Intel Serv Robotics
DOI 10.1007/s11370-012-0120-4

SPECIAL ISSUE

Navigation of mobile robot by using D++ algorithm

Pi-Ying Cheng · Pin-Jyun Chen

Received: 12 April 2012 / Accepted: 10 September 2012

Abstract The navigation of mobile robots is a vital aspect of technology in robotics. We applied the D++ algorithm, which is a novel and improved path-planning algorithm, to the navigation of mobile robots. The D++ algorithm combines Dijkstra's algorithm with the idea of a sensor-based method, such that Dijkstra's algorithm is adapted to local search, and the robot can determine its next move in real-time. Although the D++ algorithm frequently runs local search with limited ranges, it can compute optimum paths by expanding the size of the searching range to avoid local minima. In addition, we verified the performance of the D++ algorithm by applying it to a real robot in a number of environments. The use of the D++ algorithm enables robots to navigate efficiently in unknown, large, complex and dynamic environments.

Keywords Mobile robot · Path planning · Dijkstra's algorithm · Real time

1 Introduction

During the past few decades, the navigation of mobile robots has attracted notable attention, and considerable research was developed. The main technology of navigation in unknown or uncertain environments includes exploration, mapping, localization, motion control, and path-planning [1,2]. Path planning is one of the main technologies to direct a robot from a starting point to its destination. Dijkstra [3] and Hart [4] proposed Dijkstra's algorithm and A* to determine a shortest path in a known environment. However, Dijkstra's algorithm and A* explores numerous cells in an environment. Furthermore, Dijkstra's algorithm and A* require the global information of the environment in advance. Therefore, they are only suitable to static and smaller environments. Stentz [5,6] proposed the D* algorithm, which manages dynamic environments. Koenig subsequently proposed the Life Long A* (LPA*) [7] and D*-Lite algorithm [8] to improve the efficiency limitations of the D* algorithm. However, D* and D*-Lite algorithms also require information of the global environment. Therefore, these approaches involve the initial running of global searches from the start point to the destination, and lead to long delays for the first search operation in large environments. Therefore, these algorithms are only suitable for applications in which the information of the environment is determined in advance, such as in GPS navigators or video games.

In recent years, research on the navigation of mobile robots has focused on local searching in dynamic environments. Khatib [9] proposed using artificial potential fields (APF) as a type of local-searching approach to solve the path planning problem of manipulators and mobile robots. The theory of APF is quite simple and demonstrates high response times in large environments and real-time operation. Moreover, the APF can generate extremely smooth trajectories. However, the APF exhibits considerable problems in local minima. Other shortcomings of the APF include the robot oscillating in a narrow channel and being incapable of passing through small-sized doors [10,11].

Moreover, several novel and improved approaches based on fuzzy logic or neural networks have been proposed for the mobile robot navigation problem in unknown environ-

P.-Y. Cheng · P.-J. Chen (✉)
Department of Mechanical Engineering, National Chiao Tung University, EE437, 1001 Ta-Hsueh Road, Hsinchu 30010, Taiwan, R.O.C
e-mail: rexchen@mail.mirdc.org.tw; rexandimmj@hotmail.com

P.-Y. Cheng
e-mail: pycheng@cc.nctu.edu.tw

Published online: 05 October 2012

ments [12,13]. According to the varied information of the environments that are obtained by the camera, the mobile robot exhibits various behaviors (e.g., obstacle avoidance, road following, and target detection) when the algorithm of neural networks is used. In addition, numerous previous studies have solved the problem of path-planning by using fuzzy logic [14–16] and have established the definition of membership function for the direction and position of the mobile robot, having the robot select the best behavior (based on expert rules) to avoid obstacles and reach a target. However, these approaches of local searching usually present the problem of local solution. Therefore, they are often applied to collision avoidance rather than path planning. Otherwise, they must be combined with other approaches, such as the A* algorithm or the wall-following method, to escape from the local minimum area.

In this study, we propose a novel algorithm, that is, the D++ algorithm [17], which combines Dijkstra's algorithm with the idea of a sensor-based method [18], to overcome the common limitations of the path planning methods. We demonstrate the results of our experiment by using the D++ algorithm for a mobile robot to determine an efficient path to its destination in a number of static and dynamic environments.

2 D++ algorithm

The D++ algorithm was thus named because it was derived from Dijkstra's algorithm; however, it is an improvement of Dijkstra's algorithm, which was inefficient for unknown and dynamic environments. This chapter describes the concept and procedures of the D++ algorithm and demonstrates numerous simulations to illustrate its operation. The first section introduces the fundamental background of Dijkstra's algorithm and describes and explains the D++ algorithm. Finally, we describe some simulations to verify the performance of the D++ algorithm.

2.1 Dijkstra's algorithm

Dijkstra's algorithm belongs to a class of path planning approaches that involve using a one-time search in a global environment. The algorithm is a graph-search method (see Fig. 1) that involves using *OPEN* and *CLOSE* lists to save and evaluate the cost of cells in the environment. The *OPEN* list saves the cells that have been identified but not selected yet, and the *CLOSE* list saves the cells that have been selected. Figure 2 shows the workflow of Dijkstra's algorithm. The concept of Dijkstra's algorithm entails expanding child cells that are nearer the start point repeatedly until the goal is discovered. Thus, Steps 2–11 in Fig. 2 are continually executed until the goal is identified. Therefore, in large environments, Dijkstra's algorithm requires a considerable amount of time to compute the shortest path before the robot can begin moving. Furthermore, Dijkstra's algorithm requires an environment's global information in advance; therefore, it is entirely inadaptable to unknown environments.

In addition, the cost of cell in Step 2 of Fig. 2 indicates the traveling length from the *START* to the current cell [see Eq. (1)].

$$\cos t_{\mathrm{C}} = \cos t_{\mathrm{L}} + \sqrt{(X_{\mathrm{C}} - X_{\mathrm{L}})^2 + (Y_{\mathrm{C}} - Y_{\mathrm{L}})^2} \tag{1}$$

where, $\mathrm{cost}_{\mathrm{C}}$ denotes the cost of the current cell and cost_{L} denotes the cost of the previously encountered cell. X_{C} indicates the x-position of the current cell, X_{L} indicates the x-position of the previous cell, Y_{C} denotes the y-position of the current cell, and Y_{L} denotes the y-position of the previous cell.

2.2 D++ algorithm

To overcome the limitations of Dijkstra's algorithm mentioned in the previous section, the concept of a *DETECTIVE RANGE* was added to Dijkstra's algorithm, thereby requiring the robot to manage only the local environment information.

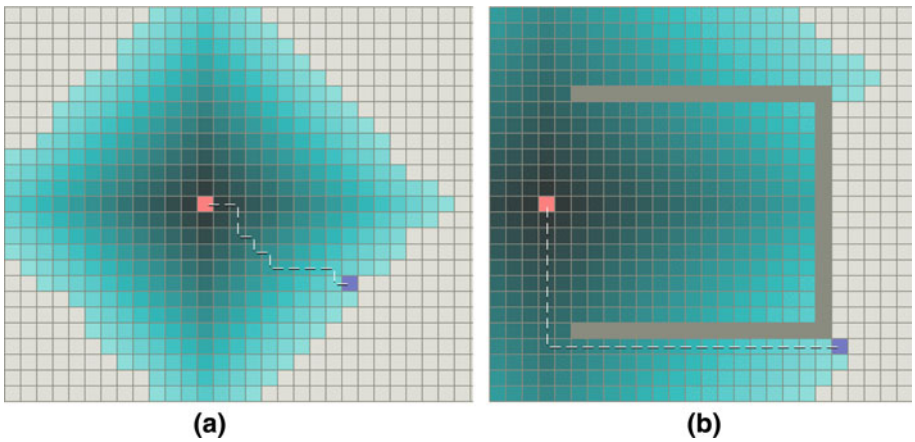

Fig. 1 Two examples of Dijkstra's algorithm: **a** no obstacle and **b** with U-shaped obstacle (from Amit's thoughts on path-finding and A-star)

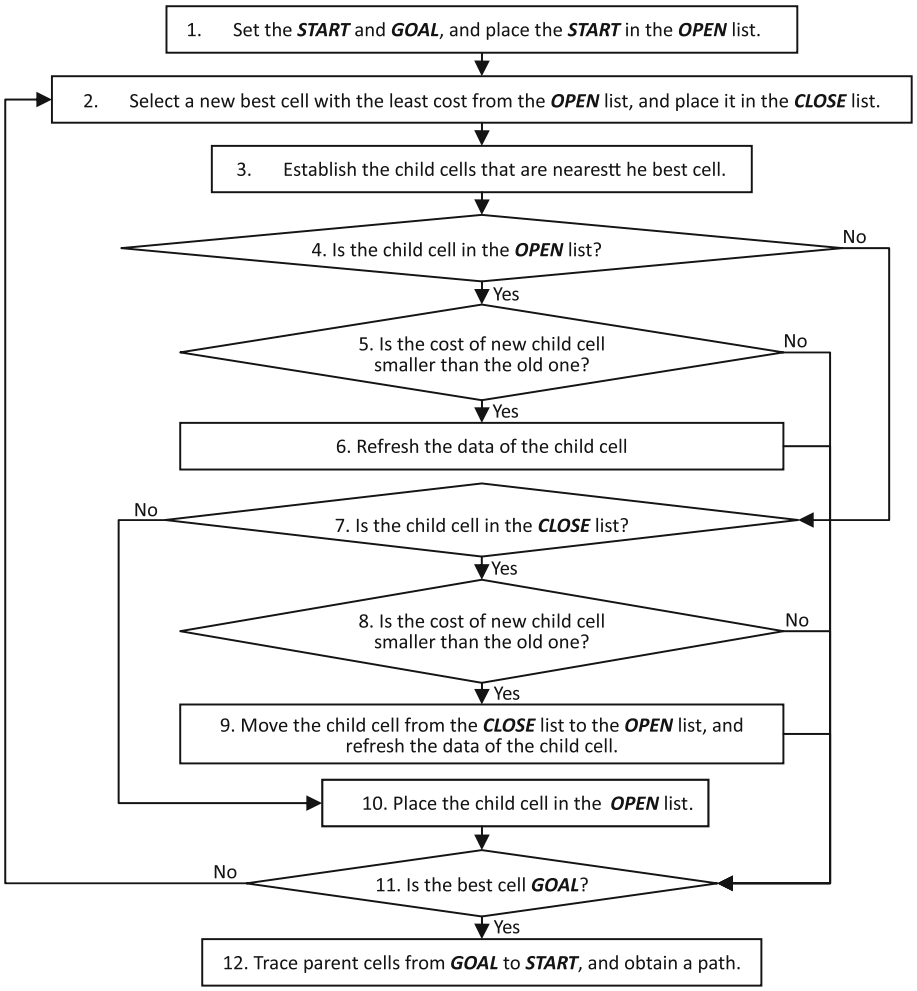

Fig. 2 Workflow of Dijkstra's algorithm

The *DETECTIVE RANGE* is similar to an area observed by a sensor. A robot must search for a *WAYPOINT* only within the current *DETECTIVE RANGE*. The *WAYPOINT* is the cell that has the shortest distance to *GOAL* [Eq. (2)].

$$\cos t_{\mathrm{E}} = \sqrt{(X_{\mathrm{G}} - X_{\mathrm{C}})^2 + (Y_{\mathrm{G}} - Y_{\mathrm{C}})^2} \tag{2}$$

where, $\mathrm{cost}_{\mathrm{E}}$ denotes the expected cost from the current cell to *GOAL*, X_{C} indicates the x-position of the current cell, Y_{C} denotes the y-position of the current cell, X_{G} denotes the x-position of the *GOAL*, and Y_{G} denotes the y-position of the *GOAL*. Only searching for a *WAYPOINT* within a limited range allows it to determine its next move immediately. Thus, the robot reaches its destination by continually searching for *WAYPOINTS* and following the consequent steps. Therefore, the D++ algorithm not only involves using Dijkstra's algorithm to expand the child cell from the robot's current position [use Eq. (1)] but involves applying the concept of the heuristics of the A* algorithm to evaluate the worth of the cells first by searching for *WAYPOINTS* [use Eq. (2)]. Figure 3 shows the legends that were used for all figures in this study, and Fig. 4 shows the workflow of the D++ algorithm. As shown in Fig. 4, the duration of the searching was considerably short because the searching range (*DETECTIVE RANGE*) for each loop (Step 2–12) was lim-

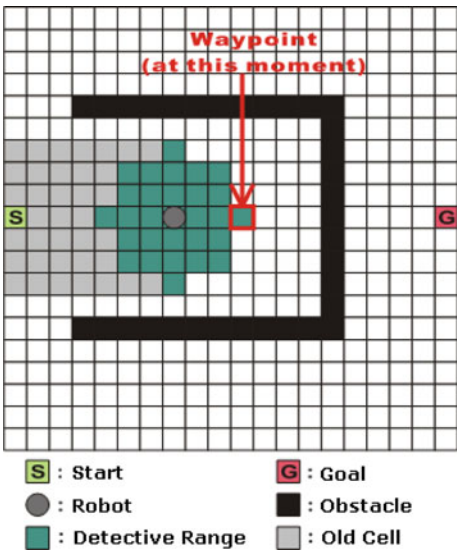

Fig. 3 Figure legends used in this paper

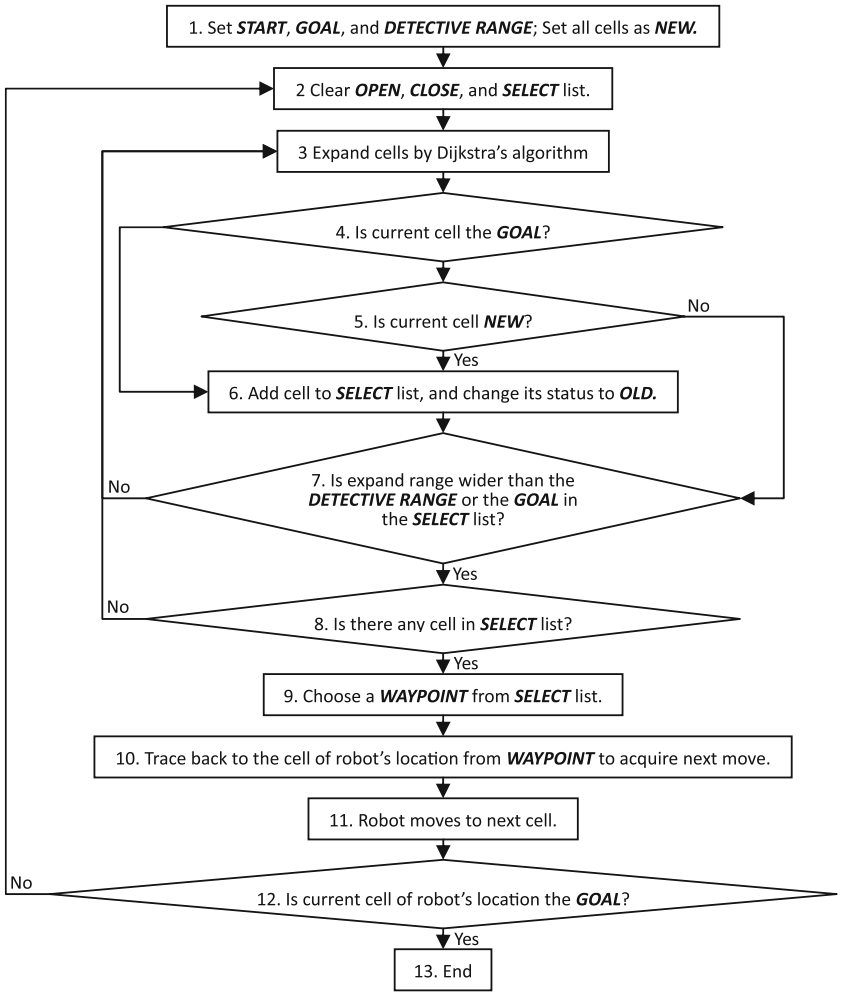

Fig. 4 Workflow of D++ algorithm

ited. Therefore, a robot can complete the first searching rapidly and begin moving. Moreover, a robot requires only the information that is relevant to the *DETECTIVE RANGE* rather than that of the entire global environment.

Because the D++ algorithm involves using only Dijkstra's algorithm for each searching loop, it must clear the contents of the *OPEN* and *CLOSE* lists every time before beginning the next loop (Step 2 in Fig. 4). Therefore, these two lists cannot record the information beyond one searching loop; that is, there are no permanent references to the global environment. This causes a robot to select repeatedly the old cells and eventually the robot tends to circle a small local environment. To avoid this situation, the D++ algorithm follows the approach of the D* algorithm [5,6] to record further the status of the cells after a loop search. When the searching process begins, all of the cells are set to a *NEW* status. If a cell has been detected or visited once, its status is changed to *OLD*. In addition, the D++ algorithm involves using a *SELECT* list. The statuses of the cells that are identified in each loop are assessed. If a cell is *NEW*, then it is moved to the *SELECT* list (Step 6 in Fig. 4). Otherwise, the cell is ignored. After all the cells in the *DETECTIVE RANGE* are identified and checked, the *WAYPOINT* that is nearest to the *GOAL* in the *SELECT* list is chosen (Step 9 in Fig. 4). Subsequently, the robot decides its next move by tracing their parent cells from this *WAYPOINT* to its current location (Step 10 in Fig. 4). Before the next loop begins, the contents of the *SELECT* list are cleared along with those of the *OPEN* and *CLOSE* lists.

In general, when the searching range for a loop reaches the size of the initial *DETECTIVE RANGE*, the process ends this loop and selects a *WAYPOINT* from the *SELECT* list. However, if the robot reaches a dead end, such as a U-shaped obstacle, then all of the cells in the *DETECTIVE RANGE* may be *OLD* (see Fig. 5b). Consequently, no more cells are available in the *SELECT* list, and the robot is unable to determine its next move. Therefore, when a robot encounters this situation, the D++ algorithm expands the size of *DETECTIVE RANGE* until a *NEW* cell is located (Step 8 in Fig. 4 and see Fig. 5c). Thus, the robot determines a pathway out of its local confinement toward the *GOAL*.

Figure 6 demonstrates a simple example of the D++ operation. According to the procedure in Figs. 4, 6a indicates the

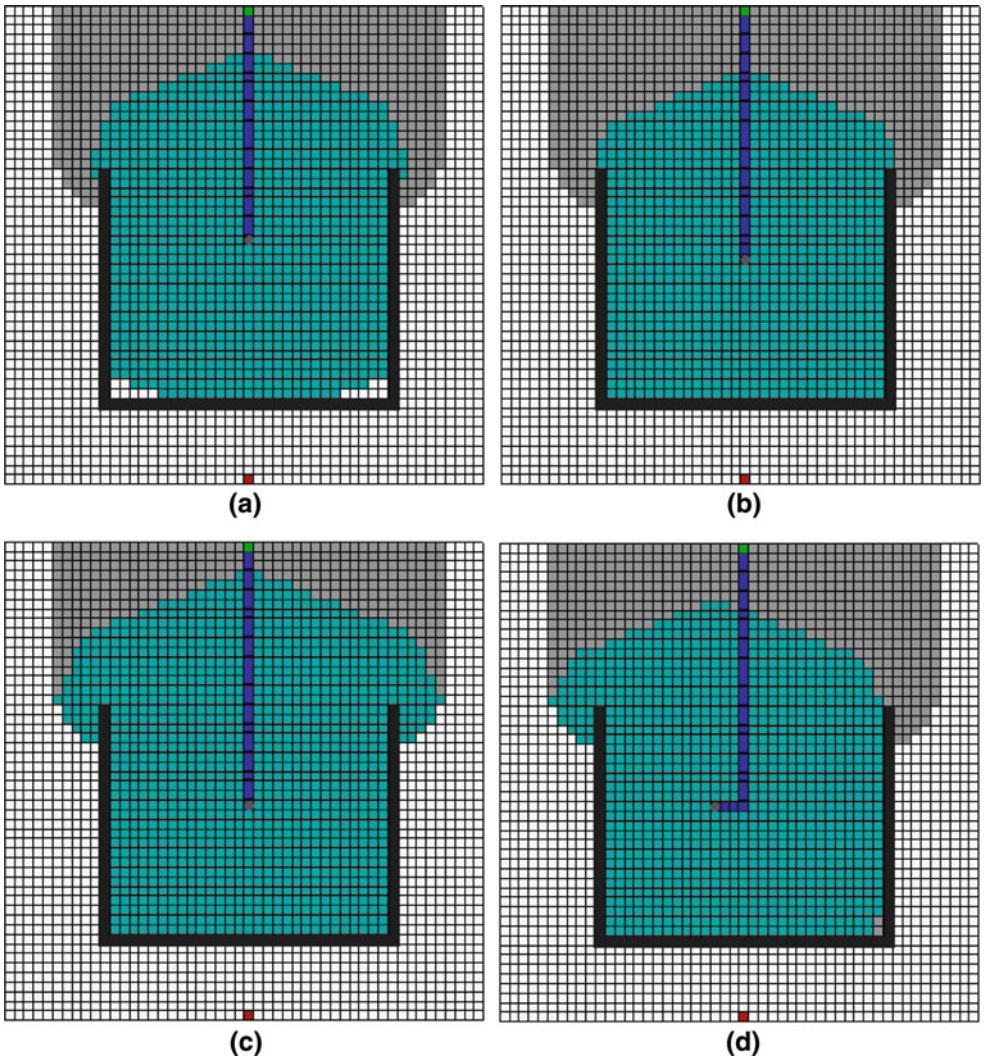

Fig. 5 Actions of D++ upon finding no cells in the *SELECT* list: **a** the robot encounters a U-shaped obstacle, **b** no *NEW* cells are present in the *SELECT* list, **c** D++ expands the *DETECTIVE RANGE* to locate *NEW* cells, and **d** the robot locates a *NEW* cell and moves toward it

initialization of the entire search process (Step 1). Figure 6b shows that the robot's *DETECTIVE RANGE* is set to one cell, and the robot has finished an initial search (Steps 2–8). Subsequently, it chooses the upper-right cell as a *WAYPOINT* from the *SELECT* list (Steps 9–11). Because the *DETECTIVE RANGE* is equal to only one cell in this example, the *WAYPOINT* is also the next location of the robot. Because the location of the robot in Fig. 6c is not the *GOAL*, it continues to execute the next searching loop (Steps 2–12). In Fig. 6d, the robot has reached the *GOAL*, thereby ending the search process. Based on the example shown in Fig. 6, the D++ algorithm with a small *DETECTIVE RANGE* is similar to the greedy best-first method [19]. This may result in the chosen path not always being the expected optimal path. To avoid this problem, magnifying the size of the *DETECTIVE RANGE* appropriately is necessary. The results for different sizes of the *DETECTIVE RANGE* are compared in the following section.

2.3 Simulation

In this section, we demonstrate a few simulations that were performed by using the D++ algorithm. In these simulations, the cycle time (the time for performing a search loop) was set at 20 ms. The specifications of the computer hardware used in this study are listed in Table 1 to provide a standard reference.

2.3.1 Static environments

In this case, the results of two simulations are demonstrated. One simulation had its *DETECTIVE RANGE* set to five cells (see Fig. 7), and the *DETECTIVE RANGE* for the other simulation was set to 20 cells (see Fig. 8). The path length in Fig. 8 is considerably greater than that in Fig. 8 because a *DETECTIVE RANGE* of five cells was not sufficiently wide

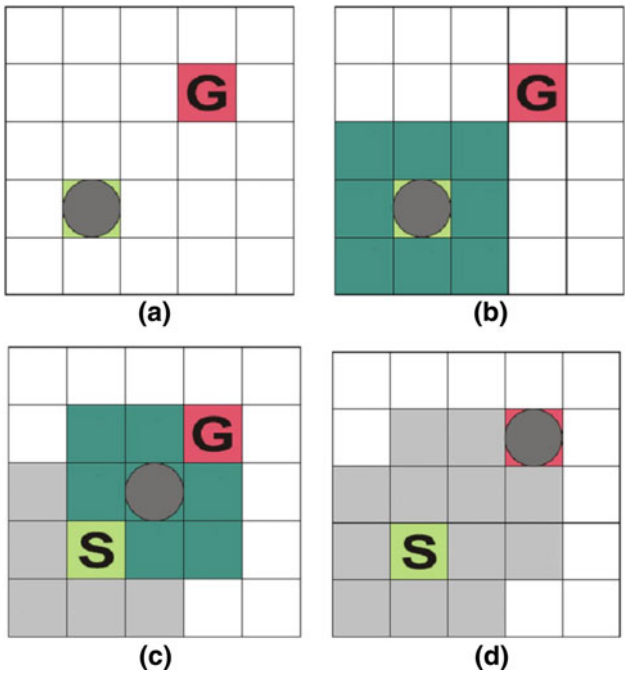

Fig. 6 An example of D++ operation with *DETECTIVE RANGE* setting of one cell

Table 1 Specifications of computer hardware used in this study

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

to enable the robot to avoid a number of vain cells (cells that were not optimally chosen). In contrast, because of the precognition of more vain cells that is possible with an increased *DETECTIVE RANGE* of 20 cells, the resultant path, as shown in Fig. 8, was shorter and more efficient.

2.3.2 Dynamic environments

In this simulation, nine objects were initially clustered in the middle of the environment (see Fig. 9); the objects were programmed to randomly move at a speed of one cell per 100 ms. As shown in Fig. 10, the robot was able to avoid objects in real-time and reached the goal without any collisions. Moreover, a definite buffer distance was observed beyond the searching range and the goal or the object because, in practical situations, the robot requires some clearance between itself and the obstacle for safety purposes.

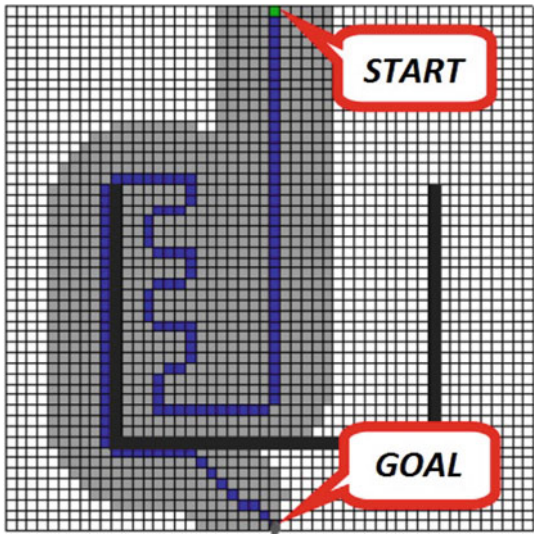

Fig. 7 Result for *DETECTIVE RANGE* setting of five cells. Total elapsed time for robot movement from *START* to *GOAL* is approximately 4 s

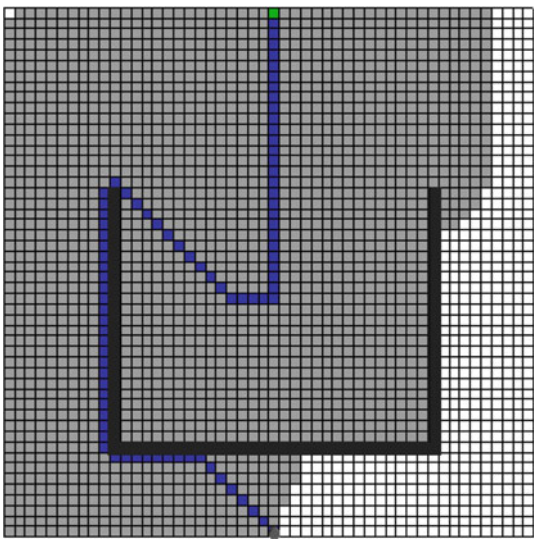

Fig. 8 Result with for *DETECTIVE RANGE* setting of 20 cells. Total elapsed time for robot movement from *START* to *GOAL* is approximately 3 s

2.4 Comparison

This section demonstrates some comparisons to distinguish the advantages and shortcomings of the D++ algorithm from other common approaches of path-planning, which are Dijkstra's algorithm, the A* algorithm, D* algorithm, and artificial potential fields.

(1) *Comparison with Dijkstra's algorithm and A*:* Fig. 11a shows a no-obstacle environment with a size of 100 × 100 cells. The *START* location was set at the lower-left corner, and the *GOAL* was set at the upper-right corner. In

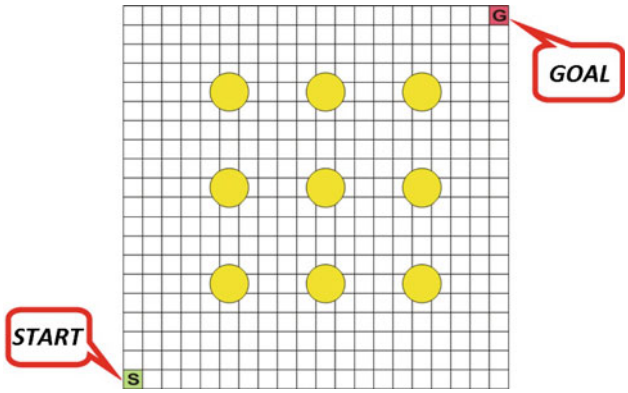

Fig. 9 Dynamic environment with nine randomly moving objects

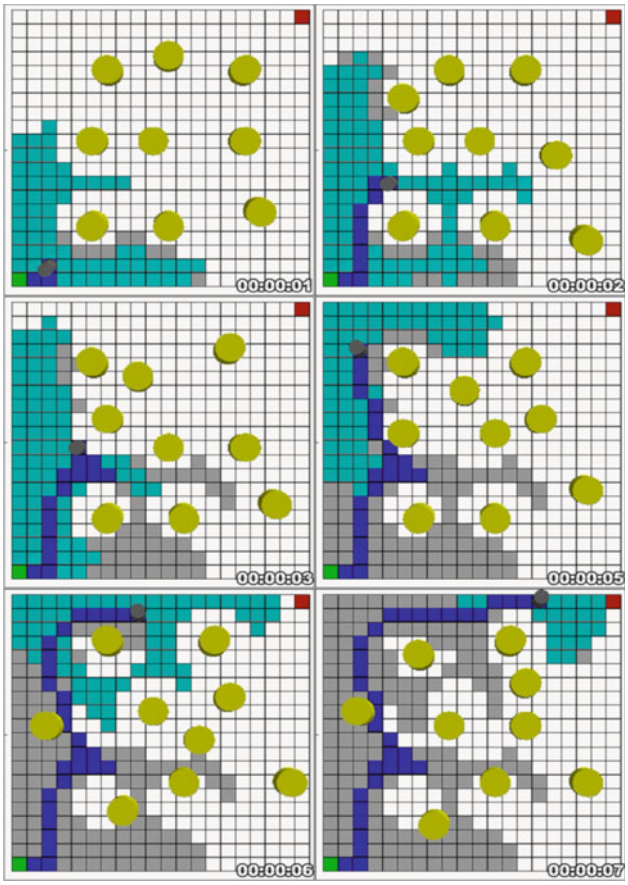

Fig. 10 Result for *DETECTIVE RANGE* setting of ten cells. Total time elapsed is approximately 8 s

Fig. 11b, the path was computed using the original Dijkstra's algorithm, and it required ("cost") approximately 12.1 s to compute the path before robot moved. In addition, the simulation using Dijkstra's algorithm visited all the cells in the environment. Figure 11c shows the results when the simulation used the A* algorithm to compute the shortest path, and the elapsed cost reduced to 5.2 s because the A* algorithm had to visit only half the total number of cells. In Fig. 11d, the use of the D++ algorithm

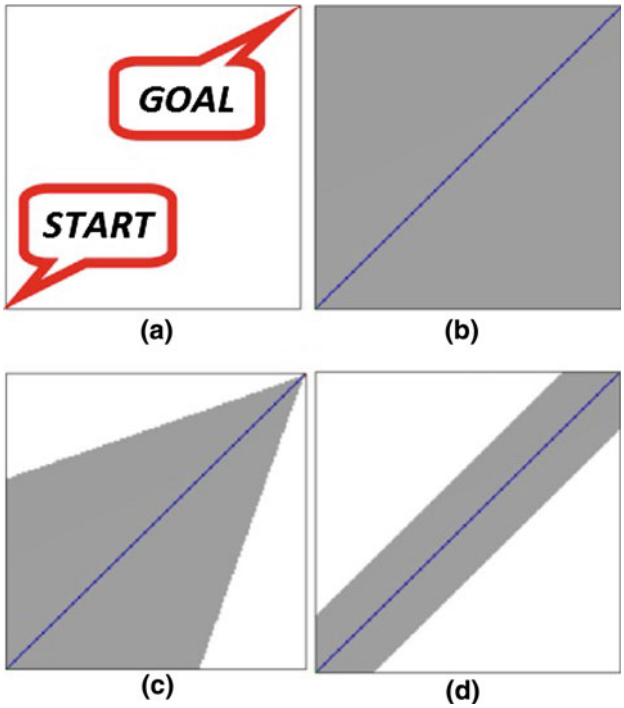

Fig. 11 **a** Pre-set *START*-*GOAL* conditions of environment. Path computation for **b** Dijkstra's algorithm, **c** A* algorithm, and **d** D++ algorithm (*DETECTIVE RANGE* setting of ten cells and cycle time of 20 ms)

cost only 20 milli-seconds before the robot could begin moving towards the *GOAL*. In addition, the D++ algorithm visited very few cells in the environment. The difference among the performance of these algorithms will become more significant for larger environment sizes.

(2) *Comparison with the D* algorithm:* Fig. 12 shows the results for the two situations that have been discussed by Stentz [6]. Figure 13 shows the result of using the D++ algorithm for these situations. It can be observed that the D++ algorithm located a lesser number of cells for the same environment than D*, and still computed a similar path. Therefore, the D++ algorithm has a faster response time than the D* algorithm. This indicates that the D++ algorithm is more practicable than the D* algorithm in dealing with rapidly changing real-time environments, such as the case in Fig. 10.

(3) *Comparison with the artificial potential fields:* In the case of Fig. 14 is a maze-like environment. The robot was able to easily determine a path to the *GOAL* by using the D++ algorithm without being trapped at any local areas (see Fig. 14a). In this case, the approach of APF will be hardly to obtain a path without any assistance of other methods like wall-following method, and then easily gets stuck at a local area (see Fig. 14b). Therefore D++ algorithm has an easy way and better ability than conventional local solution such as APF to find an effective path for complex environments.

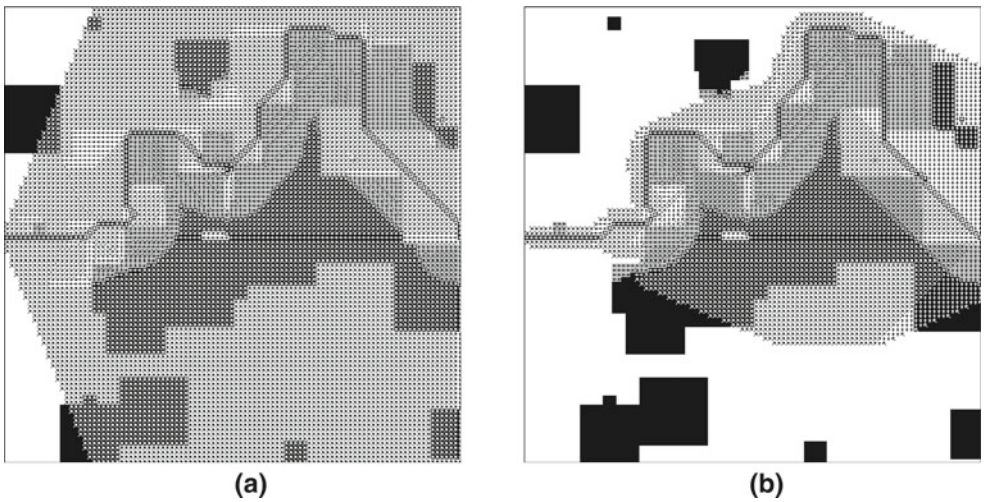

Fig. 12 Results from Stentz's study [6]: **a** basic D* algorithm and **b** focused D* algorithm

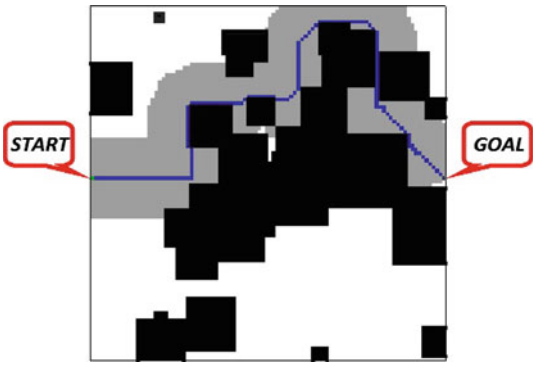

Fig. 13 Result obtained using D++ algorithm (*DETECTIVE RANGE* setting of ten cells and cycle time of 20 ms)

Therefore, from the comparisons described above, we can generalize the advantages and shortcomings of Dijkstra's algorithm, A* algorithm, D* algorithm, APF and D++ algorithm in Table 2. From Table 2, we can see that D++ algorithm has better average performance than others in term of solving speed, adaptability of varied environments, and the quality of solution. Thus D++ algorithm is more suitable than those other approaches for the mobile robots which need to deal with many kinds of environments in real world.

2.5 Optimization of *DETECTIVE RANGE*

From the cases of Figs. 7 and 8, it is assumed that the size of the *DETECTIVE RANGE* considerably affects the performance of the D++ algorithm. However, the question of the optimal *DETECTIVE RANGE* size must be addressed. The path computed by using the D++ algorithm may not be optimal because the D++ algorithm only manages the local environment in one cycle time. Therefore, the *DETECTIVE RANGE* should be set for as wide a value as possible, such that the robot can complete one local search within one cycle time. For example, for the environment in Fig. 11d, the

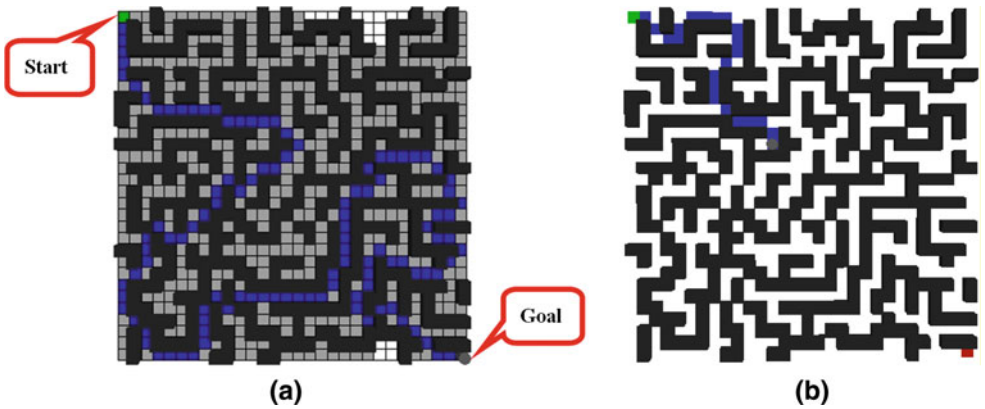

Fig. 14 Results of maze exploration **a** D++ algorithm with *DETECTIVE RANGE* setting of 20 cells, and total elapsed time is approximately 4 s, **b** artificial potential fields

Table 2 Comparisons of some mainly path-planning approaches

	Solving speed	Dynamic environment	Unknown environment	Avoidance of local solution	Quality of solution
Dijkstra	△	△	△	◎	◎
A*	○	△	△	◎	◎
D*	○	○	△	◎	○
APF	◎	◎	◎	△	△
D++	◎	◎	◎	◎	○

◎ : Good ○ : Normal △ : Poor

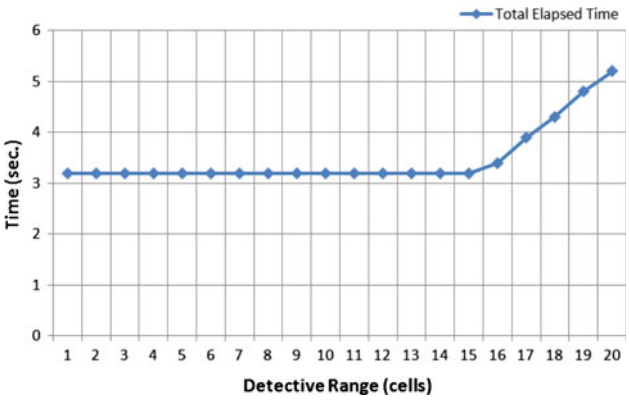

Fig. 15 Total time required for reaching the goal for the environment as specified in Fig. 11 for various *DETECTIVE RANGE* settings

DETECTIVE RANGE was varied from 1 to 20 cells, and the respective costs (times) from the start to the goal were measured and plotted as a chart (see Fig. 15). The chart shows that the response time increased for *DETECTIVE RANGE* of more than 15 cells because the robot efficiently manages computational loads only for *DETECTIVE RANGE* of less than 15 cells. Therefore, for the computer used in this study (Table 1), the *DETECTIVE RANGE* of 15 cells for a cycle time of 20 ms was optimal.

3 Experiment

To demonstrate the performance of the D++ algorithm, a self-designed mobile robot was built to navigate in unknown environments. The robot system included a host computer, a robot, a detection system, and a communication system (see Fig. 16). The specification of the host computer was the same as that in Table 1. The detection system was used to acquire the information of obstacles. The host computer applied the D++ algorithm to control the motion of the robot through the communication system by using the information of the detection system.

The size of robot is about 20 cm in length, 20 cm in width, and 30 cm in height. The robot comprised two driving wheels at the back and one swivel wheel at the front (see Fig. 17). Each driving wheel had a DC motor and a rotary encoder to drive the robot forward or to turn. The control system of the robot comprised a control board and a motor driver board. The control board was Arduino Mega 1280, which included a microprocessor of 16 MHz clock and a flash memory of 128 Kbytes. The Arduino Mega 1280 also comprises 52 digital I/O pins, 16 analog input pins, and 4 UART ports. The motor driver board has an IC of L289P, which can drive two DC motors forward and in reverse with a maximal current of 2 A. The rotative angle of the robot was detected by an

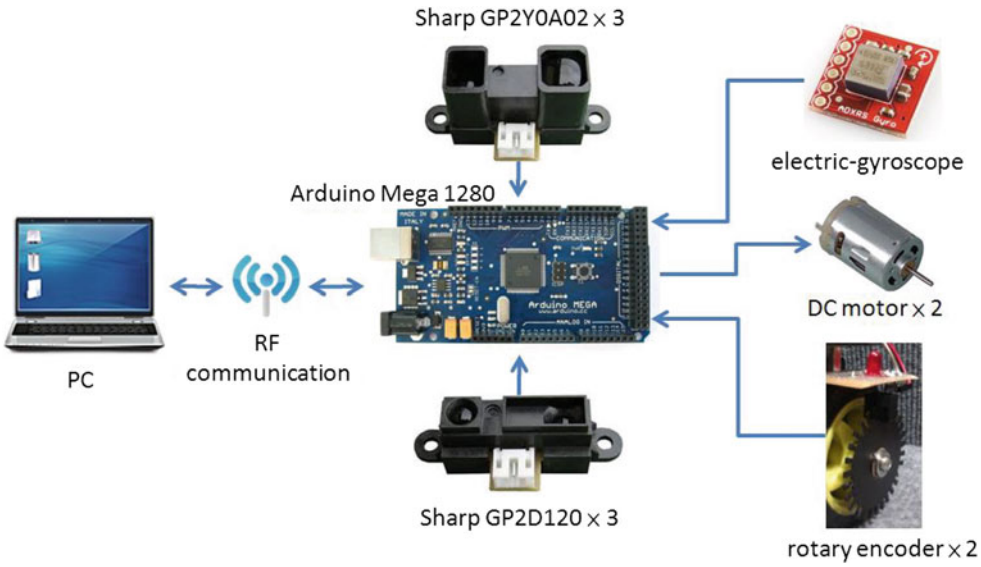

Fig. 16 The framework of the robot system used in this paper

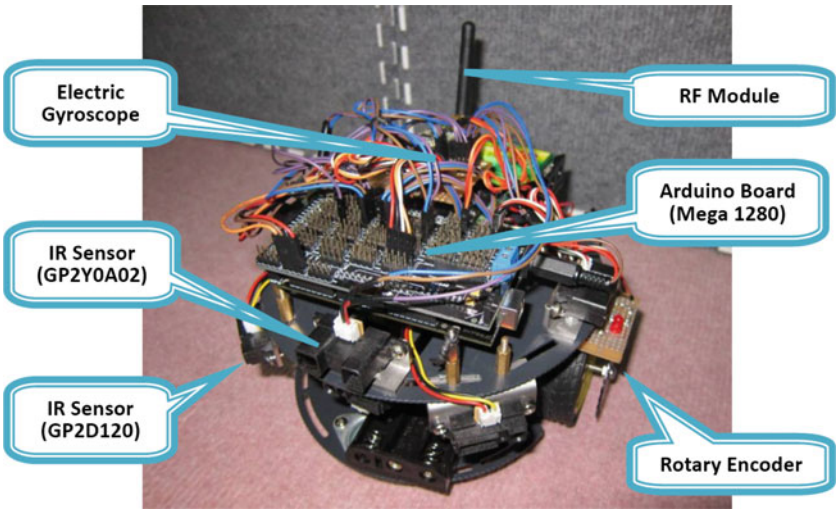

Fig. 17 The real robot that is discussed in this paper

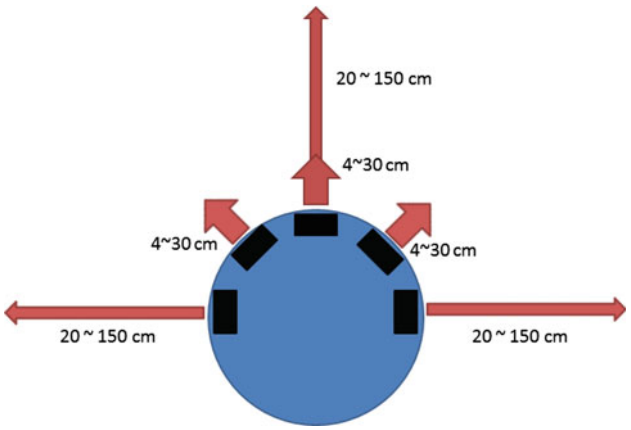

Fig. 18 The disposition and *DETECTIVE RANGE* of the IR sensors

electric-gyroscope. The detective system comprised of six infrared-ray (IR) sensors that were installed on the robot (see Fig. 18). Three sensors were Sharp GP2Y0A02 with the detective distance from 20 to 150 cm, and were used to detect far away obstacles. The other sensors were Sharp GP2D120 with the detective distance from 4 to 30 cm, and were used to detect proximity obstacles. The sensors that detected proximity obstacles not only provided the information of obstacles to the host computer, but also avoided collision, which is the cause of some errors of motion control such as miscounts of encoders, inaccuracy of electric-gyroscope, skid of wheels, etc. The communication system was a pair of radio-frequency (RF) modules. They operated at a baud rate of 9,600 bits/s, and at a work frequency of 434 MHz. They were installed on the robot and the host computer.

Figure 19 shows the workflow of the navigation control of a real robot used in these experiments. First, users must determine the size of the *DETECTIVE RANGE* and the position of the *GOAL*. The robot then sets its current position as *START*. The robot then uses IR sensors to detect obstacles and determine the next move by using the D++ algorithm. If

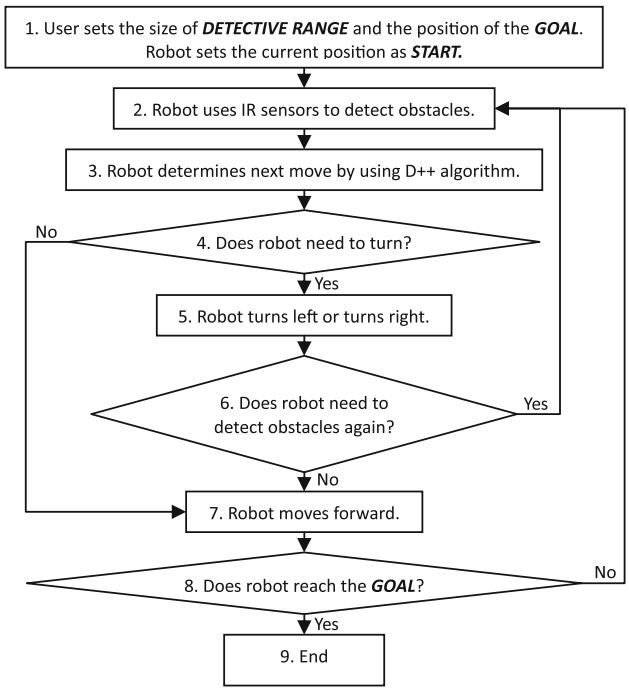

Fig. 19 Workflow of navigation control of real robot

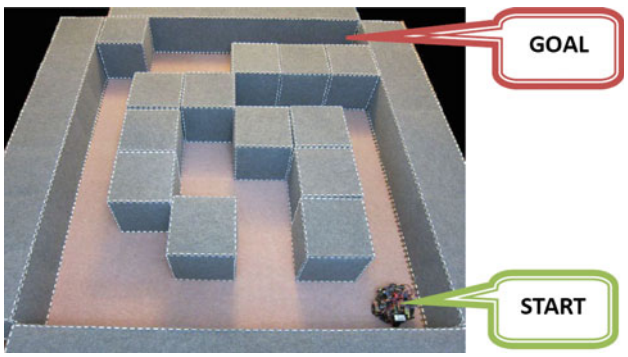

Fig. 20 A maze-type and static environment with the dimension of 7 cells × 6 cells

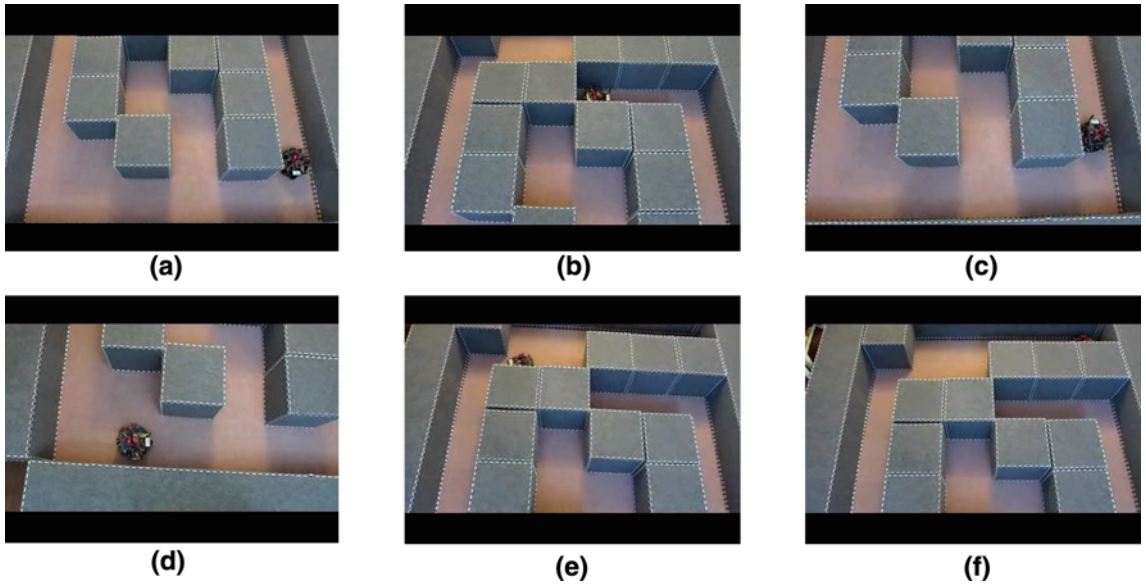

Fig. 21 The actual situation of navigation of the robot (*DETECTIVE RANGE* is three cells)

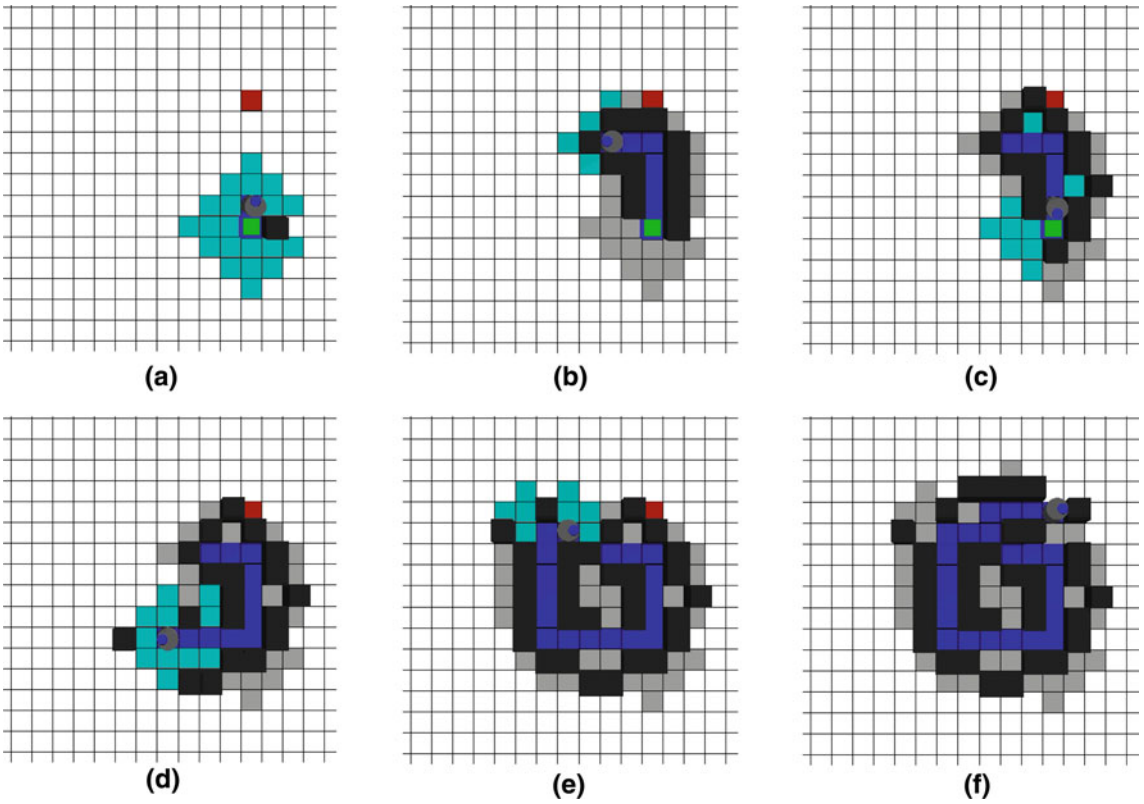

Fig. 22 The monitor of navigation of the robot (*DETECTIVE RANGE* is three cells)

the robot must turn, then the electric gyroscope guides the robot to turn left or right at an angle of 90°. If the robot must move forward, then the rotary encoders and the electric gyroscope guide the robot to move straight for the distance of one cell. When the robot reaches the *GOAL*, the navigation ends. Otherwise, the process continually returns to Step 2.

3.1 Static environment

Figure 20 shows that the robot navigated a maze-shaped environment, the information of which was not determined in advance. Figure 21 presents the actual process of robot navigation, and Fig. 22 presents the monitor of the host

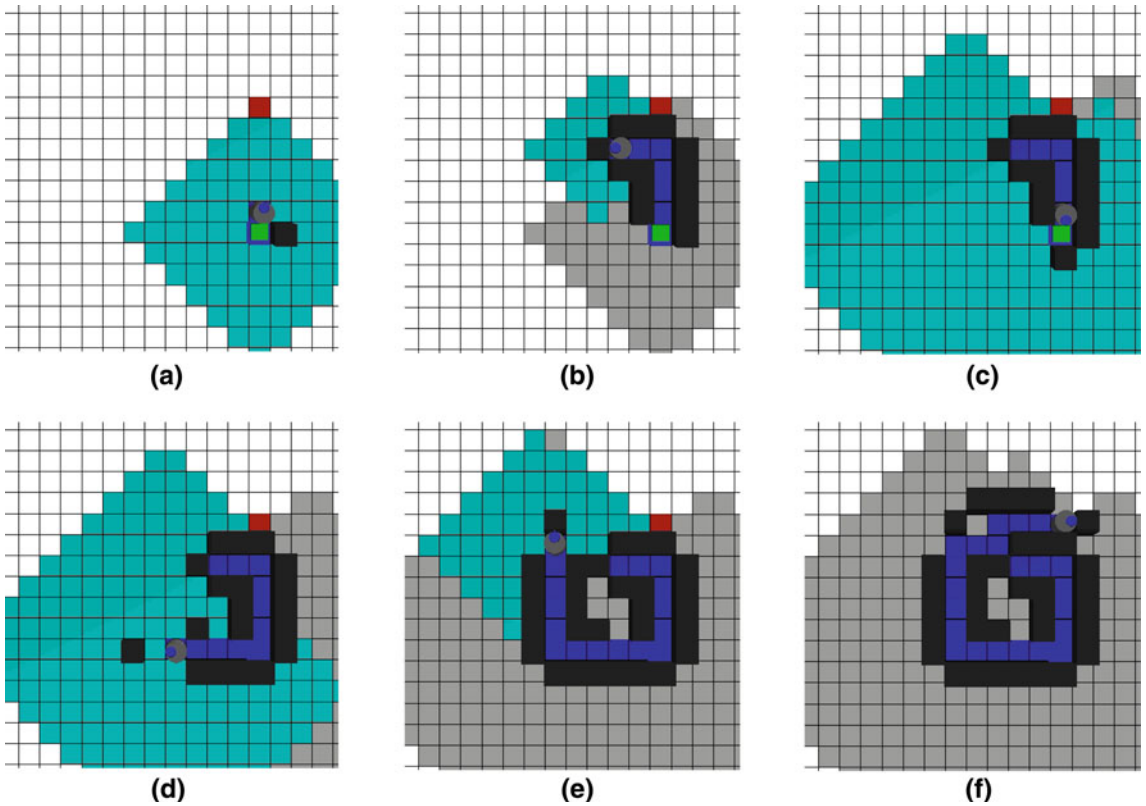

Fig. 23 The monitor of navigation of the robot (*DETECTIVE RANGE* is 20 cells)

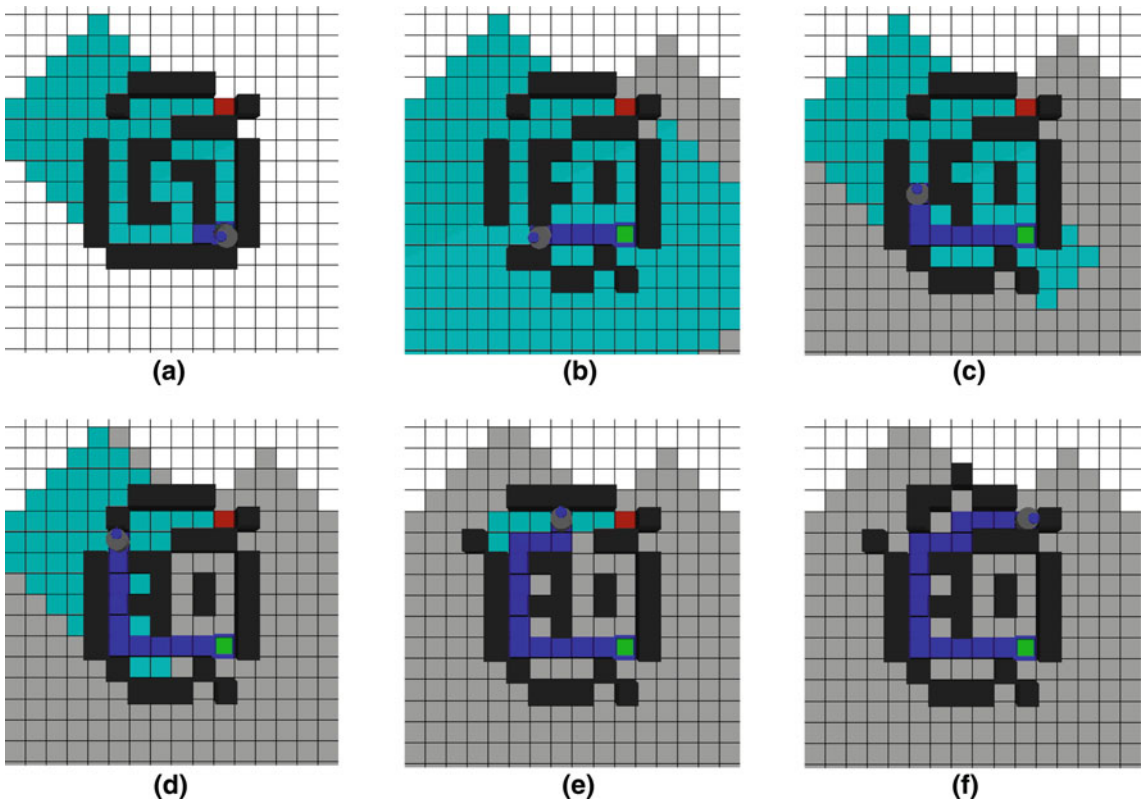

Fig. 24 The monitor of navigation of the robot (*DETECTIVE RANGE* is 20 cells and information of environment is known in advance)

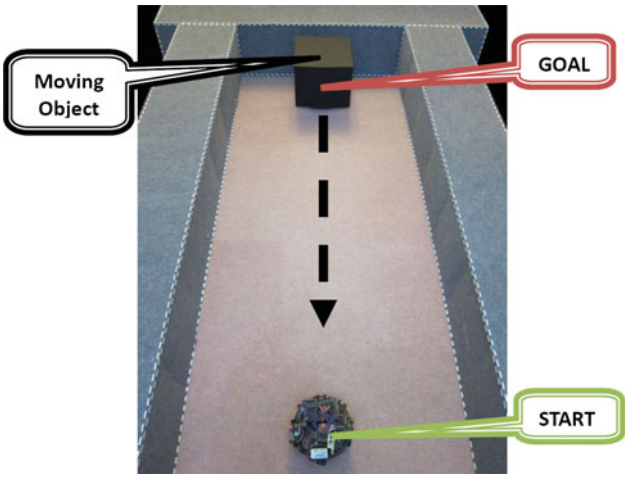

Fig. 25 A dynamic environment

computer when the robot was navigating. In this case, the *DETECTIVE RANGE* was set for three cells. The dimensions of each cell in the experiments of this chapter are 30 cm × 30 cm. The blue dot on the grey robot in Fig. 22 indicates the orientation of the robot. This example is useful for visualizing the function of the D++ algorithm in a real robot system and shows the ability of the D++ algorithm to escape from local solutions or to avoid them.

Figure 23 shows the monitor of another case in which the *DETECTIVE RANGE* was set for 20 cells in the same environment as that in Fig. 20. Figures 22 and 23 show that few differences exist between the results of the *DETECTIVE RANGES* of 3 and 20 because the limit of the sensors was that they could not detect the obstacles as wide as the *DETECTIVE RANGE* of 20 cells. However, if we use the information of the environment, which was detected in the case of Fig. 23, the robot would directly determine the shortest path to its destination (Fig. 24).

3.2 Dynamic environment

Figures 25, 26 and 27 show that the robot navigated in a dynamic environment in which an object moved forward at the middle of map. The *DETECTIVE RANGE* was set for five cells. This case shows that the D++ algorithm has a superior ability to regular global search approaches, such as Dijkstra's algorithm and A*, in rapidly responding in dynamic environments.

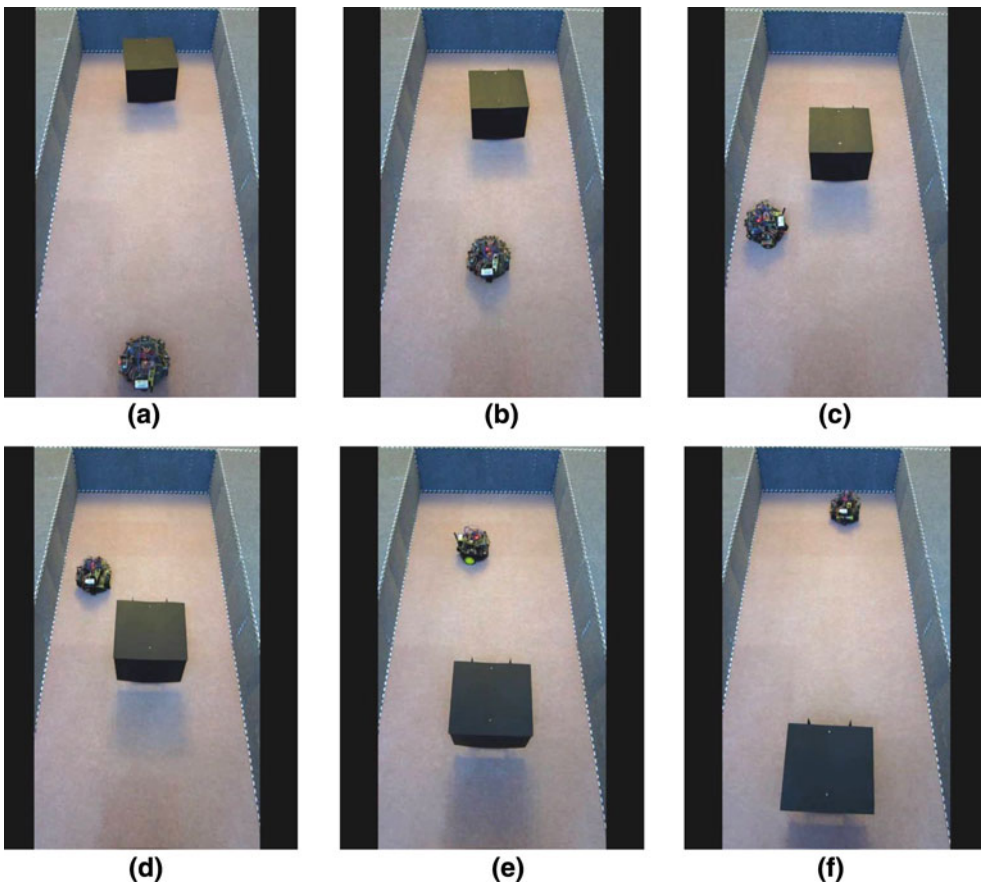

Fig. 26 The actual situation of navigation of the robot in a dynamic environment (*DETECTIVE RANGE* is five cells)

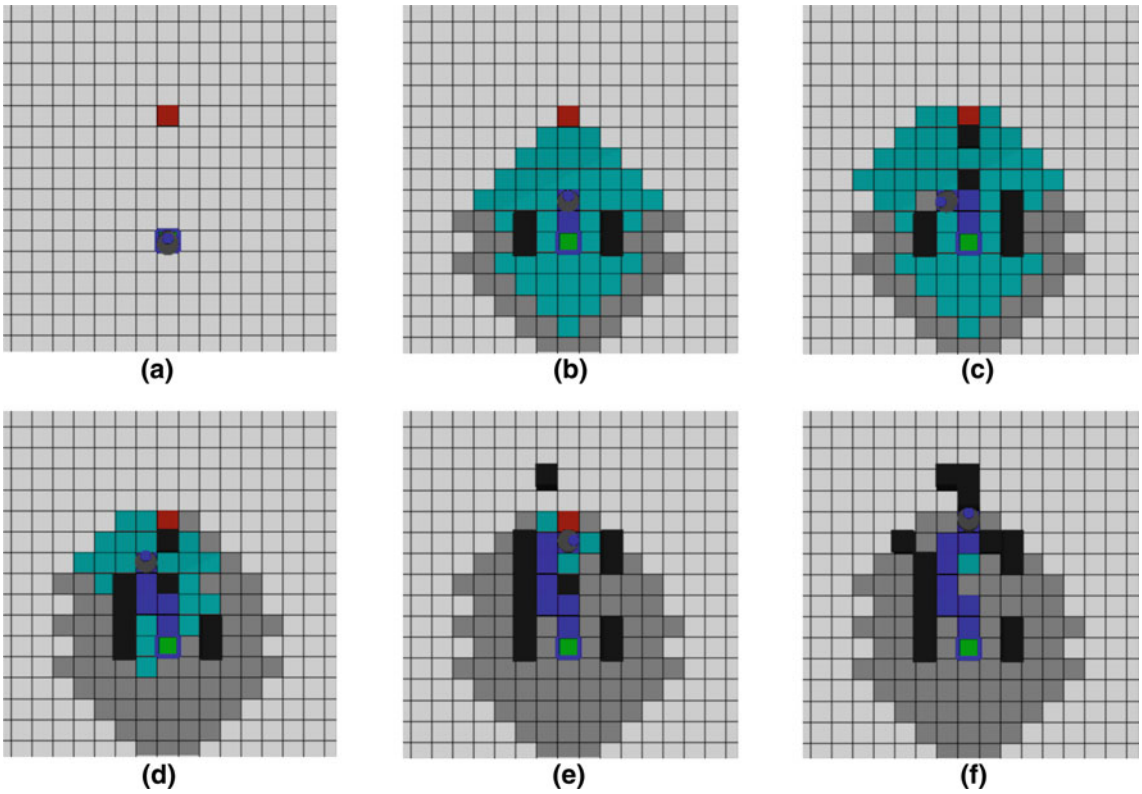

Fig. 27 The monitor of navigation of the robot in a dynamic environment (*DETECTIVE RANGE* is five cells)

4 Conclusions

This study applied the D++ algorithm, which is an optimal and efficient path-planning algorithm that combines the advantages of global and local searching approaches and applies them to a sensor-equipped robot. The concept of a *DETECTIVE RANGE* was added to the Dijkstra's algorithm. The *DETECTIVE RANGE* reduced the size of the searching space of Dijkstra's algorithm; therefore, the D++ algorithm obtained waypoints rapidly to enable the robot to move immediately. This characteristic also enabled the robot to manage the changing and dynamic environment. Moreover, the D++ algorithm maintains the property of a global search because it is based on the theory of Dijkstra's algorithm. Thus, the D++ algorithm has an optimal ability and is simpler than other local searching approaches in determining a pathway when a robot is stuck in a local area.

We also applied the D++ algorithm to a small robot in a laboratory environment. Because of the limitations of IR sensors, the performance of the D++ algorithm applied to the actual robot system was not as favorable as the results of the simulations (Figs. 7, 8), despite the *DETECTIVE RANGE* being considerably wide. Thus, the performance of the sensors, which detect the obstacles of environments, had a considerable effect on the results of the D++ algorithm for a real robot system.

We expect that the D++ algorithm is a general algorithm of path-planning to manage the problems of unknown, large, complex, and dynamic environments. However, using the algorithm requires additional experiments for real-world environments to verify the performance of the D++ algorithm. Therefore, future studies should develop and establish a more effectively functioning robot system by installing a sturdier structure, a high-speed control board, high-accuracy encoders, a laser rangefinder, or a GPS. The D++ algorithm may be applied successfully to exploratory robots in the actual field in the future.

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

References

1. LaValle SM (2006) Planning algorithms. Cambridge University Press, Cambridge
2. Russell S, Norvig P (2009) Artificial intelligence: a modern approach, 3rd edn. Prentice Hall, Englewood Cliffs
3. Dijkstra EW (1959) A note on two problems in connexion with graphs. Numer Math 1:269–271
4. Hart PE, Nilsson NJ, Raphael B (1972) Correction to a formal basis for the heuristic determination of minimum cost paths. SIGART Newslett 37:28–29

5. Stentz A (1994) Optimal and efficient path-planning for partially-known environments. In: Proceedings of the international conference on robotics and automation, pp 3310–3317
6. Stentz A (1995) The focused D* algorithm for real-time replanning. In: Proceedings of the international joint conference on artificial intelligence, pp 1652–1659
7. Koenig S, Likhachev M, Furcy D (2004) Lifelong planning A*. Artif Intell J 155(1–2):93–146
8. Koenig S, Likhachev M (2002) D* Lite. In: Proceedings of the eighteenth national conference on artificial intelligence, pp 476–483
9. Khatib O (1985) Real-time obstacle avoidance for manipulators and mobile robots. In: IEEE international conference of robotics and automation, vol 2, pp 500–505
10. Koren Y, Borenstein J (1991) Potential field methods and their inherent limitations for mobile robot navigation. In: IEEE international conference of robotics and automation, vol 2, pp 1398–1404
11. Charifa S, Bikdash M (2009) Comparison of geometrical, kinematic, and dynamic performance of several potential field methods. In: IEEE Southeastcon, pp 18–23
12. Meng M, Kak AC (1993) Mobile robot navigation using neural networks and nonmetrical environmental models. Control Syst Mag 13(5):30–39
13. Yang S, Meng M (2001) Neural network approaches to dynamic collision-free trajectory generation. IEEE Trans Syst Man Cybern B Cybern 31(3):302–318
14. Pan J, Pack DJ, Kosaka A, Kak AC (1995) FUZZY-NAV: a vision-based robot navigation architecture using fuzzy inference for uncertainty-reasoning. In: Proceedings of the world congress on, neural networks, vol 2, pp 602–607
15. Lee TL, Wu CJ (2003) Fuzzy motion planning of mobile robots in unknown environments. J Intell Robot Syst 37:177–191
16. Senthilkumar KS, Bharadwaj KK (2009) Hybrid genetic-fuzzy approach to autonomous mobile robot. In: IEEE international conference of technologies for practical robot applications (TePRA), pp 29–34
17. Cheng, PY, Chen PJ (2010) The D++ algorithm: real-time and collision-free path-planning for mobile robot. The 2010 IEEE/RSJ international conference on intelligent robots and systems, pp 3611–3616.
18. Kim M, Chong NY, Yu W (2008) Fusion of direction sensing rfid and sonar for mobile robot docking. In: IEEE international conference of automation science and engineering, pp 709–714
19. Pearl J (1984) Heuristics: intelligent search strategies for computer problem solving. Addison-Wesley, Boston