Computer Systems Science & Engineering
DOI:10.32604/csse.2021.016273
Article

Tech Science Press

An Improved Q-RRT* Algorithm Based on Virtual Light

Chengchen Zhuge[1,2,3,*], Qun Wang[1,2,3], Jiayin Liu[1,2,3] and Lingxiang Yao[4]

[1]Department of Computer Information and Cyber Security, Jiangsu Police Insitute, Nanjing, 210031, China
[2]Jiangsu Electronic Data Forensics and Analysis Engineering Research Center, Jiangsu Police Insitute, Nanjing, 210031, China
[3]Jiangsu Provincial Public Security Department Key Laboratory of Digital Forensics, Jiangsu Police Insitute, Nanjing, 210031, China
[4]University of Technology Sydney, Sydney, 2007, Australia
[*]Corresponding Author: Chengchen Zhuge. Email: zgcc1986@163.com
Received: 29 December 2020; Accepted: 28 February 2021

Abstract: The Rapidly-exploring Random Tree (RRT) algorithm is an efficient path-planning algorithm based on random sampling. The RRT* algorithm is a variant of the RRT algorithm that can achieve convergence to the optimal solution. However, it has been proven to take an infinite time to do so. An improved Quick-RRT* (Q-RRT*) algorithm based on a virtual light source is proposed in this paper to overcome this problem. The virtual light-based Q-RRT* (LQ-RRT*) takes advantage of the heuristic information generated by the virtual light on the map. In this way, the tree can find the initial solution quickly. Next, the LQ-RRT* algorithm combines the heuristic information with the optimization capability of the Q-RRT* algorithm to find the approximate optimal solution. LQ-RRT* further optimizes the sampling space compared with the Q-RRT* algorithm and improves the sampling efficiency. The efficiency of the algorithm is verified by comparison experiments in different simulation environments. The results show that the proposed algorithm can converge to the approximate optimal solution in less time and with lower memory consumption.

Keywords: Path planning; RRT*; Q-RRT*; LQ-RRT*; virtual light

1 Introduction

Path planning is a thriving research area in the field of mobile robots. It mainly involves how to find a feasible path connecting the start point to the goal point for the mobile robot when some indicators are satisfied. An increasing demand exists for intelligent mobile robots in industry, aerospace, and the military. In these areas, the mobile robot needs to complete complex tasks such as search and rescue, reconnaissance, and other activities. The working environment may be harmful to human health including conditions of strong radiation or high temperature and hypoxia. Various types of mobile robots, such as unmanned ground vehicles [1], unmanned aerial vehicles [2,3], and surface/underwater vehicles [4,5], have been designed and developed. Many path-planning algorithms have been proposed to make the mobile robot move safely in a complex environment.

Path-planning algorithms can be divided into two categories: one is based on a deterministic graph, and the other is based on random sampling. The grid-based path-planning algorithm is commonly used in deterministic graphics algorithms such as A* [6], D* [7], and their variants. In this kind of algorithm, the workspace of the mobile robot is divided into many grids. The algorithm generates a path by searching the grids that are not occupied by obstacles. This kind of algorithm can ensure the completeness of the solution, but the result of the path is affected by the resolution of the grid. The higher-resolution grid can represent a more precise environment and planning path. However, it leads to lengthy search times and large demands on memory space. There is another problem in the grid-based path-planning algorithm, namely that the generated path cannot be guaranteed smooth for mobile robots since the workspace is discretized. Some improved algorithms, for example, Theta* [8], have been proposed, but they are still difficult to apply directly to mobile robots with non-holonomic constraints.

In the random sampling-based algorithm, the planner randomly samples a state point in the workspace each time and detects the collision to obtain the obstacle information in the workspace. This kind of algorithm does not need to accurately model the workspace. Therefore, compared with the grid-based algorithm, the sampling-based path planning algorithm can also work effectively in the complex high-dimensional space. However, the sampling-based algorithm can only guarantee that it is probabilistic completeness. In other words, the solution of the problem can be found when the sampling number approaches infinity.

The classical Rapidly-exploring Random Tree (RRT) algorithm was proposed by LaValle [9]. RRT has received extensive attention, and various improved RRT algorithms have been proposed. The RRT-Connect [10] algorithm constructs two trees: one roots at the start point and the other roots at the goal point. Each tree grows towards the other one based on the greedy strategy. The algorithm is terminated when the two trees are connected. Earlier work [11] introduced the heuristic function to guide the tree grow towards the goal with the lowest possible cost. The results of experiments show that the efficiency of searching space can be effectively improved by introducing heuristic information.

The obvious disadvantage of the RRT algorithm is that the generated path is random due to the stochastic characteristic of RRT. This means that it cannot guarantee to find the optimal path in a limited time. Some algorithms have been proposed to improve the quality of the resulting path, such as those described in previous studies [12,13]. The Anytime RRT [12] algorithm generates the initial path first. Next, the cost of the initial path is used as the upper bound of the next iteration search. In the next iteration search, each sample state is evaluated. If the cost of the state is lower than the upper bound, the state will be added to the tree. The result will be improved in each iteration search for as long as the remaining time allows. Further work [13] introduced the cache point to the Anytime RRT algorithm. The cache points are chosen from the path generated in the last iteration and reused in the new iteration, which can improve the growth efficiency of the tree. However, these algorithms still cannot guarantee to find the approximate optimal solution in finite time.

RRT* [14] was proposed by Karaman and Frazzoli in 2011. It optimizes the connection of the tree by introducing two new functions. The two functions are named ChooseParent and Rewire. When the sampling times tend to be infinite, the probability of finding the optimal solution approaches one. This was proved in earlier work [14]. The obvious disadvantage of RRT* is that it needs a lot of time to converge to the optimal or approximately optimal solution. Many variants of RRT* have been proposed to accelerate the convergence and overcome this problem. RRT*-smart [15] optimizes the path according to the triangular inequality and visibility graph. After the initial solution is found, some points called beacons are chosen from the initial solution to improve the solution in the subsequent iterations. Informed-RRT* [16] introduced an elliptical heuristic sampling domain to limit the sampling space. The sampling domain gradually decreases with the convergence of the solution. Informed-RRT* increases the convergence rate significantly if the volume of the elliptical domain is smaller than that of the configuration space [17]. In other words, if the elliptical domain is larger than the configuration space, the algorithm is no longer applicable. Jeong et al. [17] modified the ChooseParent and Rewire functions and proposed the Quick-RRT* (Q-RRT*) algorithm. In ChooseParent and Rewire, the parent vertices are also taken into account for connecting and reconnecting the branches of the tree. The results of experiments show that the Q-RRT* algorithm increases the convergence rate without increasing the computation cost significantly.

In this paper, an improved Q-RRT* algorithm based on virtual light (LQ-RRT*) is proposed. The virtual light, which is set at the goal, lights up the map, and the light intensity decreases with increasing distance from the virtual light. The tree can search for a solution towards the region with increasing light intensity quickly using the characteristics of light intensity attenuation. The LQ-RRT* algorithm can find the initial solution faster than Q-RRT*. Meanwhile, it can converge to the approximate optimal solution faster and with lower consumption of memory.

2 Background

2.1 Path Planning Problem Definition

Let $X \in \mathbb{R}^n$ be the configuration space of the planning problem, where $n \in N$ and $n \geq 2$. Let X_{obs} and X_{goal}, the obstacle region and the goal region respectively, be the subset of X. Let $X_{free} = X \backslash X_{obs}$ define the collision-free space. Let $x_{start} \in X_{free}$ be the start point. Given a set X, a path in X is a continuous function $\sigma : [0, 1] \rightarrow X$, and Σ is the set of all feasible paths. A path σ is collision-free if $\forall\tau \in [0, 1]$, $\sigma(\tau) \in X_{free}$. The path-planning problem is to find a path σ such that σ is collision-free: $\sigma(0) = x_{start}$ and $\sigma(1) \in X_{goal}$.

The optimal path planning problem is to find a path, $\sigma*$, that minimizes a given cost function, $c : \Sigma \rightarrow \mathbb{R} \geq 0$, such that:

$$\sigma* = \underset{\sigma\in\Sigma}{\arg\min}\{c(\sigma)|\sigma(0) = x_{start}, \sigma(1) \in X_{goal}, \forall\tau \in [0, 1], \sigma(\tau) \in X_{free}\} \tag{1}$$

where $\mathbb{R} \geq 0$ is the set of the non-negative real numbers [11].

2.2 RRT Algorithm*

The main pseudo-code of the RRT* algorithm is presented in Algorithm 1. The RRT* algorithm begins with a tree rooted at x_{start} and extends the tree incrementally like the RRT algorithm. The basic procedures are similar to the RRT algorithm used by RRT* and are described as follows:

Algorithm 1: RRT*(V, E)

```
1:  $T \leftarrow (V, E)$;
2:  while $NotReachStop$ do
3:      $x_{rand} \leftarrow Sample()$;
4:      $x_{nearst} \leftarrow Nearest(T, x_{rand})$;
5:      $(x_{new}, \sigma) \leftarrow Steer(x_{nearst}, x_{rand})$;
6:      if $CollisionFree(\sigma)$ then
7:          $X_{near} \leftarrow Near(T, x_{new})$;
8:          $x_{parent} \leftarrow ChooseParent(X_{near}, x_{nearst}, x_{new}, \sigma)$;
9:          $T \leftarrow Connect(T, x_{parent}, x_{new})$;
10:         $T \leftarrow Rewire(T, x_{new}, X_{near})$;
11:     end if
12: end while
13: return $T$;
```

Sample: The *Sample* function samples a state $x_{\text{rand}} \in X_{\text{free}}$ randomly.

Nearest(T, x): Given tree T and a state $x \in X_{\text{free}}$, this function returns $x_{\text{nearest}} \in T.V$, which is the closest one to x in terms of a given distance function.

Steer(x, x'): Given two states $x, x' \in X$, the *Steer* function steers from x to x' along a local path $\sigma : [0,1]$ and returns a state $x_{\text{new}} \in X$ such that $\sigma(0) = x$ and $\sigma(1) = x_{\text{new}}$.

CollisionFree(x, x'): Given two states $x, x' \in X$, the Boolean *CollisionFree* function returns true if the path $\sigma : [0,1] \rightarrow X_{\text{free}}$ such that $\sigma(0) = x$ and $\sigma(1) = x_{\text{new}}$. If the *CollisionFree* function returns true, the new vertex and edge will be added to the tree.

NotReachStop: Returns true if the stop criteria for iterating or the goal point is not reached.

Note that the *Nearest* function returns all the vertices in the hypersphere with a specific radius centered at x_{new} which are used in the *ChooseParent* function.

There are two optimization procedures in RRT*: *ChooseParent* and *Rewire*. The pseudo-codes of *ChooseParent* and *Rewire* are presented in Algorithm 2 and Algorithm 3, respectively. The *ChooseParent* function searches the vertices in the neighborhood of x_{new} to find a vertex that can get to x_{new} with the lowest total cost.

Algorithm 2: ChooseParent(X_{near}, x_{nearst}, x_{new}, σ)

```
1:  $x_{\min} \leftarrow x_{\text{nearst}}$;
2:  $c_{\min} \leftarrow Cost(x_{\min}) + C(\sigma)$;
3:  for $x_{\text{near}} \in X_{\text{near}}$ do
4:      $(x_{\text{new}}', \sigma') \leftarrow Steer(x_{\text{near}}, x_{\text{new}})$;
5:      $c \leftarrow Cost(x_{\text{near}}) + C(\sigma')$;
6:      if $c < c_{\min}$ then
7:          if $CollisionFree(\sigma')$ then
8:              $c_{\min} \leftarrow c$;
9:              $x_{\min} \leftarrow x_{\text{near}}$;
10:         end if
11:     end if
12: end for
13: return $x_{\min}$;
```

The *Rewire* function tries to find the vertices in the neighborhood of x_{new}, which can be reached by x_{new} with a lower cost than their current parents and rewires them.

2.3 Q-RRT Algorithm*

There are two special changes in the Q-RRT* algorithm compared with RRT* as shown in Algorithm 4 and Algorithm 5. In the *ChooseParent* function, the parents of vertices in the neighborhood of x_{new} are also considered (Lines 8 and 9, Algorithm 4). The optimization efficiency can be improved by expanding the search scope. However, it does not increase the computation time too much since the vertices in X_{near} usually have the same parent.

Algorithm 3: Rewire(T, x_{new}, X_{near});

```
1:  for all $x_{near} \in X_{near}$ do
2:      ($x_{new}'$, $\sigma$)←Steer($x_{new}$, $x_{near}$);
3:      $c$←Cost($x_{new}$)+$C(\sigma)$;
4:      if $c$ < Cost($x_{near}$) then
5:          if CollisionFree($\sigma$) then
6:              $T$←Reconnect($T$, $x_{new}$, $x_{near}$);
7:          end if
8:      end if
9:  end for
10: return $T$;
```

Algorithm 4: Q-RRT*(V, E)

```
1:  $T$← ($V$, $E$);
2:  while NotReachStop do
3:      $x_{rand}$←Sample();
4:      $x_{nearst}$←Nearest($T$, $x_{rand}$);
5:      ($x_{new}$, $\sigma$)←Steer($x_{nearst}$, $x_{rand}$);
6:      if CollisionFree($\sigma$) then
7:          $X_{near}$←Near($T$, $x_{new}$);
8:          $X_{parent}$←Ancestry($T$, $X_{near}$);
9:          $x_{parent}$←ChooseParent($X_{near} \cup X_{parent}$, $x_{nearst}$, $x_{new}$, $\sigma$);
10:         $T$←Connect($T$, $x_{parent}$, $x_{new}$);
11:         $T$←Rewire-Q-RRT*($T$, $x_{new}$, $X_{near}$);
12:     end if
13: end while
14: return $T$;
```

In the *Rewire* function, the parents of x_{new} are considered to further improve the optimization efficiency (Line 2, Algorithm 5). The two main new functions in Q-RRT* are as follows:

ancestor: Given a graph $G = (V, E)$, a vertex x, and a natural number $d \in \mathbb{N}$, it returns the d^{th} parent of x.

Ancestry: Given a graph $G = (V, E)$ and a vertex x, it returns Ø if d = 0; otherwise, it returns $\bigcup_{i=1}^{d} ancestor(G, x, i)$.

Algorithm 5: Rewire-Q-RRT*(T, x_{new}, X_{near})

```
1:  for all $x_{near} \in X_{near}$ do
2:      for all $x_{form} \in \{x_{new}\} \cup ancestor(T, x_{new})$ do
3:      $(x_{new}', \sigma) \leftarrow Steer(x_{form}, x_{near})$;
4:       $c \leftarrow Cost(x_{form}) + C(\sigma)$;
5:       if $c < Cost(x_{near})$ then
6:          if $CollisionFree(\sigma)$ then
7:              $T \leftarrow Reconnect(T, x_{new}, x_{near})$;
8:          end if
9:       end if
10: end for
11: return $T$;
```

3 Proposed Algorithm

3.1 Virtual Light

Earlier work [18] proposed an improved A* algorithm based on virtual light, which is called LA*. In the LA* algorithm, a virtual light is set up at x_{goal}. The virtual light lights up the map according to the characteristics of light propagation along a straight-line. The areas where light cannot reach are marked as dark. The light intensity is the strongest at x_{start} and decreases as the propagation distance of the light becomes longer. Thus the light intensity can be regarded as a kind of heuristic information on the map. The planner can search for areas with higher light intensity. This kind of phototaxis mechanism can greatly speed up finding the solution.

The beam spreads from the virtual light just like the beam spreads from a flashlight. The direction of the beam is towards x_{start}, and the beam angle is less than or equal to 180°. The beam starts from x_{goal}, and each vertex that is illuminated by the beam has a light intensity noted by BRIGHT as shown in Fig. 1a. As the BRIGHT value increases, the light intensity decreases in the actual program, which facilitates processing. Therefore, the BRIGHT value of x_{goal} is 0.

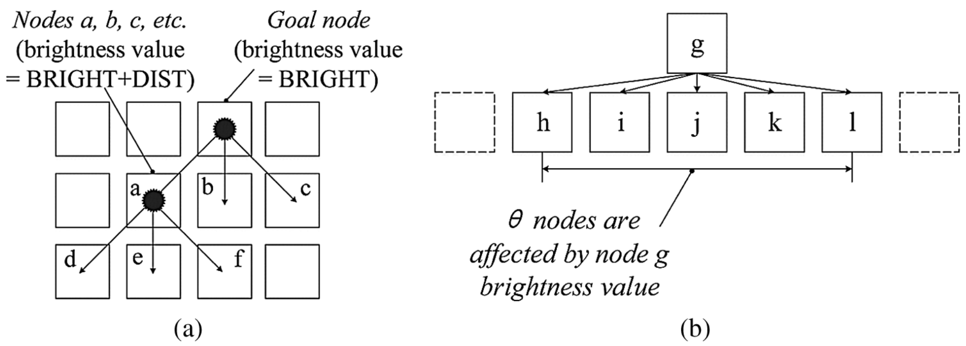
Nodes a, b, c, etc. (brightness value = BRIGHT+DIST)
Goal node (brightness value = BRIGHT)
a
b
c
d
e
f
(a)
g
h
i
j
k
l
θ nodes are affected by node g brightness value
(b)

Figure 1: Diagram of the light propagation mechanism (a) 90° light beam pattern (θ = *3*) (b) The generalized light beam angle pattern (any θ)

x_{goal} is set up at the top-middle position of the map as shown in Fig. 2b. The lighted map is represented by a grayscale image, and the light regions have a small BRIGHT value, while the dark regions have a high BRIGHT value. In the following description, the lighted map is noted by LMap for convenience. The number of successor affected vertices is given by θ (θ is an odd number). The case of when θ is 3 is shown in Fig. 1a. The successor-affected vertices of vertex a are d, e, and f. The BRIGHT value of the three vertices is equal to the BRIGHT value of a plus the distance from vertex a to each vertex. The unaffected vertices are marked DARK, which is a very large value. If a vertex is affected by multiple predecessor vertices, its BRIGHT value is set to be the minimum value, which means that the light travels along the shortest path.

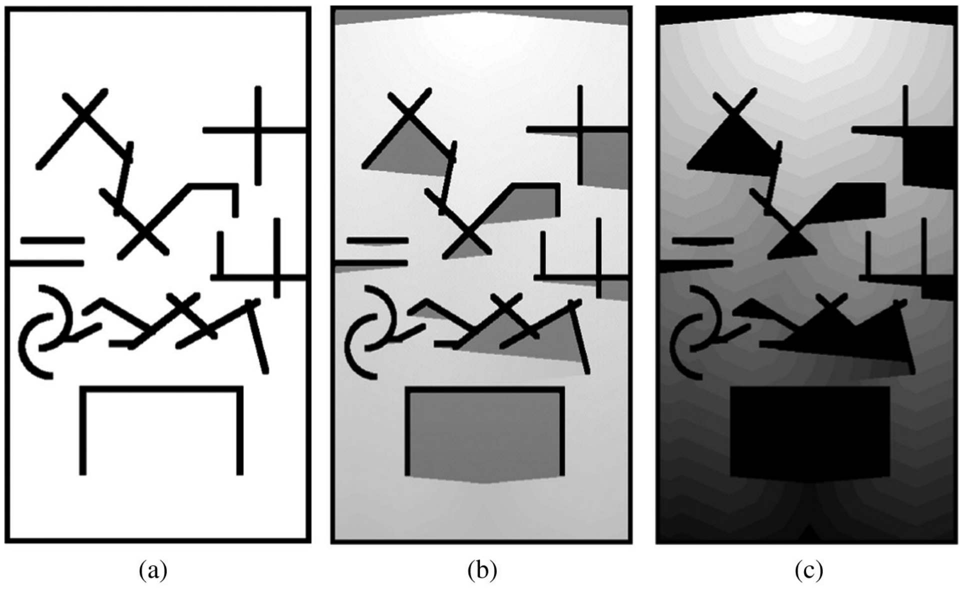

Figure 2: Maps under different representations (a) Original map (b) Lighted map (c) Contour map

The beam angle is determined by θ, $\theta \in [3, \max(w, h)]$, where w and h are the width and height of the map respectively. When θ is 3, the beam angle corresponds to the 90° pattern as shown in Fig. 1a. When θ is equal to $\max(w, h)$, it corresponds to the 180° pattern. The generalized light beam angle pattern is shown in Fig. 2b. If a vertex is blocked by an obstacle whose width is greater than θ and the light beam cannot reach the vertex, then the BRIGHT value of the vertex is set to be DARK.

*3.2 LQ-RRT**

A novel algorithm called LQ-RRT* is presented in this paper. The pseudo-code of LQ-RRT* is presented in Algorithm 6. The *LMap* function generates the lighted map according to the scheme described earlier (Line 1, Algorithm 6). A contour map, where the propagation of light intensity is depicted clearly, is shown in Fig. 2c to provide a better understanding of LMap. Unlike RRT* and Q-RRT*, LQ-RRT* constructs a light intensity perception region that is a semicircle centered at x_{nearest}. The *Resample* function is designed to resample n vertices in the perception region in different directions as shown in Fig. 3. Next, it returns x'_{rand}, which has the lowest BRIGHT value in the n vertices. In the LQ-RRT* algorithm, the *Sample* and *Nearest* functions are used to determine which vertex is to be expanded, and the *Resample* function determines the better direction to be expanded.

Algorithm 6: LQ-RRT*(V, E)

```
1:  $LMap()$; // Generate the lighted map
2:  $T \leftarrow (V, E)$;
3:  while $NotReachStop$ do
4:      $x_{rand} \leftarrow Sample()$;
5:      $x_{nearst} \leftarrow Nearest(T, x_{rand})$;
6:      $x'_{rand} \leftarrow Resample(x_{nearest})$;
5:      $(x_{new}, \sigma) \leftarrow Steer(x_{nearst}, x'_{rand})$;
6:      if $CollisionFree(\sigma)$ then
7:          $X_{near} \leftarrow Near(T, x_{new})$;
8:          $X_{parent} \leftarrow Ancestry(T, X_{near})$;
9:          $x_{parent} \leftarrow ChooseParent(X_{near} \cup X_{parent}, x_{nearst}, x_{new}, \sigma)$;
10:         $T \leftarrow Connect(T, x_{parent}, x_{new})$;
11:         $T \leftarrow Rewire\text{-}Q\text{-}RRT^*(T, x_{new}, X_{near})$;
11:     end if
12: end while
13: return $T$;
```

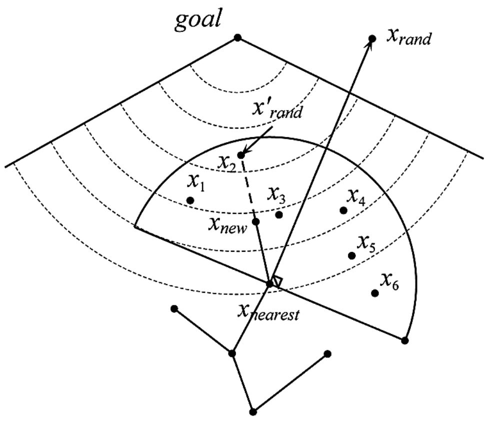

Figure 3: Diagram of *Resample* function

4 Experiments

In this section, LQ-RRT* is compared with RRT* and Q-RRT* in three different environment maps as shown in Fig. 4. Map 1 is the obstacle-free scenario shown in Figs. 4a–4c. Map 2 and Map 3 are scenarios cluttered with different types of obstacles as shown in Figs. 4d–4i, respectively. The map size is 406 m × 710 m. x_{start} is set at the bottom of the map, and x_{goal} is set at the top of the map. The parameters in experiments are the depth d, resample number n, and beam angle θ. The parameters are the same in all experiments to allow a fair comparison. In the *ancestor* and *Ancestry* functions, the value of d is 1. The resample number n is 6, and the beam angle θ is 23.

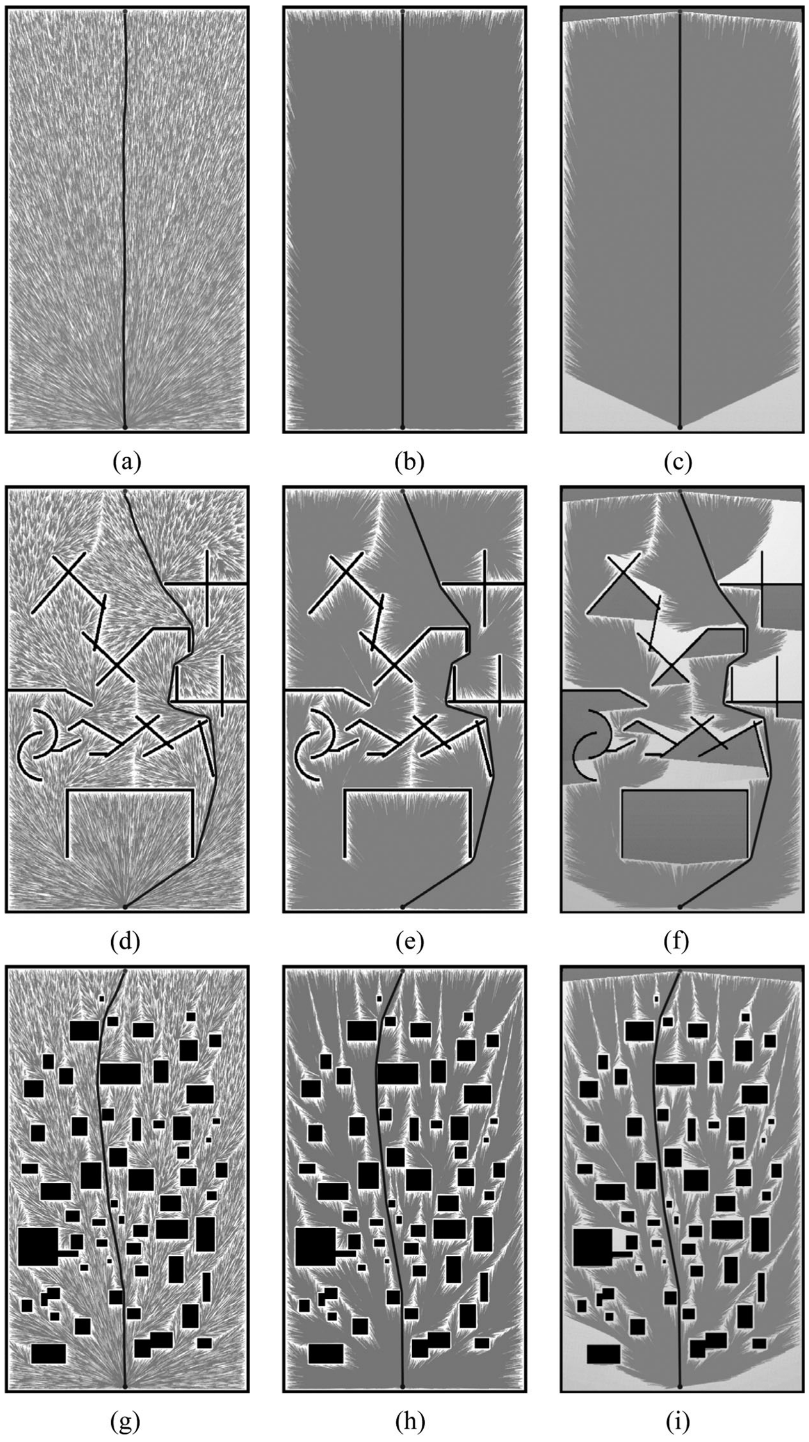

Figure 4: Generated path with 60000 samples in different maps (a) RRT* (b) Q-RRT* (c) LQ-RRT* (d) RRT* (e) Q-RRT* (f) LQ-RRT* (g) RRT* (h) Q-RRT* (i) LQ-RRT*

The final paths generated by different algorithms with 60,000 samples in the three maps are shown in Fig. 4. Although 60,000 samples have been used, the paths of RRT* are still not optimal (Fig. 4). In comparison, Q-RRT* and LQ-RRT* generate straighter paths. Meanwhile, the proposed algorithm can reduce the sample space effectively, especially when there are several areas marked with DARK on the map.

Statistical information is presented in Tabs. 1–3. Three indicators are utilized to compare the performance of the three algorithms. In Tabs. 1–3, the third and fifth columns are the length of the optimized path and the total vertices of the tree. The fourth column is the time to find a solution of 1.05*len_{optimal}, where len_{optimal} is the length of the optimal solution. len_{optimal} of the three maps are 690 m, 851.36 m, and 699.11 m.

Table 1: Results comparison of Map 1

Iterations	Algorithm	Path length(m)	Time$_{5\%}$(s)	Vertices
40000	RRT*	691.55	19.95	57282
	Q-RRT*	690	4.59	57174
	LQ-RRT*	690	**0.91**	**48981**
20000	RRT*	694.3	34.19	38076
	Q-RRT*	690	6.72	38142
	LQ-RRT*	690	**1.42**	**32792**
10000	RRT*	711.08	31.51	19091
	Q-RRT*	690	5.31	19093
	LQ-RRT*	690	**0.94**	**16391**
5000	RRT*	717.93	20.40	9568
	Q-RRT*	690	6.74	9554
	LQ-RRT*	690	**0.97**	**4233**
2500	RRT*	748.1	——	2429
	Q-RRT*	695.2	6.47	2408
	LQ-RRT*	**690**	**1.26**	**2173**

Table 2: Results comparison of Map 2

Iterations	Algorithm	Path length(m)	Time$_{5\%}$(s)	Vertices
40000	RRT*	860.74	55.55	32958
	Q-RRT*	853.5	23.19	33140
	LQ-RRT*	**852.99**	**2.61**	**22336**
20000	RRT*	900.8	90.27	15506
	Q-RRT*	855.5	38.63	16302
	LQ-RRT*	**853.81**	**3.57**	**11302**
10000	RRT*	951.25	——	6980
	Q-RRT*	881.28	42.04	6519
	LQ-RRT*	**858.62**	**2.00**	**5727**

Table 2 (continued).

Iterations	Algorithm	Path length(m)	$\text{Time}_{5\%}$(s)	Vertices
5000	RRT*	1127.9	——	3001
	Q-RRT*	1049.3	——	3380
	LQ-RRT*	**865.74**	**1.72**	**2992**
2500	RRT*	1173.3	——	1780
	Q-RRT*	1070.6	——	1703
	LQ-RRT*	**879.16**	**1.99**	**1469**

Table 3: Results comparison of Map 3

Iterations	Algorithm	Path length(m)	$\text{Time}_{5\%}$(s)	Vertices
40000	RRT*	703.08	68.35	29931
	Q-RRT*	700.82	24.42	30025
	LQ-RRT*	**700.52**	**7.46**	**25918**
20000	RRT*	718.31	32.92	14957
	Q-RRT*	701.21	11.55	14994
	LQ-RRT*	**700.61**	**1.04**	**13204**
10000	RRT*	746.18	——	7419
	Q-RRT*	701.68	18.72	7032
	LQ-RRT*	**700.89**	**0.93**	**6648**
5000	RRT*	757.4	——	3646
	Q-RRT*	711.31	20.04	3572
	LQ-RRT*	**702.8**	**1.44**	**2986**
2500	RRT*	995.82	——	1567
	Q-RRT*	745.73	——	1804
	LQ-RRT*	**702.24**	**1.06**	**1563**

Q-RRT* and LQ-RRT* converge to the optimal solution when the number of iterations reduces from 40,000 to 5000 as shown in Tab. 1. Only the LQ-RRT* algorithm converges to the optimal solution when the number of iterations reduces to 2500. The indicator "$\text{Time}_{5\%}$" shows that LQ-RRT* has a faster convergence rate than the other two algorithms, and the indicator "Vertices" shows that LQ-RRT* consumes the lowest memory. The same conclusions can be obtained for the two indicators from Tabs. 2 and 3.

LQ-RRT* generates the shortest path in all experiments as shown in Tabs. 2 and 3. The path lengths of RRT* and Q-RRT* gradually increase when the total number of iterations decreases. Meanwhile, although the path length of LQ-RRT* also increases, it is still asymptotically optimal. RRT* and Q-RRT* cannot find a solution with path length less than len_{optimal} when the number of iterations is small.

5 Conclusions

RRT* is an optimization algorithm. However, its main disadvantage is that the convergence rate is too low. An improved Q-RRT* algorithm, which we call LQ-RRT*, is proposed in this paper to overcome this problem. In Q-RRT*, *ChooseParent* and *Rewire* functions are modified to speed up the convergence. The concept of virtual light is introduced in this paper. Virtual light works like a flashlight. It lights up the map by light beam propagation and generates heuristic information on the map. Q-RRT* combined with the heuristic information can further accelerate the convergence of the solution. The results of experiments show that LQ-RRT* is quicker and consumes less memory than RRT* and Q-RRT*. In future work, it will be necessary to test the performance of LQ-RRT* in a dynamic environment.

Acknowledgement: The authors would like to thank the Jiangsu Electronic Data Forensics Analysis and Research Center (No. 2019SJPT002) and the Key Laboratory of Digital Forensics of Jiangsu Public Security Department. We also thank LetPub (www.letpub.com) for its linguistic assistance during the preparation of this manuscript.

Funding Statement: This work was supported by the Natural Science Foundation of the Jiangsu Higher Education Institutions of China [grant number 19KJB510022] and the Startup Research Foundation for Advanced Talents [grant number JSPIGKZ/2911119220].

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] Z. Chen, D. Wang, G. Chen, Y. Ren and D. Du, “A hybrid path planning method based on articulated vehicle model,” *Computers, Materials & Continua*, vol. 65, no. 2, pp. 1781–1793, 2020.

[2] N. Lin, J. Tang, X. Li and L. Zhao, “A novel improved bat algorithm in UAV path planning,” *Computers Materials & Continua*, vol. 61, no. 1, pp. 323–344, 2019.

[3] V. Mezhuyev, Y. Gunchenko, S. Shvorov and D. Chyrchenko, “A method for planning the routes of harvesting equipment,” *Intelligent Automation & Soft Computing*, vol. 26, no. 1, pp. 121–132, 2020.

[4] D. Wu, Y. Liu, Z. Xu and W. Shang, “Design and development of unmanned surface vehicle for meteorological monitoring,” *Intelligent Automation & Soft Computing*, vol. 26, no. 5, pp. 1123–1138, 2020.

[5] J. Wang, T. Zhang, B. Jin and S. Wu, “A novel sins/iusbl integration navigation strategy for underwater vehicles,” *Journal of Cyber security*, vol. 1, no. 1, pp. 1–10, 2019.

[6] P. E. Hart, N. J. Nilsson and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[7] A. Stentz, “Optimal and efficient path planning for partially known environments,” in *Proc. ICRA*, San Diego, CA, USA, pp. 3310–3317, 1994.

[8] K. Daniel, A. Nash, S. Koenig and A. Felner, “Theta*: Any-angle path planning on grids,” *Journal of Artificial Intelligence Research*, vol. 39, pp. 533–579, 2010.

[9] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” in *Technical Report*, Ames, Iowa, USA: Iowa State University, No. 98-11, 1998.

[10] J. J. Kuffner and S. M. LaValle, “RRT-connect: An efficient approach to single-query path planning,” in *Proc. ICRA*, San Francisco, CA, USA, 2, pp. 995–1001, 2000.

[11] C. Urmson and R. Simmons, “Approaches for heuristically biasing RRT growth,” in *Proc. IROS, Las Vegas*, NV, USA, 2, pp. 1178–1183, 2003.

[12] D. Ferguson and A. Stentz, “Anytime RRTs,” in *Proc. IROS*, Beijing, China, pp. 5369–5375, 2006.

[13] Q. D. Zhu, Y. B. Wu, G. Q. Wu and X. Wang, “An improved anytime RRTs algorithm,” in *Proc. Artificial Intelligence and Computational Intelligence*, Shanghai, China, pp. 268–272, 2009.

[14] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[15] F. Islam, J. Nasir, U. Malik, Y. Ayaz and O. Hasan, “RRT*-smart: Rapid convergence implementation of RRT* towards optimal solution,” in *Proc. Mechatronics and Automation*, Chengdu, China, pp. 1651–1656, 2012.

[16] J. D. Gammell, S. S. Srinivasa and T. D. Barfoot, “Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic,” in *Proc. IROS*, Chicago, IL, USA, pp. 2997–3004, 2014.

[17] I. B. Jeong, S. J. Lee and J. H. Kim, “Quick-RRT*: Triangular inequality-based implementation of RRT* with improved initial solution and convergence rate,” *Expert Systems with Applications*, vol. 123, no. 2, pp. 82–90, 2019.

[18] M. Hawa, “Light-assisted A_* path planning,” *Engineering Applications of Artificial Intelligence*, vol. 26, no. 2, pp. 888–898, 2013.