

# RRT-GD: An Efficient Rapidly-exploring Random Tree Approach with Goal Directionality for Redundant Manipulator Path Planning

Junxiang Ge<sup>1</sup>, Fuchun Sun<sup>1</sup>, Chunfang Liu<sup>1</sup>

**Abstract**—Rapidly-exploring random trees (RRT) method performs well for obstacle-avoiding motion planning. However, it is not fast enough and less goal-direction. In this paper, we propose an effective rapidly-exploring random trees method with goal directionality (RRT-GD) for redundant manipulator path planning. The presented RRT-GD model concentrates both on the status of the end actuator and on the movements of the joints that the Newton-Raphson method is employed for inverse kinematics calculation. In order to speed up the path planning, the path searching algorithm is modified to be goal-directionality and the quintic polynomial is applied to smooth the planned path. In the experiments, it shows that RRT-GD tends to be 10 or more times faster than the usual RRT algorithm when doing the same obstacle-avoiding path planning task. Therefore, RRT-GD could be more efficient and useful in doing our daily works.

## I. INTRODUCTION

Rapidly-exploring random trees (RRT) method was proposed by SM Lavalley and JJ Kuffner [1], which is a suitable way to be used in obstacle-avoiding path planning for mobile robot or multiple freedom manipulator. It constructs a graph with obstacle free points in feasible space based on random sampling, in which the optimal path is searched from the initial point to the goal point. RRT method has the following superior characteristics comparing with other obstacle-avoiding methods:

- RRT method attends to explore unknown space.
- The graph constructed by RRT will gradually fill up the feasible space if the exploring times is enough.
- RRT method don't need precise modeling of the obstacles, instead, it only needs the information if or not a point is obstacle free through doing collision detect, thus ending up with the high efficiency of obstacle free planning.

Though it has the above advantages, RRT method will still perform low success rate of planning when there is many obstacles surrounded, or the distance from the initial point to the goal point is very far to explore. For these problems, JJ Kuffner and SM Lavalley [2, 3] proposed a bi-directional RRT algorithm (bi-RRT) that two trees were built based on the initial point and the goal point separately. When expanding each time, the tree of initial point will extend to the goal-point based tree, or the contrary way, until these two trees meets. Because of goal heuristic, this approach improves the

success rate of path planning. In spite of this modification, we still find it defective sometimes for path planning.

As mentioned before, RRT attends to explore unknown space, so that it could explore the whole free space. However, for goal-reaching works, it does not need to explore the whole space in the most time. What we need to do is to find a feasible way from the initial point to the goal point without obstacle collision, i.e., only a part of the whole space is necessary for exploration, which includes the initial point, the goal point and some obstacles. Thus, in this paper, we propose a modified RRT method with goal directionality (RRT-GD) for path planning with goal-directionality exploration. The experiments demonstrate that the proposed method could quickly accomplish the obstacle avoiding path planning and the planning speed is 10 times faster than that of the original RRT method.

On the other hand, for performing the redundant manipulator motion, it is necessary to transform each point in the path tree from the configuration space (C-space) to robots' joint space by using the inverse kinematics optimization. There are lots of methods to do inverse kinematics optimization for redundant manipulator, e.g., MLG (most likely gradient method) [4, 5], WLN (weighted least norm method) [6], Genetic Algorithm [7], GP (gradient projection method), etc. In this paper, Newton-Raphson method is adopted to perform the inverse kinematics optimization work. In order to make comparison, we also implement MLG method in our experiments according to [4, 5]. Compared to other methods, Newton-Raphson method has faster converging speed and more accurate result that it takes less than 10 iteration times (usually 5 or 6 times) and has deviations less than micro-meters. Finally, the quintic polynomial approach [8] is utilized to smooth the path from the initial point to the goal point with detailed status information (pose & joint angles).

The paper is organized as follows. In Section II, the proposed RRT-GD approach is illustrated in detail; Section III introduce the Newton-Raphson method in inverse kinematics optimization; then, the quintic polynomial method is showed for smoothing path; in the section IV, we demonstrate the experiments for verifying the effectiveness of the proposed method.

## II. RRT WITH EXTENDED STEPS

In this section, the original RRT method for obstacle-avoiding path planning is introduced firstly. Then, we show the presented RRT-GD approach for faster motion planning with goal directionality.

<sup>1</sup>State Key Lab of Intelligent Technology and Systems, Tsinghua National Laboratory for Information Science and Technology (TNList), Department of Computer Science and Technology, Tsinghua University. gejx14@mails.tsinghua.edu.cn, fcsun@tsinghua.edu.cn, cfliu1985@gmail.com

### A. Rapidly-exploring Random Trees

The basic problem of RRT is how could we move the manipulator from the initial status to the goal status without collision with the obstacles in the space? To make it more clearly, we make appointments on the following mathematical symbols to express status space (S-space) and configuration space (C-space).

We use  $n$  to stand for the degree of freedom (DOF) of the robot manipulator, which is 7 in our experiment. So S-space is expressed with 7 parameters from  $q_1$  to  $q_7$ , with the expression like (1):

$$S = (q_1, q_2, \dots, q_7) \quad (1)$$

As for the C-space, i.e., pose space, we use Euler angles that obeys Euler Z-X-Z transformation rules to express the azimuth angles, which displays like  $\gamma = (\psi, \theta, \phi)$ . Position writes like  $P = (x, y, z)$ . Get the position and Euler angles together, we define  $m$  as the dimension of pose, which is 6 in this way, and the expression of pose is expressed like (2):

$$X = (P, \gamma) = (x, y, z, \psi, \theta, \phi) \quad (2)$$

Note that  $n > m$  should be guaranteed since it is a redundant system. Therefore, we get the initial condition of collision-free path planning problem as this: knowing the initial joints' angle  $S_0 \in \mathbb{R}^n$ , the initial status of the end-effector  $X_0 \in \mathbb{R}^m$  (which could be computed from  $S_0$ ), and the goal status of the end-effector  $X_g \in \mathbb{R}^m$ . The relevant notations and functions of RRT are illustrated as below:

- $A_0$ : the whole pose space which can be reached by robot manipulator in C-space.
- $A_{free}$ : free space belong to  $A_0$ , i.e.,  $A_{free} \in A_0$ , and there is no obstacle in  $A_{free}$ .
- $A_{obs}$ : space of obstacle, i.e.,  $A_{obs} = A_0 \setminus A_{free}$ .
- $B_0, B_{free}, B_{obs}$ : these are the corresponding notations with A, but in the S-space. The subscripts take same meanings as in A.

Actually, it is difficult to build the accurate boundary of  $A_0$ . In convenient, we use  $X_{min}$  and  $X_{max}$  as the two terminal vertex of  $A_0$  here.

$$A_0 = \{X | X_{min} < X < X_{max}\}. \quad (1)$$

RRT method builds a path tree  $T = (V, E)$ , with the initial point  $X_0$  for pose and  $S_0$  for status as the first tree point, while  $V$  stands for all points and  $E$  is the relation between these points (father point and its child). The basic functions that RRT method use are defined as follow:

- *sample*: generate a random point  $X_{ram}$  in pose format within  $A_{free}$ .
- *distance*: compute the distance between two points in  $A_0$  by using (??).
- *nearest\_neighbor*: given the point X, this function returns the nearest point  $X_{near}$  to X in  $T$ .
- *steer*: given the random point  $X_{ram}$  and  $X_{near}$ , this function returns a new point  $X_{new}$  in  $A_{free}$ , which is nearer to  $X_{ram}$  than  $X_{near}$ .

- *collisionTest*: given point X, return 1 for obstacle free and 0 for obstacle collision.

Then, the basic algorithm of RRT is displayed as the pseudocode in Fig. 1.

```

RRT( $S_0, A_0$ )
1.  $V \leftarrow \{S_0\}; E \leftarrow \emptyset; i \leftarrow 0;$ 
2. while  $i < N$  do
3. {
4.    $T \leftarrow (V, E);$ 
5.    $X_{rand} \leftarrow \text{sample}(i); i \leftarrow i + 1;$ 
6.    $T \leftarrow \text{extend}(T, X_{rand});$ 
7. }

 $\text{extend}(T, X_{rand})$ 
1.  $X_{near} = \text{nearest\_neighbor}(T, X_{rand});$ 
2.  $X_{new} = \text{steer}(X_{near}, X_{rand});$ 
3. if  $X_{new}$  and  $\text{collisionTest}(X_{new})$ 
4.    $V \leftarrow V \cup \{X_{new}\};$ 
5.    $E \leftarrow E \cup \{(X_{near}, X_{new})\};$ 
5.   return success;
6. else
7.   return fail;

```

Fig. 1. The pseudocode of RRT algorithm.

In our algorithm, a 'rrtDistance' is employed for calculating the distance between points as (2). It is made to concentrate more on the accuracy of position than the accuracy of Euler azimuth angles.

$$\text{rrtDistance}(X_1, X_2) = \alpha \|P_1 - P_2\| + (1 - \alpha) \|\gamma_1 - \gamma_2\|, \quad 0 < \alpha < 1. \quad (2)$$

In our experiments, we choose  $\alpha > 0.5$ , e.g.,  $\alpha = 0.8$ . The *sample* function generates a point  $A_0$  that  $X_{min} < A_0 < X_{max}$ . Then, the random point  $X_{ram}$  is produced with the *sample* function as shown in (3).

$$X_{ram} = X_{min} + t \times (X_{max} - X_{min}), \quad 0 < t < 1; \quad (3)$$

Notice that  $t$  is randomly generated in the interval (0, 1) and the *sample* function would produce all the points of  $A_0$ . As for the *steer* function, a new point will be generated depending on  $X_{near}$  and  $X_{ram}$  as shown in (4):

$$X_{new} = X_{near} + s(X_{ram} - X_{near}) / \|X_{ram} - X_{near}\|, \quad 0 < s < \|X_{ram} - X_{near}\|. \quad (4)$$

We called  $s$  in (4) the step-length. In RRT algorithm, each time when *steer* function runs, the RRT tree  $T$  will attend to step  $s$  length forward in  $A_0$ . Since  $X_{rand}$  will be anywhere in  $A_0$ , it is inferred that the direction tree  $T$  extends is all-dimension, i.e., no certain direction, but different directions with different times. These properties of RRT method will be discussed in the next subsection.

### B. Modified RRT Approach

Though RRT can explore the whole space  $A_0$  to get an obstacle-free path to reach the goal point, it still has some shortage when it actually runs. Since RRT extends only one step (with step-length  $s$ ) further each time, it seems slow sometimes when  $A_0$  is big enough to explore. On the other hand, RRT tree is built beginning with the initial point, leaving out the goal point in the tree-built process. Still, there are other insufficiencies with RRT, which results in many modified RRT methods. Bi-RRT [2, 3] is one of them. Here, we briefly introduce this modified model.

Bi-RRT extends more steps until the new point generated is in collision with obstacles or reach the random point  $X_{rand}$ . And bi-RRT maintains two trees in its memory, one starting extend function from the initial point and the other from the goal point. The two trees extend to each other interchangeably every time until they meet in the space, which is the reason the name ‘bi-RRT’ comes from. With such modification, bi-RRT takes in account of both the initial point and the goal point together with a faster extending speed. Nevertheless, it still has not solve the problem of exploring useless space since the two trees will both explore the whole space.

In our model, multiple steps extending strategy is integrated into our algorithm. The multi-extend algorithm is written as shown in Fig. 2:

```

multi_extend( $T, X_{rand}$ )
1. [ $X_{new}, result$ ] = extend( $T, X_{rand}$ );
2. if result = fail or collisionTest( $X_{new}$ ) = 0
3.   return fail;
4. while result = success
5. {
6.   [ $X_{new}, result$ ] = extend( $X_{new}, X_{rand}$ );
7.   if collisionTest( $X_{new}$ ) = 0 or
           surpass( $X_{new}, X_{rand}$ )
8.     break;
9. }
10. return success.

```

Fig. 2. Multi-step extended pseudocode

The function ‘surpass( $X_{new}, X_{rand}$ )’ detects if  $X_{new}$  is near enough to  $X_{rand}$ . If the two points are close enough, then ‘true’ returns and ‘multi\_extend’ method will be stopped; Otherwise, the ‘false’ returns and the extending process continues. It could be deduced that, if the extending direction is correct, i.e., closer to the goal point, we could reach the goal point faster; however, if extends to a bad direction frequently, this extending strategy will then take more time than one-step extending strategy. We solve this problem by presenting a RRT-GD (RRT with goal-directionality) model.

### C. RRT-GD: Trun RRT to be goal-directionality

As mentioned before, both RRT and its modified methods (including bi-RRT) are easy to explore useless space

which do not need in the daily tasks, i.e., it lacks goal-directionality. In order to solve this problem, taking the normal task in our experiment circumstance into account, we turn our sights to the ‘sample’ function in RRT. Simple to find that if we just don’t use a whole space ‘sample’ method, but limit it to a useful space, we can decide the extending direction of RRT tree  $T$ , that is the kernel principle within our modified RRT method called RRT-GD.

Originally, ‘sample’ method is random in the whole work space, thus its extending direction is all-around. The sketch map can be seen in Fig. 3. When we use a space containing the goal point  $X_g$ , called sample space  $A_{sam}$ , which can simply be a sphere with  $X_g$  centering (of course other shapes also works, e.g., cuboid), the extending direction can be limit to goal-direction, as Fig. 3 shows. It can be seen from the sketch map that, if the initial point is contained in sample space, i.e.,  $X_0 \in A_{sam}$ , then the extending direction could also be all-around just as usual RRT does, but since the probability of the goal-direction becomes larger than the other directions, the principle of RRT-GD could also work.

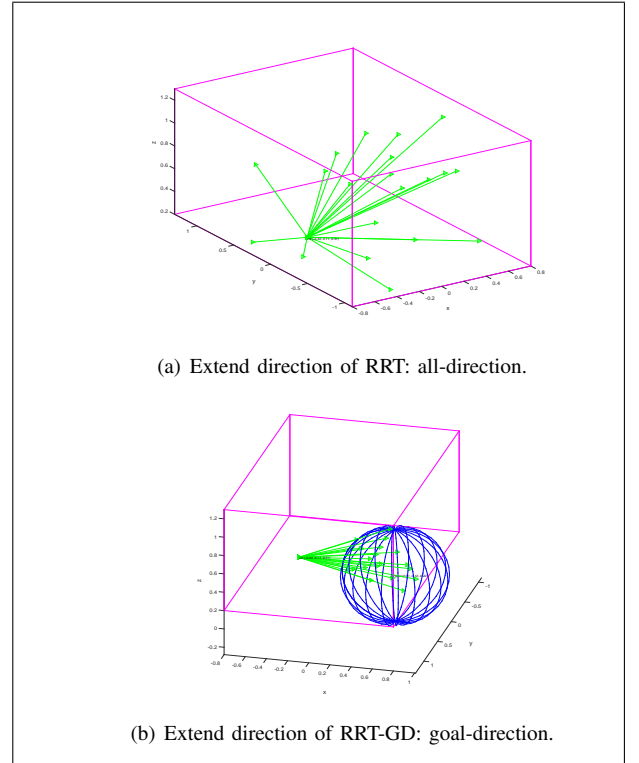


Fig. 3. Extend direction of RRT and RRT-GD. Pink cuboid is for the whole working space. Blue sphere is for sample space.

With this change, RRT-GD needs less extending times and the tree it build will go to the goal point rapidly. But on the other hand, it could also be easily seen that RRT-GD can’t get answer if all the extending direction of it can’t be explored further, e.g., a big obstacle object stands between the initial point and the goal point so that it resist all the extending directions RRT-GD could generate out. That is truly a problem in theory, but RRT-GD could work well in fact because of the following reasons:

- In fact, we could not get the accurate profile of the whole space  $A_0$  and  $A_{ram}$  most of the time, so the method of choosing a big range of  $A_0$  to satisfied our tasks itself is a RRT-GD method. In this way, RRT-GD is in the same with RRT, hence it will be all-direction exploring method and get the correct answer just as RRT will.
- As mentioned above, RRT-GD could also be all-direction if the sample space if  $A_{sam}$  include  $X_0$ . So RRT-GD could get the correct answer as RRT could.
- In fact, there will not be so many obstacles, and each obstacle will not be so big in the working space for our common tasks. As long as we make  $A_{sam}$  big enough, RRT-GD could always works correctly.

As we can see later in our exprements, RRT-GD could work faster and better than RRT most of time.

### III. INVERSE KINEMATICS WITH NEWTON-RAPHSON METHOD

This section focuses on the inverse kinematics optimization for redundant manipulator. There has been some inverse kinematics methods, e.g., MLG (most likely gradient method) [4, 5], WLN (weighted least norm method) [6], Genetic Algorithm [7], GP (gradient projection method), etc.. Because of high accuracy and speed, we utilize the Newton-Raphson method to perform the inverse kinematics optimization task.

#### A. Newton-Raphson method

Let's come back to our inverse kinematics problem: we know  $S_0$  (and so  $X_0$  can be computed) and  $X_g$ , how could we compute  $S_g$ ? Newton-Raphson method is an iteration approach that each time it just goes straight from the current point to the goal point.

$$S_{n+1} = S_n + J^+ \times minus(X_g, X_n). \quad (5)$$

The current status and pose are written as  $S_n$  and  $X_n$ , and  $J^+$  is the Jacobian matrix of current poin. Then when  $X_n$  is close enough to  $X_g$ , we get the answer  $S_{n+1}$  as result  $S_g$ . Note that though the function name called 'minus', it didn't just do simple minus operation as ' $X_g - X_n$ ', because the Euler azimuth doesn't supplement the minus operation. We should use the Rotate-By-1-Axis method to get the corret answer, since the combination of any fix-axis rotations of rigid body can be seen as one-time fix-axis rotation.

After this work, we take down the Newton-Raphson method as Fig. 4.

Here we use N (we set it to 10 in program) to limit the iteration times, for the fact that Newton-Raphson method would always get the correct answer in less than 10 iteration times with accuracy of micrometer, otherwise it would fail with bias greater than  $\epsilon$  (1 micrometer), i.e., more iteration times have no use cause it can't improve the accuracy further more.

```

Newton_Raphson( $S_0, X_g$ )
1. compute  $X_0$  and  $J^+$ ;  $i = 0$ ;  $times = 0$ ;
2. while  $\epsilon < rrtDistance(X_i, X_g)$  and  $times < N$ 
3. {
4.    $S_{i+1} = S_i + J^+ \times minus(X_g, X_i)$ ;
5.    $i = i + 1$ ;  $times = times + 1$ ;
6.   compute  $X_i$  and  $J^+$ ;
7. }

```

Fig. 4. Newton-Raphson method pseudocode

#### B. Quintic Polynomial with Path Smoothing

There comes another question after doing Newton-Raphson method: knowing  $S_0$  and  $S_g$ , how could we move the manipulator? The most direct answer might be directly adapt the joint angles to  $S_g$ , however, it might not be the optimal path in the thought of end manipulator. Thus, we need to smooth the path from  $S_0$  to  $S_g$ .

The quintic polynomial has been used in our program, the principle of which can be seen in [8]. We just use its conclusion. Supposed we know  $S_0$ ,  $S_g$ , and their distance fixed, we need to plan the medium path in 1 second (of course it can be changed by timing a factor). Then in quintic polynomial method, we plan the path as in (6):

$$S(t) = a_5 t^5 + a_4 t^4 + a_3 t^3 + a_2 t^2 + a_1 t + a_0; \quad \text{with : } \begin{cases} a_5 = 6(S_g - S_0); \\ a_4 = -15(S_g - S_0); \\ a_3 = 10(S_g - S_0); \\ a_2 = 0; \\ a_1 = 0; \\ a_0 = S_0; \end{cases} \quad (6)$$

We can see in (6) of the kinematics properties of its planning path. When  $t = 0s$ ,  $S(t)$  equals to  $S_0$ , that is the initial point; and when  $t = 1s$ ,  $S(t)$  equals to  $S_g$ , that is the goal point. And we can also compute the speed at each time in the path, just by doing differential operation. What we can deduce from (6) is that the speed of end manipulator when in initial point or goal point is 0. We could also change the coefficients from  $a_0$  to  $a_5$  according to [8] to fit different uses. These method works well on our 7-arm robot manipulator when doing exprements.

## IV. EXPERIMENTS

#### A. Experiment Environment

We use a 7-DOF (degree of freedom) robot manipulator, show in Fig. 5 to test the results in our experiment. Then we use the D-H method (Denavit and Hartenberg method) to model our robot arm, with the model axis built obeying D-H parameter method as the Fig. 6 show.

As a result of this axis coordinates, we get the DH parameters (show in TABLE I) from it, which is the key data used in our procedure.



Fig. 5. 7-DOF robot arm

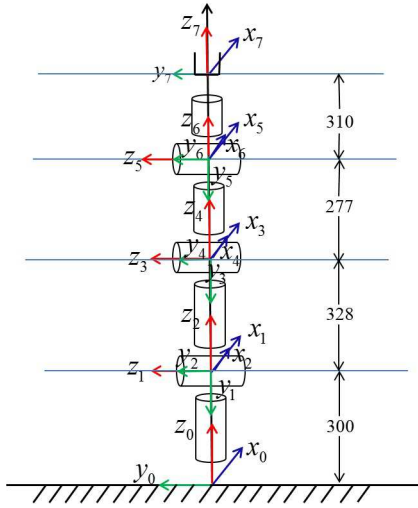


Fig. 6. Robot axis coordinates, obeying D-H parameter method. Numbers in the graph is in millimeter.

### B. Comparison between Newton-Raphson and MLG

At the beginning, we have complemented the MLG method before using Newton-Raphson method for inverse kinematics. Here, we introduce MLG method and compare it with Newton-Raphson method, to see the great properties of Newton-Raphson algorithm.

MLG method is also a iteration method, using the gradient of current point as the iteration direction, the detail of which can be seen in [4, 5]. We just make comparison between MLG and Newton-Raphson method.

We do comparison between these two methods using 4 groups of experiments. These tests start with the same initial point  $q_0$ , and so the same with  $X_0$ , and go to different end point appointed by  $X_g$ . Note that the iteration times of MLG method can be a input variable which can be assigned before MLG algorithm runs. So we take 5, 10, and 20 iteration times here as representatives. As TABLE II shows, in these groups of experiments, we make sense that:

- The time spent by Newton-Raphson method is almost the same with 5-iteration-times MLG method doing the

TABLE I

DH Parameters of robot. ( $\theta$  : joint angle;  $d$  : connecting rod skew;  $a$  : connecting rod;  $\alpha$  : tortuosity angle of connecting rod.)

Joint Number	$\theta_i$	$d_i(\text{mm})$	$a_i(\text{mm})$	$\alpha_i(^{\circ})$	area of $\theta_i(^{\circ})$
1	$q_1$	300	0	-90	-180~180
2	$q_2$	0	0	90	-90~90
3	$q_3$	328	0	-90	-180~180
4	$q_4$	0	0	90	-120~120
5	$q_5$	277	0	-90	-180~180
6	$q_6$	0	0	90	-120~120
7	$q_7$	310	0	0	-180~180

same task, sometimes even less than it.

- Newton-Raphson method uses less than 10 iteration times ending up with bias no more than  $10^{-6}m$ , but MLG method, no matter using 5, 10, or 20 iteration times, always ending up with deviation error higher than  $10^{-3}m$ , spending time in the same order of Newton-Raphson method.
- Newton-Raphson method adapts to a wider range of 'dist' change, i.e., it is suitified to more tasks when doing inverse kinematics. When 'dist' becomes bigger, Newton-Raphson works better than MLG method, that can be find in group 4, with Newton-Raphson method successful and MLG failed. Here, we limit the result bias to  $0.1m$ , i.e., when the accuracy is bigger than this number, we think it failed.
- When Newton-Raphson method fails in doing inverse kinematics, MLG method may always can't work well, because they both use the gradient of current point. Minded that whether the result fail or success doesn't depend on 'dist' parameter, it is decided by many reasons. Sometimes a small 'dist' even like  $0.02m$  may also result in failed both for Newton-Raphson method and MLG method.

One point which should be reminded is that we implement MLG method according to [5]. There exists some limitaion conditions with the end trajectory, e.g., the velocity of end manipulator can't be 0, which is set to faster than  $0.05m/s$  in our program. So more iteration times may not makes the result more accurate in MLG method. But the conclusion that MLG method always ends up with accuracy of  $10^{-3}m \sim 10^{-2}m$  when it successes is right, which has been tested by many times of experiments.

### C. Comparison between RRT-GD and RRT

We implement our 'sample' method with sample space in sphere shape of radius set to  $0.5m$  in RRT-GD. We can see the comparison between RRT and RRT-GD within iteration times and time property in TABLE III.

We take the same initial point as  $S_0$  in TABLE III, and let the end manipulator go to different goal points. The 'extending times' in table 3 is the total times of successful extending when finding the exact path to goal point, and 'failed extending times' means the total failed times when

TABLE II

Comparison between RRT and NEWTON-RAPHSON method.  $S_0 = [0.7854; 0.5236; 0; 0.5236; 0; 0.5236; 0](rad)$ ;  $X_0 = [0.5045; 0.5045; 0.7223; 2.3562; 1.5708; -1.5708]$ ;  $dist = rrtDistance(X_0, X_g)$ , all their units are meters(m). ‘ $\checkmark$ ’ means successful. ‘ $\times$ ’ means result failed, i.e., bias (accuracy) comes out more than 0.1m.

No.	task(m)	Method	iteration times	time spent( $10^{-3}s$ )	accuracy(m)	result
1	$X_g = [0.50; 0.45; 0.72; 2.35; 1.57; -1.57]$ $dist = 0.045$ .	N-R	5	6.15	$1.28 \times 10^{-9}$	$\checkmark$
			5	5.70	$6.50 \times 10^{-3}$	$\checkmark$
		MLG	10	6.52	$2.64 \times 10^{-3}$	$\checkmark$
			20	11.76	$1.25 \times 10^{-3}$	$\checkmark$
2	$X_g = [0.5; 0.48; 0.72; 2.35; 1.55; -1.55]$ $dist = 0.025$ .	N-R	4	2.38	$2.34 \times 10^{-7}$	$\checkmark$
			5	3.95	$9.74 \times 10^{-2}$	$\checkmark$
		MLG	10	6.60	$9.56 \times 10^{-2}$	$\checkmark$
			20	12.77	$9.39 \times 10^{-2}$	$\checkmark$
3	$X_g = [0.44; 0.44; 0.68; 2.30; 1.57; -1.57]$ $dist = 0.092$ .	N-R	7	3.87	$5.87 \times 10^{-8}$	$\checkmark$
			5	3.94	$1.65 \times 10^{-2}$	$\checkmark$
		MLG	10	6.64	$1.73 \times 10^{-2}$	$\checkmark$
			20	12.20	$1.78 \times 10^{-2}$	$\checkmark$
4	$X_g = [0.45; 0.55; 0.60; 2.00; 1.57; -1.57]$ $dist = 0.184$ .	N-R	9	5.70	$5.53 \times 10^{-10}$	$\checkmark$
			5	4.46	$1.37 \times 10^{-1}$	$\times$
		MLG	10	7.31	$1.22 \times 10^{-1}$	$\times$
			20	11.65	$1.12 \times 10^{-1}$	$\times$

TABLE III

Comparison between RRT and RRT-GD method.  $S_0 = [-0.2618; -0.2618; 0; -1.3090; 0; -1.3962; 0](rad)$ ;  $X_0 = [0.4011; 0.1075; 0.3115; -1.8326; 2.9671; 1.5708]$ ;  $dist = rrtDistance(X_0, X_g)$ , all their units are meters(m). ‘ $\times$ ’ means result failed.

No.	task(m)	Method	extending times	failed extending times	time spent(s)
1	$X_g = [0.42; -0.22; 0.22; -1.83; 2.97; -1.57]$ $dist = 0.7108$ .	RRT	808	8401	108.20
		RRT-GD	9	27	0.44
2	$X_g = [0.42; 0.22; 0.22; -1.83; 2.80; -1.50]$ $dist = 0.6668$ .	RRT	506	4250	47.39
		RRT-GD	3	1	0.16
3	$X_g = [0.51; 0.12; 0.22; -1.73; 2.90; -1.57]$ $dist = 0.7326$ ;	RRT	$\times$		
		RRT-GD	4	6	0.219

doing extending. The altogether iteration is their sum. One point which should be reminded is that the data in TABLE III is only one successful result among many tests, which is the best value within these tests (each group takes 10 tests). The corresponding planning path with TABLE III is show in Fig. 7. From Table III and Fig. 7, we can learn that:

- RRT-GD uses much less time and iteration times compared to RRT when doing the same work. As we can see, RRT-GD attends to be 100 times faster, speeding up the algorithm remarkably.
- RRT-GD can perform the path planning real-timely in these tasks, with planning time less than 0.5s, which is a great superior comparing to the traditional RRT method.
- RRT-GD can do the task that RRT can't, as in task 3 of TABLE III. Here, we set the iteration times limitation to 10000, surpassing which we think the algorithm failed. In task 3, we do RRT method for 10 times but none of them succeed, while RRT-GD could always succeed.
- Within our experiments, RRT-GD could succeed as long as RRT method succeed, i.e., the correctness of RRT-GD can be assured. This is because there are usually not so many obstacles in the working space, and each obstacle is usually small enough (no more than 0.1s in radius), while the radius of sample space is set to 0.5m,

that is big enough to do common works with obstacle-avoiding.

- Since RRT-GD uses less iteration times, so the planning path given by RRT-GD seems to be more directly, i.e., less zig-zag on the path, which can be seen in Fig. 7.

Note that we return the program immediately if it finds the planning path from initial point to goal point when doing the tasks in TABLE III. Of course, we could also set iteration times to a fix number, then use RRT-GD or RRT method to get multiple paths and get the best one as output. When doing these way, says iteration times limitation to 10000, RRT-GD can always get more paths out than RRT method.

One more to say, we can think of the two peaks of RRT-GD. One situation is when sample space fills up the whole working space, i.e.,  $A_{sam} = A_0$ . At this time, RRT-GD algorithm attends to be RRT method, i.e., they are the same when running. Another situation is that when sample space shrinks to only one point, i.e.,  $A_{sam} = \{X_{goal}\}$ , then the problem becomes a straight line path planning from the initial point to goal point. In fact, since we can't give the whole space an accuracy limitation, RRT method itself thus can't ensure to give a path even the path truly exists, i.e., RRT isn't even correct in theory on the account of actual use in this way. So RRT-GD is an important ideology which proves to be more pratical, press closer to the real circumstance and

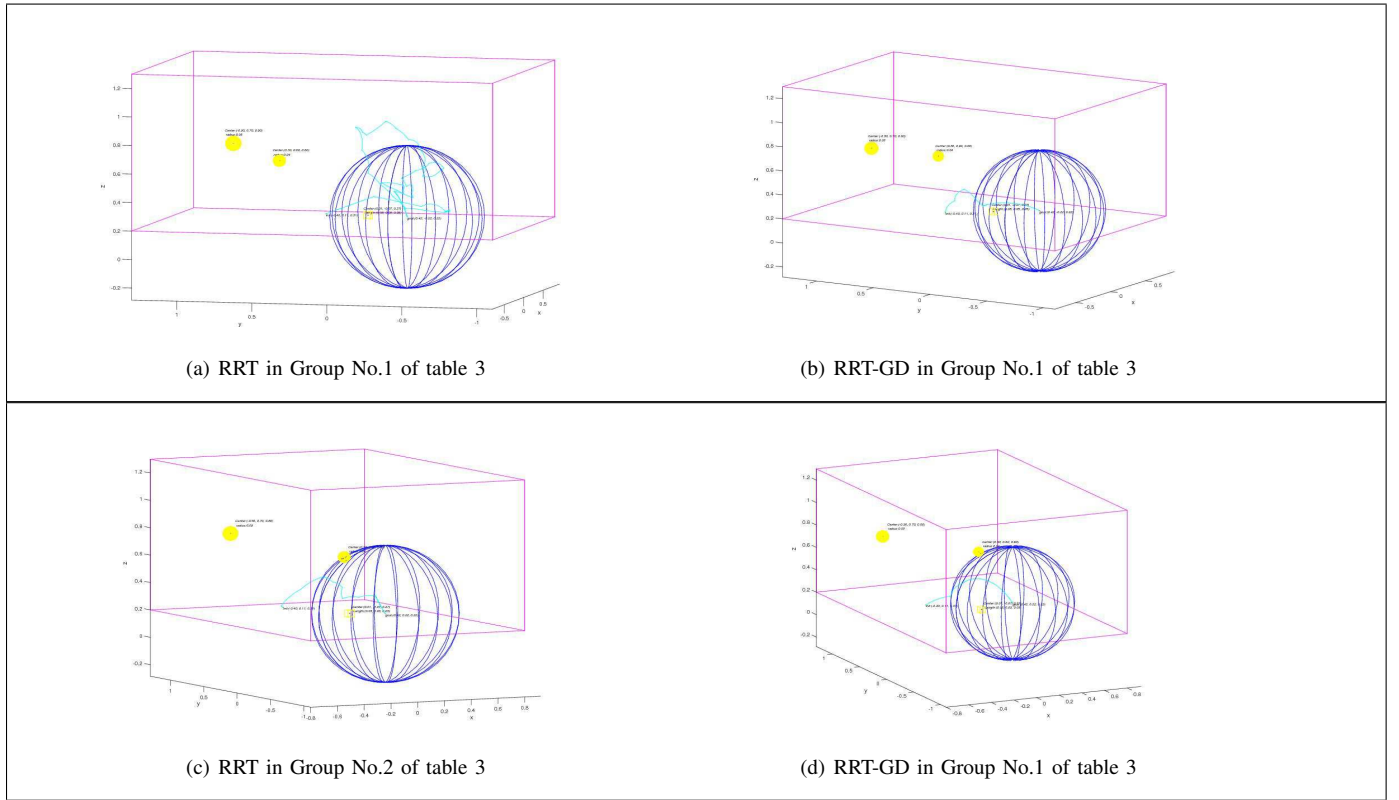


Fig. 7. Result comparison between RRT and RRT-GD. The blue sphere is the sample method sphere we use with goal point centering in red. The range of pink cuboid is our whole work space. The yellow sphere or cuboid is obstacle in the free space. The crayon line is the planning path starting from initial point to goal point.

daily common tasks.

## V. CONCLUSIONS

In this paper, we introduce a new method in obstacle-avoiding path planning, with modified RRT method doing obstacle-avoiding, Newton-Raphson doing inverse kinematics, and quintic polynomial doing path smoothing. After that, we find the less goal-direction with RRT and its modified algorithms, so we implement RRT-GD to solve this problem. With RRT-GD, we can accomplish daily works faster and more efficient, usually 10~100 times faster than the usual RRT method because of its goal-directionality property. Note that when the sample space of RRT-GD is not full of the whole working space, RRT-GD may not give a right planning path as RRT in theory, i.e., it is not a universal relevance method. However, RRT-GD works well in practice, as we can see in the experiments, for that in our common working space, there are usually not so many obstacles and the size of each obstacle are usually small enough for RRT-GD to work on well. In the future, we will make improvements on the shortcomings of RRT-GD without verifying the sample space.

## REFERENCES

- [1] Rapidly-Exploring Random Trees: Progress and Prospects, SM Lavalle, JJ Kuffner, Algorithmic & Computational Robotics New Directions, 2000:293–308.
- [2] RRT-connect: An efficient approach to single-query path planning, JJ Kuffner, SM Lavalle, Proc IEEE Intl Conf on Robotics & Automation, 2000, 2:995 - 1001.
- [3] Research on Kinematics and Obstacle Avoidance Path Planning for Redundant Manipulator, Yin Bin, Shi Shicai, Master Thesis, Harbin Institute of Technology, 2014. Classified Index:TP242.2, U.D.C:681.5.
- [4] Kinect-based Service Robot for Desktop Cleaning, Zheng Han, Meng Gao, ShiJiaZhuang Tiedao University, Master Thesis, 2013.
- [5] Planning Method Considering the Kinetic Characteristics of the End Effector of a Redundant Manipulator, Wenbing Huang, Fuchun Sun, Huaping Liu, Qinghua Daxue Xuebao/journal of Tsinghua University, 2014,54(12):1544-1548.
- [6] An Optimization Method for Inverse Kinematics of a 7-DOF Redundant Manipulator, Wenbin Yan, Lei Sun, Control Conference, 2015.
- [7] Algorithm Based on Analytical Method and Genetic Algorithm for Inverse Kinematics of Redundant Manipulator, Yun Dong, Tao Yang, Wen Li, Computer Simulation, 2012.
- [8] Trajectory Planning for a Robotic Ping-pong Player, Guowei Zhang, Bin Li, Huaibing Zhen, Haili Gong, Cong Wang, Chinese Journal of Scientific Instrument, Vol.32 No.6, Jun 2011.