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

Junxiang Ge[1], Fuchun Sun[1], Chunfang Liu[1]

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. 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 Lavalle 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. There are also many other intelligent path planning methods [9-11]. Compared with them, RRT method attends to explore unknown space so that the graph constructed by RRT will gradually fill up the feasible space as long as the exploring times is enough. Besides, 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 path 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 free space is too large to explore. For these problems, JJ Kuffner and SM Lavalle [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.

[1]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

As mentioned before, RRT attends to explore the whole space. However, for goal-reaching works, it doesn't 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. 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 likeyly 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 comparation, we also implement MLG method in our exprements 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 1 micrometers. Finally, the quintic polynomial approach [8] is utilized to smooth the planned path.

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.

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 exprement. So S-space is expressed with 7 parameters from q_1 to q_7, with the expression like (1):

$$S \;=\; (q_1, q_2,\; \dots\;, q_7) \tag{1}$$

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):

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

Note that $n > m$ should be guatanteed since it is a redundant system. Therefore, we get the initial condition of collsion-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., $A_{obs} = A_0 \backslash A_{free}$.

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

$$A_0 = \{X |\; X_{min} < X < X_{max}\}. \tag{3}$$

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. 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 (4).
- *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.

In our algorithm, an 'rrtDistance' method is employed for calculating the distance between points, show as (4). It is made to concentrate more on the accuracy of position than the accuracy of Euler azimuth angles.

$$\begin{aligned} rrtDistance(X_1, X_2) = \alpha||P_1 - P_2|| + \\ (1-\alpha)||\gamma_1 - \gamma_2||,\; 0 < \alpha < 1. \end{aligned} \tag{4}$$

In our exprements, 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 (5).

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

$extend(T, X_{rand})$
1. $X_{near} = nearest\_neighbor(T, X_{rand});$
2. $X_{new} = steer(X_{near}, X_{rand});$
3. $if\ X_{new}\ and\ collisionTest(X_{new})$
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.

$$X_{ram} = X_{min} + t \times (X_{max} - X_{min}), 0 < t < 1; \tag{5}$$

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 (6):

$$\begin{aligned} X_{new} = X_{near} + s(X_{ram} - X_{near})/||X_{ram} - \\ X_{near}||,\; 0 < s < ||X_{ram} - X_{near}||. \end{aligned} \tag{6}$$

We called s in (6) the step-length. In RRT algorithm, each time when *steer* function runs, the RRT tree T will attend s length forward. Since X_{rand} will be anywhere in A_0, it is inferred that the direction tree T extends to is all-dimension. 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 large 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 insufficients 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_{ram}. 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 hasn't solved the problem of exploring unuseful 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 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

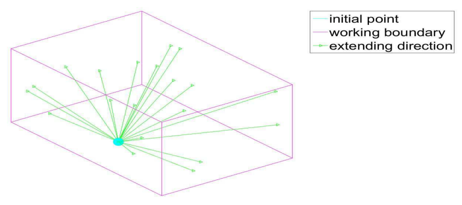
The function '$surpass(X_{new}, X_{rand})$' detects if X_{new} is near enough to X_{rand}. If the two points are close enough, then 'true' returns and 'multi_extend' method will be stopped; otherwise, the 'false' returns and the extending process continues. It could be deduced that, if the extending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

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

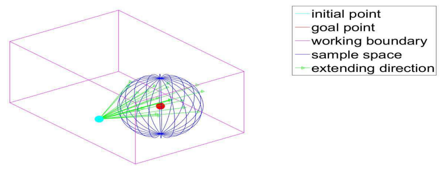
As mentioned before, both RRT and its modified methods (includeing bi-RRT) are easy to explore unuseful space which do not need in the daily tasks, i.e., it lacks goal-directionality. In order to solve this problem, taking the normal task in our experiment circumstance into account, we turn our sights 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. There exists another case: 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-directions becomes larger than the other directions, the principle of RRT-GD could also work.

With this change, RRT-GD needs less extending times and the tree it builds 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



(a) Extend direction of RRT: all-direction.



(b) Extend direction of RRT-GD: goal-direction.

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

extending directions RRT-GD could generate. 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 in RRT-GD method, i.e., RRT-GD is the same with RRT when we take the whole working space A_0 as sample space.
- As mentioned above, RRT-GD could also be all-direction if the sample space 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 daily tasks. As long as we make A_{sam} large enough, RRT-GD could always works correctly.

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

III. Inverse Kinematics with Newton-Raphson Method

Let's look on the 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 with the iteration in (7).

$$S_{n+1} = S_n + J^{+} \times minus(X_g, X_n). \tag{7}$$

The current status and pose are written as S_n and X_n, and J^{+} is the generalized inverse transformation of current Jacobian matrix. 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 write down the Newton-Raphson method as Fig. 4.

```
$Newton\_Raphson(S_0, X_g)$
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

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 ϵ (1 micrometer), i.e., more iteration times have no use cause it can't improve the accuracy further more.

The quintic polynomial has been used in our program to do path smoothing, the principle of which can be seen in [8]. Supposed we know S_0, S_g, 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 (8):

$$S(t) = a_5t^5 + a_4t^4 + a_3t^3 + a_2t^2 + a_1t + a_0;$$
$$with: \begin{cases} a_5 = & 6(S_g - S_0); \\ a_4 = & -15(S_g - S_0); \\ a_3 = & 10(S_g - S_0); \\ a_2 = & 0; \\ a_1 = & 0; \\ a_0 = & S_0; \end{cases} \tag{8}$$

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 shows.

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.

B. Comparation between Newton-Raphson and MLG

MLG (most likely gradient) 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 implement it and then make comparation between MLG and Newton-Raphson method.

Four groups of experiments has been done within this comparation. These tests start with the same initial point q_0, and so the same with X_0, and go to different goal point

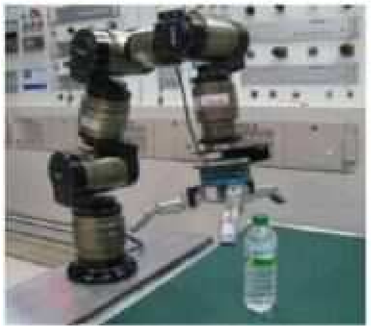


Fig. 5. 7-DOF robot arm

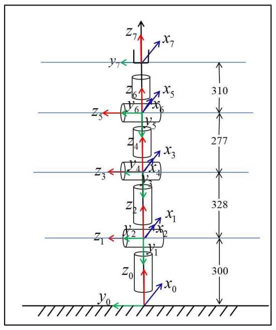


Fig. 6. Robot axis coordinates, obeying D-H parameter method. Numbers in the graph is in milimeter.

appointed by X_g. Note that the iteration times of MLG method is an input variable, so we take 5, 10, and 20 iteration times here as representatives. As Table II shows, in these groups of exprements, we make sense that:

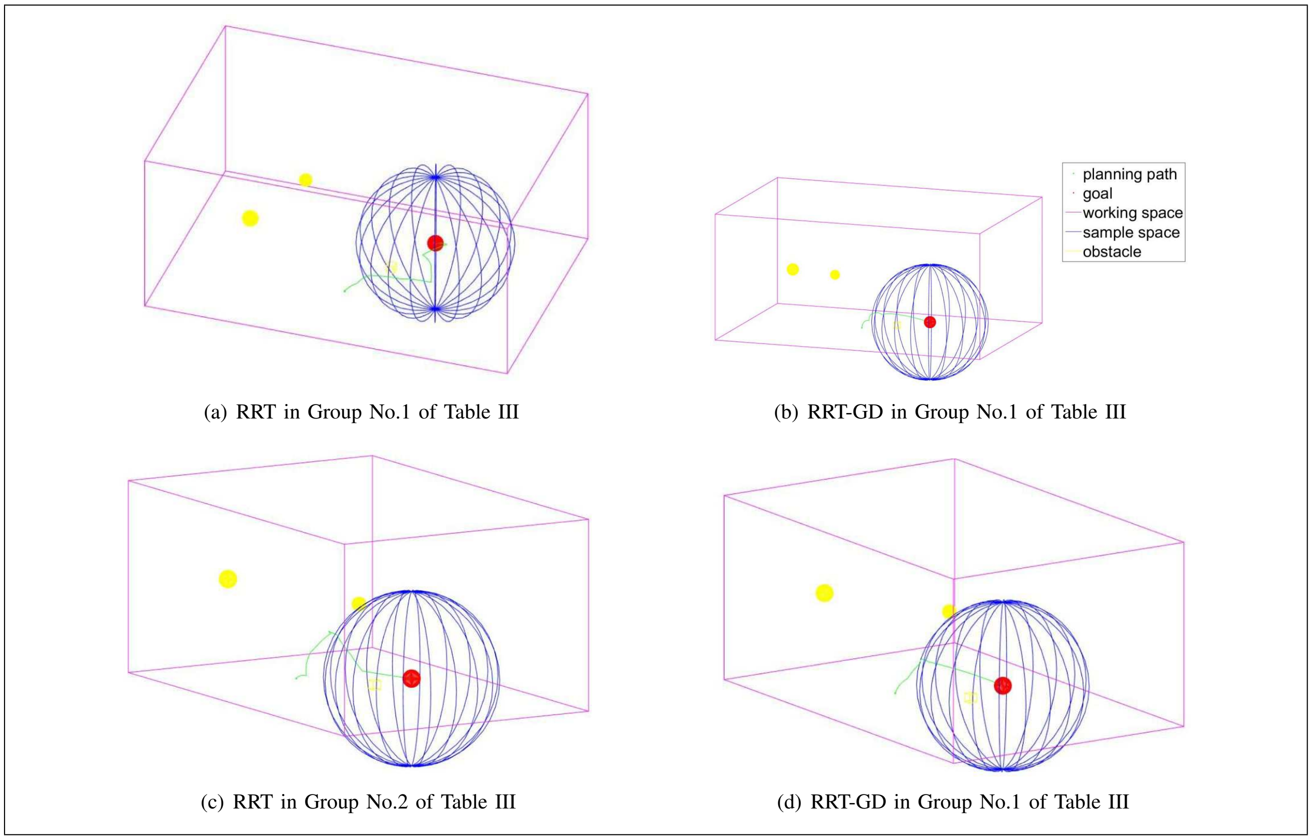
- The time spent by Newton-Rapshon method is in the same order of magnitude with 5-iteration-times MLG method.
- Newton-Raphson method usually ends up with bias no more than $10^{-6}m$, while MLG with deviation error higher than $10^{-3}m$.
- Newton-Raphson is suitified to more tasks when doing inverse kinematics, which can be proved in group 4, with Newton-Raphson method successful and MLG failed.
- When Newton-Raphson method fails, MLG method would also failed 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.

C. Comparation between RRT-GD and RRT

We implement our 'sample' method with sample space in sphere shape of radius set to $0.5m$ for RRT-GD. We can see the comparation between RRT and RRT-GD 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 total execution times for 'extend' method, on which time is mainly spent on, is the sum of 'successful extending times' and



(a) RRT in Group No.1 of Table III

(b) RRT-GD in Group No.1 of Table III

(c) RRT in Group No.2 of Table III

(d) RRT-GD in Group No.1 of Table III

Fig. 7. Result comparation between RRT and RRT-GD. The blue sphere is the sample space 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. The crayon line is the planned path starting from initial point to goal point.

TABLE I

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

Joint Number	θ_i	d_i(mm)	a_i(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

'failed extending times' in Table III. One point which should be reminded is that the data in a group in TABLE III is the best result within 10 tests (each group takes 10 tests). The corresponding planned path of group 1 and 2 in 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 10 times faster or more, 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 4 of Table III. Here, we set the iteration times limitation to 10000, surpassing which we think the algorithm failed. In task 4, 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. 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 planned path gived by RRT-GD seems to be better, i.e., less zig-zag on the path.
- Multi-steps extending strategy shows in Fig. 2 works on well with RRT-GD since the extending directions is always goal-direction, so that it can speed up the process to reach goal.

Note that we return the program immediately if it finds the planned path from initial point to goal point when doing the tasks in Table III. Of course, we could change it to get the best one as output in a certain iteration times. When doing these way, says iteration times limitation to 10000, RRT-GD can always get more paths out than RRT method, ending up with better planning path.

TABLE II

Comparation 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).'$\surd$' *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	1.28×10^{-9}	√
		MLG	5	5.70	6.50×10^{-3}	√
			10	6.52	2.64×10^{-3}	√
			20	11.76	1.25×10^{-3}	√
2	$X_g = [\ 0.5; 0.48; 0.72; 2.35; 1.55; -1.55\ ]$ $dist = 0.025.$	N-R	4	2.38	2.34×10^{-7}	√
		MLG	5	3.95	9.74×10^{-2}	√
			10	6.60	9.56×10^{-2}	√
			20	12.77	9.39×10^{-2}	√
3	$X_g = [\ 0.44; 0.44; 0.68; 2.30; 1.57; -1.57\ ]$ $dist = 0.092.$	N-R	7	3.87	5.87×10^{-8}	√
		MLG	5	3.94	1.65×10^{-2}	√
			10	6.64	1.73×10^{-2}	√
			20	12.20	1.78×10^{-2}	√
4	$X_g = [\ 0.45; 0.55; 0.60; 2.00; 1.57; -1.57\ ]$ $dist = 0.184.$	N-R	9	5.70	5.53×10^{-10}	√
		MLG	5	4.46	1.37×10^{-1}	×
			10	7.31	1.22×10^{-1}	×
			20	11.65	1.12×10^{-1}	×

TABLE III

Comparation 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	successful 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	426	3006	32.74
		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	$X_g = [\ 0.32; 0.02; 0.20; -1.80; 2.80; -1.57\ ]$ $dist = 0.5858;$	RRT	221	1181	12.66
		RRT-GD	3	12	0.220
4	$X_g = [\ 0.51; 0.12; 0.22; -1.73; 2.90; -1.57\ ]$ $dist = 0.7326;$	RRT	×		
		RRT-GD	4	6	0.219

V. CONCLUSIONS

In this paper, we introduce a new method in obstacle-avoiding path planning, with RRT-GD method doing obstacle-avoiding, Newton-Raphson doing inverse kinematics, and quintic polynomial doing path smoothing. 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 exprements, 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 varifying 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.

[9] A Subdivision Algorithm in Configuration for Findpath with Rotation, Rodney A. Brooks, Tomas Lozano-Perez, Systems Man & Cybemetics IEEE Transactions on, 1985, SMC-15(2):244-233.

[10] New Heuristic Algorithms for Efficient Hierarchical Path Planning, David Zhu and Jean-Claude Latombe, IEEE Transactions on Robotics & Automation, 1991.7(1):9-20.

[11] A Methodology for Intelligent Path Planning, Suman Chakravorty, John L. Junkins, Proceedings of the 2005 IEEE International Symposium on Mediterrean Conference on Control and Automation, 10.1109/.2005.1467081, 13 March 2006.