Knowledge-Based Systems 27 (2012) 302–313

Contents lists available at SciVerse ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

A new hybrid navigation algorithm for mobile robots in environments with incomplete knowledge

Yi Zhu*, Tao Zhang, Jingyan Song, Xiaqin Li

Department of Automation, Tsinghua University, Beijing 100084, China
Division of Control Science and Engineering, Tsinghua National Laboratory for Information Science and Technology, Beijing 100084, China
National Key Laboratory of Flight Vehicle Control Integrated Technology, Xi'an, China

ARTICLE INFO

Article history:
Received 26 March 2011
Received in revised form 26 October 2011
Accepted 7 November 2011
Available online 12 November 2011

Keywords:
Mobile robots
Hybrid navigation
Incomplete knowledge
System architecture
Bug algorithms

ABSTRACT

Focusing on the navigation problem of mobile robots in environments with incomplete knowledge, a new hybrid navigation algorithm is proposed. The novel system architecture in the proposed algorithm is the main contribution of this paper. Unlike most existing hybrid navigation systems whose deliberative layers usually play the dominant role while the reactive layers are only simple executors, a more independent reactive layer that can guarantee convergence without the assistance of a deliberative layer is pursued in the proposed architecture, which brings two benefits. First, the burden of the deliberative layer is released, which is beneficial to guaranteeing real-time property and decreasing resource requirement. Second, some possible layer conflicts in the traditional architecture can be resolved, which improves the system stability. The convergence of the new algorithm has been proved. The simulation results show that compared with three traditional algorithms based on different architectures, the new hybrid navigation algorithm proposed in this paper performs more reliable in terms of escaping from traps, resolving conflicts between layers and decreasing the computational time for avoiding time out of the control cycle. The experiments on a real robot further verify the validity and applicability of the new algorithm.

1. Introduction

As one of the key techniques in the applications of mobile robots, autonomous navigation endows a robot with the capability of reaching expected positions without collisions for executing various tasks, e.g., transportation, surveillance, or exploration. Since the motion performance can strongly affect the task performance, a reliable navigation algorithm is very necessary.

From the perspective of the possible application scenarios, the existing navigation algorithms can be categorized into two types that are respectively applied in environments with complete or incomplete knowledge. Most early works [1] belong to the first type. In these works, a collision free path for reaching the goal is previously planned in a map constructed by the knowledge of the workspace. What the robot should do is just accurately executing the plan. This concept is usually termed the deliberative navigation. However, complete knowledge is unrealistic for many applications and therefore comparatively recent works mainly focus on the second situation, i.e., how to navigate the robot in incompletely known environments. There are two main solutions. One is the reactive navigation [2–9] whose concept is directly mapping the sensory data to the on-line control command of the robot in every control cycle. Such algorithms need no prior knowledge and have good reactivity. Nevertheless, they may be trapped in troublesome scenarios, e.g., U-shape obstacles, since greedy search strategies are usually adopted in these methods to select the next motion direction and the global view is a lack. Furthermore, they have not taken the possible prior knowledge of the workspace into account. Actually, instead of possessing either complete, or no knowledge, having partial knowledge in advance is the most common circumstance in real applications. To overcome these limitations, another solution [10–24] attempts to design systems consisting of both the deliberative and reactive layer, since intuitively the deliberative layer can utilize the existing knowledge to generate long-term plans for avoiding trap situations, and the reactive layer can provide a good reactivity for executing the plan. Such concept is termed the hybrid navigation and is focused on in this paper.

The concept of the hybrid navigation presents the possibility to acquire both the global view and the reactivity. However, a new problem comes out: how to cooperate the two layers with each other in one synthesized system. A common strategy [10–13] is that the deliberative layer periodically computes a path as a reference for the reactive layer to generate the final motion commands.

* Corresponding author at: Department of Automation, Tsinghua University, Beijing 100084, China.
E-mail address: zhu-y07@mails.tsinghua.edu.cn (Y. Zhu).

0950-7051/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.knosys.2011.11.009

The reference path can also be computed occasionally, e.g., only when a trap-situation caused by the reactive navigation occurs [14]. In some other works, the deliberative layer generates a free channel [15] or sub-goals [16,17] off line based on the prior knowledge, and the reactive layer works on line independently with executive freedom. Another concept [18–20] is that the initial path can be deformed to some extent by the deliberative layer during execution when local changes in the environment are detected by sensors. Furthermore, a recent work [23] allows the robot to finish the navigation task by exploring. The deliberative layer will make the exploration plan on-line and the reactive layer just executes the plan. Nevertheless, to our knowledge, most previous works have either or both of the two following limitations.

First, a dominant view in most previous works is that guaranteeing convergence (find a path to reach the goal if it exists) is the task of the deliberative layer (note that no algorithm is strictly convergent in a changing environment since the future is unpredictable [25], and therefore "convergence" mentioned in this paper refers to convergence in static environments), and the reactive layer is only a simple controller that has some degree of executive freedom but cannot deal with complex trap-situations without the assistance of the deliberative layer. As a result, the deliberative layer must work frequently to correctly guide the robot when the prior knowledge is incomplete, e.g., re-planning paths periodically or once a trap is encountered. Such operative mode requires large computational resources, which also brings unstable factors to the entire system. For example, the time for re-planning can easily exceed the control cycle when the workspace is large enough or there are high-accuracy requirements. If re-planning cannot be finished in time, the system may fail since the reactive layer cannot independently guarantee convergence. Furthermore, a time-consuming system will degrade the safety and efficiency of the system. The second limitation of many previous works is the possible navigation failure caused by layer conflicts in the system. Since each layer has its own world model and algorithm, they may generate different solutions under the same condition. For example, the deliberative layer can conclude that there is no solution when there are errors in the prior knowledge, but the reactive layer may detect a safe way during execution. Also, the deliberative layer may generate a plan that the reactive layer cannot execute. Such conflicts may cause navigation failure. Implementers usually handle this problem by "fine tuning" the system according to empirical results [13]. Notably, in a recent dissertation [13], a learning mechanism is proposed to resolve layer conflicts and improve system performance for the traditional hybrid system. In this mechanism, the deliberative layer will adapt its world model (mainly in terms of the cost model) to better predict the execution result of the reactive layer by learning from experience. Hence more feasible plan can be generated for the reactive layer after learning. However, besides some remained problems about the learning mechanism itself as discussed in [13], a possible drawback is that a learning mechanism integrated in the deliberative layer of a traditional architecture may further increase the burden of the deliberative layer and the system's dependence on it, whose disadvantages have been pointed out above.

In this paper, focusing on the navigation problem of mobile robots in environments with incomplete knowledge, a new hybrid navigation algorithm is proposed. The main contribution is that we present a novel architecture in this algorithm for cooperating the two layers to overcome the two aforementioned limitations in many previous works. The new architecture differs from most previous works in its stress on a reliable and independent reactive layer rather than a powerful deliberative layer, which is demonstrated to be helpful for releasing the burden of the deliberative layer, guaranteeing the real-time property of the system and ensuring navigation success when there are layer conflicts. The remainder of this paper is organized as follows: the new algorithm is presented with the convergence proof in Section 2. In Section 3, simulations are carried out to compare the performance of our algorithm with related works, and the results of experiments on a real robot are also presented.

2. New hybrid navigation algorithm

Before the algorithm description, the environmental conditions should be formulated at first. The environment discussed in this paper is a two-dimension plane which is populated by finite static obstacles with arbitrary shape and it may also contain several autonomous moving obstacles which are discs with finite radius (a safe distance can be set for other shaped obstacles to imitate discs). There is some prior knowledge which contains the positions of some but not all the static obstacles, and the knowledge may also be inaccurate. The trajectories of moving obstacles are previously unknown. However, through specific sensors, e.g., range sensors and cameras, the robot can on-line detect the distance, velocity and radius of any obstacle in any direction within a detectable distance as long as the view has not been obstructed by other obstacles.

Fig. 1 describes the architecture of the new hybrid navigation algorithm proposed in this paper. It works as follows: the deliberative layer previously generates a reference path for reaching a goal from a start point based on the prior knowledge; the reactive layer then executes the plan with the freedom of revising it for dealing with the uncertainty in the prior knowledge. The key trait of our architecture is that the deliberative layer has not taken the key role of making decision to guarantee convergence. On the contrary, we pursue a more reliable and independent reactive layer that can guarantee convergence by itself. The deliberative layer only uses the existing knowledge to provide a long-term plan for optimizing the robot's motion. This plan is not necessary (therefore it is labeled as a dash arrow in Fig. 1) and can be changed during execution. Based on this architecture, the validity and robustness of the new algorithm no longer depend on the deliberative layer. The deliberative layer has not to frequently re-compute the reference path and a rough plan is enough, which significantly releases the computational burden of the system and is beneficial to ensuring high reactivity and decreasing resource requirement. Hence the first limitation mentioned before has been overcome. As to the second limitation, in our system, the result generated by the reactive layer will be adopted as the final command when conflicts occur between the two layers. We do not use the result of the deliberative layer partly due to its long computational time, but mainly because practical trials during execution are more reliable than plans based on incomplete knowledge that may contain errors and unpredictable changes. The correctness of such rule depends on the advantage that our reactive layer can guarantee convergence

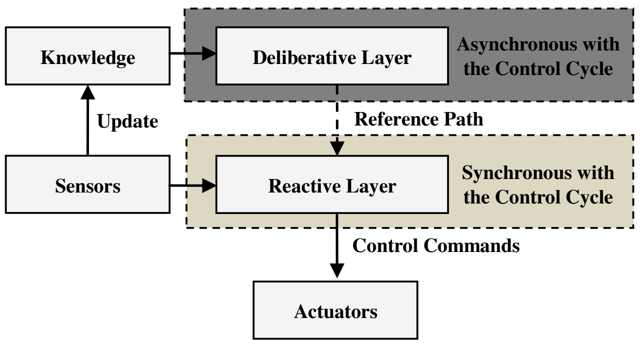

Fig. 1. Architecture of the new hybrid navigation algorithm.

without the assistance of the deliberate layer. Next we will respectively describe the two layers and how to integrate them in the new algorithm.

2.1. Deliberative layer

The task of the deliberative layer is to generate a reference path which optimizes the robot's motion for reaching a goal from a start point in a global view based on the prior knowledge. A binary grid map as shown in Fig. 2 is used to represent the workspace. In the map, the state of each grid is either free or occupied (if it is unknown, it will be processed as free). Each grid has eight neighboring grids but whether they are connectable to constitute a path depends on their comparative positions and states. As shown in Fig. 3, any neighboring gird in the horizontal or vertical direction is connectable as long as it is free, while any neighboring grid in the diagonal direction is connectable if and only if it is free and at least one of its two neighboring grids is free. The map is initially built by the prior knowledge and the reference path is calculated by searching the map. During the motion of the robot, the map will be updated on line by sensory data for the navigation tasks in the future. The grid map has been widely used in practical applications due to its ease of implementation, but it has the limitation that it requires enough resolution (the grid size should be small enough) to guarantee that a path for reaching the goal can always be found if one exists [26]. On the other hand, large resolution will occupy large memory space and it increases the time for searching the path. However, the resolution will only affect the optimization but not the convergence of our algorithm since the latter is guaranteed by the reactive layer instead of the deliberative layer. Hence we can use comparatively smaller resolution (0.5 m in our experiments) than many related works (0.05 m in [10–12]) to decrease the computational and spatial expense.

To search the reference path in the map, we use the classical A* algorithm [1]. The optimization object is to minimize the following function:

$$\alpha_1 \sum_i \|c_i - c_{i-1}\| + \alpha_2 N + \alpha_3 \sum_i n_i \tag{1}$$

where $\|c_i - c_{i-1}\|$ represents the Euclidean distance between the ith and $i-1$th grid center in the path, N is number of turning points in the path, n_i is the number of all the occupied grids whose center is within a safe distance (0.5 m in our experiments) of the ith grid center in the path, α_1, α_2 and α_3 are the constants ($\alpha_1 = \alpha_2 = \alpha_3 = 1$ in our experiments). The object described by (1) considers the factor

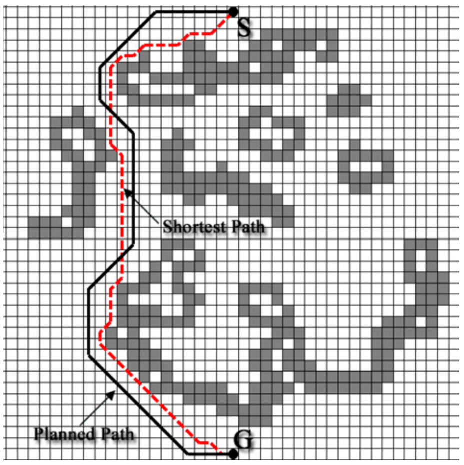

Fig. 2. An example of the grid map and the planned path (S is the start point; G is the goal; grey grids are occupied; white grids are free or unknown).

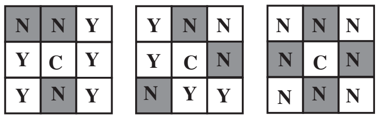

Fig. 3. Three examples for illustrating the connectivity of the neighboring grids (C is the current grid; Y is connected grid; N is disconnected grid; grey grids are occupied; white grids are free or unknown).

of path length, the number of turning points and the obstacle distance. Turning points increase the complexity of execution since the robot needs to slow down in advance for turning and therefore more time is required for reaching the goal. Keeping enough distance to obstacles guarantees safety and the robot can increase its speed to shorten motion time instead of moving cautiously. A comparison between the optimal path calculated by (1) and the shortest path is presented in Fig. 2. Note that other factors (e.g., the risk of special regions [27]) can also be considered according to the application requirements. To describe and store the reference path, we only use the turning points of a path and the final goal rather than all the grids in the path. They are stored in sequence and such a description obviously saves large space. Additionally, if the goal is not exactly located in a grid center, its real position will be stored instead of the corresponding gird center, which leads to a small error but can be ignored.

It should be noted that in practical applications, the goal is usually generated on line by a task layer or remote client and there may be multiple goals. Hence the deliberative layer should on-line compute a reference path for each goal. In this paper, we only consider the situation that there is a precedence constraint among all the goals (the goal generated earlier should be reached earlier), and therefore the last goal will be the start point of the next goal. The reference paths generated for each goal will be sequentially stored in a list based on the precedence. If there is no precedence constraint, the sequence of all the goals should also be considered and all the reference paths of previous goals should be recomputed for optimization once a new goal is generated.

2.2. Reactive layer

As described in Fig. 1, we require an independent reactive layer which possesses both the property of reactivity and convergence. Nevertheless, as pointed out before, most existing reactive navigation methods may get stuck in traps without the assistance of a deliberative layer, which is the well-known local minima problem. While many efforts [28–30] have been made to enhance the robustness of the reactive navigation in the face of trap-situations, to our knowledge, only a family of algorithms termed the Bug algorithms [31] have strictly proved their convergence and therefore thoroughly solve the local minima problem in theory. The Bug algorithms navigate the robot by two reactive motion modes: moving toward the goal if no obstacle is encountered, otherwise following the obstacle boundary until a leaving condition is satisfied. Based on a proper leaving condition, the Bug algorithms can ensure convergence by decreasing the goal distance step by step. As an example, in the classical Bug2 algorithm shown in Fig. 4 (S is the start point; G is the goal; Hi is the "hit point" at which the robot ends the motion toward the goal for the ith time and Li is the "leave point" at which the robot ends boundary following for the ith time), a leave point is defined as a point L that belongs to line section SG but line section LG does not cross the followed boundary, and meanwhile L must have a minor goal distance than the initial point of the current boundary following motion. This definition is proved to ensure convergence [32].

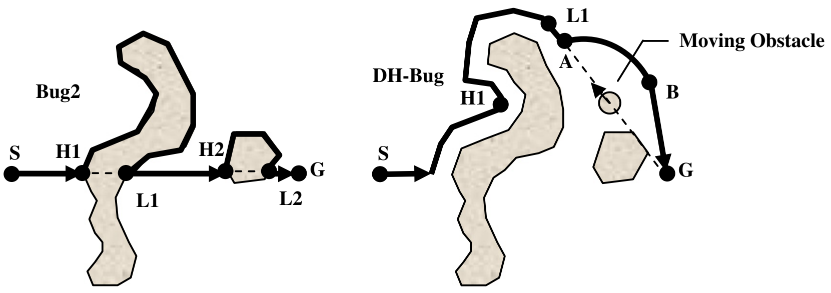

Fig. 4. An example comparing DH-Bug with the classical Bug2 algorithm.

However, previous Bug algorithms also have some limitations. Their generated path can be much longer than the optimal path in complex environments. Furthermore, almost all the previous Bug algorithms have not considered how to avoid moving obstacles. Additionally, previous works are usually designed in an abstract view that ignores implementation issues. To overcome these limitations, we have proposed a new Bug-type algorithm termed DH-Bug (Distance Histogram Bug) [33] and it is used for our reactive layer in this paper. To shorten the path, DH-Bug permits the robot to move in any direction whose bias to the goal direction is less than 90° instead of moving directly toward the goal in the motion toward the goal. The revision delays the motion of boundary following which is the main reason for generating a long path. Additionally, a new leaving condition (the goal and the followed obstacle are respectively located on the opposite side of the robot, and the current goal distance is less than the goal distance at the initial point of the current boundary following motion) is designed for ending the motion of boundary following earlier. For dealing with moving obstacles, a new motion mode and the related switching criteria are designed. Furthermore, DH-Bug presents not only an abstract concept, but also a control method for realizing this concept on real robots. As shown in Fig. 4, compared with the classical Bug2 algorithm, the occurrence of H1 is delayed but L1 occurs earlier in DH-Bug. Intuitively, it can shorten the path of boundary following and this conclusion has been verified by plenty of simulations. Additionally, after detecting that a moving obstacle may hit the robot if the robot keeps its current direction (at point A), DH-Bug will switch to the mode of avoiding moving obstacles. It will then resume the motion toward the goal when the situation is safe (at point B). DH-Bug is proved to be convergent in static environments. It can also adapt the speed and direction of the robot to smoothly avoid moving obstacles and finally reach the goal if some conditions are satisfied (these conditions can be easily satisfied in usual circumstances [33]).

Note that other Bug algorithms can also be used in the reactive layer, but their applications are limited in static environments. Similarly, any other reactive algorithm that guarantees convergence is also applicable to our reactive layer.

2.3. Integration of two layers

For integrating the two layers, three main issues have to be taken into account. The first issue is their cooperative mode (synchronous or asynchronous). The second issue is how to utilize the reference path in the reactive layer and the third one is how to resolve the conflicts between the two layers. Next we will sequentially illustrate theses issues.

In our hybrid navigation algorithm, the deliberative and reactive layer layers are integrated in an asynchronous way. The two layers independently work. They only interact through the list of reference paths. The deliberative layer computes a reference path once there is a new goal and stores it in the tail of the list. The reactive layer will sequentially navigate the robot to reach the goals stored in the list based on their corresponding reference paths. Once a goal (not sub-goal) is reached, its corresponding reference path will be deleted from the list. If all the goals are reached and there is not a new one (or the deliberative layer has not computed a solution yet), the reactive layer will be idle (the robot stops) until the list is not void.

As to the second issue, a reference path is described by all its turning points and the goal as illustrated in Section 3.1 and these points will be used as sub-goals for the reactive layer. More sub-goals are not necessary since the path between any two turning points is already a straight line, and fewer sub-goals are not enough to trace the reference path. If a sub-goal is detected to be unreachable (it may be located in an obstacle due to the incomplete knowledge), it will be skipped. Additionally, we do not require the reactive layer to reach all the sub-goals one by one since it is not so wise when there are errors in the prior knowledge. Two such cases are shown in Fig. 5. In Fig. 5, there are four sub-goals for bypassing the obstacle described in the prior knowledge. However, in the first case, the obstacle actually does not exist (it

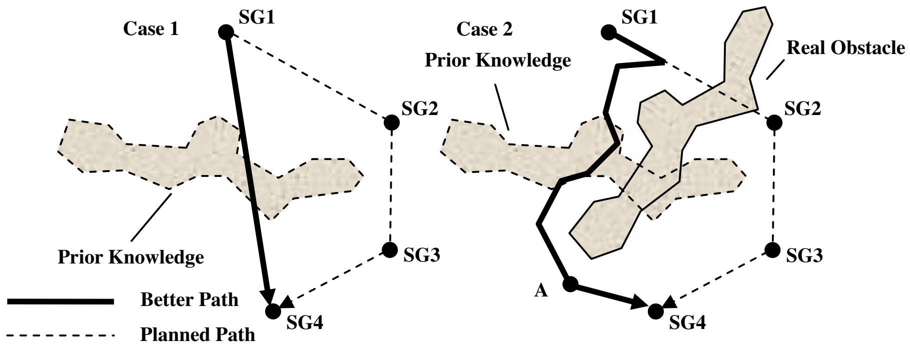

Fig. 5. Two cases that there are errors in the prior knowledge.

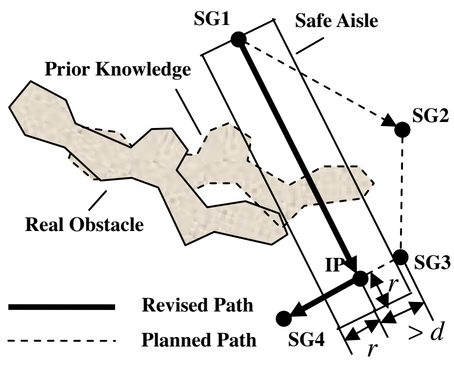

Fig. 6. Illustration of the conditions for skipping sub-goals in the reference path.

may be removed after the last update of the knowledge). Obviously, the path that directly links SG1 and SG4 is much shorter than the reference path. In the second case, there are errors in the position of the obstacle and therefore the robot cannot directly move to SG2 from SG1. Based on the DH-Bug algorithm in the reactive layer, it will try to bypass the obstacle by following its boundary. However, if the robot selects its following direction as shown in Fig. 5, it will find a safe path to SG4 at A before it reaches SG2. Therefore it seems much wiser to directly go to SG4 rather than reach the sub-goals one by one according to the plan. As illustrated above, for optimizing the path in the face of uncertainty, some sub-goals should be skipped if a "safe" path is detected to reach a sub-goal that ranks behind them. We define a path as "safe" when there is a collision free aisle whose width is $2r$ (the diameter of the robot) from the start point to the end point of the path (an example is shown in Fig. 6). The above skip condition can be further extended. As shown in Fig. 6, if there is a safe path (from SG1 to IP) to reach the line section between any two adjacent sub-goals (SG3 and SG4) and at least one (SG4) of them ranks behind some other sub-goals (SG2 and SG3), these sub-goals (SG2 and SG3) can be skipped and the intersection point (IP) of the detected safe path and the reference path should be added as a new sub-goal. However, frequently adding new sub-goals may lead to oscillations, and therefore we add another constraint which requires that the distance between the new sub-goal and the first sub-goal before it in the reference path should be larger than a threshold, or the new sub-goal is just a sub-goal already in the reference path. If this constraint is not satisfied, the new sub-goal will not be added and its previous sub-goals will not be skipped. The added constraint can be described as

$$\|NSG - SG_i\| = 0 \vee \|NSG - SG_{i-1}\| > d \tag{2}$$

where NSG is the new sub-goal, SG_i and SG_{i-1} are the first sub-goal after and before it in the reference path, "$\vee$" is logic or, d is the distance threshold (in our experiments, $d = 2$ m). When there is a new sub-goal, the robot will switch to the motion toward the goal if the mode of boundary following is active, or remain in the current mode if the other two modes is active. Additionally, not all the sub-goals but only finite ones (10 in our experiments) that rank between the current sub-goal will be checked for determining whether they should be skipped since checking all the sub-goals is usually unnecessary and it will increase the computational time.

The possible conflict between the two layers in our architecture is how to determine whether a goal is unreachable since the conclusions of the two layers may be different. If such a conflict occurs, the conclusion generated by the reactive layer will be adopted. We do not use the results of the deliberative layer for two main reasons. First, the deliberative layer needs to search the entire workspace to make such a conclusion. It is rather time-consuming and adverse for the reactivity. Therefore, we require the deliberative layer to generate a path directly linking the start point and the goal if no solution has been computed yet in a sustainable time. Then the reactive layer will guide the robot to reach the goal as in a completely unknown environment since such a reference path is the same as no reference path. The second and the most important reason is that the prior knowledge is incomplete and it may contain errors and unpredictable changes. Moreover, in order to decrease computational burden, the gird map used in our deliberative layer is rather rough as illustrated in Section 3.1 and therefore it is unreliable for making the conclusion. On the other side, the result of the reactive layer is trustable since it is based on the latest sensor data and our reactive layer can guarantee convergence by itself without the assistance of the deliberate layer.

2.4. Proof of convergence

Before the proof, the definition of convergence is firstly presented below.

Definition 1. If a navigation algorithm can guide the robot to reach any goal in the workspace after generating a path of finite length as long as there is a collision free path linking the start point and the goal (otherwise it will report fail), the algorithm is convergent.

Note that Definition 1 is only valid for static environments since a current collision free path for reaching the goal may no longer exist after a period in dynamic environments. It is hard to define convergence in dynamic environments. To our knowledge, there is no such a definition. As pointed out in [25], no algorithm can guarantee that it can always guide the robot to reach the goal when there are moving obstacles whose trajectories are previously unknown since the future is unpredictable. Therefore, in a dynamic environment, we can only attempt to design an algorithm that can robustly avoid moving obstacles and reaching the goal in most circumstances, and "convergence" only refers to static environments in this paper. Based on Definition 1, the following theorem is easy to be proved.

Theorem 1. *If the navigation algorithm applied in the reactive layer is convergent, the new hybrid navigation algorithm will also be convergent.*

Proof. As described in Section 2.3, the new algorithm divides the process of reaching a goal into a series of sub-processes $\{SP_i\}$ and each sub-process SP_i corresponds to a sub-goal SG_i. Due to the convergence of the reactive layer, after generating a path of finite length, there are only three cases for any SP_i. First, SG_i is reached. Second, SG_i is detected to be unreachable and it will be skipped. Third, a shorter path is detected and SG_i is skipped. Therefore, no matter which case occurs, SP_i will be ended after generating a path of finite length. On the other side, there are only finite turning points in a reference path. Additionally, there will be only finite new sub-goals since a distance threshold is required for generating a new sub-goal in the reference path according to (2) and the total length of the reference path is finite. As a conclusion, the number of SP_i is finite and any SP_i will be ended after generating a path of finite length. Since the last sub-goal (the real goal) will not be skipped, the hybrid navigation algorithm can always guide the robot to reach the real goal in the last sub-process as long as there is a collision free path, otherwise it will report failure. The length of the entire path is finite since there are finite sub-paths and their lengths are all finite. □

Theorem 1 indicates that the convergence of the new hybrid navigation algorithm depends on the convergence of the reactive layer. Obviously, the performance of the hybrid navigation algorithm when there are moving obstacles is also determined by the reactive layer since they have not been considered at all in our deliberative layer due to the assumption that their trajectories are previously unknown. The DH-Bug algorithm used in our reactive layer is convergent in static environments, and therefore the new algorithm is also convergent according to Theorem 1. As to dynamic environments, since DH-Bug can robustly avoid moving obstacles and reach the goal in usual circumstances, the new algorithm can also perform well.

3. Simulations and experiments

To test the performance of the new hybrid navigation algorithm, plenty of simulations have been carried out in MobileSim which is the special simulation software for Pioneer robots. The algorithm has also been implemented on a real Pioneer3-AT robot for verifying its practicability. Next the simulation and experimental results will be presented, respectively.

3.1. Simulation studies

Based on the model of a Pioneer3-AT robot in MobileSim, the new algorithm has been tested in two complex environments shown in Fig. 7 with incomplete knowledge that may also contain errors. The first environment (25×25 m) is populated by scattered obstacles and the second environment (45×30 m) is like a maze. More than one hundred valid pairs of start and goal points (a pair is valid if both the start point and the goal are not within obstacles and the goal is reachable) are randomly selected as navigation tasks in each environment. For each navigation task, there is some prior knowledge in the form of a grid map that is constructed based on the position data of obstacles. The knowledge is incomplete and there may be errors. An error is assumed to be a common position bias (it is random but limited by two meters) of all the points in an obstacle in any one axis or both of them. Each obstacle is randomly selected with a certain probability to be accurately known, unknown or inaccurately known. Each navigation task will be carried out twice with initial maps generated under different probabilities. The first one is 80% accurately known, 15% unknown and 5% inaccurately known. The second one is 30% accurately known, 35% unknown and 35% inaccurately known. Obviously, the incompleteness of the knowledge in the first case is much lower than the second one.

The simulation results show that the new hybrid navigation algorithm can robustly finish all the navigation tasks in both of the two environments and both the two cases, which verifies the convergence of the new algorithm. Two simulation results are presented in Fig. 7. It can be seen that the new algorithm does not only simply follow the reference path. It will adapt the motion of the robot to deal with the uncertainty in the prior knowledge. For example, due to the position errors in the prior knowledge, the sub-goal SG6 in the simulation shown in Fig. 7(a) and (b) is too close to the real obstacle and it is unreachable. At first, the robot attempts to follow the boundary of the corresponding obstacle to detect whether there is a collision free path to reach SG6 or it is unreachable. However, during the motion of boundary following, the robot detects a collision free path to reach the goal and therefore it skips SG6 and directly reaches the goal. Note that even there is no such a path, the robot can also skip SG6 after finishing a loop around the obstacle to conclude that SG6 is unreachable. In the simulation shown in Fig. 7(c) and (d), two unforeseen obstacles

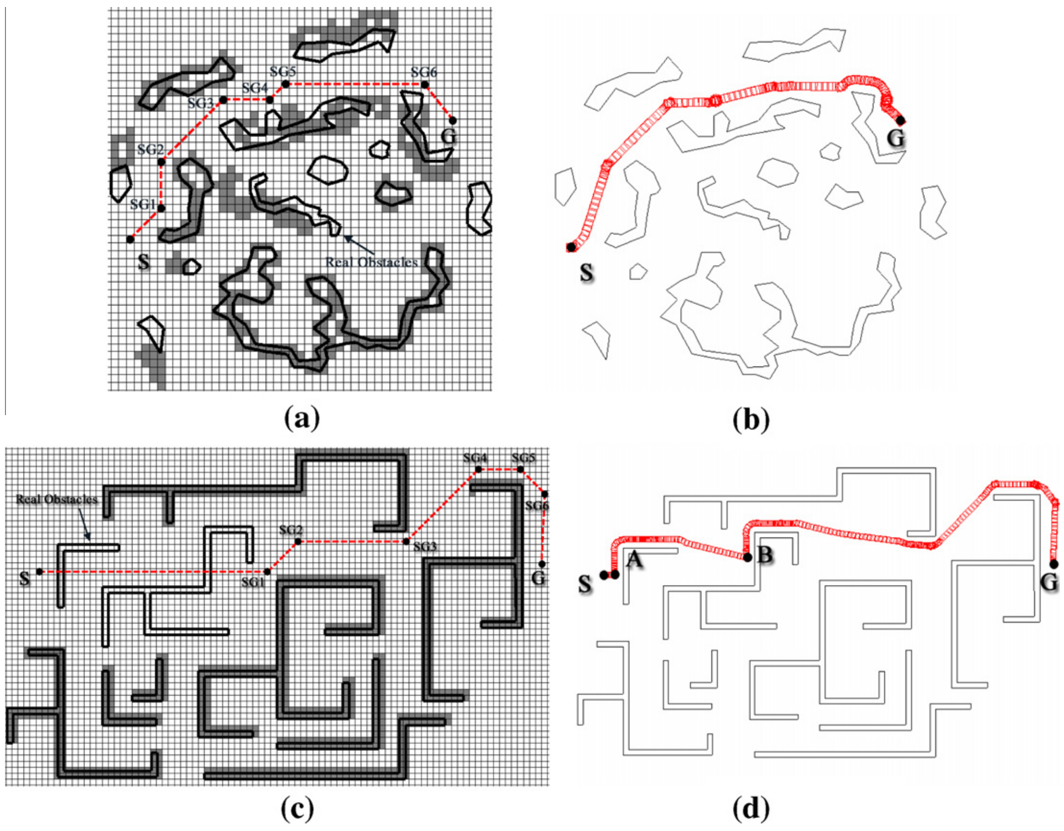

Fig. 7. Two simulation results of the new hybrid navigation algorithm (S is the start point; G is the goal; SG*i* is the *i*th sub-goal): (a) and (c) are the maps generated based on the prior knowledge (grey grids are occupied; white grids are free or unknown; the dash line is the reference path generated by the deliberative layer); (b) and (d) are the motion trajectories of the robot.

are encountered, respectively, at A and B, but the robot bypasses them by following their boundaries. Furthermore, it can be seen that instead of sequentially reaching sub-goals one by one, the robot will skip some of them (e.g., SG1, SG5 in Fig. 7(a) and SG1, SG2 in Fig. 7(c)) to shorten the path length as long as the skip condition described in Section 2.3 is satisfied.

For comparison, four other algorithms have also been respectively simulated in the same conditions. The first algorithm is the artificial potential field (APF) method [2]. As a classical reactive navigation algorithm, the APF method has attracted attentions for many years and a lot of its variations [28] have been proposed due to its simplicity and real-time property. In the APF method, each obstacle is assumed to generate a repulsive force and the goal generates an attractive force. The robot then moves in the direction of their composite force. The second algorithm is the DH-Bug algorithm that is applied in our reactive layer. It is compared with the new hybrid algorithm to verify whether there is an improvement after integrating it with a deliberative layer. The third algorithm is a hybrid navigation algorithm which integrates the APF method with a deliberative layer. The deliberative layer previously generates a series of sub-goals based on the prior knowledge. Then the robot moves to them one by one and avoids unforeseen obstacles under the control of the APF method. Such or similar concepts have been proposed in some previous works [15,16] to utilize both the reactivity of the APF method and the possible knowledge. In this paper, we use the same sub-goals in the reference path generated by our deliberative layer to guide the APF method and we will term it the first hybrid navigation algorithm (FHNA). The fourth algorithm is another hybrid navigation algorithm which also integrates the APF method with a deliberative layer. However, the deliberative layer will re-compute the series of sub-goals based on the current knowledge in every control cycle. As mentioned in the Section 1, such architecture is widely applied in previous works [10–12]. Therefore we use it to compare with the new algorithm proposed in this paper and we will term it the second hybrid navigation algorithm (SHNA). The successful ratios (the times that the robot finishes its navigation task divided by the times of simulations) of the five algorithms, i.e., APF, DH-Bug, FHNA, SHNA and the new hybrid navigation algorithm (NHNA) under two probabilities are respectively presented in Tables 1 and 2. Figs. 8 and 9 show the corresponding results of APF, DH-Bug, FHNA and SHNA under the same conditions as the simulations in Fig. 7.

It can be seen that APF performs worse than the other four algorithms in most cases. The reason is that it can be easily trapped in troublesome scenarios before reaching the goal. In the example shown in Fig. 9(a), though the robot difficultly escapes from A after a loop, it is still trapped at B. Such circumstance is the local minima problem [28] which is a well known limitation of most reactive navigation algorithms since they often use greedy strategies to search the locally optimal motion direction but have no global view. However, DH-Bug finishes all the tasks and it also works in a reactive mode. The reason is that DH-Bug will record the minimal distance to the goal in its previous trajectories to determine its future motion whose purpose is decreasing the distance. This information has actually provided a global view for it. FHNA finishes more tasks than APF in most cases especially in the maze environment which contains much more traps. It is because FHNA can avoid many obstacles in advance before being trapped by following the reference path generated by its deliberative layer. Nevertheless, its performance will rapidly decrease when the incompleteness of the prior knowledge grows higher. As the example shown in Fig. 9(c), the robot is trapped at A just due to an unforeseen obstacle. Furthermore, FHNA is not only suffered from the local minima problem but also conflicts between the two layers. Two main conflicts have been detected in our simulations. The first type is that the sub-goals generated by the deliberative layer may be unreachable for the reactive layer. For example, the robot is trapped at A in Fig. 8(c) since SG6 shown in Fig. 7(a) is too close to the obstacle. As a comparison, NHNA resolves such conflict by skipping the corresponding sub-goal as shown in Fig. 7(b). The second type is that the deliberative layer concludes that the goal is unreachable based on the prior knowledge and therefore FHNA stops running. However, the conclusion may be deduced by the errors in the prior knowledge. Such conflict will not occur in NHNA since only our reactive layer can make such a conclusion based on on-line data. Additionally, a reference path generated from highly incomplete knowledge sometimes may make the situation even worse rather than better. For example, the robot is trapped at A by using FHNA in Fig. 9(c) but it escapes from the same point in Fig. 9(a) by using APF. The reason can be detected by Fig. 7(c). As shown in Fig. 7(c), compared with the direction from A to G, the direction from A to SG1 is more against the direction from A to the corridor above it, and therefore it is easier for SG1 to pull the robot to A by its attractive force. When there are more unknown obstacles and errors, the conflict mentioned above occurs more frequently and therefore FHNA even performs worse than APF in the first environment as shown in Table 2.

SHNA performs much better than APF and FHNA in most cases since it can re-compute the reference path on line. Due to this advantage, SHNA will not be suffered from the local minima problem and the incompleteness and errors in the prior knowledge does not significantly affect the performance of SHNA. As shown in Fig. 8(d), SHNA can skip SG6 by re-computing the reference path after updating the map by the sensory data. However, the expensive computational time caused by frequently re-computing the reference path becomes the main problem of SHNA. The computational time in one control cycle should be limited since a long cycle will degrade the reactivity and lead to oscillations and collisions. It

Table 1
The successful ratios under lowly incomplete knowledge.

Environment	APF (%)	DH-Bug (%)	FHNA (%)	SHNA (%)	NHNA (%)
Scattered obstacles	78	100	85	93	100
Maze	30	100	70	64	100

Table 2
The successful ratios under highly incomplete knowledge.

Environment	APF (%)	DH-Bug (%)	FSNA (%)	SHNA (%)	NHNA (%)
Scattered obstacles	78	100	62	91	100
Maze	30	100	40	63	100

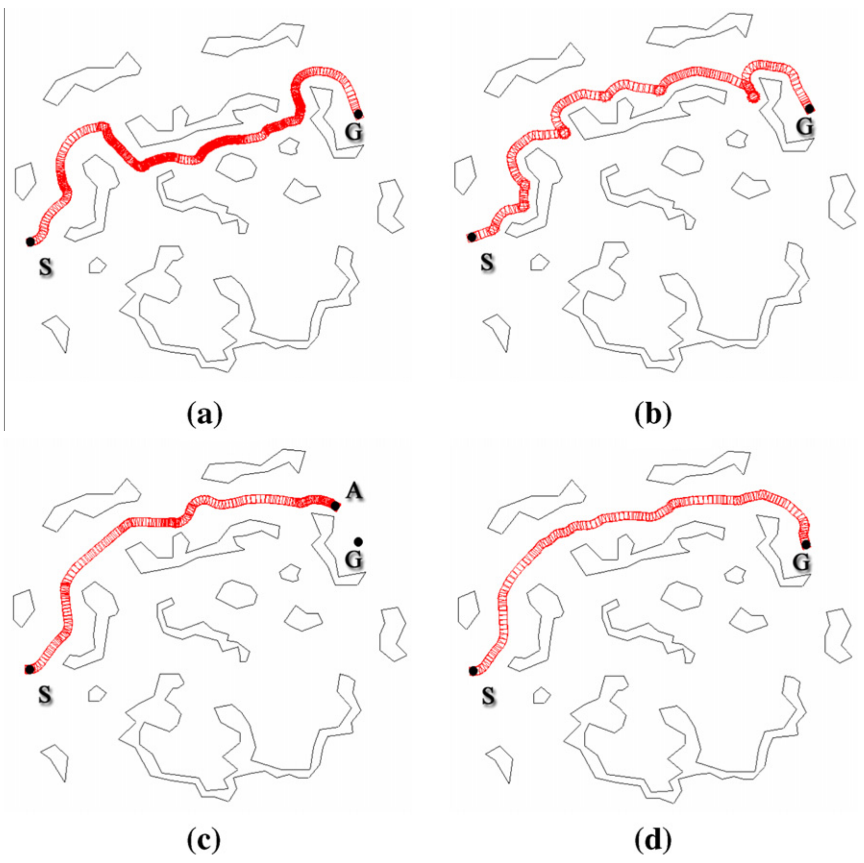

Fig. 8. The simulation results of the other four algorithms in the environment containing scattered obstacles under the same conditions as the corresponding simulation in Fig. 7: (a) APF; (b) DH-Bug; (c) FHNA; (d) SHNA.

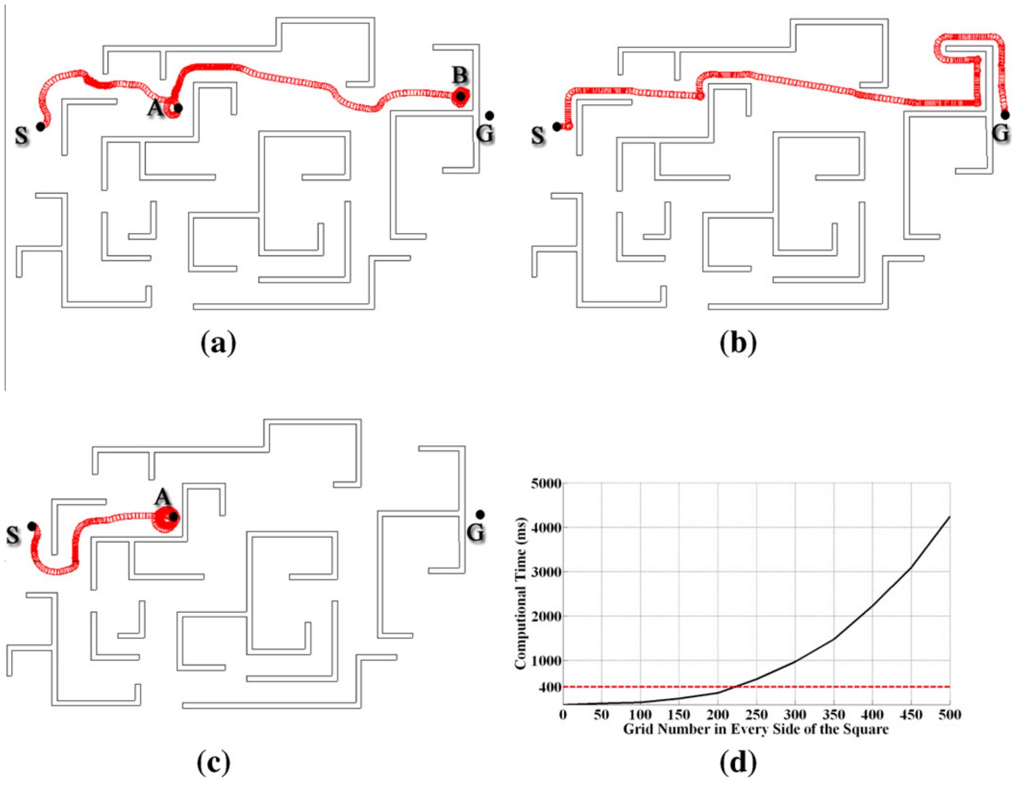

Fig. 9. The simulation results of the other four algorithms in the maze environment under the same conditions as the corresponding simulation in Fig. 7: (a) APF; (b) DH-Bug; (c) FHNA; (d) the computational time to compute a path between the two endpoints in the diagonal line of a square map where all the grids are free (SHNA fails since its computational time exceeds the control cycle).

Table 3
The average path lengths under lowly incomplete knowledge.

Environment	DH-Bug	SHNA	NHNA
Scattered obstacles	1.95	1.25	1.27
Maze	2.29	1.29	1.88

Table 4
The average path lengths under highly incomplete knowledge.

Environment	DH-Bug	SHNA	NHNA
Scattered obstacles	1.97	1.34	1.78
Maze	2.23	1.38	2.16

is required to be within 0.2 s (the time that a sensor updates its data) in a previous work [11]. In our simulations, even though we extend it to 0.4 s and the algorithm will stop only when it exceeds 0.4 s in five successive cycles, SHNA still fails a lot of tasks in the maze environment as shown in Tables 1 and 2 due to this reason. For example, SHNA fails the task shown in Fig. 7(c) since the average time of its first five cycles is 1.19 s which seriously exceeds the upper limit and is long enough for a robot without control to hit a nearby obstacle. SHNA performs better in the environment containing scattered obstacles since it is much simpler (there will be fewer occupied grids) and smaller (the possible distance between the start point and goal is shorter) than the maze environment, which decreases the time for searching a path. Fig. 9(d) depicts the times for computing a path between the two endpoints in the diagonal line of a square map where all the grids are free when the grid number in the map grows. As shown in Fig. 9(d), the computational time grows rapidly and it exceeds 0.4 s after the grid number in the side of the map is larger than about 220. If not all the grids are free, more time will be required. As a comparison, the average computational cycles of APF, DH-Bug, FHNA and NHNA in all the simulations are, respectively 0.13, 0.14, 0.13 and 0.15 s. To keep the computational time within the control cycle, a simple approach is to limit the size and resolution of the map [11], but it cannot solve the problem in essence. For example, the goal may be outside of the map. Furthermore, though SHNA can avoid the first type of conflicts in FHNA in most cases by re-computing the reference path, it is still suffered from the second type of conflicts in FHNA, i.e., it may make a wrong conclusion that the goal is unreachable and stop.

As described above, an approximate order of reliability can be concluded as follows (from low to high): APF, FHNA, SHNA, DH-Bug and NHNA. Furthermore, we have further compared DH-Bug,

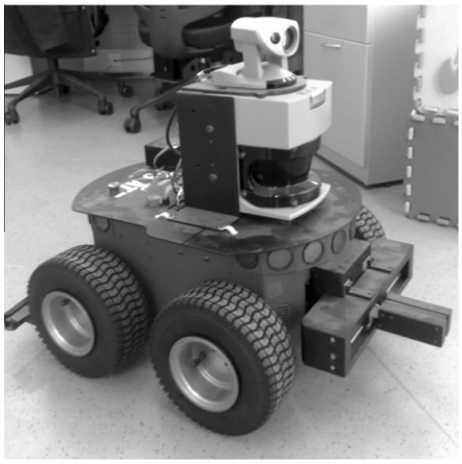

Fig. 11. The Pioneer3-AT robot used in the experiments.

SHNA and NHNA by their average path lengths generated in the simulations that all of them finish the tasks. The result is shown in Tables 3 and 4. From the results, an approximate order can be concluded as (from long to short): DH-Bug, NHNA, SHNA. NHNA performs better than DH-Bug since it has a deliberative layer to generate a reference path based on the prior knowledge but DH-Bug can only grope its way without any guidance. SHNA generates the shortest average path length since it frequently re-computing the reference path, which can correct the errors in the prior knowledge. Additionally, the performances of both SHNA and NHNA will be degraded when the incompleteness in the prior knowledge increased, but SHNA is less sensitive to the changes because it will frequently update the reference path. The results indicate that making full use of the knowledge, especially the accurate knowledge is significant for shortening the path, and SHNA performs better than NHNA in this aspect. However, it does not mean the hybrid architecture of SHNA is more efficient than NHNA since NHNA can easily shorten the path length based on its architecture, e.g., by re-computing the reference path in every control cycle as long as the process time does not exceed an upper limit, but SHNA cannot work without re-computing even when it has exceeded the control cycle since APF may be trapped in a local minimum. In other words, from the perspective of design space [34], the independent reactive layer of NHNA remains a larger free space for the design of the deliberative layer, and how to make full use of the deliberative layer when it is idle will be our future works. For example,

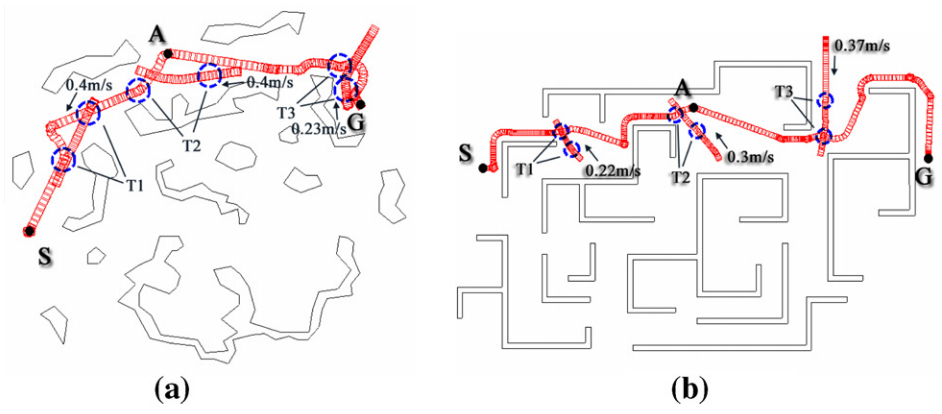

Fig. 10. Two simulation results of the new hybrid navigation algorithm when there are moving obstacles.

a learning mechanism similar to the work presented in [13] can be integrated in our deliberative layer to further improve the system performance with less consideration of the computational burden that it may impose.

Finally, we also test the validity of the new algorithm proposed in this paper when there are moving obstacles through more than fifty simulations. The new algorithm performs well in these simulations. Two results are shown in Fig. 10 (the positions of the static obstacles and the prior knowledge are the same as the simulations in Fig. 7). In Fig. 10(a), the robot detects two moving obstacles and attempts to avoid them respectively at the moment of T1 and T2. The robot then skips SG4 and SG5 after reaching A and detects the third moving obstacle at the moment of T3. After avoiding it, the robot reaches the goal. In Fig. 10(b), the robot detects the three moving obstacles respectively at the moment of T1, T2, T3, and it has skipped SG1 and SG2 at A. Finally, it successfully reaches the goal. Note that the existence of moving obstacles will not significantly changes the order of the successful ratios shown in Tables 1 and 2, since the main influential factors are the local minima problem, conflicts between the two layers and timeout of the control cycle, rather than moving obstacles.

3.2. Experiments on a Pioneer3-AT robot

To further verify the applicability of the new hybrid navigation algorithm proposed in this paper, we have implemented it on a real Pioneer3-AT robot (shown in Fig. 11) in an office environment populated by tables, chairs and various boxes. Laser and sonar sensors equipped on the robot are applied as range sensors to detect the distance to obstacles. Additionally, several other Pioneer3-AT robots are used as moving obstacles to disturb the navigation task in some experiments. These moving obstacles will send their positions, velocities and radiuses to the robot through a wireless network as long as they are within the view of the robot, which imitates the application scenario that several robots equipped with communication devices work in the same environment. For different application scenarios, there are also other approaches to acquire such information, e.g., using cameras [8].

Fig. 12 presents two results of our experiments. In the experiment shown in Fig. 12(a), we have a rough knowledge of the workspace (shown in Fig. 12(c)) in advance. After receiving a navigation task (G1), the robot computes a reference path and starts to

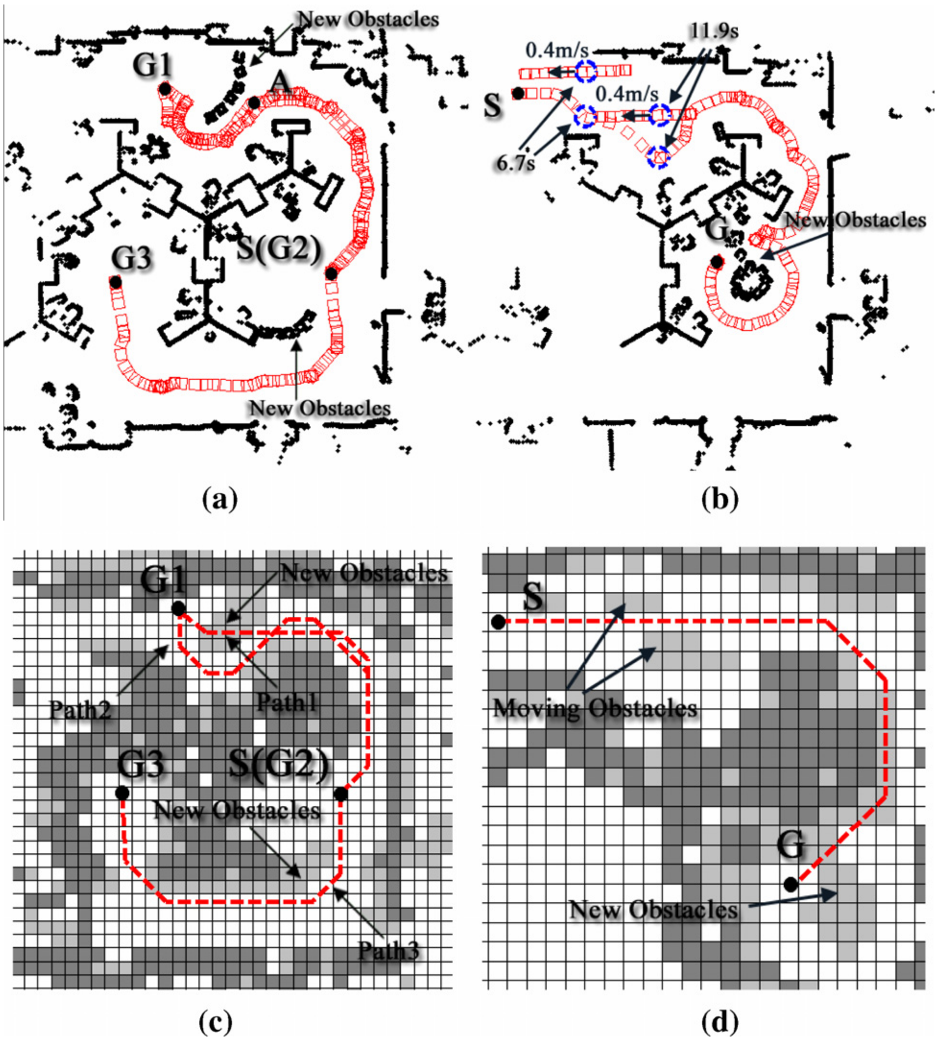

Fig. 12. Two experimental results of the new hybrid navigation algorithm: (a) the robot sequentially reaches three goals in a complex environment; (b) the robot reaches the goal after avoiding two moving obstacles; (c) and (d) present the grid map (S is the start point; G is the goal; dark-grey grids are occupied according to the prior knowledge; light-grey grids are free or unknown according to the prior knowledge but occupied according to the sensors; white grids are free or unknown according to both the prior knowledge and sensors) and reference paths (red dash line) of the experiments depicted in (a) and (b). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

approach the goal. However, it detects several unforeseen obstacles that obstruct its path at A. Therefore it avoids them by following their boundaries and finally reaches G1. Then it receives the second task (G2) and computes a reference path that has taken the new obstacles mentioned above into account. Hence it directly avoids these obstacles by following the reference path this time and finally reaches G2. Before reaching G2, the robot detects several new obstacles again. Therefore when the third task (G3) is received, the reference path has considered these obstacles and the robot can easily reach G3. In the experiment shown in Fig. 12(b), besides several new obstacles out of the prior knowledge, there are two moving obstacles. After computing the reference path (shown in Fig. 12(d)), the robot detects that these obstacles whose speeds are about 0.4 m/s have obstructed the path and threatened it. It then rapidly avoids them after 6.7 and 11.9 s. After that, the robot follows the reference path, avoids some other unforeseen obstacles and finally reaches the goal. In the two experiments, the highest speed of the robot is up to more than 0.6 m/s. The control cycle is about 0.1 s. Note that the control cycle in the real experiments is shorter than that in simulations since there are other system threads (e.g., the operating and anti-virus software) running in the computer used for simulations and they will take up the process time.

4. Conclusions

A new hybrid navigation algorithm is proposed in this paper for mobile robots working in environments with incomplete knowledge. In the new algorithm, a binary grid map with the A^* algorithm is used in the deliberative layer, and the DH-Bug algorithm that belongs to the Bug family is applied in the reactive layer. The deliberative layer generates a reference path in the form of several sub-goals based on the incomplete knowledge and uses it to guide the reactive layer. To deal with the incompleteness and possible errors in the prior knowledge, the reactive layer does not simply follow the reference path but can revise it by skipping some sub-goals once a shorter collision free path is detected. The convergence of the algorithm has been proved.

The new hybrid navigation algorithm differs from many previous works in its novel architecture which is also the main contribution of this paper. Unlike most existing hybrid navigation algorithms whose deliberative layers usually play the dominant role while the reactive layers are only simple executors, the DH-Bug algorithm used in our reactive layer can guarantee convergence by itself. The new architecture brings two benefits. First, the deliberative layer has not to frequently re-compute the reference path for preventing the robot from getting stuck in traps, which significantly releases the computational burden of the system. Second, some possible layer conflicts in the traditional architecture are resolved. In the new architecture, the result of the reactive layer will be adopted when there are conflicts between the two layers since practical trials based on on-line sensory data are more reliable than plans based on incomplete knowledge. The correctness of such rule depends on the advantage that our reactive layer can guarantee convergence without the assistance of the deliberate layer.

The simulation results indicate that the new algorithm performs more reliably than the APF, FHNA and SHNA method which represent three main types of the existing architectures. The latter three algorithms are suffered from either the local minima problem or the conflicts between layers and the expensive computational time. Furthermore, the new algorithm can acquire a shorter average path length than DH-Bug after utilizing the possible knowledge of the environment by integrating DH-Bug with a deliberative layer. The experiments on a real robot have further verified the validity and applicability of the new algorithm proposed in this paper. However, as discussed in Section 3.1, while having large improvement space, the new algorithm currently has not made full use of the deliberative layer based on its reliable architecture and therefore its average path length is longer than SHNA, and therefore how to further exploit the deliberative layer in the architecture proposed in this paper will be our future works.

References

[1] J.C. Latombe, Robot Motion Planning, Kluwer Academic Publishers, Boston, MA, 1991.
[2] O. Khatib, Real-time obstacle avoidance for manipulators and mobile robot, International Journal of Robotics Research 5 (1) (1986) 90–98.
[3] J. Borenstein, Y. Koren, The vector field histogram – fast obstacle avoidance for mobile robots, IEEE Journal of Robotics and Automation 7 (3) (1991) 278–288.
[4] D. Fox, W. Burgard, S. Thrun, The dynamic window approach to collision avoidance, IEEE Robotics and Automation Magazine 4 (1) (1997) 23–33.
[5] B. Beaufrere, S. Zeghloul, A mobile robot navigation method using a fuzzy logic approach, Robotica 13 (1995) 437–448.
[6] J. Minguez, L. Montano, Nearness diagram (ND) navigation: collision avoidance in troublesome scenarios, IEEE Transactions on Robotics and Automation 20 (1) (2004) 45–59.
[7] C. Ye, P. Webb, A sub goal seeking approach for reactive navigation in complex unknown environments, Robotics and Autonomous Systems (2009) 877–888.
[8] S.S. Ge, Y.J. Cui, Dynamic Motion planning for mobile robots using potential field method, Autonomous Robots 13 (3) (2002) 207–222.
[9] L. Huang, Velocity planning for a mobile robot to track a moving target – a potential field approach, Robotics and Autonomous Systems 57 (2009) 55–63.
[10] O. Brock, O. Khatib, High-speed navigation using the global dynamic window approach. Proceedings of the IEEE International Conference on Robotics and Automation, 1999, pp. 341–346.
[11] J. Mingues, L. Montano, Sensor-based robot motion generation in unknown dynamic and troublesome scenarios, Robotics and Autonomous Systems 52 (2005) 290–311.
[12] J. Mingues, Integration of planning and reactive obstacle avoidance in autonomous sensor-based navigation. Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005, pp. 2486–2492.
[13] M.D. Power. Applying inter-layer conflict resolution to hybrid robot control architectures. Ph.D. Dissertation, Georgia Institute of Technology, 2010.
[14] A. Ranganathan, S. Koenig, A reactive robot architecture with planning on demand. Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003, pp. 1462–1468.
[15] W. Choi, J. Latombe. A reactive architecture for planning and executing robot motion with incomplete knowledge. Proceedings of the IEEE/RSJ International Workshop on Intelligent Robots and Systems, 1991, pp. 24–29.
[16] B. H. Krogh, C. E. Thorpe. Integrated path planning and autonomous vehicles. Proceedings of the IEEE International Conference on Robotics and Automation, 1986, pp. 1664–1669.
[17] T. Zhang, Y. Zhu, J. Song, A hybrid path planning method for partly unknown environment, ICCAS-SICE (2009) 2485–2490.
[18] S. Quinlan, O. Khatib, Elastic bands: connecting path planning and control. Proceedings of the IEEE International Conference on Robotics and Automation, 1993, pp. 802–807.
[19] O. Brock, O. Khatib, Real-time replanning in high-dimensional configuration spaces using sets of homotopic paths. Proceedings of the IEEE International Conference on Robotics and Automation, 2000, pp. 550–555.
[20] S. Koenig, M. Likhachev, Fast replanning for navigation in unknown terrain, IEEE Transactions on Robotics 3 (2005) 354–363.
[21] R. Simmons, R. Goodwin, K.Z. Haigh, S. Koenig, J. O'Sullivan. A layered architecture for office delivery robots. Proceedings of the International Conference on Autonomous Agents, 1997.
[22] E. Gat, On three-layer architectures. Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems, Menlo Park: AAAI Press, 1998, pp. 195–210.
[23] Y.-C. Chang, Y. Yamamoto, Path planning of wheeled mobile robot with simultaneous free space locating capability, Intelligent Service Robotics 2 (1) (2009) 9–22.
[24] K. Berns, K.-D. Kuhnert, C. Armbrust, Off-road robotics — an overview, Künstliche Intelligenz 25 (2011) 109–116.
[25] V. Lunelsky, K. Harinarayan, Decentralized motion planning for multiple mobile robots: the cocktail party model, Autonomous Robots 4 (1997) 121–135.
[26] S.S. Ge, F.L. Lewis, Autonomous Mobile Robots: Sensing, Control, Decision Making and Applications, Taylor & Francis, Boca Raton, 2006.
[27] S. Bayili, F. Polat, Limited-Damage A^*: a path search algorithm that consider damage as a feasibility criterion, Knowledge-Based Systems 24 (2011) 501–512.
[28] T. Zhang, Y. Zhu, J. Song, Real time motion planning for mobile robots by means of artificial potential field method in unknown environment, Industrial Robot: An International Journal 37 (4) (2010) 384–400.

[29] O.R.E. Motlagh, T.S. Hong, N. Ismail, Development of a new minimum avoidance system for a behavior-based mobile robot, Fuzzy Sets and Systems 160 (2008) 1929–1946.
[30] C. Ordonez Jr, E.G. Collins, M.F. Selekwa, D.D. Dunlap, The virtual wall approach to limit cycle avoidance for unmanned ground vehicles, Robotics and Autonomous Systems 56 (2008) 645–657.
[31] J. Ng, T. Braunl, Performance comparison of bug navigation algorithms, Journal of Intelligent and Robotic Systems 50 (2007) 73–84.
[32] V. Lumelsky, A. Stepanov, Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape, Algorithmica 2 (1987) 403–430.
[33] Y. Zhu, T. Zhang, J. Song, X. Li, A new Bug-type algorithm for navigation of mobile robots in unknown environments containing moving obstacles. Industrial Robot: An International Journal (in press).
[34] N. Hawes, J. Wyatt, A. Sloman, Exploring design space for an integrated intelligent system, Knowledge-Based Systems 22 (2009) 509–515.