Check for updates

Research Article

INTERNATIONAL JOURNAL OF ADVANCED ROBOTIC SYSTEMS

International Journal of Advanced Robotic Systems
July-August 2018: 1–11
© The Author(s) 2018
DOI: 10.1177/1729881418787049
journals.sagepub.com/home/arx
SAGE

Optimal randomized path planning for redundant manipulators based on Memory-Goal-Biasing

Dong Han[1], Hong Nie[1], Jinbao Chen[1] and Meng Chen[2]

Abstract

Planning path rapidly and optimally is one of the key technologies for industrial manipulators. A novel method based on Memory-Goal-Biasing–Rapidly-exploring Random Tree is proposed to solve high-dimensional manipulation planning more rapidly and optimally. The tree extension of Memory-Goal-Biasing–Rapidly-exploring Random Tree can be divided into random extension and goal extension. In the goal extension, the nodes extended to the goal are recorded in a memory, and then the node closest to the goal is selected in the search tree excepting the nodes in the memory for overcoming the local minimum. In order to check collisions efficiently, the manipulator is simplified into several key points, and the obstacle area is appropriately enlarged for safety. Taking the redundant manipulator of Baxter robot as an example, the proposed algorithm is verified through MoveIt! software. The results show that Memory-Goal-Biasing–Rapidly-exploring Random Tree only takes a few seconds for the path planning of the redundant manipulator in some complex environments, and within an acceptable time, its optimization performance is better than that of traditional optimal method in terms of the obtained path costs and the corresponding standard deviation.

Keywords

Rapidly-exploring Random Tree, manipulation planning, local minimum, redundant manipulator, MoveIt!

Date received: 16 May 2017; accepted: 10 June 2018

Topic: Robot Manipulation and Control
Topic Editor: Andrey V Savkin
Associate Editor: Steven Su

Introduction

In industrial manufacturing and other domains, collision-free path planning plays an important role in performing mobile robot navigation[1] and complex manipulator operation.[2] Especially for redundant manipulators, multiple degrees of freedom (DOFs) result in high-dimensional search spaces because of generally searching in joint configuration spaces. Thus, it is a computational challenge to rapidly find an optimal feasible path for redundant manipulators.

In response to this challenge, researchers have presented sampling-based algorithms that are capable of realizing high-dimensional path planning,[3–8] such as probabilistic roadmaps, Rapidly-exploring Random Tree (RRT), and

[1] State Key Laboratory of Mechanics and Control of Mechanical Structures, Nanjing University of Aeronautics and Astronautics, Nanjing, People's Republic of China
[2] Shanghai Aerospace System Engineering Research Institute, Shanghai, People's Republic of China

Corresponding author:
Dong Han, State Key Laboratory of Mechanics and Control of Mechanical Structures, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, People's Republic of China.
Email: han_dongnuaa@126.com

Creative Commons CC BY: This article is distributed under the terms of the Creative Commons Attribution 4.0 License (http://www.creativecommons.org/licenses/by/4.0/) which permits any use, reproduction and distribution of the work without further permission provided the original work is attributed as specified on the SAGE and Open Access pages (https://us.sagepub.com/en-us/nam/open-access-at-sage).

their variants. Despite their feasibility for probabilistic completeness, it is difficult to achieve an expected balance between optimization performance and planning efficiency. For example, some variants can greatly improve the efficiency of path planning by changing extension strategies,[9] but their optimization performance is poor. Some other variants adopt specific optimization methods to lower the path costs and even tend to the optimal path,[10–12] but they easily fall into the local minimum or consume much more time.

Therefore, the main contribution of this article is to balance the conflict between optimization and efficiency and provide a more practical way for the path planning of redundant manipulators. The proposed path planner in this article is a greedy RRT algorithm based on Memory-Goal-Biasing (MGB-RRT). It can solve the local minimum problem by storing goal-extended nodes in a memory. In addition, the generated paths by MGB-RRT are post-processed for further path optimization. In order to quickly detect collisions in path planning, several key coordinate points are used to represent the pose of manipulators, and the obstacle areas are simultaneously enlarged.

The remainder of this article is organized as follows. The second section discusses the related work about RRTs. The third section describes the MGB-RRT algorithm and the fourth section presents the simple collision detection method. The fifth section shows the planning results. A discussion based on the results is presented in the sixth section. Finally, a conclusion is given in the last section.

Related work

RRTs are proved to be well suited to handle high-dimensional configuration space and non-holonomic constraints.[4,13] However, the basic RRT has several limitations. For example, the planned paths are unpredictable and suboptimal due to randomness, and the planning time may be too long to get a feasible path owing to excessive constraints or dimensions. Therefore, various variants of RRT exist to overcome the above limitations.[14,15]

Efficient RRTs

In order to speed up path planning, a bidirectional version of RRT (RRT-Connect or Bi-RRT) was proposed, which grows two trees: one starting from the goal and the other starting from the initial configuration.[9] The two trees advance toward each other until they are connected. Bi-RRT has been demonstrated to be very efficient in many highly constrained environments, such as high-dimensional spaces and narrow spaces. However, Bi-RRT is only suitable for the cases where the goal remains unchanged, and its planned paths are far from optimal.

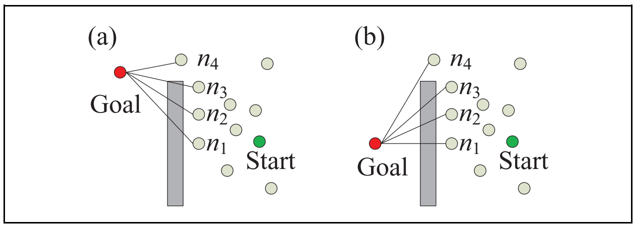

Figure 1. The diagram illustrates the reason for the local minimum problem in the path planning of manipulators: (a) no local minimum and (b) local minimum.

Asymptotically optimal RRTs

Recently, Rapidly-exploring Random Tree Star (RRT*) has been developed. It adjusts the parent–child relationship near the new nodes to reduce the path costs.[10] RRT* is shown to be asymptotically optimal in theory, namely the planned paths tend to be optimal with the increase of the planning time. However, RRT* consumes much memory and its convergence rate is slow. To overcome these limitations, Qureshi and Ayaz[16] proposed a Potential Function Based-RRT* (P-RRT*) that incorporates the artificial potential field algorithm in RRT*. Jeong et al.[17] improved the RRT* algorithm using the ancestor nodes to efficiently enlarge the pool of parent candidates. Combining the advantages of Bi-RRT and RRT* (Bi-RRT*) was also proved to converge faster to the optimal path solutions.[18,19] However, the speed of these algorithms is not fast enough, especially in high-dimensional spaces.

Heuristic RRTs

Unlike the RRT* algorithms, the heuristic RRTs reduce the randomness and the costs of the planned paths by changing the way of generating new nodes. Urmson and Simmons[11] developed a heuristically guided RRT (hRRT) algorithm. It incorporates heuristic functions to generate valid nodes and thus guides the tree toward low-cost solutions. Another heuristic search method is based on goal-biasing (Goal-Biasing RRT). It generally includes random extension and goal extension.[12,20,21] Which extension to be chosen is determined by a threshold within the range of 0 to 1. The random extension is the basic RRT algorithm for avoiding obstacles. In the goal extension, firstly, the node with the minimum distance to the goal configuration is selected in the search tree, then the tree continues to be extended from this nearest node toward the goal configuration until a collision is checked or the goal is reached. This greedy approach performs better in reducing randomness and time consumption. However, the local minimum problem may result in planning failure due to the heuristic searches and no memory, as shown in Figure 1.

As mentioned above, in the goal extension, the node closest to the goal is always selected in the search tree for extending to the goal rapidly. Based on this strategy, the desired path will be calculated by the goal-biasing RRT

```
MGB-RRT():
  SearchTree.init($q_{start}$);
  MemoryTree.init(NULL);
  for $i$=1 to $K$ do
      $p$ = RandomReal([0.0,1.0]);
      if($p$<$p_g$) do
          ExtendRandomly();
      else
          ExtendToGoal();
      end
  end

ExtendRandomly():
  $q_{rand}$ = Sample_Random();
  $q_{near}$ = Nearest_Neighbor($q_{rand}$,SearchTree);
  $q_{dir}$ = ($q_{rand}$-$q_{near}$)/||$q_{rand}$-$q_{near}$||;
  $q_{new}$ = $q_{near}$ +$L$*$q_{dir}$;
  if(Collision_Check($q_{new}$)) do
      SearchTree.add_node($q_{new}$)
      if(Norm($q_{goal}$ - $q_{new}$)<$error$) do
          return Reached;
      end
  end

ExtendToGoal():
  RestTree = SearchTree.remove(MemoryTree);
  $q_{old}$ = Nearest_Neighbor($q_{goal}$ ,RestTree);
  MemoryTree.add_node($q_{old}$);
  flag = true;
  while(flag) do
      $q_{dir}$ = ($q_{goal}$ - $q_{old}$)/||$q_{goal}$ - $q_{old}$||;
      $q_{new}$ = $q_{old}$ + $L$*$q_{dir}$;
      if(Collision_Check($q_{new}$)) do
          SearchTree.add_node($q_{new}$);
          MemoryTree.add_node($q_{new}$);
          $q_{old}$= $q_{new}$;
          if(Norm($q_{goal}$ - $q_{new}$ )<$error$) do
              Return Reached;
          end
      else
          flag = false;
      end
  end
```

Figure 2. The MGB-RRT algorithm. MGB-RRT: Memory-Goal-Biasing–Rapidly-exploring Random Tree.

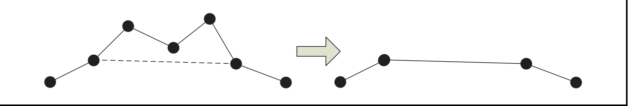

Figure 3. Removing the redundant nodes in the path.

approach in the case as shown in Figure 1(a), because the node (n_4) avoiding the obstacle is the nearest one to the goal. In Figure 1(b), the node (n_1) is always the nearest one, but the new nodes extended from this node are always in collision, resulting in the failure of the goal extension, that is to say, the planning becomes stuck in the local minimum.

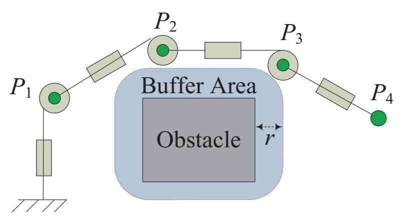

Figure 4. Fast collision detection for 7-DOF manipulator. DOF: degree of freedom.

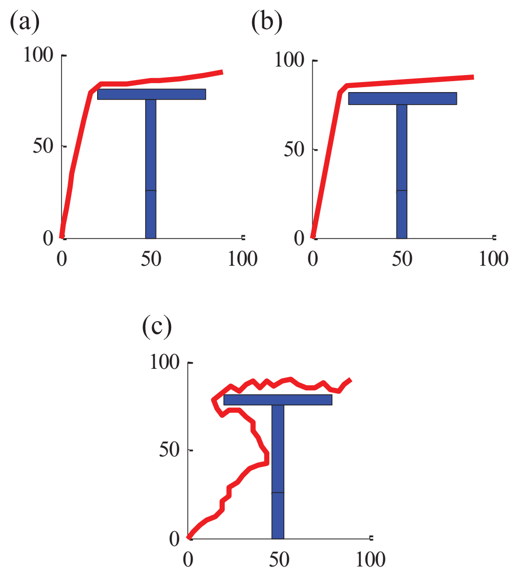

Figure 5. The planning results with (a) Bi-RRT*, (b) MGB-RRT, and (c) Bi-RRT. Bi-RRT*: bidirectional version of Rapidly-exploring Random Tree Star; MGB-RRT: Memory-Goal-Biasing–Rapidly-exploring Random Tree; Bi-RRT: bidirectional version of Rapidly-exploring Random Tree.

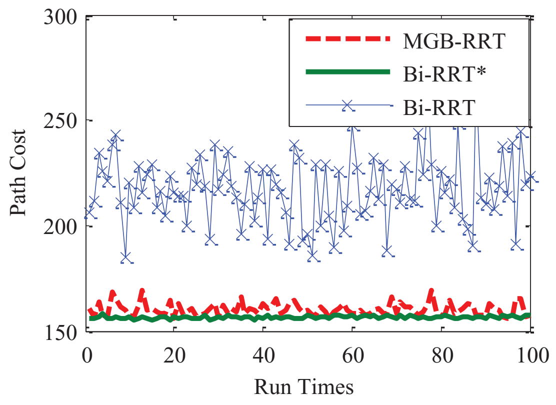

Figure 6. The path costs with Bi-RRT*, MGB-RRT, and Bi-RRT. Bi-RRT*: bidirectional version of Rapidly-exploring Random Tree Star; MGB-RRT: Memory-Goal-Biasing–Rapidly-exploring Random Tree; Bi-RRT: bidirectional version of Rapidly-exploring Random Tree.

Table 1. The planning results after 100 planning runs using Bi-RRT*, MGB-RRT, and Bi-RRT in two-dimensional space.

Approach	Path cost	StDev of path costs	Time (s)	Random extensions	Goal extensions	Collisions
Bi-RRT	217.4936	16.3317	0.02688	341	165	324
MGB-RRT	160.1395	2.9873	0.05807	177	288	161
Bi-RRT*	156.4534	0.6526	23.0	2000	1406	2219

Bi-RRT*: bidirectional version of Rapidly-exploring Random Tree Star; MGB-RRT: Memory-Goal-Biasing–Rapidly-exploring Random Tree; StDev: standard deviation; Bi-RRT: bidirectional version of Rapidly-exploring Random Tree.

Therefore, Kalisiak and van de Panne[22] introduced a non-regressing flood-fill mechanism into RRT (RRT-blossom) for escaping from the local minimum. This method needs to determine whether a new node is in the explored area per iteration, which needs to consume more time. Cheng and LaValle[23] developed a strategy based on recording the exploration information of each RRT node for excluding repeated states and solving the local minimum problem.

Unlike the literature,[22,23] MGB-RRT just memorizes the exploration information in the goal extension rather than in all extensions, because the probability of repeated exploration is very small in the random extension for high-dimensional manipulation planning. This improvement can make MGB-RRT take less planning time.

Baxter robot is used to test the performance of the proposed methods with MoveIt! software [version ROS], where Baxter is an industrial robot with two 7-DOF arms built by Rethink Robotics (Boston, Massachusetts),[24,25] and MoveIt! is the most widely used open-source software for motion planning, manipulation, kinematics, and other advanced robotics applications.[26] Furthermore, MoveIt! provides two efficient inverse kinematics (IK) solvers, which are KDLKinematicsPlugin and IKFastKinematicsPlugin. KDLKinematicsPlugin is a universal IK solver based on Newton–Raphson iterative method. IKFastKinematicsPlugin proposed by Diankov[27] can analytically and fast solve the kinematic equations of some complex kinematic chains. Both solvers can solve the IK of Baxter's 7-DOF manipulators for manipulation planning in this article.

MGB-RRT algorithm

Aiming at the possible local minimum problem of Goal-Biasing or other heuristic RRTs, the proposed MGB-RRT algorithm can record some special nodes in a separate memory so as to find an appropriate nearest node in the search tree excepting these special nodes. This is conducive to escaping from the local minimum. Consequently, determining which nodes to be memorized is the key and innovation of MGB-RRT.

As introduced earlier, during the goal extension, the new closest node to the goal is not expected to be one of the nodes generated by prior goal extension steps. Because if any goal-extended node is used for goal extension again, a collision will be detected anyway. So, all of the nodes extended to the goal should be memorized, and then the

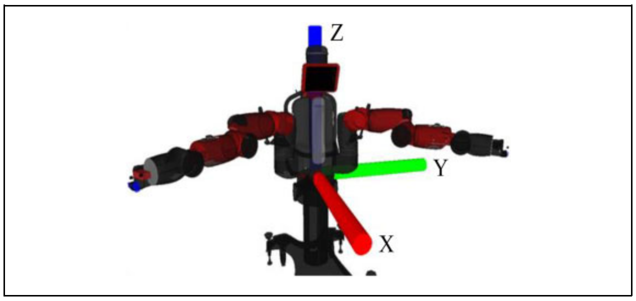

Figure 7. Baxter robot and its base coordinate system.

node nearest to the goal is found in the search tree without the memorized nodes in the next goal extension step.

The detailed MGB-RRT algorithm is shown in Figure 2. In this pseudocode, the search tree is grown until a newly added node is within the desired distance threshold of the goal or the maximum number K of iterations is exceeded. In the extension stage, the tree is grown randomly with a probability of p_g (line 6) and is grown to the goal with a probability of $1 - p_g$ (line 8). L represents the extension step. In the *ExtendToGoal* function, the node q_{old} with the shortest distance to the goal q_{goal} is found in *RestTree* rather than *SearchTree* (lines 21 and 22). This node is added to *MemoryTree* (line 23). Then, the tree is grown toward the goal q_{goal} step by step, and the new node q_{new} is also stored in *MemoryTree* (line 30) until a collision is checked or the goal q_{goal} is reached (lines 32 and 33).

After generating a feasible path, a post-processing method is implemented to further optimize the path, reduce the randomness of the path, and improve the motion stability of manipulators.

First, the redundant nodes in the path are removed. Two nodes are randomly chosen in the path according to a certain range. If there are no obstacles when the two nodes are connected directly, the nodes between the two are regarded as the redundant nodes that need to be removed, as shown in Figure 3. The path cost can be reduced to minimum after multiple loops.

Then, Bézier curve is adopted to smooth the above path for enhancing the stability of the manipulator's movement.

Collision detection

It is well known that collision detection will consume much time in the path planning of multi-DOF manipulators

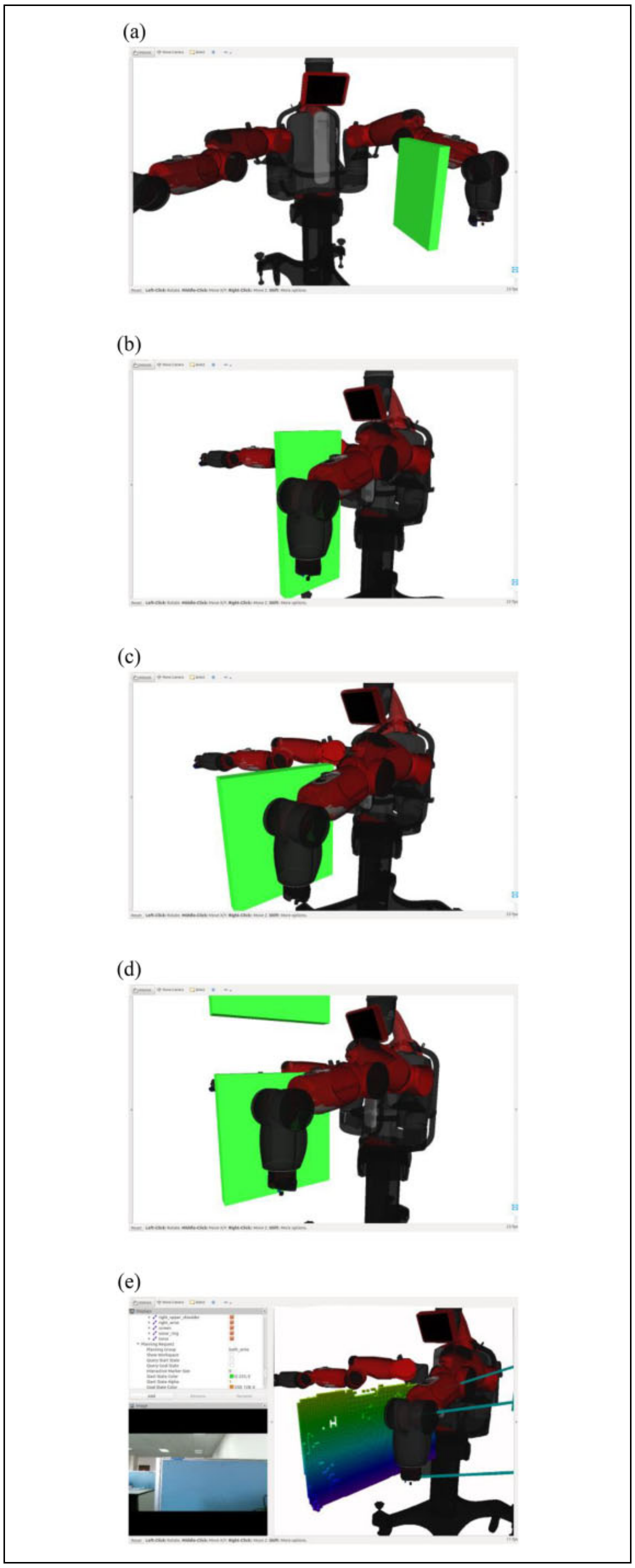

Figure 8. Different obstacles in the scenarios: (a) *Obstacle1,* (b) *Obstacle2,* (c) *Obstacle3,* (d) *Obstacle4,* and (e) *Obstacle5.*

Table 2. The parameters of the obstacles in the planning environments.

Scenario	Size (m)	Position (m)
Obstacle1	[0.27, 0.05, 0.45]	[0.70, 0.26, 0.22]
Obstacle2	[0.27, 0.05, 0.65]	[0.70, 0.26, 0.22]
Obstacle3	[0.42, 0.05, 0.45]	[0.78, 0.26, 0.22]

because of multiple links. In order to check collisions rapidly, Baxter's manipulator is simplified as multiple points, as shown in Figure 4. P_1, P_2, P_3, and P_4 are selected as the four key points of the manipulator. They are located in shoulder joint, elbow joint, wrist joint, and end-effector, respectively. Hence, the pose of the manipulator is uniquely determined by the positions of these four key points. In addition, more point positions can be calculated by interpolating between two key points (P_1 and P_2, P_2 and P_3, P_3 and P_4) for precise collision detection.

Forward kinematics is applied for getting the positions of P_1, P_2, P_3, and P_4. The Kinematics and Dynamics Library (KDL) provides a robotic framework for modeling and calculation of kinematic chains.[28] Therefore, the kinematic chains from Baxter's base to each key point are respectively represented by the objects of the KDL Chain class. The KDL solver (KDL::ChainFkSolverPos_recursive) is used to calculate the forward kinematics of these chains. This method makes it more convenient to solve forward kinematics.

Furthermore, in order to avoid obstacles successfully, the obstacle area is properly expanded according to the radius of the manipulator, as shown in Figure 4. Hence, we just need to judge whether the points are within the expanded obstacle area for collision detection. It can improve the efficiency of path planning and is one of the advantages of this article.

Results

The performance of the MGB-RRT algorithm was compared against Bi-RRT* (which tends to global optimum but has low efficiency) and Bi-RRT (which is fast but has poor optimization performance) over different planning tasks.

All of the tests were performed on an Intel(R) Core(TM) i7-3770 CPU. For each task, by performing 100 planning runs, we calculated the following values:

1. The average path cost (*Path Cost*). It is represented by Euler distance in joint configuration spaces.
2. The standard deviation of 100 path costs (*StDev of Path Costs*). It can describe the path randomness.
3. The average planning time (*Time*(*s*)). It includes the time consumption of post-processing for MGB-RRT.
4. The average number of random extensions (*Random Extensions*).
5. The average number of goal extensions (*Goal Extensions*). It should be explained that *Goal Extensions* in Bi-RRT and Bi-RRT* means the extension number of one tree grown to the other tree.[9]

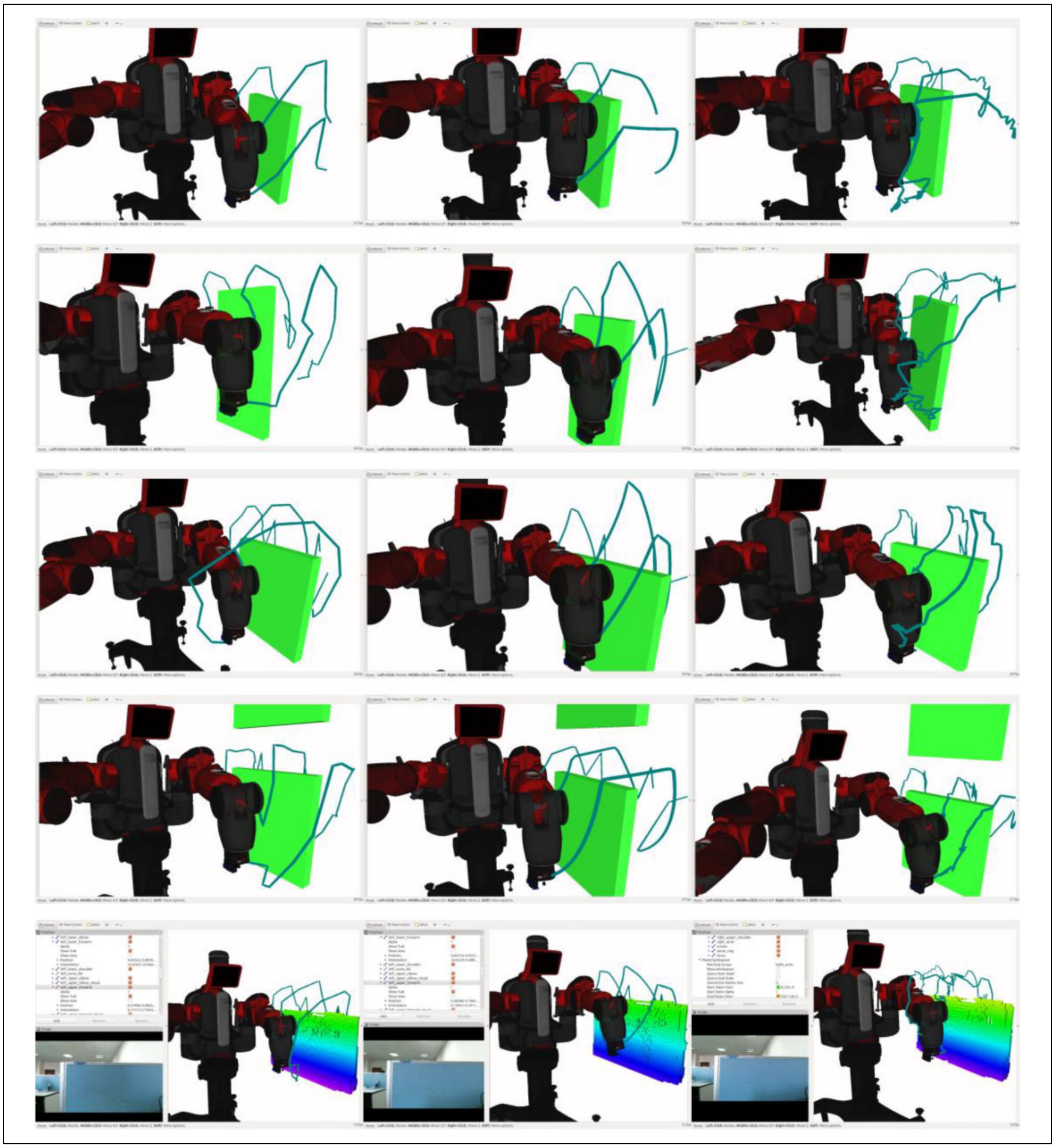

Figure 9. The planning results using Bi-RRT*, MGB-RRT, and Bi-RRT, among which the left are planned by Bi-RRT*, the middle are planned by MGB-RRT, and the right are planned by Bi-RRT. Bi-RRT*: bidirectional version of Rapidly-exploring Random Tree Star; MGB-RRT: Memory-Goal-Biasing–Rapidly-exploring Random Tree; Bi-RRT: bidirectional version of Rapidly-exploring Random Tree.

6. The average number of checking collisions (*Collisions*).

Planning in two-dimensional space

First of all, the planning algorithms were implemented in two-dimensional space using MATLAB [version 2014a] software. The extension step size and the error were set to 5, respectively. The start position was [0, 0] and the goal position was [90, 90]. The obstacle was *T*-shaped that could result in the local minimum problem. The results with different algorithms are shown in Figure 5.

In Figure 5, the path planned by MGB-RRT is very close to the optimal path planned by Bi-RRT*; however, Bi-RRT

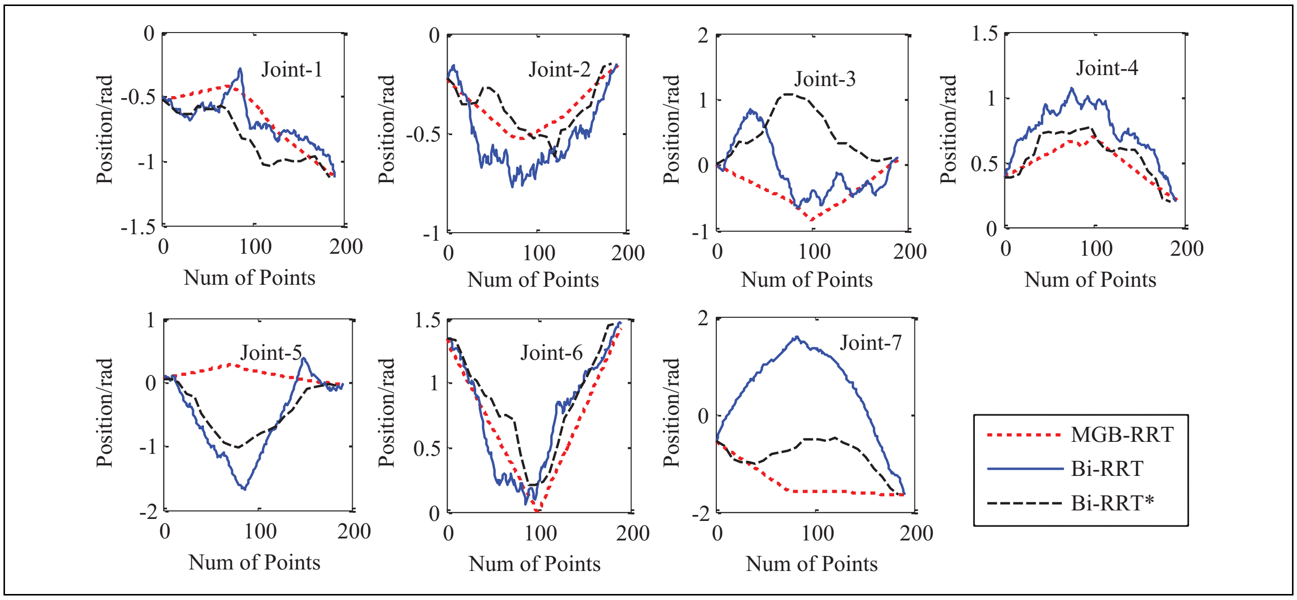

Figure 10. The joint trajectories for avoiding *Obstacle4* with MGB-RRT, Bi-RRT, and Bi-RRT*, respectively. Bi-RRT*: bidirectional version of Rapidly-exploring Random Tree Star; MGB-RRT: Memory-Goal-Biasing–Rapidly-exploring Random Tree; Bi-RRT: bidirectional version of Rapidly-exploring Random Tree.

performs poorly on optimization. Moreover, the path costs of 100 runs are drawn in Figure 6, where the path costs by Bi-RRT* are almost unchanged, and the path costs by MGB-RRT have a little change, however the path costs by Bi-RRT have much greater variations due to randomness.

The average results after 100 planning runs are shown in Table 1. Among the three methods, Bi-RRT* consumes the most time although its path cost and standard deviation are minimum; Bi-RRT achieves a relatively high planning rate, however its path cost and standard deviation indicate that the paths generated by Bi-RRT are far from the optimal solution and have large uncertainties; MGB-RRT realizes a satisfactory balance between the path cost and the planning time, which is significant for practical applications.

Planning for redundant manipulators

The performances of these three methods in high-dimensional spaces were compared over different planning scenarios involving the redundant manipulator of Baxter robot. The base coordinate system of Baxter robot is shown in Figure 7. The algorithms were implemented with Move-It! packages embedded in the Indigo version of Robot Operating System (ROS). ROS is a flexible robotics middleware for developing robot software.[29] The planning results were visualized by RViz, which is a three-dimensional visualization tool for ROS.

The initial position of the end-effector of Baxter's manipulator was [0.861, 0.485, 0.096] m in the base coordinate system. The initial orientation was represented by quaternion, which was [0.0471, −0.3779, 0.9243, 0.0249] ($q = d + ai + bj + ck$, $[d, a, b, c]$). The goal position was [0.855, 0.008, 0.107] m and the goal orientation was [0.0376, −0.6201, 0.7835, −0.0151]. Using the IK solvers provided by MoveIt!, we could respectively get the initial joint positions [−0.5245, −0.2454, 0.0011, 0.4120, 0.0553, 1.3122, −0.5411] rad and the goal joint positions [−1.1242, −0.1526, 0.0957, 0.1977, −0.0481, 1.4602, −1.6628] rad. Each joint error of reaching the goal was set as less than 0.034 rad.

In the following tests, we did not change the initial state and the goal state of the manipulator but the types of obstacles, so as to compare the performances of the planning algorithms aiming at different scenarios.

The obstacles in the scenarios are shown in Figure 8. *Obstacle1*, *Obstacle2*, and *Obstacle3* are different-sized rectangular solids. They are used to prove the capability of moving over large obstacles by MGB-RRT. The sizes and the positions of the three obstacles are shown in Table 2. *Obstacle4* is a narrow passage with a width of 0.18 m (the diameter of the manipulator is about 0.13 m). This obstacle is used to verify that MGB-RRT is able to plan a feasible path in narrow spaces. *Obstacle5* is the three-dimensional occupied map of a real obstacle obtained from Kinect v2 camera (Microsoft, US).[30] *Obstacle5* is used to demonstrate the practicability of MGB-RRT.

The planning results in these five scenarios are shown in Figure 9. The curves in Figure 9 are the trajectories of the three key points in the manipulator. These curves keep a certain distance with the obstacles. It means that each link of the manipulator avoided the obstacles and reached the goal state safely. Consequently, the proposed method of collision detection is proved to be reliable. Additionally,

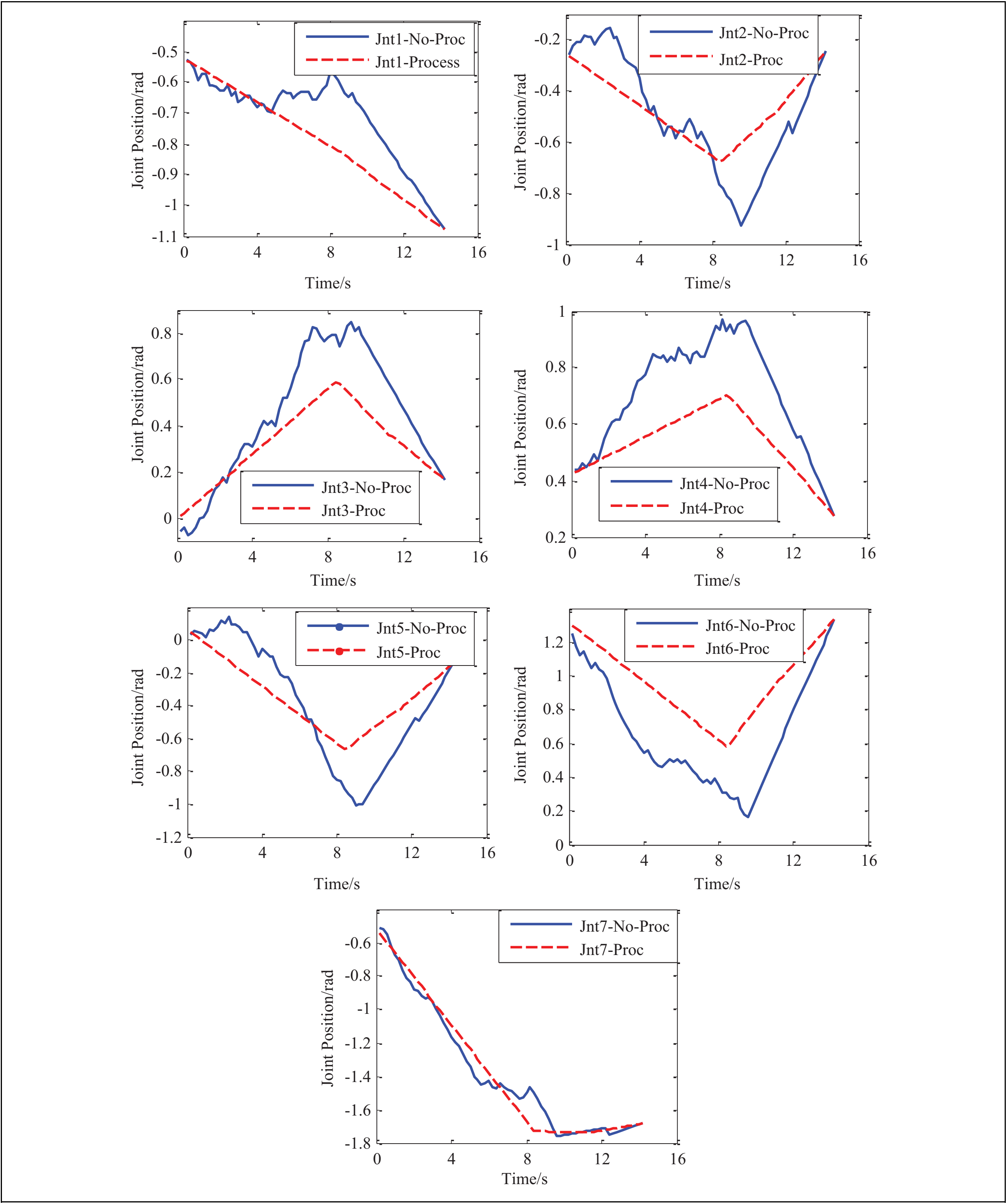

Figure 11. The comparison of the joint trajectories before and after the processing, Jnt*i*-No-Proc is the trajectory of *i*th joint before the processing, Jnt*i*-Proc is the trajectory of *i*th joint after the processing.

the joint trajectories for avoiding *Obstacle4* are used as an example to present the differences in the planned trajectories, as shown in Figure 10. It can be seen that the trajectories planned by MGB-RRT have fewer fluctuations and lower path costs compared with those by Bi-RRT and Bi-RRT*.

Table 3. The planning results after 100 planning runs using Bi-RRT*, MGB-RRT, and Bi-RRT for Baxter's manipulator.

Approach	Scenario	Path cost	StDev of path costs	Time (s)	Random extensions	Goal extensions	Collisions
Bi-RRT*	*Obstacle1*	3.6866	0.6997	125	12,000	10,678	3186
MGB-RRT	*Obstacle1*	3.3431	0.7085	0.5819	786	1215	395
Bi-RRT	*Obstacle1*	6.8795	2.0414	0.2285	521	383	486
Bi-RRT*	*Obstacle2*	5.4020	0.8038	110	12,000	10,059	5178
MGB-RRT	*Obstacle2*	4.5857	0.7109	1.7677	2109	3997	1160
Bi-RRT	*Obstacle2*	10.7210	1.9133	0.4906	1523	1034	1467
Bi-RRT*	*Obstacle3*	4.9906	0.8347	120	12,000	10,485	3925
MGB-RRT	*Obstacle3*	4.1487	0.6627	1.0453	1307	2388	698
Bi-RRT	*Obstacle3*	9.5445	1.9171	0.3759	1035	717	987
Bi-RRT*	*Obstacle4*	6.2865	0.8883	100	12,000	9462	6815
MGB-RRT	*Obstacle4*	4.8743	0.8283	2.4024	2466	5541	1478
Bi-RRT	*Obstacle4*	10.0835	2.0253	0.5542	1687	1108	1624
Bi-RRT*	*Obstacle5*	5.0794	0.8611	150	12,000	10,026	4536
MGB-RRT	*Obstacle5*	4.3457	0.7019	2.2981	1546	3297	848
Bi-RRT	*Obstacle5*	8.6764	1.8538	0.4673	771	547	719

Bi-RRT*: bidirectional version of Rapidly-exploring Random Tree Star; MGB-RRT: Memory-Goal-Biasing–Rapidly-exploring Random Tree; StDev: standard deviation; Bi-RRT: bidirectional version of Rapidly-exploring Random Tree.

In order to show the effects of the post-processing method in MGB-RRT, the comparison of the joint trajectories before and after the processing is shown in Figure 11. According to the deviation between the two types of curves in Figure 11, it is illustrated that by processing the initial planned paths (the solid curves in Figure 11), the path costs are further reduced, and the harmful random fluctuations are also eliminated, leading to more optimal paths (the broken curves in Figure 11). Therefore, the post-processing method is conducive to the stable and natural movement of manipulators.

The average results after 100 planning runs are shown in Table 3. They are somewhat different from the results in Table 1. In Table 3, the path cost and the corresponding standard deviation with MGB-RRT are more optimal than those with Bi-RRT*. It is because the given iteration times of Bi-RRT* are not enough to generate the optimal paths. Maybe we could increase the iteration times to make Bi-RRT* better than MGB-RRT in optimization performance, but it is not useful for the practical path planning of manipulators, because the planning time with Bi-RRT* is too long and the work environment may have been changed during the path planning. Furthermore, Bi-RRT still has advantage in the planning time over the other two algorithms, but the generated paths are too random to be directly applied to motion control. MGB-RRT has satisfactory performance on both path costs and planning time.

Discussion

As is well known, Bi-RRT* and Bi-RRT are two typical variants of RRT, which is also the reason that we choose the two methods to compare with MGB-RRT. The above planning results show that these two methods have some drawbacks:

1. The paths solved by Bi-RRT* are optimal or near-optimal, but the convergence rate of Bi-RRT* is pretty slow and unacceptable, especially for the path planning of redundant manipulators, which needs more than 100 s, as shown in Table 3.
2. Bi-RRT can quickly find a feasible solution for redundant manipulators. For example, it only takes about 0.5542 s for the narrow passage, but it has a poor performance on optimization, as shown in Tables 1 and 3.

One reason for these drawbacks should be that the tree growth has no clear goal and no memory, resulting in repeated and useless extensions. However, the proposed MGB-RRT algorithm adopts the following strategies to overcome the drawbacks:

1. A greedy strategy is adopted that extends the tree toward the goal continuously until a collision is detected. It provides a clear goal to avoid useless extensions and improve the efficiency of path planning.
2. A memory mechanism is used to overcome repeated extensions. It can memorize the exploration information in the goal extension and does not need complex computation in each iteration.

Therefore, in the planning process using MGB-RRT, the goal-extension times are more than the random-extension times, and the collision-checking times are relatively fewer, as shown in Tables 1 and 3. In addition, the post-processing method is implemented in MGB-RRT, which can reduce random fluctuations to obtain more optimal paths, as shown in Figures 10 and 11.

The above strategies make MGB-RRT get satisfactory optimization results (lower path costs and lower standard deviation) within an acceptable time (a few seconds), namely a better balance between optimization and efficiency is achieved. This is the advantage of MGB-RRT over Bi-RRT* and Bi-RRT.

Conclusion

In order to generate more optimal and faster solutions for the path planning of redundant manipulators, an improved MGB-RRT is proposed. The main features of this approach are to provide a memory mechanism and a post-processing method for generating the paths with satisfactory performances on optimization and efficiency. In addition, MGB-RRT adopts a fast collision-checking method. This method just needs to judge whether multiple specific points in the manipulator are within the enlarged obstacle areas.

The proposed MGB-RRT algorithm is compared with Bi-RRT* and Bi-RRT over a range of different planning scenarios using Baxter's redundant manipulator. The simulation planning results demonstrate the following:

1. Although the planning time with MGB-RRT is more than that with Bi-RRT, it should be acceptable in practice. The optimization performance of MGB-RRT is much better than that of Bi-RRT.
2. Within the acceptable time range, MGB-RRT has better optimization performance and lower computational complexity than Bi-RRT*.

Therefore, MGB-RRT is more applicable to the path planning of redundant manipulators in the actual industrial production.

MGB-RRT adopts a greedy strategy that extends the tree toward the goal until a collision is detected. However, if the goal extension can be terminated in advance according to the previous collision information, the extension times can be further decreased, resulting in higher planning efficiency. Therefore, the future work will aim at this idea to develop an intelligent MGB-RRT algorithm based on machine learning.

Acknowledgements

The authors express their sincere thanks to all the editors and the anonymous reviewers for their critical and constructive review of this article.

Declaration of conflicting interests

The authors declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This study was supported by the National Natural Science Foundation of China (No. 51675264 and No. 61773265), Open Foundation of Shanghai Key Laboratory of Spacecraft Mechanism, Funding of Jiangsu Innovation Program for Graduate Education (No. KYLX16_0388), and Fundamental Research Funds for the Central Universities.

ORCID iD

Dong Han http://orcid.org/0000-0002-0646-408X

References

1. Savkin AV and Hoy M. Reactive and the shortest path navigation of a wheeled mobile robot in cluttered environments. *Robotica* 2013; 31: 323–330.
2. Zang XZ, Yu WT, Zhang L, et al. Path planning based on Bi-RRT algorithm for redundant manipulator. In: *International conference on electrical, automation and mechanical engineering* (EAME 2015), Phuket, Thailand, 26–27 July 2015, pp. 189–191.
3. Kavraki LE, Svestka P, Latembe JC, et al. Probabilistic roadmaps for path planning in high-dimensional configuration space. *IEEE Trans Robot Autom* 1996; 12: 566–580.
4. LaValle SM. Rapidly-exploring random trees: a new tool for path planning. Report, Computer Science dept., Iowa State University, October 1998.
5. Dobson A and Bekris KE. Sparse roadmap spanners for asymptotically near-optimal motion planning. *Int J Robot Res* 2014; 33: 18–47.
6. Hsu D, Kindel R, Latombe JC, et al. Randomized kinodynamic motion planning with moving obstacles. *Int J Robot Res* 2002; 21: 233–255.
7. Janson L, Schmerling E, Clark A, et al. Fast marching tree: a fast marching sampling-based method for optimal motion planning in many dimensions. *Int J Robot Res* 2015; 34: 883–921.
8. Kalakrishnan M, Chitta S, Theodorou E, et al. STOMP: Stochastic trajectory optimization for motion planning. In: *IEEE international conference on robotics & automation*, Shanghai, China, 9–13 May 2011, pp. 4569–4574. IEEE.
9. Kuffner JJ and LaValle SM. RRT-connect: an efficient approach to single-query path planning. In: *IEEE international conference on robotics and automation*, San Francisco, USA, 24–28 April 2000, pp. 995–1001. IEEE.
10. Karaman S and Frazzoli E. Sampling-based algorithms for optimal motion planning. *Int J Robot Res* 2011; 30: 846–894.
11. Urmson C and Simmons R. Approaches for heuristically biasing RRT growth. In: *IEEE/RSJ international conference on intelligent robots and systems*, Las Vegas, USA, 27–31 October 2003, pp. 1178–1183. IEEE.
12. LaValle SM and Kuffner JJ. Rapidly-exploring random trees: progress and prospects. In: *Workshop on algorithmic foundations of robotics*, 2000.
13. Abbadi A. *Expert systems and advanced algorithms in mobile robots path planning*. PhD Thesis, Brno University of Technology, Czech Republic, 2015.
14. Viseras A, Losada RO, and Merino L. Planning with ants: efficient path planning with rapidly exploring random trees and ant colony optimization. *Int J Adv Robot Syst* 2016; 13: 1–16.

15. Yang L, Xiao J, Qi J, et al. GART: an environment-guided path planner for robots in crowded environments under kinodynamic constraints. *Int J Adv Robot Syst* 2016; 13: 1–18.
16. Qureshi AH and Ayaz Y. Potential functions based sampling heuristic for optimal path planning. *Auton Robot* 2015; 40: 1079–1093.
17. Jeong IB, Lee SJ, and Kim JH. RRT*-Quick: a motion planning algorithm with faster convergence rate. *Adv Int Syst Comput* 2015; 345: 67–76.
18. Qureshi AH and Ayaz Y. Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments. *Robot Auton Syst* 2015; 68: 1–11.
19. Shum A, Morris K, and Khajepour A. Direction-dependent optimal path planning for autonomous vehicles. *Robot Auton Syst* 2015; 70: 202–214.
20. Weghe MV, Ferguson D, and Srinivasa SS. Randomized path planning for redundant manipulators without inverse kinematics. In: *IEEE-RAS international conference on humanoid Robots*, Pittsburgh, USA, 29 November–1 December 2007, pp. 477–482. IEEE.
21. Vahrenkamp N, Berenson D, Asfour T, et al. Humanoid motion planning for dual-arm manipulation and re-grasping tasks. In: *IEEE/RSJ international conference on intelligent robots and systems*, St Louis, MO, USA, 10–15 October 2009, pp. 2464–2470. IEEE.
22. Kalisiak M and van de Panne M. RRT- blossom: RRT with a local flood-fill behavior. In: *IEEE international conference on robotics & automation*, Orlando, USA, 15–19 May 2006, pp. 1237–1242. IEEE.
23. Cheng P and LaValle SM. Reducing metric sensitivity in randomized trajectory design. In: *IEEE/RSJ international conference on intelligent robots and systems*, Maui, USA, 29 October–3 November 2001, pp. 43–48. IEEE.
24. Reddivari H, Yang C, Ju Z, et al. Teleoperation control of baxter robot using body motion tracking. In: *International conference on multisensor fusion and information integration for intelligent systems (MFI)*, Beijing, China, 28–29 September 2014, pp. 1–6. IEEE.
25. Ju Z, Yang C and Ma H. Kinematics modeling and experimental verification of baxter robot. In: *33rd Chinese control conference (CCC)*, Nanjing, China, 28–30 July 2014, pp. 8518–8523. IEEE.
26. Moveit! software. http://moveit.ros.org (accessed 5 June 2017).
27. Diankov R. *Automated construction of robotic manipulation programs*. PhD Thesis, Carnegie Mellon University, USA, 2010. http://www.programmingvision.com/rosendiankovthesis.pdf (accessed 3 May 2017).
28. Kinematics and Dynamics Library. http://www.orocos.org/kdl (accessed 10 January 2018).
29. Quigley M, Gerkey B, Conley K, et al. ROS: an open-source robot operating system. In: *ICRA workshop on open source software*, January 2009.
30. Hornung A, Wurm KM, Bennewitz M, et al. OctoMap: an efficient probabilistic 3D mapping framework based on octrees. *Auton Robot* 2013; 34(3):189–206.