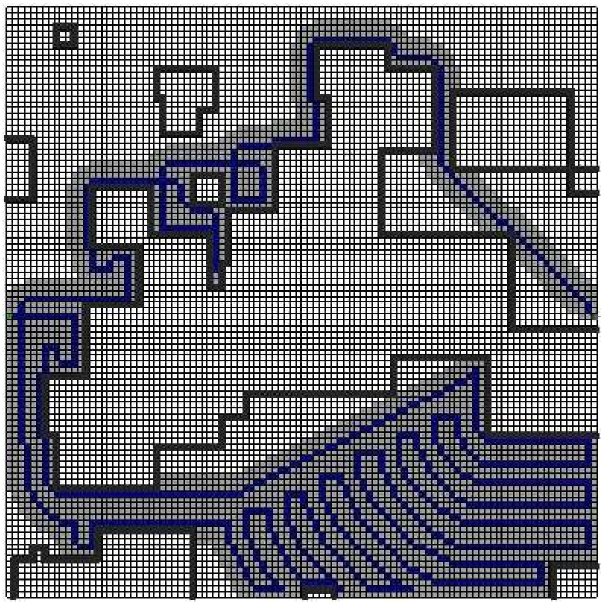
Fig. 6. The result with the detective range of **3** grids, and the cycle time is **20** milli-seconds. The total time is about **26** seconds.

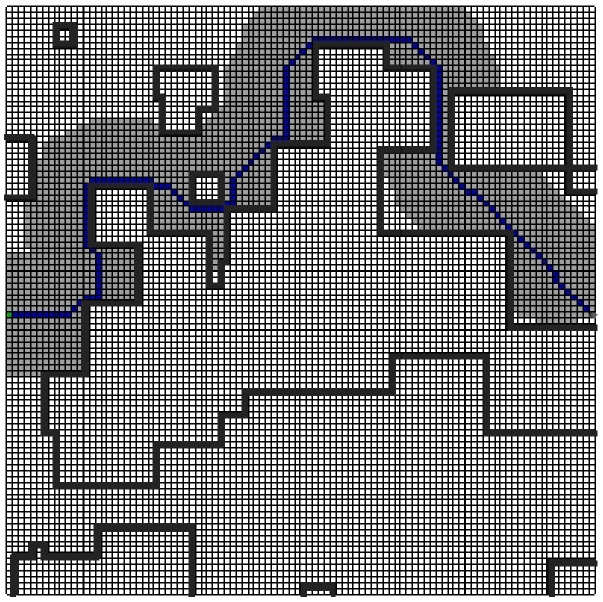
Fig. 7. The result with the detective range of **10** grids, and the cycle time is **20** milli-seconds. The total time is about **5** seconds.

Then we also adopted the case which was discussed by Willms, AR [14] (shown in Figure 8). It is a maze, and for some path-planning methods, such as artificial potential fields, fuzzy, neural network, and genetic algorithm, is very difficult to obtain a good and efficient solution. Even it may result in a local solution and never reach a goal. We also set the cycle time for twenty milli-seconds, and set the detective range for twenty grids. The result is shown in Figure 7. We acquired the same path which was also obtained by Willms, AR. However, the method of Willms, AR has poor efficiency like Dijkstra's and A* algorithm especially for large maps.

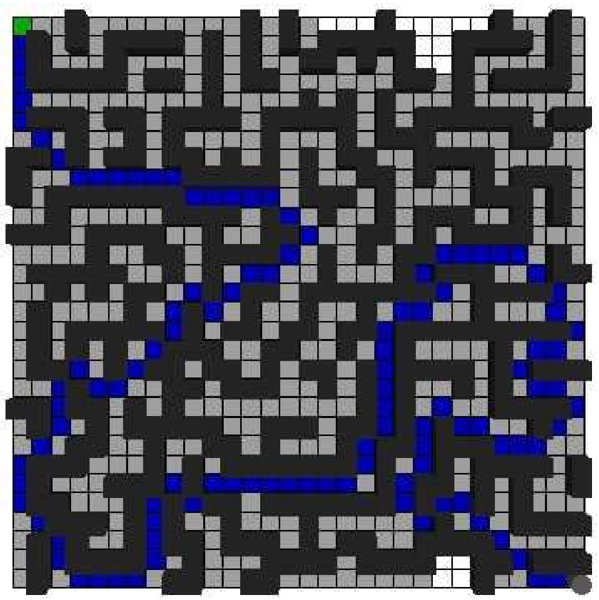
Fig. 8. The result with the detective range of **20** grids, and the cycle time is **20** milli-seconds. The total time is about **4** seconds.

B. Dynamic problem

In this simulation, there were nine objects which were in the middle of map and moved one grid randomly per cycle time (shown in Fig. 9). We set the detective range for ten grids, and set the cycle time for two hundreds milli-seconds. The result of simulation is shown in Fig. 10. The key technology which D++ algorithm can solve dynamic problem is that D++ algorithm detects environment and runs a search in every cycle time. Thus, if the cycle time is small enough then it will almost become a real-time system to deal with dynamic environment.

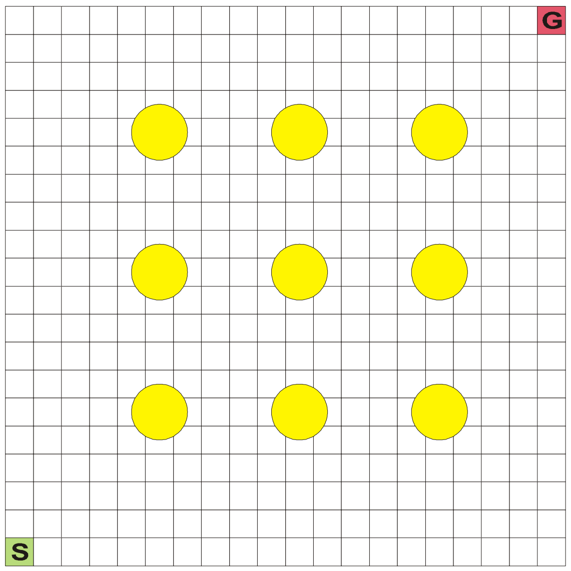

Fig. 9. The case of dynamic problem. There are nine objects in the middle of map, and they move one grid randomly per cycle time.

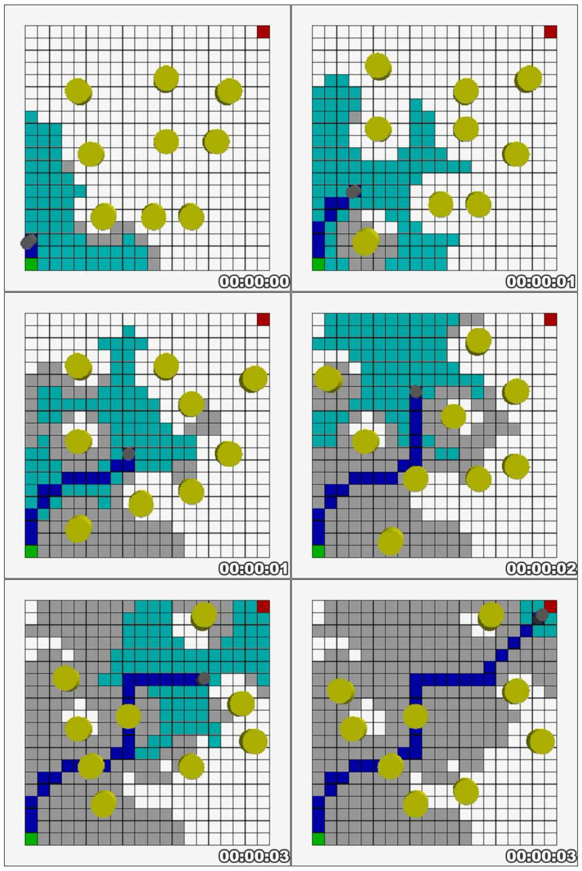

Fig. 10. The result with the detective range of **10** grids, and the cycle time is **100** milli-seconds. The total time is about **3** seconds.

Acknowledgment

The authors would like to thank Kun- Min Huang and Chuen-Fu Wu for their help. This work was supported by Metal Industries Research & Development Centre, Taiwan.

IV. Conclusion

In this paper, we proposed the D++ algorithm to solve general problems of path-planning methods at present for mobile robot. The main idea of D++ algorithm is that combining Dijkstra's algorithm with sensor-based method to make robot run Dijkstra's algorithm with a small range of searching space which we call detective range in this paper within one cycle time. Through the design of detection range, D++ algorithm will be more flexible than Dijkstra's algorithm to make robot have higher response or better capacity of finding the shortest path. Therefore, adjusting the size of detective range will have a great effect on the performance of D++ algorithm. Comparing with Dijkstra's and A* algorithms, D++ algorithm can avoid a waiting time for a large map, and also can deal with dynamic problem which is difficult to be solved by Dijkstra's and A* algorithms generally. Besides, comparing with artificial potential fields, D++ algorithm can easier leave the region of local solution, even won't approach some regions of local solution such as a trap if the detective range is wide enough. Therefore, D++ algorithm is not only very suitable for large maps, but also complex maps, such as a maze. Because D++ algorithm is based on Dijkstra's algorithm, it can ensure reaching a goal like Dijkstra's algorithm if there is at least one path to goal. In future, we will continue this research to apply D++ algorithm to real mobile robot.

References

[1] Dijkstra, E. W. "A note on two problems in connexion with graphs," Numerische Mathematik 1: 269–271, 1959.

[2] Hart, P. E.; Nilsson, N. J.; Raphael, B. (1972). "Correction to A Formal Basis for the Heuristic Determination of Minimum Cost Paths," SIGART Newsletter 37: 28–29.

[3] A. Stentz, "Optimal and Efficient Path-planning for Partially-Known Environments," Proceedings of the International Conference on Robotics and Automation, 1994,3310–3317.

[4] A. Stentz. "The Focused D* Algorithm for Real-Time Replanning," Proceedings of the International Joint Conference on Artificial Intelligence, 1652–1659, 1995.

[5] S. Koenig, M. Likhachev and D. Furcy. "Lifelong Planning A*," Artificial Intelligence Journal, 155, (1-2), 93-146, 2004.

[6] S. Koenig and M. Likhachev, "D* Lite," In Proceedings of the Eighteenth National Conference on Artificial Intelligence, pp. 476-483, July 2002.

[7] Khatib, O. "Real-time obstacle avoidance for manipulators and mobile robots," Robotics and Automation. Proceedings. 1985 IEEE International Conference on Volume 2, Mar 1985 Page(s):500 – 505

[8] Koren, Y.; Borenstein, J., "Potential field methods and their inherent limitations for mobile robot navigation," Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on 9-11 April 1991 Page(s):1398 - 1404 vol.2, Digital Object Identifier 10.1109/ROBOT.1991.131810

[9] Charifa S., Bikdash M., "Comparison of geometrical, kinematic, and dynamic performance of several potential field methods," Southeastcon, 2009. SOUTHEASTCON '09. IEEE Digital Object Identifier: 10.1109/SECON.2009.5174043, Publication Year: 2009 , Page(s): 18 - 23

[10] Ismail AL-Taharwa, Alaa Sheta and Mohammed Al-Weshah, "A Mobile Robot Path-planning Using Genetic Algorithm in Static Environment," Journal of Computer Science 4 (4): 341-344, 2008, ISSN 1549-3636

[11] Simon X. Yang, Max Meng., "Neural Network Approaches to Dynamic Collision-Free Trajectory Generation," IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics, Vol.31 No.3 pp.302-318, June, 2001

[12] Senthilkumar K.S., Bharadwaj K.K., "Hybrid Genetic-Fuzzy Approach to Autonomous Mobile Robot," Technologies for Practical Robot Applications, 2009. TePRA 2009. IEEE International Conference on Digital Object Identifier: 10.1109/TEPRA.2009.5339649, Publication Year: 2009 , Page(s): 29 – 34

[13] Hassanzadeh I., Sadigh S.M., "Path-planning for a mobile robot using fuzzy logic controller tuned by GA," Mechatronics and its Applications, 2009. ISMA '09. 6th International Symposium on Digital Object Identifier: 10.1109/ISMA.2009.5164798, Publication Year: 2009 , Page(s): 1 - 5

[14] Willms, A.R.; Yang, S.X., "An efficient dynamic system for real-time robot-path-planning ," Volume 36, Issue 4, Aug. 2006 Page(s):755 - 766 , Digital Object Identifier 10.1109/TSMCB.2005.862724

[15] Myungsik Kim, Nak Young Chong, Wonpil Yu, "Fusion of direction sensing RFID and sonar for mobile robot docking," Automation Science and Engineering, 2008. CASE 2008. IEEE International Conference on Digital Object Identifier: 10.1109/COASE.2008.4626436, Publication Year: 2008 , Page(s): 709 - 714