DOI:10.15918/j.jbit1004-0579.2012.03.006

Journal of Beijing Institute of Technology, 2012, Vol. 21, No. 3

Variable dimensional state space based global path planning for mobile robot

ZHANG Hao-jie(张浩杰)✉, CHEN Hui-yan(陈慧岩), JIANG Yan(姜岩), GONG Jian-wei(龚建伟), XIONG Guang-ming(熊光明)
(School of Mechanical Engineering, Beijing Institute of Technology, Beijing 100081, China)

Abstract: A variable dimensional state space (VDSS) has been proposed to improve the re-planning time when the robotic systems operate in large unknown environments. VDSS is constructed by uniforming lattice state space and grid state space. In VDSS, the lattice state space is only used to construct search space in the local area which is a small circle area near the robot, and grid state space elsewhere. We have tested VDSS with up to 80 indoor and outdoor maps in simulation and on segbot robot platform. Through the simulation and segbot robot experiments, it shows that exploring on VDSS is significantly faster than exploring on lattice state space by Anytime Dynamic A* (AD*) planner and VDSS is feasible to be used on robotic systems.

Key words: variable dimensional state space; lattice state space; Anytime Dynamic A* (AD*); path planning

CLC number: TP 242.6 **Document code**: A **Article ID**: 1004-0579(2012)03-0328-08

Wheeled mobile robot that operates in real world needs to respond very quickly to changes in the environment so that the resulting maneuvers can be executed in a timely manner, especially when the environment is unknown, dynamic, or dangerous. Because sensors are imperfect, wheeled mobile robot navigating in unknown environments must re-plan whenever it receives new sensory data in order to ensure a safe, low cost path. In particular, planning a smooth and optimal path within a minimal time is becoming more and more important for the wheeled mobile robot in such environments.

One of the most common implementations of path planning algorithms for wheeled mobile robot utilize cost maps (or, in other word, 2D grid worlds) to represent the surrounding environment. It is easy to find the path in 2D grid state space. But the path may be unfeasible, since it does not consider about the constraints of the movement of the robot, shown in Fig. 1a. If we add the orientation for each state in 2D grid state space, called lattice state space, shown in Fig. 1b, the planning may be computationally challenging. Especially, when the environment becomes very large, the planning time will be very long, since we add orientation as an additional dimension to lattice state space.

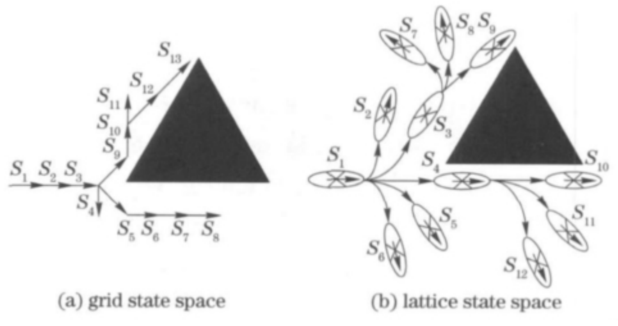

S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8 S_9 S_{10} S_{11} S_{12} S_{13}
(a) grid state space
S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8 S_9 S_{10} S_{11} S_{12}
(b) lattice state space

Fig. 1 Example of two different state spaces

1 Related work

In order to reduce the planning time, roboti-

Received 2011-06-04
Supported by the National Natural Science Foundation of China(90920304)
✉ Author for correspondence
E-mail: haojie.bit@gmail.com

cists have concentrated on improving planning algorithms and designing state space for several decades, providing a large body of research. Basically there are two main types of planners used: heuristic search algorithms and randomized search algorithms. Heuristic search algorithms, such as most notably A* [1], can find optimal paths, but typically do not run fast enough to re-plan in real time when the robot receives new sensory data. Many variations based on A* have been proposed in order to improve the re-planning efficiency, such as Lifelong Planning A* (LPA*) [2], Anytime Repairing A* (ARA*) [3] and Anytime Dynamic A* (AD*) [4], typically by saving and reusing state values from previous searches. A common approach using randomization is rapidly exploring random trees (RRTs) [5-6], which is designed to explore the environment quickly. RRTs is guaranteed to find some path to the goal, but not necessarily an optimal path. The quality of the path generated by RRTs is hard to identify.

Roboticists have also separately used 2D grid state space and lattice state space to represent the environment for planning. The 2D grid state space only has two dimensions (x, y), where (x, y) represents the position of the robot in the world. However, the path generated by planning on 2D grid state space is sometimes unfeasible, since it doesn't take into account the robot's other constraints. A novel state space called variable sized grid cells was introduced for rapid re-planning in dynamic environments[7]. The state space was divided into different kinds of resolution. Close to the robot, it used high resolution grid. Further away from the robot, it switched to a coarser resolution grid. This approach made up the lost of orientation to a certain degree. A lattice state space with three dimensions (x, y, θ) is used. It introduced orientation θ to the state space. However, the planning time is increasing along with the dimension θ is introduced. Besides, a multi-resolution lattice state space is also designed for planning[8]. This improved state space is helpful to reduce planning time and obtain a smooth path when the deliberation time is limited.

2 Approach description

Our approach is to modify the search state space explored by AD* planner. In this way, the planner can be used unchanged. Instead of performing the entire search either on a single grid state space or on a single lattice state space, we perform the search on a variable dimensional state space (VDSS). That is, only the areas near the robot are searched carefully through using lattice state space; areas further away from the robot are searched coarsely through using grid state space. Because this results in fewer states that need to be explored, planning can be accomplished rapidly. The lattice state space will move as the robot is moving, so that the robot will always have a smooth path defined for its next action.

The key idea of this approach is that, the planner will always regenerate a new and smooth path before reaching grid state space by keeping lattice state space centered over the robot's current position and moving with the robot. If the search can be done quickly, the robot can re-plan at each time step.

3 Algorithm

We define a small range of lattice state space around the robot and grid state space elsewhere as variable dimensional state space. By aligning the search between the two different state spaces, our search space remains compatible with standard heuristic search algorithms and is capable of producing path planning in unknown environments.

3.1 Graph construction

3.1.1 Improved grid and lattice motion primitives

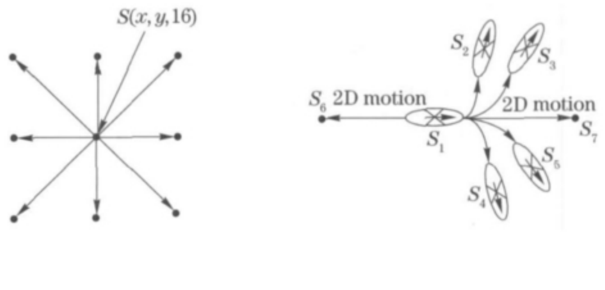
In order to make VDSS applicable in standard search algorithms, our approach is to add a virtual orientation coordinate for each state in grid state space. The angle should be different from that in lattice state space. The orientation is divided into 16 pieces with interval $\pi/8$ in lattice state space. We have chosen $[2\pi, 17\pi/8]$ as the interval of virtual orientation angle and it can be represented by discrete digital quantity 16. As a result, we use an 8-connected control set as motion primitives in grid state space and the states in it can be represented by three dimensions, where orientation coordinate 16 is a constant (Fig. 2a). The improved control set is a little different from regular 8-connected control set in grid state space[9], since each state has three dimensions (x, y, θ) and θ is a constant orientation coordinate ($[2\pi, 17\pi/8] \rightarrow 16$).

In order to align the motion between grid state space and lattice state space, we add two additional 2D motion primitives for states in lattice state space (Fig. 2b). The two additional 2D motion primitives are added based on the orientation of 3D states in lattice state space. They will be reserved if they are in grid state space and will be dropped if they are in lattice state space.

3.1.2 Handling boundary challenges

The biggest implementation challenge in planning with VDSS is handling the boundaries between grid state space and lattice state space. Actions within each state space should move the robot from one state to another state. It is easy to handle when the robot just moves either in grid state space or in lattice state space separately. However, at the boundaries between the two different dimensional state spaces, actions may move the robot to a different dimensional state space, such as from grid state space to lattice state space or from lattice state space to grid state space. These are illustrated by Fig. 3. The dashed arrow represents 2D motion, and the solid arrow represents 3D motion.

As shown in Fig. 3, when we expand state from grid state space to lattice state space, for example, state S_5 is expanded, for each predecessors of state S_5, we judge whether it is in grid state space or in lattice state space. Most predecessors of state S_5 are still in grid state space



(a) 2D motion primitives (b) 3D motion primitives

Fig. 2 Improved motion primitives

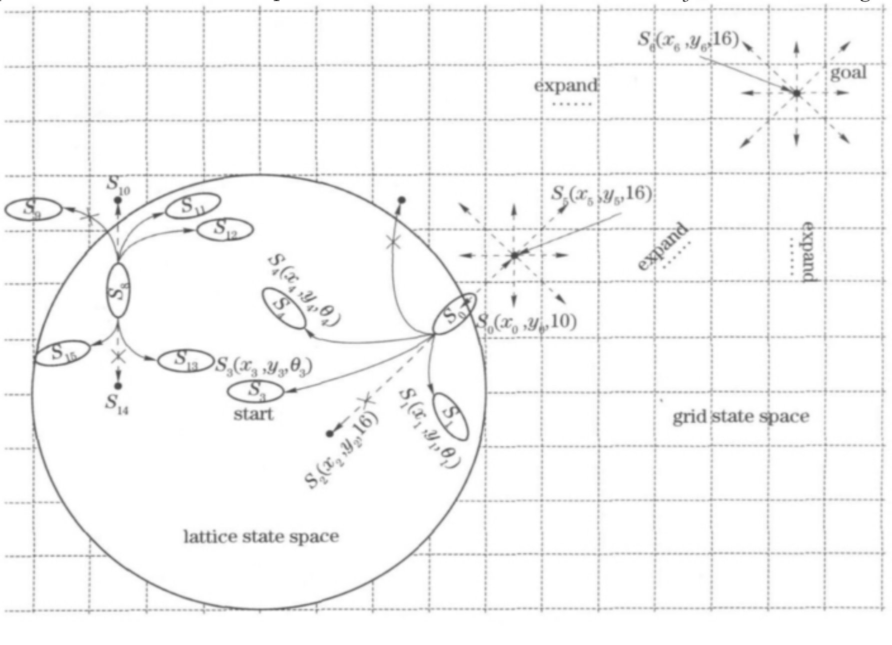


Fig. 3 Variable dimensional state space for planning

except state S_0, we add a constant orientation coordinate 16 to the predecessors which are still in grid state space and update the orientation coordinate of state S_0 with 10. The updated orientation coordinate is calculated by the state's direction relative to its parent state. For example, the orientation of state S_0 is $5\pi/4$ relative to its parent state S_5 and discrete it to 10((int)$(5\pi/4/\pi/8)$ = 10). While we need to expand S_0, we use improved 3D motion primitives since the state S_0 is in lattice state space.

When we expand state from lattice state space to grid state space, for example, state S_8 is expanded. It has seven motion primitives, including two 2D motion primitives and five 3D motion primitives. We just keep 2D motion primitives that are in grid state space and 3D motion primitives that are in lattice state space. Accurately, we drop 3D state S_9 that is in grid state space and 2D state S_{14} that is in lattice state space. Then the actions at the boundaries are defined, regardless of they are from grid state space to lattice state space and vice versa.

3.2 Anytime dynamic A*

In order to verify the advantages of VDSS, we use AD* planner[10] to search with VDSS and lattice state space separately.

The AD* planner exploits a property of A* that can result in much faster generation of solutions, namely that if consistent heuristics are used and multiplied by an inflation factor $\varepsilon > 1$, then A* can often generate a solution much faster than if no inflation factor is used[11], and the cost of the solution generated by A* will be at most ε times the cost of an optimal solution[12]. AD* operates by performing a series of inflated A* searches with decreasing inflation factors, where each search reuses information from previous searches. By doing so, it is able to provide suboptimal bounds on all solutions generated and allows for control of these bounds, since the user can decide how much the inflation factor will be decrease between searches.

3.3 Planning with variable dimensional state space

After the initialization, we run the AD* planner, shown in Algorithm 1, which runs as a usual expect for how it updates the membership of a state (Algorithm 2) and how it updates the states in the 3D space-old and 3D space-new lattice state spaces before each re-plan (Algorithm 3).

Algorithm 1 Backward AD* with VDSS

```
1 procedure key($s$)
2 if($v(s) \geqslant g(s)$)
3   return $[g(s) + \varepsilon * h(s); g(s)]$;
4 else
5   return $[v(s) + h(s); v(s)]$
6 procedure Main()
7 $g(s_{goal}) = v(s_{goal}) = \infty$; $v(s_{start}) = \infty$; $g(s_{start}) = 0$;
8 $bs(s_{goal}) = bs(s_{start}) = \varnothing$;
9 OPEN = CLOSED = INCONS = $\varnothing$; $\varepsilon = \varepsilon_0$;
10 insert $s_{start}$ into OPEN with key($s_{start}$);
11 forever
12   ComputePath();
13   publish $\varepsilon$-suboptimal solution;
14 UpdateStatesofLattice();
15 if $\varepsilon = 1$
16 detect the changes in edge cost;
17 for all directed edges $(u, v)$ with changed edge
   cost
18   update the edge cost $c(u, v)$;
19   if($v \neq s_{start}$ and $v$ was visited by AD* before)
20     $bs(v) = \arg\min_{s' \in succs(v)}(v(s' + c(s', v)))$;
21     $g(v) = v(bs(v)) + c(bs(v), v)$;
22   if significant edge cost changes were observed
23     increase $\varepsilon$ or re-plan from scratch;
24 else if $\varepsilon > 1$
25     decrease $\varepsilon$
26 Move states from INCONS into OPEN
27 Update the priorities for all $s \in$ OPEN according to key($s$);
28 CLOSED = $\varnothing$;
```

Algorithm 2 UpdateSetMembership

```
1 procedure UpdateSetMembership($s$)
2 if($v(s) \neq g(s)$)
3   if($s \notin$ CLOSED) insert/update $s$ in OPEN with
```

```
      key($s$);
4     else if ($s \notin$ INCONS) insert $s$ into INCONS;
5 else
6     if($s \in$ OPEN) remove $s$ from OPEN;
7     else if($s \in$ INCONS) remove $s$ from INCONS;
```

Algorithm 3 UpdatestatesofLattice

```
1 procedure UpdateStatesofLattice()
2 for $\forall s \in$ 3D space-old $\cup$ 3D space-new
3 set $g(s) = \infty$ and $v(s) = \infty$; UpdateSetMembership($s$);
4 for all the state $s$ ($s \neq s_{\text{start}}$) that was generated on the frontier
5     set $g(s) = \infty$
6     $bs(s) = \arg\min_{s' \in \text{succs}(s)} (v(s') + c(s', s))$;
7     $g(s) = v(bs(s) + c(bs(s), s))$;
8     UpdateSetMembership($s$);
9 set new search start;
10 for $\forall s$ on the frontier
11 if $v(s) \neq \infty$
12     for $\forall s' \in$ Preds($s$)
13         if $s' \in$ 3D space-old $\cup$ 3D space-new
14         set $g(s') = \infty$
15         $bs(s') = \arg\min_{s'' \in \text{suces}(s') (v(s'') + c(s'', s'))}$;
16         $g(s') = v(bs(s')) + c(bs(s'), s')$;
17         UpdateSetMembership($s'$);
```

As shown in Fig. 4, the lattice state space moves as the robot (black square) moves to a new position. We will get two lattice state spaces, referred to 3D space-old and 3D space-new.

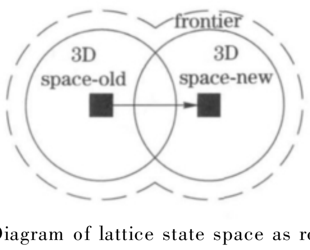


Fig. 4 Diagram of lattice state space as robot moves

First, we check all the states that are generated in the 3D space-old or 3D space-new and set their g value and v value to infinite (line 3, Algorithm 3). Then we move them away from the search tree as the robot never saw the two areas.

Second, we check all the states on the frontier, which may expand into lattice state space and set their g value to infinite (line 5, Algorithm 3). For these states, we find their best successor again (line 6 and 7, Algorithm 3). At the same time, we put them to the search tree according to their g value and v value again through function UpdatesetMembership in Algorithm 2.

Third, we get the predecessors of the states whose v value are not infinite. It indicates that these states have been expanded during previous planning. We only set those predecessors' g value to infinite if they are in the 3D space-old or 3D space-new (line 13 and 14, Algorithm 3). Then we find their best successor again (line 15 and 16, Algorithm 3). At the same time, we put them to the search tree according to their g value and v value again through function UpdatesetMembership in Algorithm 2.

In the algorithm, it only needs to re-expand some states on the frontier, 3D space-old and 3D space-new during AD* search. It can reuse the state value from previous searches. As a result, the number of expanding state will be less and the planning time will decrease largely.

4 Experiment results

We ran experiments in the simulation and on a real robot to evaluate the efficiency of VDSS, while planning in unknown environments.

4.1 Planning implementation

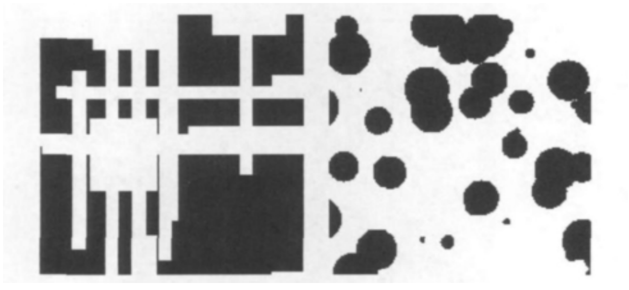
In VDSS, the actions used to get successors for states in lattice state space are a set of "motion primitives"[13], which are improved based on the motions sequences[8] used in lattice-based planner, shown in Fig. 2b. The motion primitives used to get successors for states in grid state space are 8-connected grids, shown in Fig. 2a. We choose AD* as our planner. For planning with lattice state space, we initially run a 16-connected 2D Dijkstra search from the goal to all the (x, y) cells in the environment to get the heuristic, assuming the robot is a circular with a radius equal to the actual robot's inscribed circle. For planning with VDSS, we choose Euclidean distance as our heuristic, since most of the states in the search space are 2D grid states.

4.2 Simulation

4.2.1 Experiment design

We compare VDSS against lattice state space by using AD* planner. In lattice state space, it is a 3D search space (x, y, θ). In VDSS, we set a circle (radius = 4 m) around the robot as lattice state space and grid state space elsewhere. As a result, it was a 2D search outside the circle and a 3D search inside the circle.

In order to test the efficiency of our algorithm, we made two sets of 80 randomly generated maps to simulate indoor and outdoor type environments. All the environments are 500×500 cells, $1\,000 \times 1\,000$ cells, $1\,500 \times 1\,500$ cells and $2\,000 \times 2\,000$ cells with a resolution of 0.025 m. The amount of the maps is 10 for each different size environment. The robot's footprint occupies a single cell on the map and it has a random start and goal for each map. Then the robot navigated toward the goal, while periodically re-planning after traversing to the next state on the previous path. The indoor environments (Fig. 5a) are composed of a series of randomly placed narrow hallways and rooms on a grid. The outdoor environments (Fig. 5b) are very open, with randomly placed circle obstacles (representing trees, rocks etc.) that occupy roughly 30% of the map.



(a) indoor map (b) outdoor map

Fig. 5 An example of maps used in our experiments

Assuming the environments that we generated above are unknown previously, we set a constant inflated factor ε ($\varepsilon = 1.0, 2.0, 3.0$) separately for AD* and use it to search based on VDSS and lattice state space. In the experiments, the robot needs to detect the environment and re-plan a new path as it moves.

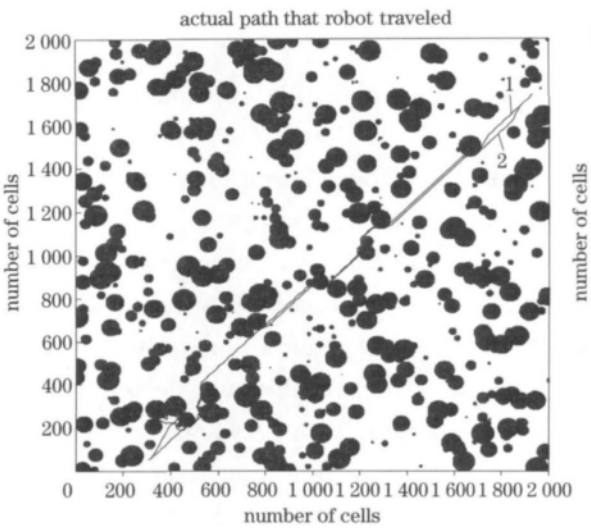
All the simulation experiments were run on an Intel (R) Core (TM) i7 CPU running at 2.93 GHz, under Ubuntu Linux operating system.

4.2.2 Experiment analysis

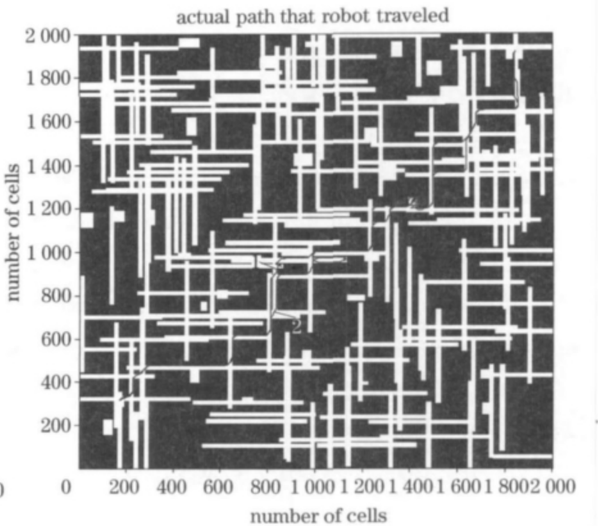
In the simulation experiments, we assume that the robot has a perception region limited to 10×10 cells, centered around it. No perception information is available outside this horizon. Also, we planned with VDSS and lattice state space on the same environments, and allocate the limit time for each re-planning is 10 s. For clarity, Fig. 6a shows the actual path that the robot traveled in outdoor environment and Fig. 6b shows the actual path that the robot traveled in indoor environment when the environments are $2\,000 \times 2\,000$ cells. Black dots and blocks are the obstacles. The path 1 is the actual traveling path generated by the lattice planner. The path 2 is the actual traveling path generated by our novel planner. The paths generated by the two planners are very similar.

Tabs. 1 – 2 show the comparison of planning with VDSS versus planning with lattice state space in more details. In particular, Tab. 1 shows the average planning time spent by the algorithm as it decreased its suboptimality bound ε. Tab. 2 gives the actual solution cost generated by the planner in each of the environment. As expected, the data shows that planning with VDSS is much faster than planning with lattice state space. The actual solution cost of planning with VDSS is a little more than planning with lattice state space when we always find a provably optimal solution ($\varepsilon = 1.0$). In real environments, it is impossible to find the optimal solution during the entire travelling, since the deliberation time for each re-plan is limited and may be run out before finding the optimal solution. On the other hand, the solution cost of planning with VDSS is much less than planning with lattice state space when we didn't find the optimal solution (for example $\varepsilon = 2.0$,

3. 0) .



(a) Simul ation results of outdoor environment



(b) Simulation results of indoor environment

Fig. 6 Simulation experiment results of traversing among previously unknown environments

Tab. 1 Average planning time in simulation experiment s

size of map/cell	ε = 1. 0		ε = 2. 0		ε = 3. 0	
	VDSS planner	lattice planner	VDSS planner	lattice planner	VDSS planner	lattice planner
500 × 500	0. 065 2	0. 079 3	0. 006 5	0. 053 5	0. 003 7	0. 045 3
1 000 × 1 000	0. 146 2	0. 247 2	0. 032 2	0. 170 7	0. 011 5	0. 170 2
1 500 × 1 500	0. 131 3	0. 493 8	0. 014 4	0. 409 5	0. 005 9	0. 4020
2 000 × 2 000	0. 148 5	0. 886 9	0. 018 3	0. 696 6	0. 008 5	0. 686 5

Tab. 2 Actual solution cost in simulation experiment cm

size of map/cell	ε = 1. 0		ε = 2. 0		ε = 3. 0	
	VDSS planner	lattice planner	VDSS planner	lattice planner	VDSS planner	lattice planner
500 × 500	43 591	37 876	45 768	52 147	57 750	66 009
1 000 × 1 000	71 130	63 375	77 049	94 632	107 442	101 940
1 500 × 1 500	102 699	84 594	114 430	131 724	124 610	161 290
2 000 × 2 000	133 640	122 040	149 060	170 750	165 600	187 760

4. 3 Tests on the segbot robot

In order to show our VDSS planner works in a real world environments, we implemented the planner in ROS and tested it on segbot robot platform, as shown in Fig. 7.

① Implementation detail

We wrote a global planner node which plugs in to ROS' s navigation stack. The navigation stack provides the planner with the robot's pose and a map of the environment and expects a path in return. VDSS planner will plan based on the map that navigation stack provides and returns a global path to the segbot robot to follow.

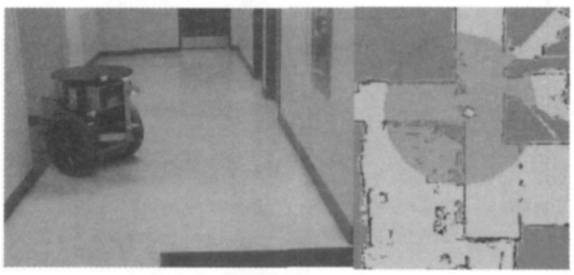


Fig. 7 Segbot robot traveling through right angle corner

② Results

In our experiments, the segbot robot had to pass the hallway and right angle turn to get to its goal. We have picked a scenario that the segbot robot needs to pass right angle turn, as shown in Fig. 7. In Fig. 7, the shadow circle represents the 3D lattice state space and the solid line means the path we planned. We get a 3D path inside the shadow circle and 2D path outside it. The shadow circle will move as the segbot moves. As a result, the segbot robot always executes the 3D path that is feasible and smooth. The planning time for each re-plan is decreasing largely, since the 2D state space decreases the amount of the states that need to be expanded.

5 Conclusion

In this paper we have introduced a novel graph structure called VDSS for decreasing the planning time in very large environments. VDSS emerges by uniforming the grid state space and the lattice state space. We use a lattice state space in the vicinity of the robot, and a grid state space elsewhere. This results in a coarse path close to the goal, but a fine path near the robot.

We have implemented VDSS in simulation experiment and on segbot robot. Compare to lattice state space planning, the results of the experiments show that the advantage of VDSS planner is obvious and it decreases the planning time largely.

References:

[1] Hart P, Nilsson N, Raphael B. A formal basis for the heuristic determination of minimum cost paths in graphs [J]. IEEE Transactions on Systems Science and Cybernetics, 1968, 2(30): 100 - 107.

[2] Koenig S, Likhachev M, Liu Yaxin, et al. Incremental heuristic search in artificial intelligence [J]. Artificial Intelligence Magazine, 2004, 25: 99 - 112.

[3] Likhachev M, Gordon G, Sebastian Thrum. ARA*: Anytime A* with provable bounds on sub-optimality [C] // The Seventeenth Annual Conference on Neural Information Processing Systems. Cambridge: MIT Press, 2004: 1 - 8.

[4] Likhachev M, Ferguson D, Gordon G, et al. Anytime dynamic A*: An anytime, replanning algorithm [C] // The International Conference on Automated Planning and Scheduling. Monterey: AAAI Press, 2005: 262 - 271.

[5] LaValle S. Rapidly-exploring random trees: A new tool for path planning [D]. Ames, Iowa, USA: Iowa State University, 1998.

[6] LaValle S, Kuffner J. Randomized kinodynamic planning [C] // IEEE International Conference on Robotics and Automation. Detroit, MI, USA: IEEE, 1999: 473 - 479.

[7] Kirby R, Simmons R, Forlizzi J. Variable sized grid cells for rapid replanning in dynamic environments [C] // IEEE/RSJ International Conference on Intelligent Robots and Systems. St. Louis, MO, USA: IEEE, 2009: 4913 - 4918.

[8] Likhachev M, Dave Ferguson. Planning long dynamically-feasible maneuvers for autonomous vehicles [J]. The International Journal of Robotics Research, 2009, 28 (8): 933 - 945.

[9] Pivtoraiko M, Knepper R, Kelly A. Optimal, smooth, nonholonomic mobile robot motion planning in state lattices, technical report CMU-RI-TR-07-15 [R]. Pittsburgh, Pennsylvania, USA: Carnegie Mellon University, 2007.

[10] Likhachev M, Ferguson D, Gordon G, et al. Anytime search in dynamic graphs [J]. Artificial Intelligence Journal, 2008, 172(14): 1613 - 1643.

[11] Gaschnig J. Performance measurement and analysis of certain search algorithms [D]. Pittsburgh, Pennsylvania, USA: Carnegie Mellon University, 1979.

[12] Davis H, Bramanti-Gregor A, and Wang Jin. The advantages of using depth and breadth components in heuristic search [J]. Methodologies for Intelligent Systems, 1988, 3: 19 - 28.

[13] Cohen B, Chitta S, Likhachev M. Search-based planning for manipulation with motion primitives [C] // IEEE International Conference on Robotics and Automation. Anchorage, AK, USA: IEEE, 2010: 2902 - 2908.

(**Edited by** Cai Jianying)