 doi:10.20944/preprints202011.0494.v1

Article

On the Enhancement of Optimality in the RRT-Connect Robot Path Planning Algorithm

Jin-Gu Kang [1], Dong-Woo Lim [1], Yong-Sik Choi [2], Woo-Jin Jang [1] and Jin-Woo Jung [1,*]

[1] Department of Computer Science and Engineering, Dongguk University, Seoul 04620, Korea; kanggu12@dongguk.edu (J.-G.K.); aehddn@gmail.com (D.-W.L.); skwndbth159@dongguk.edu (W.-J.J.)
[2] Department of Artificial Intelligence, Dongguk University, Seoul 04620, Korea; sik2230@dongguk.edu (Y.-S.C.)
* Correspondence: jwjung@dongguk.edu (J.-W.J.); Tel.: +82-2-2260-3812

Abstract: This paper proposed a Triangular Inequality based rewiring method for the RRT(Rapidly exploring Random Tree)-Connect robot path planning algorithm that guarantees the convergence time than the RRT algorithm, to enhance the optimality. To check the performance of the proposed algorithm, this paper compared with the RRT and RRT-Connect algorithms in various environments through simulation. From these experimental results, the proposed algorithm shows both quick convergence time and better optimality than the RRT algorithm, and more optimal than RRT-Connect algorithm with the similar number of sampling and convergence time.

Keywords: Robot path planning; RRT-Connect; Triangular inequality; Rewiring; Optimality

1. Introduction

With the recent 4th Industrial Revolution, interest in mobile robots is increasing in various fields such as robotics, smart factories, and autonomous driving [1]. Classical mobile robot path planning algorithms could be classified into three broad categories [2]. The first is the Road Map Approach algorithm [3], that easy to implement by designing a map that represents a path can be moved and plan through it. The second is Cell Decomposition algorithm [4], that creates a path by dividing the configuration space to cell and connecting each cell using a graph. The last is the Artificial Potential Field algorithm [5], that creates an artificial potential field and moves the robot to the goal according to the flow of potential power.

These classical algorithms include the ***Optimality***, which means how close the created path is to the shortest distance, the ***Clearance***, which means how lower probability of collision between obstacles and a robot, the ***Completeness***, which means that if a path exists it can always be found are considered important and these have been mainly studied [6].

Recently, sampling based path planning algorithms [7-12] such as Rapidly exploring Random Tree (RRT) [13] which is quickly and less computationally than classical algorithms, has been attracting attention. The main purpose of the sampling based algorithms is to find a path that can reach the goal as quickly as possible using randomly extracted sample points (random sampling). Unlike classical algorithms, the sampling based algorithm is difficult to fully reflect the optimality and the completeness. Therefore, most of the sampling based algorithms claim ***Probabilistic Completeness***, which explains that can be probabilistically close to completeness when random sampling is repeated infinitely [14]. This means that it is difficult to guarantee the ***Convergence time***, which means how quickly plan the path from a start point to a goal point, and if a path planning is required within a short time, the path differ from the optimal path can be created. Even so, the sampling based algorithm is mainly used in a dynamic environment because it enables more quickly path planning with very little convergence time than the classical algorithms.

To overcome this limitation, many studies are being conducted to expand the RRT algorithm. The RRT-Connect [15] algorithm is an algorithm to find a connected path more quickly than RRT

algorithm by setting the starting point and the goal point as the roots of separate trees and expanding the two trees alternately. In addition, there is an algorithm of optimizing a path based on the principle of triangular inequality, such as RRT*-Smart algorithm [16] and Quick-RRT* algorithm [17], to derive a path that is close to the optimal. Many algorithms [18-21] that extend the RRT algorithm have been studied.

The above algorithms show more efficient performance by improving the RRT algorithm in order to overcome the limitations of the sampling based method, but they are not perfect, such as not being able to derive the optimal distance, or there is room for improvement in terms of number of operations or time. Therefore, in order to overcome the limitations of this sampling based algorithm, this paper proposes a Triangular Inequality based RRT-Connect algorithm that finds and wires the highest ancestral existing node can be wired while alternately expanding in two trees rooted at the starting point and the goal point. The proposed algorithm shortens the convergence time and at the same time pursues optimization through rewiring. In addition, we will verify the efficiency by comparing the RRT algorithm and RRT-Connect algorithm, which are previous studies, through simulation experiments (This paper proposes a methodology and an algorithm that applies the Triangular Inequality principle to the RRT-Connect algorithm. The RRT*-Smart and Quick-RRT* algorithms that applied the Triangular Inequality principle to the RRT algorithm are not treated as comparative experiments in this paper).

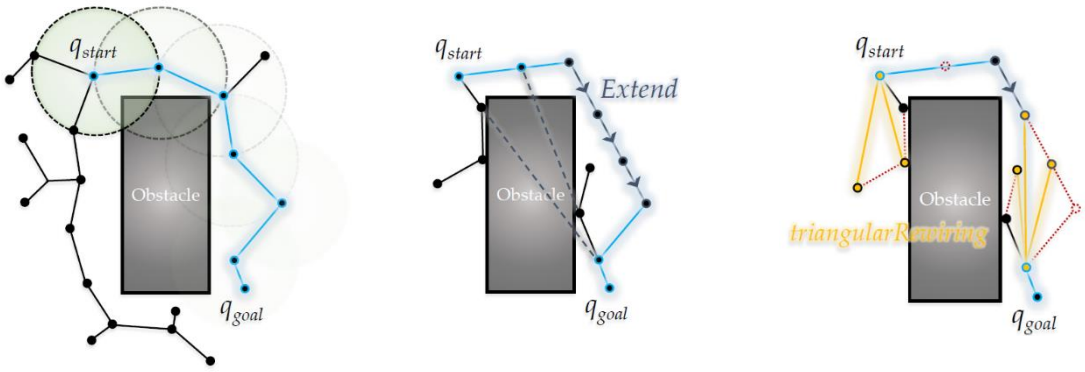


Figure 1. Overview of the algorithms in this paper: (**a**) RRT; (**b**) RRT-Connect; (**c**) The proposal.

Figure 1 shows an overview of the three algorithms mainly covered in this paper: RRT, RRT-Connect, and the proposal. In this figure, the start point is q_{start}, and the goal point is q_{goal}. The RRT algorithm in Figure 1 (a) shows that the path is expanded in a tree structure, and the RRT-Connect algorithm in Figure 1 (b) shows that the trees that are expanded at the start and goal points each attract and connect each other. The proposed algorithm in Figure 1 (c) shows that the RRT-Connect algorithm was rewired into a Triangular Inequality during the path planning.

In this paper, Chapter 2 introduces the RRT algorithm, Chapter 3 introduces the RRT-Connect algorithm, and the Triangular Inequality based RRT-Connect algorithm proposed in Chapter 4. In detail, Section 4.1 shows the pseudocode of the proposed rewire method through the principle of triangular inequality, which can be applied to the RRT-Connect algorithm, Section 4.2 shows the mathematical modeling of the proposed algorithm, and Section 4.3 shows the pseudocode of each method of the RRT-Connect algorithm applying the proposed rewire method. The experimental environment and results to check the performance of the proposed algorithm in Chapter 5 are shown, and the conclusion is shown in Chapter 6.

2. The Rapidly exploring Random Tree (RRT) Algorithm

The **Rapidly exploring Random Tree (RRT)** algorithm [13] is the most representative sampling based path planning algorithm. the RRT algorithm is an algorithm of planning this path by gradually

expand a tree with a root node at the start point by using random sampling. It is designed to handle Non-holonomic constraints and high degrees of freedom [12].

When a random sample is generated in the configuration space, it tries to connect at a point separated by a preset step length from the node nearest to the random sample among the nodes constituting the tree with the step length. If tree connections are possible, add nodes to create an extended tree.

As mentioned in introduction, this sampling based path planning algorithm uses randomly generated sample points to find one path that can reach the goal as quickly as possible, so it is difficult to sufficiently reflect the optimality and completeness.

2.1. Pseudocode of the RRT Algorithm

This section shows the pseudocode of the RRT algorithm used in the experiment of this paper, designed based on the paper [13] in which the RRT algorithm was proposed. The RRT algorithm can be represented by one main algorithm (A1) and two additional functions (A2-3).

Algorithm 1. Pseudocode of the RRT Algorithm.

Input:
q_{start} ← Position of Start Point
q_{goal} ← Position of Goal Point
λ ← Step Length
C ← Position Set of All Boundary Points in All Obstacles
N ← Number of Random Samples
Output:
R ← Result of Path R
Initialize:
T ← ***Null*** Tree
d_{optima} ← 0

Begin *RRT* **Procedure**
While 1 **to** N **do**
  Generate Random Sample
  q_{rand} ← Position of Random Sample
  q_{near} ← **Find** Position of Nearest Node in T from q_{rand}
  If Not *isInside*(q_{near}, q_{rand}, λ) **then**
    q_{new} ← Position of Intersection Point between Line Segment connecting q_{rand} and q_{near}, and Circle with Radius λ centered at q_{near} // **2D:** Circle, **3D:** Sphere, **…**
  Else
    q_{new} ← q_{rand}
  If Not *isTrapped*(q_{new}, q_{near}, C) **then**
    T ← **Insert** Node<q_{new}> and Edge<q_{new}, q_{near}> to T
    If *isInside*(q_{new}, q_{goal}, λ) **then**
      T ← **Insert** Node<q_{goal}> and Edge<q_{new}, q_{goal}> to T
      P_{reach} ← Path from Last Inserted Node [q_{goal}] to Root Node [q_{start}] in T
      d_{reach} ← Distance of P_{reach}
      If d_{optima} = 0 **or** d_{optima} > d_{reach} **then**

17 $R \leftarrow P_{reach}$

18 $d_{optima} \leftarrow d_{reach}$

19 $T \leftarrow$ **Delete** Node<q_{goal}> and Edge<q_{new}, q_{goal}> from T

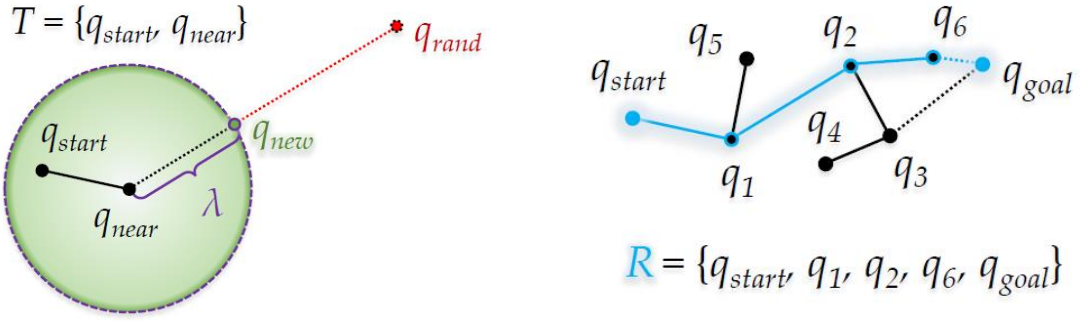
End *RRT* **Procedure**

Algorithm 1 shows the pseudocode of RRT algorithm. The root node of the initial tree T has q_{start}, and this q_{start} is the start point. From this q_{start} to the goal position q_{goal}, random sample is generated N times and aims to reach while the tree is expanded. q_{rand} is position of generated random sample, and in this q_{rand}, a node nearest to the tree T is q_{near}. At this time, the position q_{new} created later varies depending on whether q_{rand} is located inside a circle (or n-sphere) with q_{near} as the center and step length λ as the radius. The function that determines this is *isInside* (A2), and if q_{rand} is located inside the circle (or n-sphere) (*True*), q_{new} becomes q_{rand}, and if it is not located inside (*False*), q_{new} becomes intersection point between line segment connecting q_{rand} and q_{near}, and circle with λ centered at q_{near}. If there is no obstacle between q_{new} and q_{near} (*False*), q_{new} is inserted into the tree as a child node of q_{near} of T. Currently, the function that determines whether an obstacle exists between q_{new} and q_{near} is *isTrapped* (A3) (in the *isTrapped* function, C refers to the set of obstacles).

If there is the q_{goal} inside the radius of the λ with the newly inserted q_{new} as the center, it is considered to have reached the goal point (by *isInside* function). If it is reached (*True*), q_{goal} is inserted as a child node of node q_{new} of T.

For the tree T thus completed, the distance d_{reach} is calculated for the path P_{reach} to q_{start} and q_{goal}. Currently, if d_{reach} is smaller than the optimal path distance d_{optima} or reached first ($d_{optima} = 0$), the optimal path R becomes P_{reach}, and d_{optima} becomes d_{reach}. At the end of the next N sampling, R becomes the final planned path.

If the number of random sampling remains, the above process is repeated. At this time, q_{goal} and the edge connected to this node must be deleted from the tree T. Otherwise, the tree structure will break due to cyclic. As a result, when a graph structure is formed, the cost of path search increases rapidly.



(**a**) (**b**)

Figure 2. The RRT algorithm: (**a**) Process when q_{new} is created; (**b**) After the random sampling ends.

Figure 2 shows the path planning process of the RRT algorithm. Figure 2 (a) shows that q_{new} is created at the node position q_{near} of the tree T nearest to the random sample position q_{rand}. Figure 2 (b) shows the optimal path R among several candidate paths to the start position q_{start} and the goal position q_{goal}.

2.2. Pseudocode of the functions used in the RRT Algorithm

This section introduces additional functions used in pseudocode of the RRT algorithm (A1) in Section 2.1. The *isTrapped* function determines whether an obstacle collides, and the *isInside* function determines if the point exists inside the radius.

Algorithm 2. Pseudocode of the *isInside* Function from the RRT Algorithm.

Input:
q_{center} ← Position {q_{near} / q_{new}} from *RRT*
q_{target} ← Position {q_{rand} / q_{goal}} from *RRT*
λ ← Step Length λ from *RRT*
Output:
f ← Result of Boolean f
Initialize:
f ← ***False***

Begin *isInside* **Procedure from** *RRT*
1 d ← Distance of q_{center} to q_{target}
2 **If** $\lambda \geq d$ **then**
3 f ← ***True***
End *isInside* **Procedure from** *RRT*

Algorithm 2 shows the *isInside* function among RRT pseudocodes shown in Algorithm 1. In RRT algorithm, it is determined whether q_{rand} exists inside a circle (or n-sphere) with q_{near} as the center and λ as the radius, or q_{goal} exists inside a circle (or n-sphere) with q_{new} as the center and λ as the radius.

In the *isInside* function, the position to be determined (q_{rand}, q_{goal}, ...) is called q_{target}, and the center of radius (q_{near}, q_{new}, ...) is called q_{center}. When the distance between q_{center} and q_{target} is d, if this d is less than or equal to λ, it is determined that q_{target} is the inside position (*True*).

Algorithm 3. Pseudocode of the *isTrapped* Function from the RRT Algorithm.

Input:
q_{new} ← Position q_{new} from *RRT*
q_{near} ← Position q_{near} from *RRT*
C ← Position Set of All Boundary Points in All Obstacles C from *RRT*
Output:
f ← Result of Boolean f
Initialize:
n ← 1
f ← ***True***

Begin *isTrapped* **Procedure from** *RRT*
1 l_q ← Line Segment connecting q_{new} and q_{near}
2 c ← Position Set of All Boundary Points of n-th Inserted Obstacle in C
3 l_c ← Line Segment connecting Last Inserted Position and 1st Inserted Position in c
4 i ← 1
5 **While Not** Intersect between l_q and l_c **do**
6 l_c ← Line Segment connecting i-th Inserted Position and (i + 1)-th Inserted Position in c
7 i ← i + 1
8 **If** i = (Number of Position in c) – 1 **then**
9 **If** Intersect between l_q and l_c **then**
10 ***Break***
11 n ← n + 1
12 **If** n > (Number of Position Set in C) **then**

```
13              f ← False
14              Break
15          Else
16              c ← Position Set of All Boundary Points of n-th Inserted Obstacle in C
17              l_c ← Line Segment connecting Last Inserted Position and 1st Inserted Position in c
18              i ← 1
End isTrapped Procedure from RRT
```

Algorithm 3 shows the *isTrapped* function among RRT algorithm pseudocodes shown in Algorithm 1. In RRT algorithm, it is used to determine whether an obstacle exists between the line segment connecting q_{new} and q_{near}.

If the line segment connecting q_{new} and q_{near} is l_q, and the set of positions formed by the n-th obstacle in the set of obstacles C is c, l_c is the i-th and $(i + 1)$-th positions inserted in c (and the last and 1st position). It is determined whether it intersects with l_q for all line segments l_c in the set of positions of all obstacles c that C has. At this time, if any intersect occurs, it returns *True* and stops the procedure immediately. Otherwise, it determines all the line segments that can be l_c and returns *False*.

3. The RRT-Connect Algorithm

Path planning through the RRT algorithm may have a disadvantage in that since random samples appear with the same probability in all regions, the tree easily extends even in a direction irrespective of the goal, resulting in a long convergence time and inefficient. The **RRT-Connect** algorithm [15] proposed later has two new ideas as the method to compensate for the disadvantage of the RRT algorithm.

The first is the start point and the goal point are inserted as each root nodes and extended in each direction alternately. The two trees extending from the start point and the goal point expand as if attracting each other, so that preventing the tree, which is the disadvantage of RRT algorithm, is in a direction irrespective of the goal. This has the enhancement of disadvantage the convergence time required to search for a path. The second is the concept of *Extend*, which continues to extend to the other side of the tree if there is no collision with an obstacle when the tree extends. Through this, unlike the RRT algorithm that extends the maximum extension length when the sample is generated and is inserted to the tree, the tree continues to expand in the direction of the goal if there is no collision with an obstacle, so that a path can be planned more quickly.

Path planning through the RRT-Connect algorithm can find a path at a quickly than the RRT algorithm, but in a complex environment with narrow paths and many obstacles, the *Extend* method does not work properly and it can be difficult. Also, the path planned through the RRT-Connect algorithm is far from the optimal distance, so it does not properly reflect the optimality.

3.1. Pseudocode of the RRT-Connect Algorithm

This section shows the pseudocode of the RRT-Connect algorithm used in the experiment of this paper, designed based on the paper [15] in which the RRT-Connect algorithm was proposed. The RRT-Connect algorithm can be represented by one main algorithm (A4) and two main methods (A5-6).

Algorithm 4. Pseudocode of the RRT-Connect Algorithm.

```
Input:
q_start ← Position of Start Point
q_goal ← Position of Goal Point
λ ← Step Length
C ← Position Set of All Boundary Points in All Obstacles
```

N ← Number of Random Samples
Output:
R ← Result of Path R
Initialize:
T_a ← ***Null*** Tree
T_b ← ***Null*** Tree
d_{optima} ← 0

Begin *RRT-Connect* **Procedure**
T_a ← **Insert** Node<q_{start}> to T_a
T_b ← **Insert** Node<q_{goal}> to T_b
While 1 **to** N **do**
  Generate Random Sample
  q_{rand} ← Position of Random Sample
  If Not *Extend*(T_a, T_b, q_{newB} ← ***Null***, q_{rand}, λ, C) **then**
    If *Connect*(P_{reach} ← ***Null*** Path, T_a, T_b, q_{newB}, λ) **then**
      d_{reach} ← Distance of P_{reach}
      If d_{optima} = 0 **or** d_{optima} > d_{reach} **then**
        R ← P_{reach}
        d_{optima} ← d_{reach}
  Swap(T_a, T_b)
End *RRT-Connect* **Procedure**

Algorithm 4 shows the pseudocode of RRT-Connect algorithm. Each of the initial two trees T_a and T_b has q_{start} and q_{goal} as root nodes, and these two trees randomly sample N times and aim to reach each other during expansion. Unlike RRT algorithm, RRT-Connect algorithm is divided into two methods: *Extend* and *Connect*. The ***Extend*** method (A5) creates q_{new} from q_{rand} in T_a and extends from T_b to the q_{new} direction of T_a, and the ***Connect*** method (A6) determines whether the two trees T_a and T_b have reached each other, and if they do, merge them into one tree to obtain a path P_{reach} between the root nodes q_{start} and q_{goal} of the two trees.

If a path is created in the *Connect* method, the optimal path R in N sampling times is obtained for this path P_{reach} in the same way as RRT algorithm.

3.2. Pseudocode of the Extend method from the RRT-Connect Algorithm

This section introduces the *Extend* method used in pseudocode (A4) of the RRT-Connect algorithm in Section 3.1.

Algorithm 5. Pseudocode of the Original Extend Method from the RRT-Connect Algorithm.

Input:
T_a ← Tree T_a from *RRT-Connect*
T_b ← Tree T_b from *RRT-Connect*
q_{newB} ← Position q_{newB} from *RRT-Connect*
q_{rand} ← Position q_{rand} from *RRT-Connect*
λ ← Step Length λ from *RRT-Connect*
C ← Position Set C from *RRT-Connect*
Output:
f_{trap} ← Result of Boolean f_{trap}

T_a ← Result of Tree T_a **// Return by Reference**
T_b ← Result of Tree T_b **// Return by Reference**
q_{newB} ← Result of Position q_{newB} **// Return by Reference**
Initialize:
f_{trap} ← ***False***

Begin *Extend* **Procedure from** *RRT-Connect*
1 q_{near} ← **Find** Position of Nearest Node in T_a from q_{rand}
2 **If Not** *isInside*(q_{near}, q_{rand}, λ) **then**
3 q_{newA} ← Position of Intersection Point between Line Segment connecting q_{rand} and q_{near}, and Circle with Radius λ centered at q_{near} **// 2D:** Circle, **3D:** Sphere, **…**
4 **Else**
5 q_{newA} ← q_{rand}
6 **If** *isTrapped*(q_{newA}, q_{near}, C) **then**
7 f_{trap} ← ***True***
8 **Else**
9 T_a ← **Insert** Node<q_{newA}> and Edge<q_{newA}, q_{near}> to T_a
10 q_{near} ← **Find** Position of Nearest Node in T_b from q_{newA}
11 **If** *isInside*(q_{near}, q_{newA}, λ) **then**
12 q_{newB} ← q_{near}
13 **Else**
14 q_{newB} ← Position of Intersection Point between Line Segment connecting q_{newA} and q_{near}, and Circle with Radius λ centered at q_{near} **// 2D:** Circle, **3D:** Sphere, **…**
15 **While Not** *isTrapped*(q_{newB}, q_{near}, C) **do**
16 T_b ← **Insert** Node<q_{newB}> and Edge<q_{newB}, q_{near}> to T_b
17 **If Not** *isInside*(q_{newA}, q_{newB}, λ) **then**
18 q_{near} ← q_{newB}
19 q_{newB} ← Position of Intersection Point between Line Segment connecting q_{newA} and q_{near}, and Circle with Radius λ centered at q_{near} **// 2D:** Circle, **3D:** Sphere, **…**
20 **Else**
21 ***Break***
End *Extend* **Procedure from** *RRT-Connect*

Algorithm 5 shows the pseudocode of the *Extend* method in the RRT-Connect algorithm. The initial part is the same as the RRT algorithm (A1). *isInside* function determines whether q_{rand} is inside a circle (or n-sphere) with the node position q_{near} of the tree T_a nearest to the q_{rand} position as the center and λ as the radius. If it is not located inside (*False*), q_{newA} becomes the intersection of the circle (or n-sphere) with q_{near} as the center and λ as the radius, and the line segment connecting q_{rand} and q_{near}. If it is determined that there is no obstacle between q_{newA} and q_{near} by the *isTrapped* function (*False*), q_{newA} is inserted into the tree as a child node of q_{near} of T_a. If there is an obstacle (*True*), the *Extend* method returns *True* (f_{trap}) and terminates. If not, proceeds with the remain process, and returns *False* (f_{trap}) when the process is over.

This is the process of making T_a and T_b reach each other: First, the node of T_b nearest to q_{newA} becomes the new q_{near}. At this time, by the *isInside* function, it is determined whether q_{newA} is inside a circle (or n-sphere) with q_{near} as the center and λ as the radius, and if it is located inside (*True*), q_{newB} becomes q_{near} and is located inside If not (*False*), q_{newB} becomes the intersection of the circle (or n-sphere)

with q_{near} as the center and λ as the radius, and the line segment connecting q_{newA} and q_{near}. If q_{newB} is created, then repeats the following process until it determine whether there is an obstacle between q_{newB} and q_{near} by the *isTrapped* function and if there is the obstacle between them (*True*) or that q_{newB} reaches q_{newA} by the *isInside* function.

If there is no obstacle between these q_{newB} and q_{near} (*False*), insert q_{newB} into T_b as a child node of q_{near}. At this time, if it is determined by the *isInside* function that q_{newB} has not reached the λ radius with q_{newA} as the center (*False*), q_{near} becomes the q_{newB}, and a new q_{newB} will created from this q_{near}.

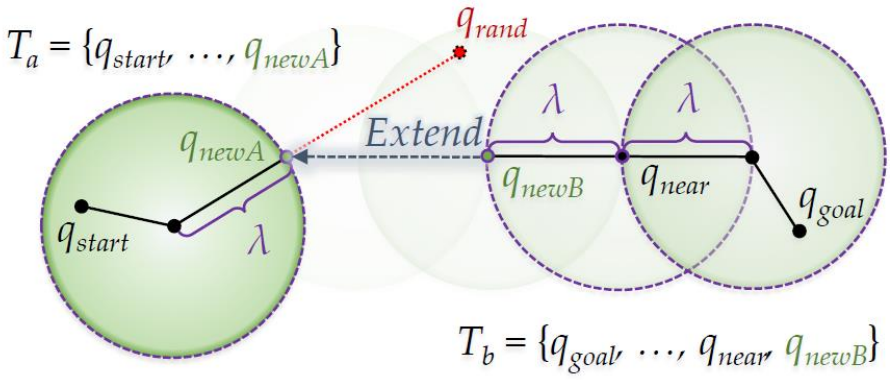


Figure 3. The *Extend* method from RRT-Connect algorithm.

Figure 3 shows the *Extend* method in the RRT-Connect algorithm. In detailed, it shows that the first q_{newA} is created, and q_{newB} is created by a radius of λ length in the direction of q_{newA} from the q_{near} position in the figure. It can be seen that T_b extends in the T_a direction for reach.

3.3. Pseudocode of the Connect method from the RRT-Connect Algorithm

This section introduces the *Connect* method used in pseudocode (A4) of the RRT-Connect algorithm in Section 3.1.

Algorithm 6. Pseudocode of the Original *Connect* Method from the RRT-Connect Algorithm.

Input:
P_{reach} ← Path P_{reach} from *RRT-Connect*
T_a ← Tree T_a from *RRT-Connect*
T_b ← Tree T_b from *RRT-Connect*
q_{newB} ← Position q_{newB} from *RRT-Connect*
λ ← Step Length λ from *RRT-Connect*
Output:
f_{reach} ← Result of Boolean f_{reach}
P_{reach} ← Result of Path P_{merged} **// Return by Reference**
Initialize:
f_{reach} ← ***False***

Begin *Connect* **Procedure from** *RRT-Connect*
1 **If** *isInside*(q_{newA}, q_{newB}, λ) **then**
2 P_a ← Path from Root Node [q_{start}] to Last Inserted Node [q_{newA}] in T_a
3 P_b ← Path from q_{newB} to Root Node [q_{goal}] in T_b
4 $P_{connect}$ ← Path from Last Inserted Node [q_{newA}] in T_a to q_{newB} in T_b
5 P_{merged} ← **Merge Path** P_a to P_b via $P_{connect}$
6 f_{reach} ← ***True***

End *Connect* **Procedure from** *RRT-Connect*

Algorithm 6 shows the pseudocode of the *Connect* method in the RRT-Connect algorithm. these T_a, T_b and q_{newB} from the *Extend* method (A5).

The tree merging process is as follows: Create a path P_a from the root node (q_{start}) of T_a to the last inserted node (q_{newA}), and a path P_b from q_{newB} of T_b to the root node (q_{goal}). Then, create the path $P_{connect}$ from q_{newB} of P_b to the last inserted node (q_{newA}) of T_a, and merge in the order of P_a, $P_{connect}$, and P_b, thereby complete planning the path P_{merged} from q_{start} to q_{goal}. After that, it returns *True* (f_{trap}), and the *Connect* method ends.

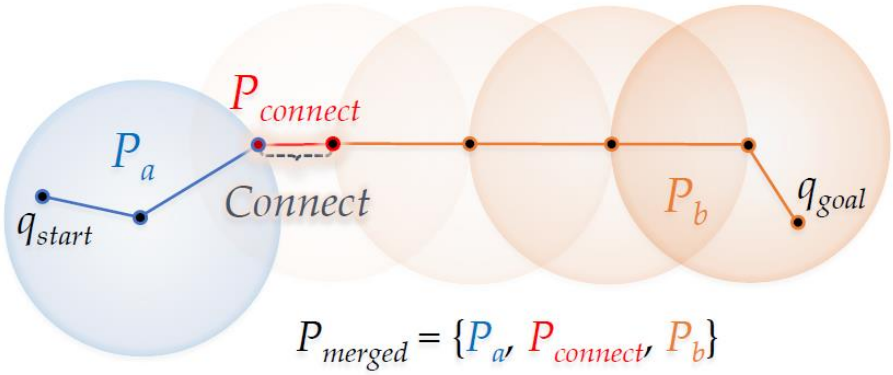


Figure 4. The *Connect* method from RRT-Connect algorithm.

Figure 4 shows the *Connect* method in the RRT-Connect algorithm. If the q_{newB} of T_b is extended in the direction of the q_{newA} by the *Extend* method shown in Figure 3, the point where the two trees merge (when q_{newB} expanded in the direction of q_{newA} of T_a enters the λ radius with centered at q_{newA}) with each other is the part marked as *Connect*. As a result, the path P_a becomes from the position q_{start} to the position q_{newA} in T_a, the path $P_{connect}$ becomes from the position q_{newA} to the position q_{newB}, and the path P_b becomes from the position q_{newB} to the position q_{goal} in T_b. The merged path P_{merged} becomes from the q_{start} point to the q_{goal} point.

4. Proposed Triangular Inequality based RRT-Connect Algorithm

The **Proposed Triangular Inequality based RRT-Connect** algorithm is rewire based on the principle of Triangular Inequality between nodes on a path planned in the RRT-Connect algorithm, thereby increasing the optimality compared to the RRT-Connect. This is similar to the RRT*-Smart algorithm [16] and Quick-RRT* [17] algorithms, which have improved their optimality by using the Triangular Inequality principle for the RRT algorithm. In this paper, the part to rewire based on the Triangular Inequality principle is called the ***Triangular Rewiring*** method.

Therefore, this chapter introduce the proposed *Triangular Rewiring* method for the RRT-Connect algorithm, and mathematical modeling is performed to confirm the validity that the proposed *Triangular Rewiring* method is always more optimal when applied to the RRT-Connect algorithm. After checking through, we will propose how to apply the *Triangular Rewiring* method to the RRT-Connect algorithm.

The method of applying the RRT-Connect algorithm of the proposed *Triangular Rewiring* method is proposed when a new node is inserted into the tree in the *Extend* method (A5) and *Connect* method (A6), the main methods of the RRT-Connect algorithm introduced in Chapter 3. It is inserted after rewiring (or after determining) through the *Triangular Rewiring* method. That is, *Extend* method and *Connect* method to which the proposed *Triangular Rewiring* method is applied are introduced in this chapter.

4.1. Pseudocode of Proposed Triangular Rewiring Method for the RRT-Connect Algorithm

This section introduces the *Triangular Rewiring* method for the proposed Triangular Inequality based RRT-Connect algorithm.

Algorithm 7. Pseudocode of the Proposed *Triangular Rewiring* Method for the RRT-Connect Algorithm.

Input:

q_{child} ← Position {q_{new} / q_{newA} / q_{newB}} from {*Extend* / *Connect*}

q_{parent} ← Position q_{near} from {*Extend* / *Connect*}

T ← Tree {T_{merged} / T_a / T_b} from {*Extend* / *Connect*}

C ← Position Set C from {*Extend* / *Connect*}

Output:

{T_{merged} / T_a / T_b} ← Result of T

Begin *triangularRewiring* **Procedure from** *Extend*, *Connect*

1 $q_{ancestor}$ ← Position of Parent Node of q_{parent} in T

2 **If Not** *isTrapped*($q_{ancestor}$, q_{child}, C) **then**

3 T ← **Delete** Node<q_{parent}>, Edge<q_{parent}, q_{child}> and Edge<q_{parent}, $q_{ancestor}$> from T

4 q_{parent} ← $q_{ancestor}$

5 $q_{ancestor}$ ← Position of Parent Node of $q_{ancestor}$ in T

6 **While Not** $q_{ancestor}$ = ***Null*** **do**

7 **If Not** *isTrapped*($q_{ancestor}$, q_{child}, C) **then**

8 T ← **Delete** Node<q_{parent}> and Edge<q_{parent}, $q_{ancestor}$> from T

9 q_{parent} ← $q_{ancestor}$

10 $q_{ancestor}$ ← Position of Parent Node of $q_{ancestor}$ in T

11 **Else**

12 ***Break***

13 T ← **Insert** Edge<q_{parent}, q_{child}> to T

14 **Else**

15 T ← **Insert** Node<q_{child}> and Edge<q_{child}, q_{parent}> to T

End *triangularRewiring* **Procedure from** *Extend*, *Connect*

Algorithm 7 shows the pseudocode of the *Triangular Rewiring* method applicable in the *Extend* (A5) and *Connect* (A6) methods of the RRT-Connect algorithm. When inserting a new node and edge in T_a or T_b in *Extend* method (A8), when a tree T_{merged} (P_{merged}) in which T_a and T_b trees are merged in *Connect* method is created (A9), rewiring is performed on the tree T.

In the *Extend* or *Connect* method, q_{new} (or q_{newA} or q_{newB}) as a q_{child} and q_{near} as a candidate for the parent node of the node are insert. And from q_{parent}, the parent node of the node (a second ancestor node candidate based on q_{child}) is called a $q_{ancestor}$. Next, it is determined whether an obstacle exists between $q_{ancestor}$ and q_{child} (by *isTrapped* function), and if there is an obstacle (*True*), the *Triangular Rewiring* process is skipped, and q_{child} is inserted into the child node of q_{parent} in T so that the contents of the *Extend* and *Connect* method from the RRT-Connect algorithm are the same. If there is no obstacle (*False*), the *Triangular Rewiring* process proceeds.

The *Triangular Rewiring* process is as follows: Delete node where position q_{parent} and the edges between q_{child} and $q_{ancestor}$ nodes connected to the q_{parent}. In other words, it disconnects the existing q_{parent} and q_{child}, and prepares to connect the q_{child} to the $q_{ancestor}$, the parent node candidate of the q_{child}. Again, q_{parent} becomes its parent node $q_{ancestor}$, and the $q_{ancestor}$ becomes the parent node of $q_{ancestor}$. Then, as previously done, determine an obstacle exists between $q_{ancestor}$ and q_{child} (by *isTrapped* function). This iterative process is continued until no $q_{ancestor}$ exists (When the parent node of the previous $q_{ancestor}$ does

not exist, that is, when $q_{ancestor}$ is q_{start}) or an obstacle exists between q_{child} and $q_{ancestor}$. Then, in the tree T, the last created q_{parent} is inserted as the parent node of q_{child}.

4.2. Mathematical Modeling of Proposed Triangular Inequality based RRT-Connect Algorithm

This section introduces the mathematical modeling of the proposed Triangular Inequality based RRT-Connect algorithm. As a result, it is shown that the proposed algorithm is more efficient in terms of optimality than the RRT-Connect algorithm. For reference, this mathematical modeling is based on a two-dimensional Euclidean space.

4.2.1. Basic Mathematical Modeling of the RRT Algorithm

This paragraph introduces basic mathematical modeling in the RRT algorithm. The following Equation 1-5 shows that the coordinate value of q_{new} is calculated from the coordinate value of q_{rand} in the RRT algorithm:

$$d = \sqrt{(q_{rand}.x - q_{near}.x)^2 + (q_{rand}.y - q_{near}.y)^2}, \tag{1}$$

$$|q_{rand}.x - q_{near}.x| : |q_{new}.x - q_{near}.x| = d : \lambda, \tag{2}$$

$$|q_{rand}.y - q_{near}.y| : |q_{new}.y - q_{near}.y| = d : \lambda. \tag{3}$$

Equation 2-3 shows the relationship between d and λ in Equation 1 through the similarity ratios from q_{near} to q_{rand} and from q_{near} to q_{new}:

$$\therefore q_{new}.x = \begin{cases} q_{rand}.x, & d \le \lambda \\ |(\lambda |q_{rand}.x - q_{near}.x|/d) + q_{near}.x|, & d > \lambda' \end{cases} \tag{4}$$

$$\therefore q_{new}.y = \begin{cases} q_{rand}.y, & d \le \lambda \\ |(\lambda |q_{rand}.y - q_{near}.y|/d) + q_{near}.y|, & d > \lambda' \end{cases} \tag{5}$$

Through these equations, the x and y coordinate values of q_{new} can be derived as shown in Equation 4-5. In this case, $d \le \lambda$ refers to a case where q_{rand} exists in a position inside the λ radius.

The following Equation 6-7 shows that the path distance $\mathbb{d}_n(q_i)$ between an arbitrary node q_i and its parent node in RRT algorithm:

$$D(q_i, \xi(q_i)) = \sqrt{(\xi(q_i).x - q_i.x)^2 + (\xi(q_i).y - q_i.y)^2}, \tag{6}$$

$$\therefore \mathbb{d}_n(q_i) = D\left(\xi^n(q_i), \xi^{n+1}(q_i)\right). \tag{7}$$

q_i refers to the i-th inserted arbitrary node and has the x and y coordinate values of the node as an element. The ξ function receives an arbitrary node as a variable and returns the parent node of this node. In Equation 6, the distance between an arbitrary node q_i and its parent node is obtained, which can be summarized as a function $\mathbb{d}_n$ as in Equation 7. At this time, n is the distance between the ancestor node and the parent node of the ancestor node, based on an arbitrary node. That is, the ξ function to the power of n ($n \ge 0$) can be represented as $\xi^n(q_i) := (\overbrace{\xi \circ \xi \circ \dots \circ \xi}^{n})(q_i)$, and when n is 0, $\xi^0(q_i) := q_i$ holds.

In addition, suppose that starting with an arbitrary node q_i and going back to the parent node to find the distance between the n-th ancestor node and the (n + 1)-th ancestor node, this can be represented as $D\left(\xi^n(q_i), \xi^{n+1}(q_i)\right)$.

Equation 8-9 shows that the total path distance $\mathbb{D}_R$ from the start position q_{start} to the goal position q_{goal} by RRT algorithm:

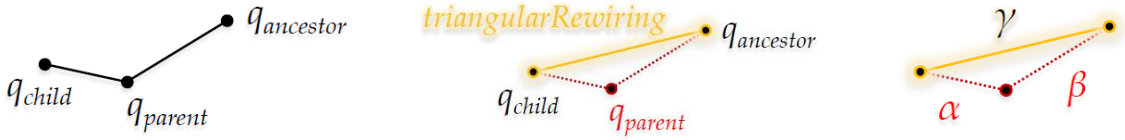
$$\xi^{\delta+1}(q_{goal}) = q_{start}, \tag{8}$$

$$\therefore \mathbb{D}_R = \sum_{n=0}^{\delta} \mathbb{d}_n(q_{goal}). \tag{9}$$

Equation 8 shows when the (δ + 1)-th ancestor node from q_{goal} is q_{start}, where δ is the upper limit of $\sum_{n=0}^{\delta} \mathbb{d}_n(q_{goal})$ in obtaining the total distance $\mathbb{D}_R$ of Equation 9. In other words, Equation 9 is the sum of the distances from q_{goal} to the first ancestor node (parent node) of q_{goal}, and the distance from the first ancestor node (parent node) of q_{goal} to the second ancestor of q_{goal}, …, and (δ – 1)-th ancestor node to the δ-th ancestor node (q_{start}).

4.2.2. Mathematical Modeling of Proposed Triangular Rewiring Method for the RRT-Connect Algorithm

This paragraph shows the mathematical modeling when the *Triangular Rewiring* method (A7) of Section 4.1 is applied to the RRT algorithm and the mathematical modeling when applied to the RRT-Connect algorithm.



(**a**) (**b**) (**c**)

Figure 5. Abstract process of the *Triangular Rewiring* method: (**a**) Example for tree; (**b**) After rewired between q_{child} and $q_{ancestor}$; (**c**) At this time, the α is a distance between q_{child} and q_{parent}, the β is a distance between q_{parent} and $q_{ancestor}$, and the γ is a distance between q_{child} and $q_{ancestor}$.

Figure 5 shows an abstract process of the Triangular Rewiring method. As shown in Figure 5 (a), if parent node of q_{child} is q_{parent} and parent node of q_{parent} is $q_{ancestor}$, and $q_{ancestor}$ is the second ancestor of q_{child}, this can be represented as Equation 10:

$$q_{ancestor} = \xi(q_{parent}) = \xi^2(q_{child}). \tag{10}$$

If the distances of the edge connecting each node are, the α between q_{child} and q_{parent}, the β between q_{parent} and $q_{ancestor}$, and the γ between q_{child} and $q_{ancestor}$ as shown in Figure 5 (c), it can be represented as Equation 11 using the principle of the triangular inequality:

$$\alpha + \beta \geq \gamma. \tag{11}$$

The following Equations 12-13 show the distance relation between ancestor nodes of q_{child}:

$$D(q_{child}, \xi(q_{child})) = \alpha,\ D(\xi(q_{child}), \xi^2(q_{child})) = \beta,\ D(q_{child}, \xi^2(q_{child})) = \gamma, \tag{12}$$

$$\therefore D(q_{child}, \xi(q_{child})) + D(\xi(q_{child}), \xi^2(q_{child})) \geq D(q_{child}, \xi^2(q_{child})), \tag{13}$$

Equation 12 can be summarized to Equation 13 by substituting Equation 10, which represents the relationship between the n-th ancestor nodes of q_{child}, with the distance as Equation 6 in Equation 11, which represents the distance between each node as a triangular inequality.

The following Equation 14-20 shows that the path of the RRT algorithm applying the *Triangular Rewiring* method is always shorter or equal to the path planned by the original RRT algorithm. Equation 14 shows the sequence index k_j for comparing the distance $\mathbb{u}$ when the Triangular Rewiring method is applied with the distance $\mathbb{d}$ when this method is not applied:

$$k_j = \tau_j + k'_j,\ k'_j = \begin{cases} 0,\ j = 0 \\ k_{j-1} + 1,\ j \geq 1' \end{cases} \tag{14}$$

In this case, j is a sequence index for $\mathbb{u}$. That is, k_j can be said to be a sequence index for $\mathbb{d}$. Currently, τ_j is the number of times that rewire occurs in the j-th.

If this is summarized by Equation 6 for a distance based on an arbitrary node q_i, it is as Equation 15. For example, as shown in Figure 5, if j is 0 and 1 rewire occurs ($\tau_0 = 1$), it can be represented in combination with the distance relationship of Equation 13 for q_{child}, as in Equation 16:

$$\mathbb{u}_{k_j}(q_i) = D\left(\xi^{k'_j}(q_i), \xi^{k_j+1}(q_i)\right), \tag{15}$$

$$\mathbb{d}_0(q_{child}) + \mathbb{d}_1(q_{child}) = \sum_{n=0}^{1} \mathbb{d}_n(q_{child}) \geq \mathbb{u}_{k_0=1}(q_{child}). \tag{16}$$

The result of Equation 16 can be generalized as shown in Equation 17:

$$\therefore \sum_{n=0}^{k_j} \mathbb{d}_n(q_i) \geq \mathbb{u}_{k_j}(q_i). \tag{17}$$

For $\mathbb{d}$ based on an arbitrary node q_i, it is shown that the total distance $\sum_{n=j}^{k_j} \mathbb{d}_n$ from the j-th to k_j-th arbitrary sequence index is always longer or equal to the distance $\mathbb{u}_{k_j}$ of the k_j-th sequence index. That is, in an arbitrary path, it can be confirmed that the distance $\mathbb{u}$ rewired by the T*riangular Rewiring* method is at least the equal (If the distances of $\mathbb{d}$ and $\mathbb{u}$ are the same, the rewired line segments are on a straight line) or always shorter in distance than $\mathbb{d}$ when not rewired.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 19 November 2020 doi:10.20944/preprints202011.0494.v1

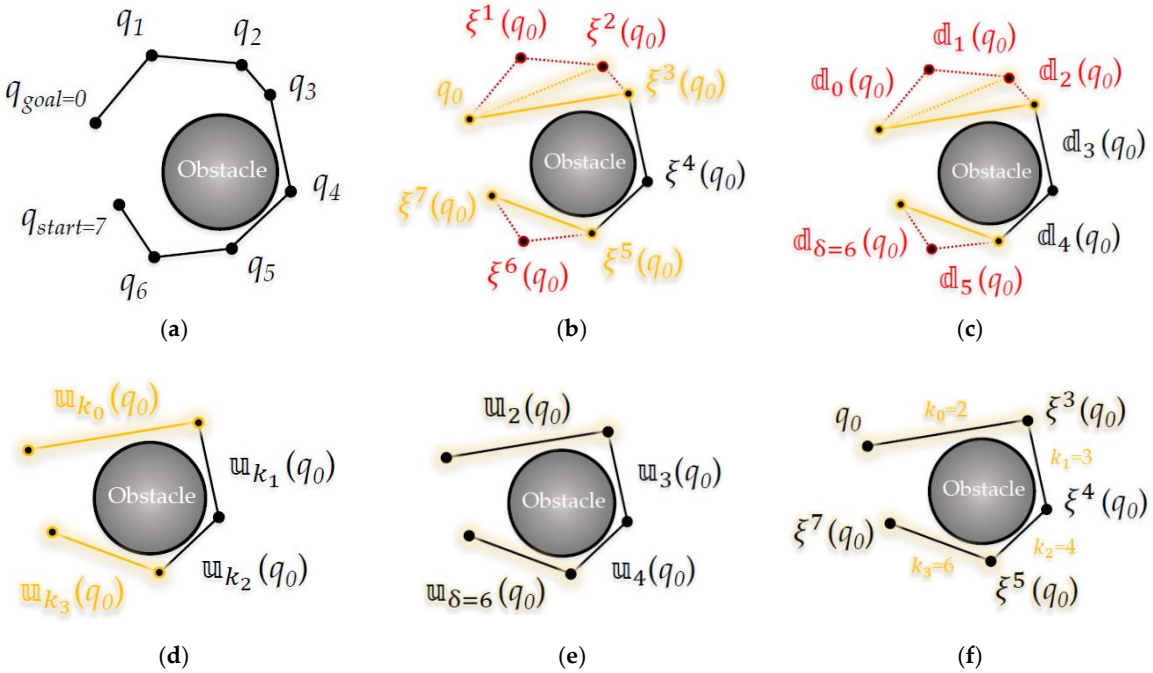


Figure 6. Detailed process of the *Triangular Rewiring* method: (**a**) Each node q for index i (at this time, q_{start} is same as q_7 and q_{goal} is same as q_0); (**b**) Represent each node by n-th ancestor ξ^n of q_0; (**c**) Each distance $\mathbb{d}_n$ between the n-th and $(n+1)$-th ancestor nodes of q_0; (**d**) When the *Triangular Rewiring* method is applied and rewired by the distance $\mathbb{u}_{k_j}$; (**e**) Represent as the value of k_j; (**f**) Represent each node by n-th ancestor ξ^n of q_0 after method is applied.

Figure 6 shows the *Triangular Rewiring* process for the path from q_{start} to q_{goal} based on Equation 10-17 (at this time, it is assumed that the node of the path shown in the figure does not positioned in a straight line). As shown in Figure 6 (b), a total of 2 rewires occurred ($\tau_0 = 2$) between q_0 and q_3 ($\xi^3(q_0)$), and a total of 1 rewire occurred ($\tau_3 = 1$) between q_5 ($\xi^5(q_0)$) and q_7 ($\xi^7(q_0)$). In that case, as shown in Figure 6 (e), k_0 is 2, k_1 is 3, k_2 is 4, and k_3 is 6 by Equation 14.

Comparing Figure 6 (c) and Figure 6 (e), according to Equation 17, the rewired distance $\mathbb{u}_2(q_0)$ is shorter than the total distance $\sum_{n=0}^{2}\mathbb{d}_n(q_0)$ from $\mathbb{d}_0$ to $\mathbb{d}_2$, the rewired distance $\mathbb{u}_6(q_0)$ is shorter than the total distance $\sum_{n=5}^{6}\mathbb{d}_n(q_0)$ from $\mathbb{d}_5$ to $\mathbb{d}_6$. That is, when comparing before applying the *Triangular Rewiring* method in Figure 6 (a) and after applied this method in Figure 6 (f), applied the method looks shorter.

The following Equation 18-19 shows the total distance $\mathbb{D}_R$ when the *Triangular Rewiring* method is not applied and the total distance $\mathbb{U}_R$ when the method is applied, for an arbitrary path (start position: q_{start}, goal position: q_{goal}), as shown in Figure 6:

$$k_\varphi = \delta, \tag{18}$$

$$\mathbb{D}_R = \sum_{n=0}^{\delta}\mathbb{d}_n(q_{goal}) = \sum_{j=0}^{\varphi}\sum_{n=k'_j}^{k_j}\mathbb{d}_n(q_{goal}),\ \mathbb{U}_R = \sum_{j=0}^{\varphi}\mathbb{u}_{k_j}(q_{goal}), \tag{19}$$

Equation 18 shows the upper limit when the index n of d is δ in Equation 8, and when this is substituted into the sequence index k_j, if k_j is δ, j becomes φ. In that case, as in Equation 19, $\mathbb{D}_R$ is to compare the $\sum_{n=0}^{\delta}\mathbb{d}_n(q_{goal})$ shown in Equation 9 with $\mathbb{U}_R$, reflecting the sequence k_j, it can be represented as $\sum_{j=0}^{\varphi}\sum_{n=k'_j}^{k_j}\mathbb{d}_n(q_{goal})$, and $\mathbb{U}_R$ can be represented as $\sum_{j=0}^{\varphi}\mathbb{u}_{k_j}(q_{goal})$.

The following Equation 20 shows that the equation summarized in Equation 19 is substituted into Equation 17:

$$\therefore \mathbb{D}_R \geq \mathbb{U}_R. \tag{20}$$

Finally, as can be confirm in Equation 20, $\mathbb{U}_R$ as the result of applying the *Triangular Rewiring* method to the distance of an arbitrary path (start position: q_{start}, goal position: q_{goal}) is at least equal (If the distances of $\mathbb{D}$ and $\mathbb{U}$ are the same, when the rewired line segments are on a straight line) to or always shorter than $\mathbb{D}_R$ as the result of this method is not applied.

The following Equation 21-23 shows the total distance $\mathbb{D}_A$ of the path from the start position (root node) of T_a to the last (inserted node) position q_{newA}, and the total distance $\mathbb{U}_A$ when the *Triangular Rewiring* method is applied to the path. Also, it shows that $\mathbb{U}_A$ is at least equal to or always shorter than $\mathbb{D}_A$:

$$\xi^{\delta_A+1}(q_{newA}) = q_{start},\ k_{\varphi_A} = \delta_A, \tag{21}$$

$$\mathbb{D}_A = \sum_{j=0}^{\varphi_A}\sum_{n=k'_j}^{k_j} \mathbb{d}_n(q_{newA}),\ \mathbb{U}_A = \sum_{j=0}^{\varphi_A} \mathbb{u}_{k_j}(q_{newA}), \tag{22}$$

$$\therefore \mathbb{D}_A \geq \mathbb{U}_A. \tag{23}$$

The following Equation 24-26 shows the total distance $\mathbb{D}_B$ of the path from the start position (root node) of T_b to the last (inserted node) position q_{newB}, and the total distance $\mathbb{U}_B$ when the *Triangular Rewiring* method is applied to the path. Also, it shows that $\mathbb{U}_B$ is at least equal to or always shorter than $\mathbb{D}_B$:

$$\xi^{\delta_B+1}(q_{newB}) = q_{goal},\ k_{\varphi_B} = \delta_B, \tag{24}$$

$$\mathbb{D}_B = \sum_{j=0}^{\varphi_B}\sum_{n=k'_j}^{k_j} \mathbb{d}_n(q_{newB}),\ \mathbb{U}_B = \sum_{j=0}^{\varphi_B} \mathbb{u}_{k_j}(q_{newB}), \tag{25}$$

$$\therefore \mathbb{D}_B \geq \mathbb{U}_B. \tag{26}$$

Therefore, Equations 21 and 24 can be derived from Equations 8 and 18, Equations 22 and 25 from Equation 19, and Equations 23 and 26 from Equation 20.

As a result, the following Equations 27-28 show that RRT-Connect to which the proposed *Triangular Rewiring* method is applied is at least the same or better in terms of optimality (total path distance) than RRT-Connect algorithm to which the method is not applied:

$$\mathbb{D}_R = \mathbb{D}_A + \mathbb{D}_B + D(q_{newA}, q_{newB}),\ \mathbb{U}_R \leq \mathbb{U}_A + \mathbb{U}_B + D(q_{newA}, q_{newB}), \tag{27}$$

$$\therefore \mathbb{D}_R \geq \mathbb{D}_A + \mathbb{D}_B \geq \mathbb{U}_A + \mathbb{U}_B \geq \mathbb{U}_R. \tag{28}$$

$\mathbb{D}_R$ (Eq. 9), which refers to the total distance of the RRT-Connect algorithm path when the *Triangular Rewiring* method is not applied, represented by the sum of the distance $\mathbb{D}_A$ of the partial path P_a (Eq. 22), the distance $\mathbb{D}_B$ of the partial path P_b (Eq. 25), and the distance $D(q_{newA}, q_{newB})$ between q_{newA} and q_{newB} as shown in Equation 27.

And $\mathbb{U}_R$ (Eq. 19), which refers to the total distance of the RRT-Connect algorithm path when the *Triangular Rewiring* method is applied, represented by equal to or shorter than the sum of the distance $\mathbb{U}_A$ of the partial path P_a of the RRT-Connect (Eq. 22), the distance $\mathbb{U}_B$ of the partial path P_b (Eq. 25), and the distance $D(q_{newA}, q_{newB})$ between q_{newA} and q_{newB} as shown in Equation 27.

In that case, Equation 28 shows that $\mathbb{U}_R$ is at least equal to or shorter than $\mathbb{D}_R$ in the RRT algorithm summarized in Equation 20, and it is also efficiently used in the RRT-Connect algorithm.

4.3. Pseudocode of Proposed Extend Method for the RRT-Connect Algorithm

This section introduces the *Extend* method in the proposed Triangular Inequality based RRT-Connect algorithm. This proposed *Extend* method (A8) replaces the *Extend* method (A6) in pseudocode of the RRT-Connect algorithm (A5).

Algorithm 8. Pseudocode of the Proposed *Extend* Method for the RRT-Connect Algorithm.

Input:
T_a ← Tree T_a from *RRT-Connect*
T_b ← Tree T_b from *RRT-Connect*
q_{newB} ← Position q_{newB} from *RRT-Connect*
q_{rand} ← Position q_{rand} from *RRT-Connect*
λ ← Step Length λ from *RRT-Connect*
C ← Position Set C from *RRT-Connect*
Output:
f_{trap} ← Result of Boolean f_{trap}
T_a ← Result of Tree T_a **// Return by Reference**
T_b ← Result of Tree T_b **// Return by Reference**
q_{newB} ← Result of Position q_{newB} **// Return by Reference**
Initialize:
f_{trap} ← ***False***

Begin *Extend* **Procedure from** *RRT-Connect*
1 q_{near} ← **Find** Position of Nearest Node in T_a from q_{rand}
2 **If Not** *isInside*(q_{near}, q_{rand}, λ) **then**
3 q_{newA} ← Position of Intersection Point between Line Segment connecting q_{rand} and q_{near}, and Circle with Radius λ centered at q_{near} **// 2D:** Circle, **3D:** Sphere, **…**
4 **Else**
5 q_{newA} ← q_{rand}
6 **If** *isTrapped*(q_{newA}, q_{near}, C) **then**
7 f_{trap} ← ***True***
8 **Else**
9 T_a ← *triangularRewiring*(q_{newA}, q_{near}, T_a, C)
10 q_{near} ← **Find** Position of Nearest Node in T_b from q_{newA}
11 **If** *isInside*(q_{near}, q_{newA}, λ) **then**
12 q_{newB} ← q_{near}
13 **Else**
14 q_{newB} ← Position of Intersection Point between Line Segment connecting q_{newA} and q_{near}, and Circle with Radius λ centered at q_{near} **// 2D:** Circle, **3D:** Sphere, **…**
15 **While Not** *isTrapped*(q_{newB}, q_{near}, C) **do**
16 T_b ← *triangularRewiring*(q_{newB}, q_{near}, T_b, C)
17 **If Not** *isInside*(q_{newA}, q_{newB}, λ) **then**
18 q_{near} ← q_{newB}
19 q_{newB} ← Position of Intersection Point between Line Segment connecting q_{newA} and q_{near}, and Circle with Radius λ centered at q_{near} **// 2D:** Circle, **3D:** Sphere, **…**
20 **Else**
21 ***Break***
End *Extend* **Procedure from** *RRT-Connect*

Algorithm 8 is the application of the *Triangular Rewiring* method (A7) to the original *Extend* method (A5) of the RRT-Connect algorithm. Compared to the original *Extend* method, in lines 9 and

16, the part where a node is newly inserted in the tree is inserted through the *Triangular Rewiring* method. Other than that, the contents are the same as the original *Extend* method.

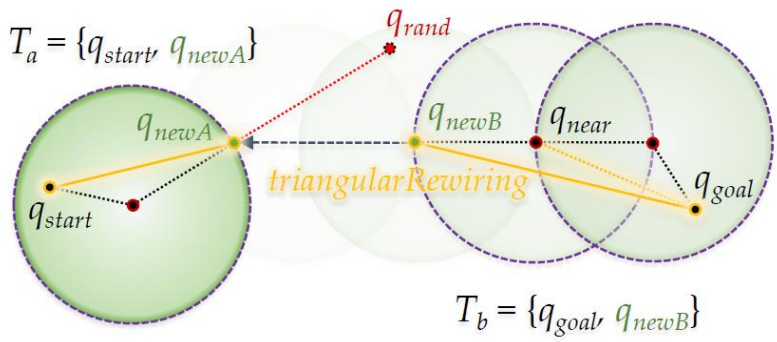


Figure 7. Proposed *Extend* method for the RRT-Connect algorithm.

Figure 7 shows the application of the *Triangular Rewiring* method to Figure 3, which shows the *Extend* method of the RRT-Connect algorithm. In T_a, q_{newA} and q_{start} are rewired, and q_{near} and q_{goal}, and q_{newB} and q_{goal} are rewired sequentially in the process of extending from T_b to T_a.

4.4. Pseudocode of Proposed Connect Method for the RRT-Connect Algorithm

This section introduces the *Connect* method in the proposed Triangular Inequality based RRT-Connect algorithm. This proposed *Connect* method (A9) replaces the *Connect* method (A7) in pseudocode of the RRT-Connect algorithm (A5).

Algorithm 9. Pseudocode of the Proposed *Connect* Method for the RRT-Connect Algorithm.

Input:
P_{reach} ← Path P_{reach} from *RRT-Connect*
T_a ← Tree T_a from *RRT-Connect*
T_b ← Tree T_b from *RRT-Connect*
q_{newB} ← Position q_{newB} from *RRT-Connect*
λ ← Step Length λ from *RRT-Connect*
Output:
f_{reach} ← Result of Boolean f_{reach}
P_{reach} ← Result of Path P_{merged} **// Return by Reference**
Initialize:
f_{reach} ← ***False***

Begin *Connect* **Procedure from** *RRT-Connect*
1 **If** *isInside*(q_{newA}, q_{newB}, λ) **then**
2 P_a ← Path from Root Node [q_{start}] to Last Inserted Node [q_{newA}] in T_a
3 P_b ← Path from q_{newB} to Root Node [q_{goal}] in T_b
4 $P_{connect}$ ← Path from Last Inserted Node [q_{newA}] in T_a to q_{newB} in T_b
5 T_{merged} ← Tree Structure with **Merge Path** P_a to P_b via $P_{connect}$
// **1st Insert:** q_{start}, …, ***n*-th Insert:** q_{newA}, **(*n* + 1)-th Insert:** q_{newB}, …, **Last Insert:** q_{goal} **to** T_{merged}
6 **For** i ← Inserted Index of q_{newA} in T_{merged} **to** (Number of Node in T_{merged}) – 1 **do**
7 q_{new} ← (i – 1)-th Inserted Node in T_{merged}
8 q_{near} ← i-th Inserted Node in T_{merged}
9 T_{merged} ← *triangularRewiring*(q_{new}, q_{near}, T_{merged}, C)

10 P_{merged} ← Path from Root Node [q_{start}] to Last Inserted Node [q_{goal}] in T_{merged}

11 f_{reach} ← ***True***

End *Connect* **Procedure from** *RRT-Connect*

Algorithm 9 is an application of the *Triangular Rewiring* method (A7) to the *Connect* method (A6) of the RRT-Connect algorithm. Compared to the original *Connect* method, it has been changed to applying the method to the merged tree by considering the *Triangular Rewiring* method when merging the path, which is lines 5-10. Other than that, the contents are the same as the original *Connect* method.

When paths P_a and P_b are merged in a tree structure of line 5, nodes on the path are inserted in the order of P_a, $P_{connect}$, and P_b in the merged tree T_{merged}. That is, in T_{merged}, the root node becomes q_{start}, and when the n-th inserted node at a certain point is q_{newA}, which is the last inserted node of T_a, the (n + 1)-th inserted node becomes q_{newB}, which is the last inserted node of T_b. Also, the last inserted node of T_{merged} becomes q_{goal}.

Then, *Triangular Rewiring* method is applied to this T_{merged}. Since it is applied to the tree itself, it determined whether rewire is possible for all nodes inserted in the tree, and if possible, rewires and updates the tree. However, since each node from T_a to T_b is inserted into T_{merged}, it is not necessary to rewire T_a for which the *Triangular Rewiring* process has already been performed. Therefore, the *Triangular Rewiring* process proceeds in the direction of T_b from the q_{newA} sequence inserted in T_{merged}. At this time, if q_{newA} is the i-th inserted node, the first node pair to be determined is (i - 1)-th node q_{new} (as q_{child}) and i-th node q_{near} (as q_{parent}). When all the nodes inserted in T_{merged} are determined, the tree structure T_{merged} is converted to the path P_{merged}, and the method is terminated (*True*).

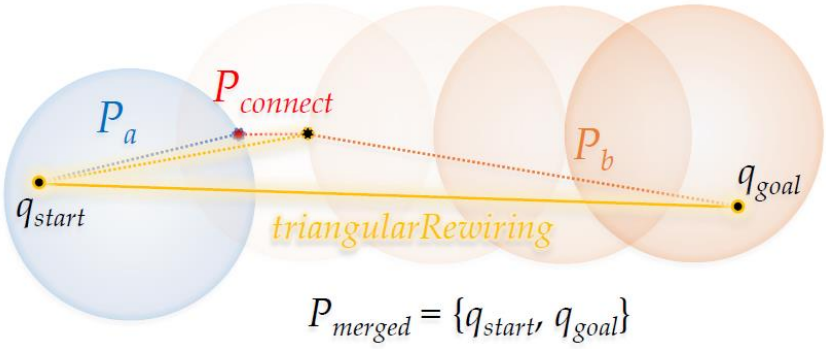


Figure 8. Proposed *Connect* method for the RRT-Connect algorithm.

Figure 8 shows the *Triangular Rewiring* method applied to Figure 4, which shows the *Connect* method of the RRT-Connect algorithm. When the paths P_a and P_b created from the trees T_a and T_b are merged and the *Triangular Rewiring* method is applied (assuming there is no obstacle between q_{start} and q_{goal}), the result is a P_{merged} in which q_{start} and q_{goal} are connected in a straight line.

4.4. Process of Proposed Triangular Inequality based RRT-Connect Algorithm

Figure 9 in this section shows the path planning process of the proposed algorithm by applying the *Triangular Rewiring* method to the *Extend* and *Connect* methods of the RRT-Connect algorithm.

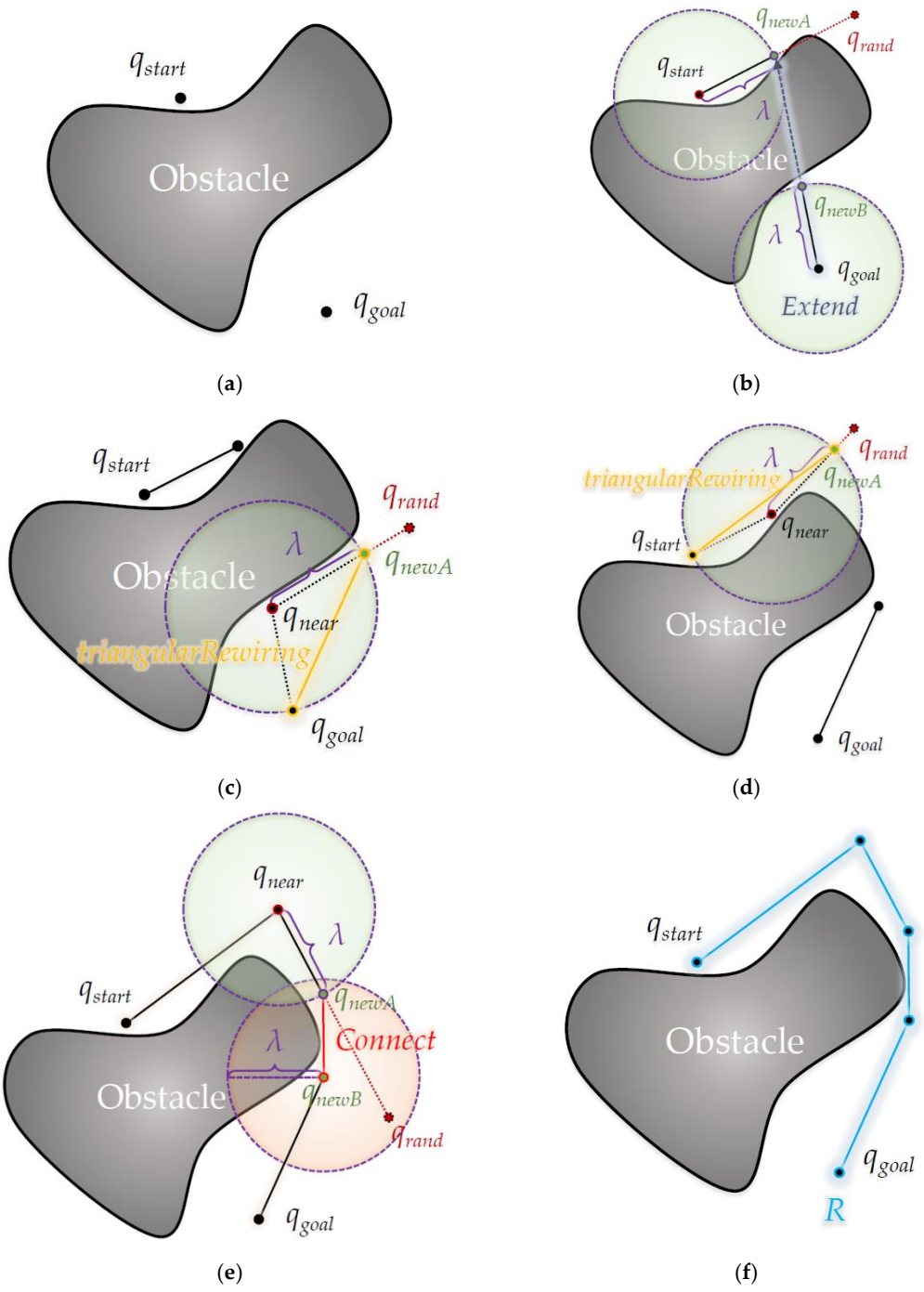


Figure 9. Detailed process of the proposed algorithm: (**a**) Start position q_{start} from tree T_a and goal position q_{goal} from tree T_b; (**b**) Create q_{newA} nearest to T_a from 1st random sampling position q_{rand}, and create q_{newB} from q_{goal} nearest to T_b; (**c**) Create new q_{newA} from q_{near} nearest to T_b from 2nd random sampling position q_{rand}, and rewire between the q_{newA} and q_{goal} the ancestor of the q_{newA}; (**d**) Create new q_{newA} from q_{near} nearest to T_a from 3rd random sampling position q_{rand}, and rewire between the q_{newA} and q_{start} the ancestor of the q_{newA}; (**e**) Create new q_{newA} from q_{near} nearest to T_a from 5th random sampling position q_{rand}, and connect between the q_{newA} and q_{newB} nearest to T_b from the q_{newA}; (**f**) Result of Path R from q_{start} to q_{goal}.

To do in Figure 9 is to plan a path from the start position q_{start} to the goal position q_{goal} through the proposed algorithm, as shown in Figure 9 (a).

In Figure 9 (b), the first random sample is generated at the q_{rand} position, and q_{newA} is created at a position separated by the length of λ from q_{start} in the direction of the position, and q_{newA} is extended

once by the length of λ in the direction of q_{newA} from q_{goal}. At this time, since there is no intermediate node between q_{newA} and q_{start}, the *Triangular Rewiring* process is skipped.

In Figure 9 (c), a second random sample is generated at the q_{rand} position, and in the direction of the position, q_{newA} is updated at a location separated by λ length from the nearest node q_{near} in the tree and rewired between q_{newA} and q_{goal}. In this case, since the tree on the opposite side collides with an obstacle to extend in the q_{newA} direction, the *Extend* process is skipped. Also, it is assumed that *Swap* occurs between T_a with initial q_{start} as the root node and T_b with initial q_{goal} as the root node between each figure.

In Figure 9 (d), as shown in Figure 9 (c), a third random sample is created at the q_{rand} position, and at a position that is separated by the length of λ in the position direction, at the node q_{near} nearest among the nodes in the tree in the position direction, It shows updating q_{newA} to a position that is the length of λ and rewires it between q_{newA} and q_{start}. In this case, also since it collides with an obstacle to extend in the direction q_{newA} from the tree on the opposite side, the *Extend* process is skipped.

In Figure 9 (e), the fifth random sample is generated at the q_{rand} position, and q_{newA} is located at a position separated by the length of λ in the direction of the position, and q_{newA} at a position separated by the length of λ from the nearest node q_{near} among nodes in the tree toward the position It is shown that updating is and that q_{newA} is merged into one tree through the *Connect* process because q_{newA} has entered the range with the center of q_{newB} and the radius of λ. It is assumed that the fourth random sample between Figure 9 (d) and Figure 9 (e) is generated inside the obstacle, so that the q_{newA} generation process is skipped. The last Figure 9 (f) shows the result of path R created as a merged tree by *Connect* as shown in Figure 9 (e).

5. Experimental Results

To verify the performance of the proposed Triangular Inequality based RRT-Connect algorithm in this paper, the RRT algorithm, the RRT-Connect algorithm, and the proposed algorithm are compared with each other in various environment maps shown in the experimental environment through the simulator.

Each algorithm was implemented based on the pseudocode shown in Chapters 2 to 4, and the compared performance measure is the average, ***Number of sampling***, ***Optimal path distance*** (pixels), and ***Convergence time*** (milliseconds) for all trials for 50 times (from the start point to the goal position until the first path plan is completed). Among the performance measures, as the number of sampling decreases, the cost of recalculation in a dynamic environment is decreases, and the optimal path distance is a measure of the optimality of path planning algorithm.

5.1. Experimental Environment

This section introduces the environment map used in the simulation, and the simulator used in the simulation with the performance of the computer.

The following Figure 10 shows the eight environmental maps used in the experiment. The green circle (S) means to the start position, the purple circle (G) means to the goal position, and the black polygon on the yellow (blue in the analysis of the experimental results) border means to the obstacle. And all maps are 600 (horizontal) * 600 (vertical) pixels.

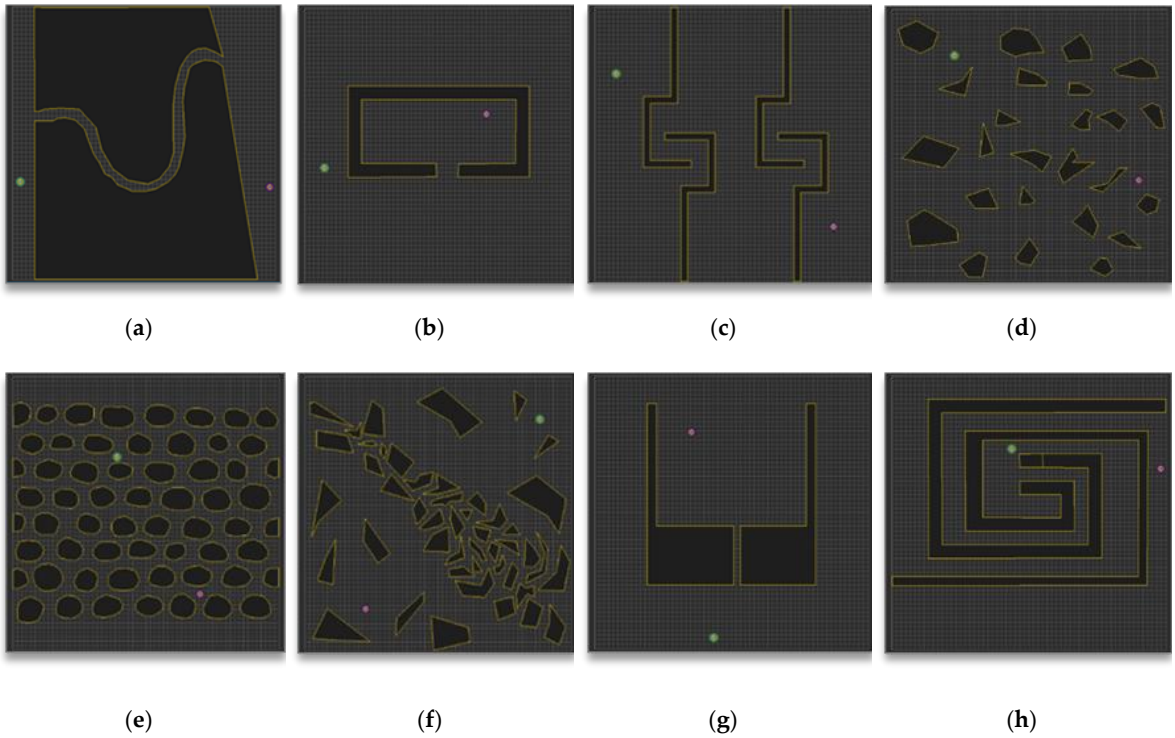


Figure 10. Maps for experiment: (**a**) Map 1; (**b**) Map 2; (**c**) Map 3; (**d**) Map 4; (**e**) Map 5; (**f**) Map 6; (**g**) Map 7; (**h**) Map 8.

To verify the performance of various path planning algorithms including the RRT algorithm, many environmental maps were considered and used [22-25]. It is important which environment map to use because the expected performance measure varies depending on the placement, shape, and other property of obstacles.

In this paper, to check the performance of the proposed algorithm, the eight maps shown in Figure 10 were benchmarked in the experimental environment of the paper [26] proposed by Jihee Han in 2017, and each map is expected to have the following features:

Map 1 in Figure 10 (a) seems to be an environment that it is easy to verify the completeness of the path planning algorithm. Map 2 in Figure 10 (b) seems to be an environment that it is also easy to verify the completeness of the path planning algorithm, and it is the environment mainly used to show the solution of the Local Minima problem in the Artificial Potential Field algorithm [25]. Map 3 in Figure 10 (c) seems to be an environment that is easy to verify the optimality and completeness of the path planning algorithm and is an environment that is unfavorable to random sampling path planning algorithms such as the RRT algorithm. Map 4 of Figure 10 (d) seems to be an environment that it is easy to verify the optimality and convergence time of the path planning algorithm, and the Cell Decomposition algorithm, which increases the computation cost as the angle of obstacle increases, is an unfavorable environment [27]. Map 5 in Figure 10 (e) seems to be an environment that it is also easy to verify the optimality and convergence time of the path planning algorithm, and for the same reason as Map 4, the Cell Decomposition algorithm is an unfavorable environment. Map 6 in Figure 10 (f) seems to be an environment that it is easy to verify the optimality, completeness and convergence time of the path planning algorithm, and it is an environment for comprehensively evaluating the performance. Map 7 in Figure 10 (g) seems to be an environment that it is easy to verify the completeness and optimality of the path planning algorithm, and for the same reason as Map 2, it can be seen as the environment used in the Artificial Potential Field algorithm. Lastly, Map 8 in Figure 10 (h) seems to be an environment that is easy to verify the completeness and convergence time of the path planning algorithm and is an environment unfavorable to random sampling path planning algorithms such as the RRT algorithm.

Since random sampling path planning algorithms such as the RRT algorithm rely on probabilistic completeness, the number of sampling and the convergence time are extremely increased and longer as there are narrow or fewer entrances of directions to the goal.

Table 1. Computer performance for simulation.

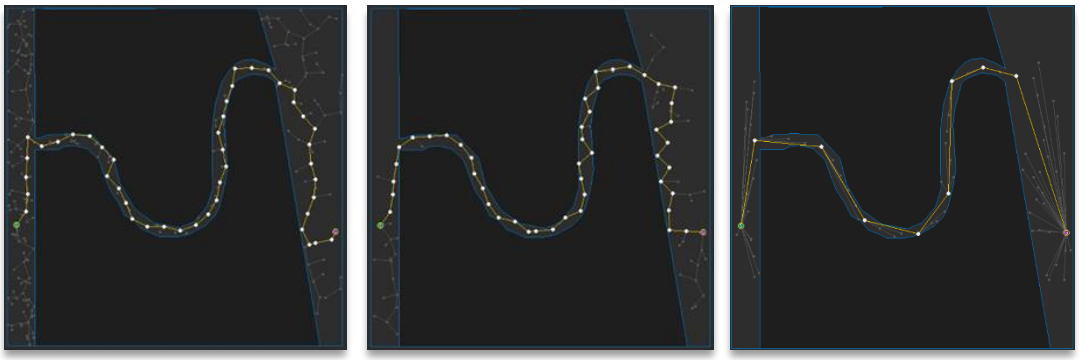
H/W	**Specification**
CPU	Intel Core i7-6700k 4.00GHz (8 CPUs)
RAM	32768MB (32GB DDR4)
VGA	Nvidia GeForce GTX 1080 (VRAM 8GB) SLI (x2)

Table 1 shows the performance of the computer used in the simulation. The simulator was developed in C# language (Microsoft Visual Studio Community 2019 version 16.1.6; Microsoft .NET Framework version 4.8.03752), and except for the visual part, only a single thread is used for calculation. The difference in convergence time may occur depending on the computer performance.

5.2. Analysis of Experimental Results of Each Map

This section checks the experimental results (on average, number of sampling, optimal path distance, convergence time) of each algorithm: RRT, RRT-Connect, the proposal in the eight environment maps (Fig. 10) presented in the experimental environment. In each map, a figure of the path planning result (of one trial) for each algorithm is shown, and the experimental results for the performance measure are shown numerically in a table (The figure for each algorithm is for one trial, not the average of repeated trials, and it may be different from the performance measure by visually and the average numerical performance measure of the repeated trials shown in the table. In particular, the number of sampling differs a lot).

In each path planning result figure, the white circle means the node on the path, and the yellow line segment means the edge between each node. Gray circles and segments are paths (trees) that are excluded during path planning. In each path planning result table, based on 100% of the RRT algorithm for each performance measure, the difference is indicated along with the value of the corresponding performance measure unit.



(**a**) (**b**) (**c**)

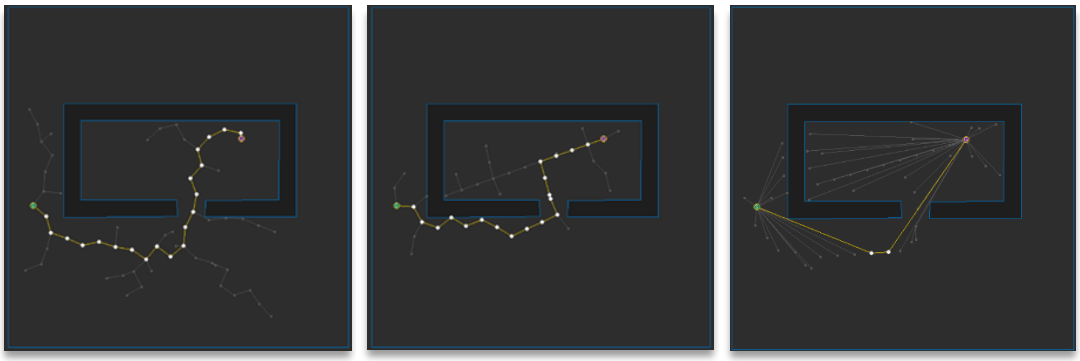
Figure 11. Experimental result of Map 1: (**a**) RRT; (**b**) RRT-Connect; (**c**) The proposed algorithm.

Figure 11 shows the path planning results of Map 1 among the environmental maps for each algorithm. In visually, the number of sampling looks similar for the RRT-Connect algorithm in Figure 11 (b) and the proposed algorithm in Figure 11 (c) compared to the RRT algorithm in Figure 11 (a), and the optimal path distance looks similar for all three algorithms.

Table 2. Experimental result of Map 1 (The parentheses to the right of each value are relative ratios based on RRT 100%).

Performance	RRT	RRT-Connect	Proposed Algorithm
Avg. Num. of Sampling [samples]	1216 *(100)*	729 *(60)*	823 *(68)*
Avg. Optimal Path Distance [px]	1341 *(100)*	1343 *(100)*	1200 *(89)*
Avg. Convergence time [ms]	12 *(100)*	7 *(58)*	10 *(83)*

Table 2 shows the result (for repeated trial 50 times) of path planning in Map 1 for each algorithm. The average number of sampling is the least in RRT-Connect algorithm at 60%, and the proposed algorithm is 68% compared to RRT algorithm, which is 8% less efficient than RRT algorithm compared to RRT-Connect algorithm. The average optimal path distance is the shortest for the proposed algorithm at 89% compared to the RRT algorithm, with little difference in the RRT-Connect algorithm at 100%, and 11% less efficient than the proposed algorithm. The average convergence time is the shortest by the RRT-Connect algorithm at 58% compared to the RRT algorithm, and the proposed algorithm is 83% compared to the RRT algorithm, which is 15% less efficient than the RRT algorithm.



(**a**) (**b**) (**c**)

Figure 12. Experimental result of Map 2: (**a**) RRT; (**b**) RRT-Connect; (**c**) The proposed algorithm.

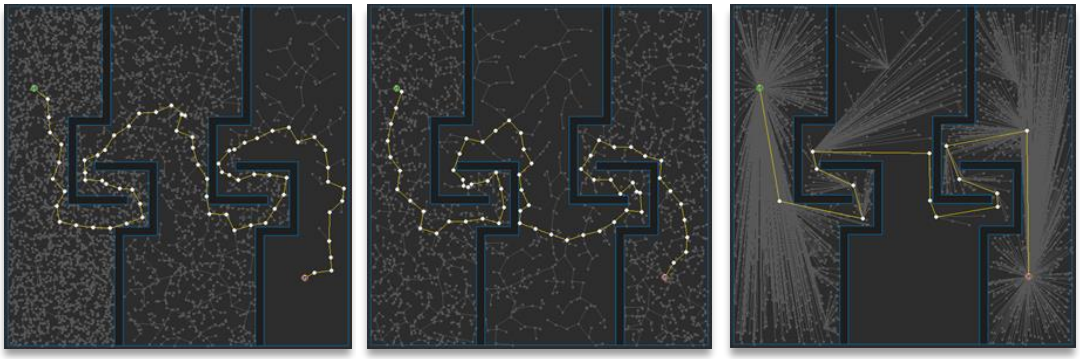
Figure 12 shows the path planning results of Map 2 among the environmental maps for each algorithm. In visually, the number of sampling looks similar for the RRT-Connect algorithm in Figure 12 (b) and the proposed algorithm in Figure 12 (c) compared to the RRT algorithm in Figure 12 (a), and the optimal path distance looks shortest for the proposed algorithm.

Table 3. Experimental result of Map 2 (The parentheses to the right of each value are relative ratios based on RRT 100%).

Performance	RRT	RRT-Connect	Proposed Algorithm
Avg. Num. of Sampling [samples]	271 *(100)*	101 *(37)*	113 *(42)*
Avg. Optimal Path Distance [px]	598 *(100)*	613 *(98)*	484 *(81)*
Avg. Convergence time [ms]	6 *(100)*	3 *(50)*	3 *(50)*

Table 3 shows the result (for repeated trial 50 times) of path planning in Map 2 for each algorithm. The average number of sampling is the least in RRT-Connect algorithm at 37%, and the proposed algorithm is 42% compared to RRT algorithm, which is 5% less efficient than RRT algorithm compared to RRT-Connect algorithm. The average optimal path distance of the proposed algorithm is the shortest at 81% compared to the RRT algorithm, and the RRT-Connect algorithm is 98%, 17% less efficient than the RRT algorithm compared to the proposed algorithm. The average convergence

time of the proposed algorithm and the RRT-Connect shows the same performance over the RRT algorithm.



(**a**) (**b**) (**c**)

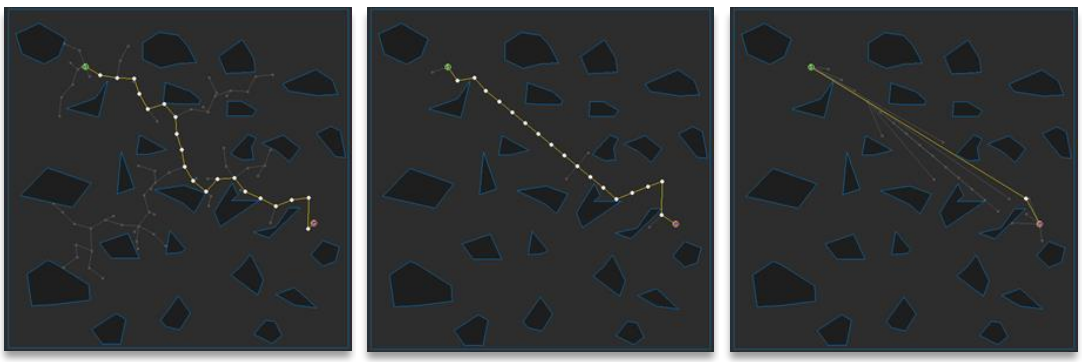
Figure 13. Experimental result of Map 3: (**a**) RRT; (**b**) RRT-Connect; (**c**) The proposed algorithm.

Figure 13 shows the path planning results of Map 3 among the environmental maps for each algorithm. In visually, the number of sampling looks similar for the RRT-Connect algorithm in Figure 13 (b) and the proposed algorithm in Figure 13 (c) compared to the RRT algorithm in Figure 13 (a), and the optimal path distance looks shortest for the proposed algorithm.

Table 4. Experimental result of Map 3 (The parentheses to the right of each value are relative ratios based on RRT 100%).

Performance	**RRT**	**RRT-Connect**	**Proposed Algorithm**
Avg. Num. of Sampling [samples]	6106 *(100)*	4574 *(75)*	4679 *(77)*
Avg. Optimal Path Distance [px]	1934 *(100)*	1871 *(97)*	1489 *(77)*
Avg. Convergence time [ms]	866 *(100)*	299 *(35)*	313 *(36)*

Table 4 shows the result (for repeated trial 50 times) of path planning in Map 3 for each algorithm. The average number of sampling is the least in RRT-Connect algorithm at 75%, and the proposed algorithm is 77% compared to RRT algorithm, which is 2% less efficient than RRT algorithm compared to RRT-Connect algorithm. The average optimal path distance of the proposed algorithm is the shortest at 77% compared to the RRT algorithm, and the RRT-Connect algorithm is 97%, 20% less efficient than the RRT algorithm compared to the proposed algorithm. The average convergence time is the least in RRT-Connect algorithm at 35%, and the proposed algorithm is 36% compared to RRT algorithm, 1% less efficient than RRT algorithm compared to RRT-Connect algorithm.



(**a**) (**b**) (**c**)

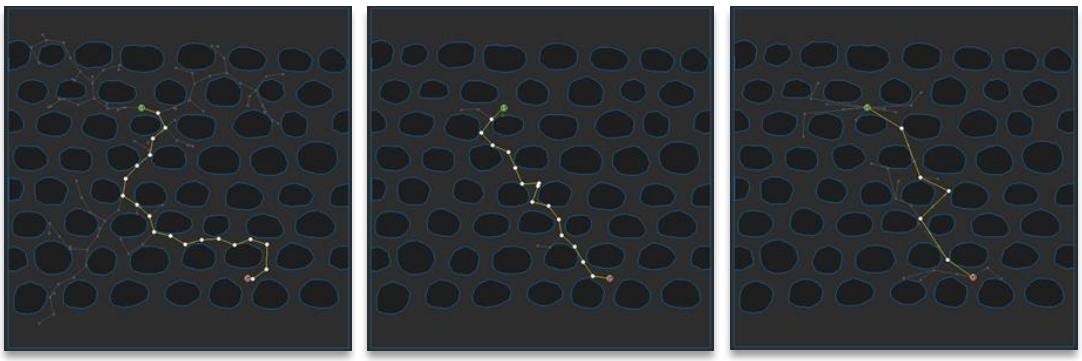
Figure 14. Experimental result of Map 4: (**a**) RRT; (**b**) RRT-Connect; (**c**) The proposed algorithm.

Figure 14 shows the path planning results of Map 4 among the environmental maps for each algorithm. In visually, the number of sampling looks least for the RRT-Connect algorithm in Figure 14 (b) compared to others, and the optimal path distance looks shortest for the proposed algorithm in Figure 14 (c).

Table 5. Experimental result of Map 4 (The parentheses to the right of each value are relative ratios based on RRT 100%).

Performance	RRT	RRT-Connect	Proposed Algorithm
Avg. Num. of Sampling [samples]	290 *(100)*	28 *(10)*	32 *(11)*
Avg. Optimal Path Distance [px]	711 *(100)*	588 *(83)*	534 *(75)*
Avg. Convergence time [ms]	3 *(100)*	3 *(100)*	4 *(133)*

Table 5 shows the result (for repeated trial 50 times) of path planning in Map 4 for each algorithm. The average number of sampling is the least in RRT-Connect algorithm at 10%, and the proposed algorithm is 11% compared to RRT algorithm, 1% less efficient than RRT algorithm compared to RRT-Connect algorithm. The average optimal path distance of the proposed algorithm is the shortest at 75% compared to the RRT algorithm, and the RRT-Connect algorithm is 83%, 8% less efficient than the RRT algorithm compared to the proposed algorithm. The average convergence time is not different by 100% compared to the RRT algorithm, and the proposed algorithm is 133% compared to RRT algorithm, 33% less efficient than others.



(**a**) (**b**) (**c**)

Figure 15. Experimental result of Map 5: (**a**) RRT; (**b**) RRT-Connect; (**c**) The proposed algorithm.

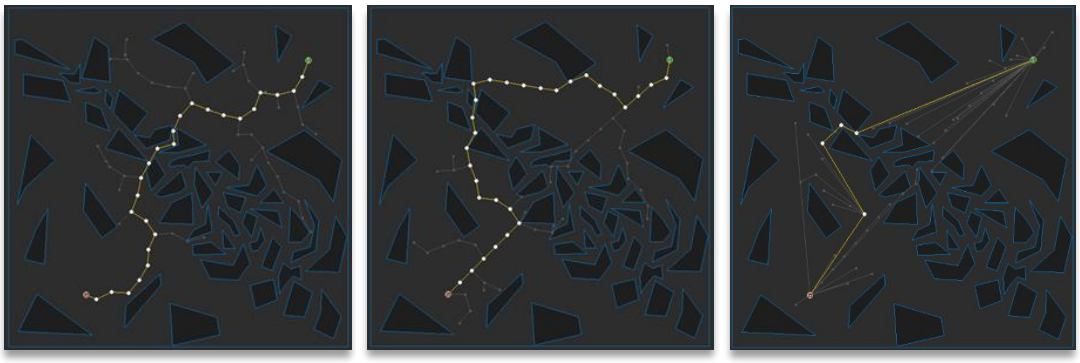
Figure 15 shows the path planning results of Map 5 among the environmental maps for each algorithm. In visually, the number of sampling looks similar for the RRT-Connect algorithm in Figure 15 (b) and the proposed algorithm in Figure 15 (c) compared to the RRT algorithm in Figure 15 (a), and the optimal path distance also looks similar for the RRT-Connect algorithm and proposed algorithm.

Table 6. Experimental result of Map 5 (The parentheses to the right of each value are relative ratios based on RRT 100%).

Performance	RRT	RRT-Connect	Proposed Algorithm
Avg. Num. of Sampling [samples]	371 *(100)*	68 *(18)*	74 *(20)*
Avg. Optimal Path Distance [px]	554 *(100)*	588 *(106)*	465 *(84)*
Avg. Convergence time [ms]	13 *(100)*	2 *(15)*	2 *(15)*

Table 6 shows the result (for repeated trial 50 times) of path planning in Map 5 for each algorithm. The average number of sampling is the least in RRT-Connect algorithm at 18%, and the proposed

algorithm is 20% compared to RRT algorithm, 9% less efficient than RRT algorithm compared to RRT-Connect algorithm. The average optimal path distance of the proposed algorithm is the shortest at 84% compared to the RRT algorithm, and the RRT-Connect algorithm is 106%, which is 22% less efficient compared to the proposed algorithm. The average convergence time of the proposed algorithm and the RRT-Connect algorithm is 15% over the RRT algorithm, showing the same performance.



(**a**) (**b**) (**c**)

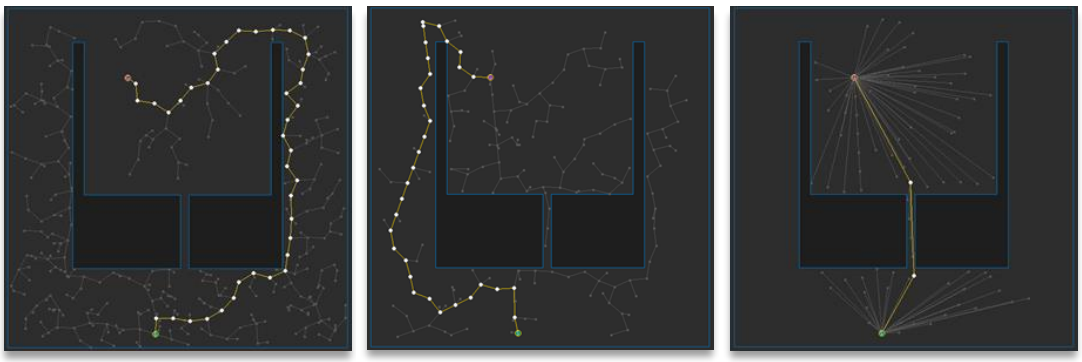
Figure 16. Experimental result of Map 6: (**a**) RRT; (**b**) RRT-Connect; (**c**) The proposed algorithm.

Figure 16 shows the path planning results of Map 6 among the environmental maps for each algorithm. In visually, the number of sampling looks least for the proposed algorithm in Figure 16 (c) compared to others, and the optimal path distance also looks shortest for the proposed algorithm.

Table 7. Experimental result of Map 6 (The parentheses to the right of each value are relative ratios based on RRT 100%).

Performance	**RRT**	**RRT-Connect**	**Proposed Algorithm**
Avg. Num. of Sampling [samples]	541 *(100)*	184 *(34)*	140 *(26)*
Avg. Optimal Path Distance [px]	886 *(100)*	778 *(88)*	668 *(75)*
Avg. Convergence time [ms]	9 *(100)*	6 *(67)*	4 *(44)*

Table 7 shows the result (for repeated trial 50 times) of path planning in Map 6 for each algorithm. The average number of sampling is the least in proposed algorithm at 26%, and the RRT-Connect algorithm is 34% compared to RRT algorithm, 8% less efficient than RRT algorithm compared to proposed algorithm. The average optimal path distance of the proposed algorithm is the shortest at 75% compared to the RRT algorithm, and the RRT-Connect algorithm is 88%, which is 13% less efficient compared to the proposed algorithm. The average convergence time is the least in the proposed algorithm at 44%, and the RRT-Connect is 67% compared to RRT algorithm, 23% less efficient than RRT algorithm compared to proposed algorithm.



(**a**) (**b**) (**c**)

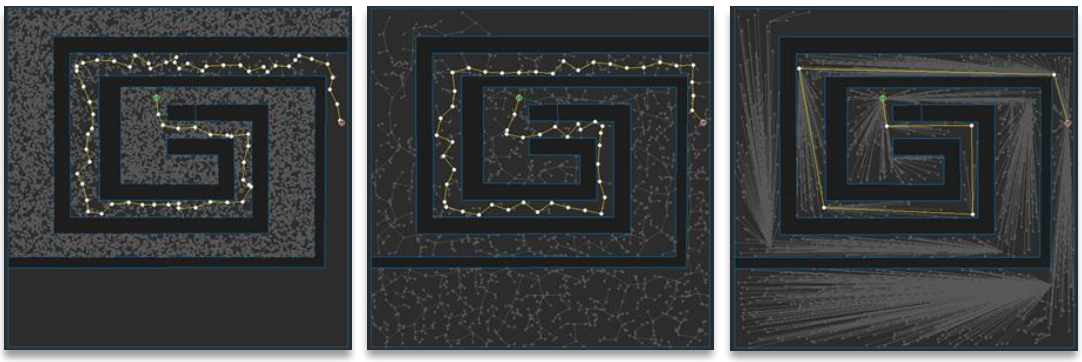
Figure 17. Experimental result of Map 7: (**a**) RRT; (**b**) RRT-Connect; (**c**) The proposed algorithm.

Figure 17 shows the path planning results of Map 7 among the environmental maps for each algorithm. In visually, the number of sampling looks least for the proposed algorithm in Figure 17 (c) compared to others, and the optimal path distance also looks shortest for the proposed algorithm.

Table 8. Experimental result of Map 7 (The parentheses to the right of each value are relative ratios based on RRT 100%).

Performance	**RRT**	**RRT-Connect**	**Proposed Algorithm**
Avg. Num. of Sampling [samples]	436 *(100)*	235 *(54)*	244 *(56)*
Avg. Optimal Path Distance [px]	898 *(100)*	862 *(96)*	674 *(75)*
Avg. Convergence time [ms]	5 *(100)*	4 *(80)*	3 *(60)*

Table 8 shows the result (for repeated trial 50 times) of path planning in Map 7 for each algorithm. The average number of sampling is the least in RRT-Connect algorithm at 54%, and the proposed algorithm is 56% compared to RRT algorithm, 2% less efficient than RRT algorithm compared to RRT-Connect algorithm. The average optimal path distance of the proposed algorithm is the shortest at 75% compared to the RRT algorithm, and the RRT-Connect algorithm is 96%, which is 21% less efficient compared to the proposed algorithm. The average convergence time is the least in the proposed algorithm at 60%, and the RRT-Connect is 80% compared to RRT algorithm, 20% less efficient than RRT algorithm compared to proposed algorithm.



(**a**) (**b**) (**c**)

Figure 18. Experimental result of Map 8: (**a**) RRT; (**b**) RRT-Connect; (**c**) The proposed algorithm.

Figure 18 shows the path planning results of Map 8 among the environmental maps for each algorithm. In visually, the number of sampling looks similar for the RRT-Connect algorithm in Figure

18 (b) and the proposed algorithm in Figure 18 (c) compared to the RRT algorithm in Figure 18 (a), and the optimal path distance looks shortest for the proposed algorithm.

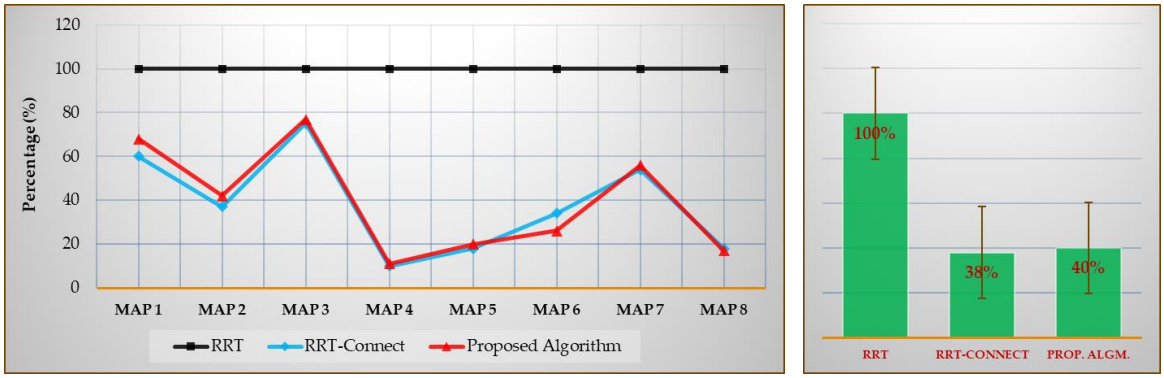
Table 9. Experimental result of Map 8 (The parentheses to the right of each value are relative ratios based on RRT 100%).

Performance	RRT	RRT-Connect	Proposed Algorithm
Avg. Num. of Sampling [samples]	17033 *(100)*	3031 *(18)*	2954 *(17)*
Avg. Optimal Path Distance [px]	1611 *(100)*	1576 *(98)*	1358 *(84)*
Avg. Convergence time [ms]	4501 *(100)*	119 *(3)*	125 *(3)*

Table 9 shows the result (for repeated trial 50 times) of path planning in Map 8 for each algorithm. The average number of sampling is the least in proposed algorithm at 17%, and the RRT-Connect algorithm is 18% compared to RRT algorithm, 1% less efficient than RRT algorithm compared to proposed algorithm. The average optimal path distance of the proposed algorithm is the shortest at 84% compared to the RRT algorithm, and the RRT-Connect algorithm is 98%, which is 14% less efficient compared to the proposed algorithm. The average convergence time of the proposed algorithm and the RRT-Connect algorithm is 3% over the RRT algorithm, showing the same performance.

5.3. Analysis of the Total Experimental Results

This section comprehensively present the experimental results (on average, number of sampling, optimal path distance, convergence time) of each algorithm: RRT, RRT-Connect and Proposed Triangular Inequality based RRT-Connect Algorithm, in the 8 environmental maps (Fig. 10) shown in Section 5.2.



(**a**) (**b**)

Figure 19. Total experimental results on average number of sampling (for first path finding): (**a**) result of each map compared with RRT algorithm; (**b**) average result compared with RRT algorithm.

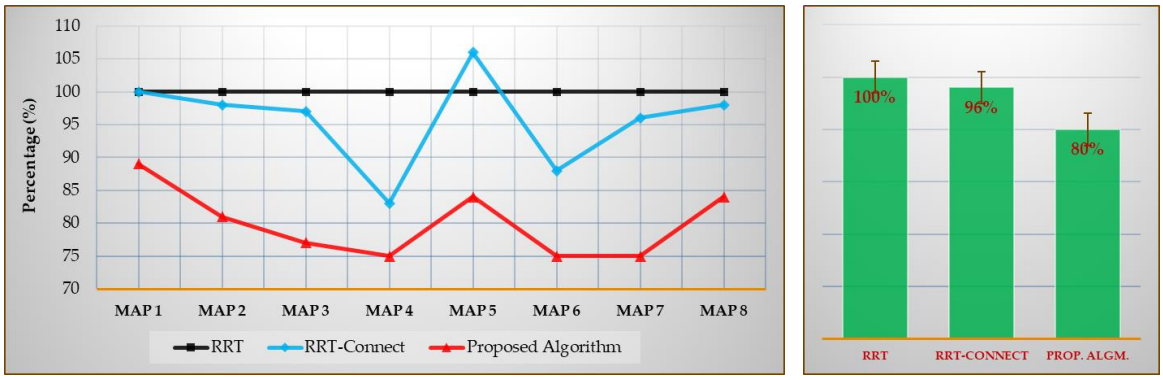
Figure 19 shows the average number of sampling [%] compared with the RRT algorithm for each map from Map 1 to Map 8 (for repeated trial 50 times), and the average number of sampling [%] compared with the average result of each algorithms for each map (for repeated trial 50 times) when the result of RRT algorithm is considered as 100%.

As shown in Figure 19 (b), the average number of sampling for all environment maps was 38% less in the RRT-Connect algorithm and 40% less in the proposed algorithm compared to the RRT algorithm. The proposed algorithm is 2% less efficient than the RRT-Connect algorithm.

Table 10. Total experimental results on average number of sampling (for first path finding) [%].

Algorithm	**Map 1**	**Map 2**	**Map 3**	**Map 4**	**Map 5**	**Map 6**	**Map 7**	**Map 8**	Avg.
RRT	100	100	100	100	100	100	100	100	100
RRT-Connect	60	37	75	10	18	34	54	18	38
Proposal	68	42	77	11	20	26	56	17	40

Table 10 is the data table of Figure 19 (a). The proposed algorithm shows better performance for Map 6 and Map 8 than the RRT-Connect algorithm, and the RRT-Connect algorithm shows better performance than the proposed algorithm in Map 1, Map 2, Map 3, Map 4, Map 5 and Map 7. However, most of the map there is no significant difference, such as showing a 2% difference from the map average. There are cases where the proposed algorithm is max 8% to min 1% better than the RRT-Connect algorithm, and there are cases where RRT-Connect algorithm is max 8% to min 1% better than the proposed algorithm.



(**a**) (**b**)

Figure 20. Total experimental results on average optimal path distance: (**a**) result of each map compared with RRT algorithm; (**b**) average result compared with RRT algorithm.

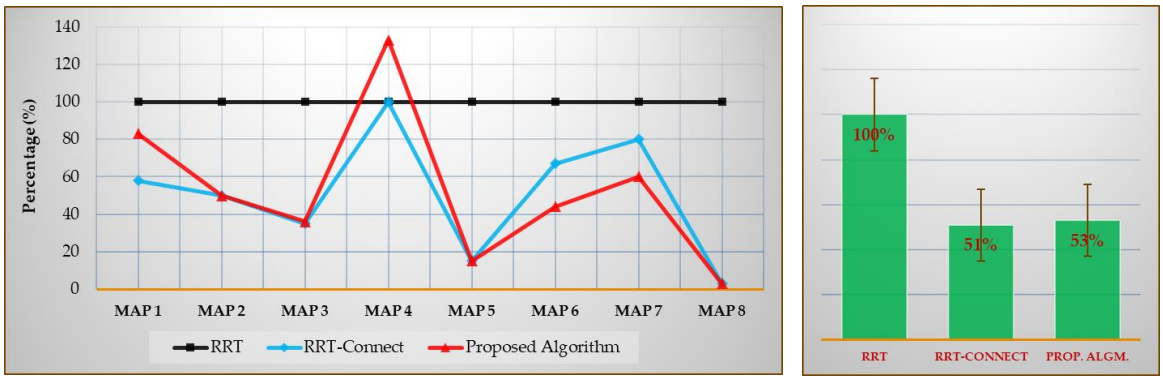
Figure 20 shows the average optimal path distance [%] compared with the RRT algorithm for each map from Map 1 to Map 8 (for repeated trial 50 times), and the average optimal path distance [%] compared with the average result of each algorithms for each map (for repeated trial 50 times) when the result of RRT algorithm is considered as 100%.

As shown in Figure 20 (b), the average optimal path distance for all environment maps was 96% less in the RRT-Connect algorithm and 80% less in the proposed algorithm compared to the RRT algorithm. The proposed algorithm is 16% more efficient than the RRT-Connect algorithm.

Table 11. Total experimental results on average optimal path distance [%].

Algorithm	**Map 1**	**Map 2**	**Map 3**	**Map 4**	**Map 5**	**Map 6**	**Map 7**	**Map 8**	Avg.
RRT	100	100	100	100	100	100	100	100	100
RRT-Connect	100	98	97	83	106	88	96	98	96
Proposal	89	81	77	75	84	75	75	84	80

Table 11 is the data table of Figure 20 (a). The proposed algorithm shows better performance for all maps than the RRT-Connect algorithm. The proposed algorithm is max 21% to min 8% better than the RRT-Connect algorithm.



(**a**) (**b**)

Figure 21. Total experimental results on average convergence time: (**a**) result of each map compared with RRT algorithm; (**b**) average result compared with RRT algorithm.

Figure 21 shows the average convergence time [%] compared with the RRT algorithm for each map from Map 1 to Map 8 (for repeated trial 50 times), and the average convergence time [%] compared with the average result of each algorithms for each map (for repeated trial 50 times) when the result of RRT algorithm is considered as 100%.

As shown in Figure 21 (b), the average convergence time for all environment maps was 51% less in the RRT-Connect algorithm and 53% less in the proposed algorithm compared to the RRT algorithm. The proposed algorithm is 2% less efficient than the RRT-Connect algorithm.

Table 12. Total experimental results on average convergence time [%].

Algorithm	**Map 1**	**Map 2**	**Map 3**	**Map 4**	**Map 5**	**Map 6**	**Map 7**	**Map 8**	Avg.
RRT	100	100	100	100	100	100	100	100	100
RRT-Connect	58	50	35	100	15	67	80	3	51
Proposal	83	50	36	133	15	44	60	3	53

Table 12 is the data table of Figure 21 (a). The proposed algorithm shows the same or better performance for Map 2, Map 5, Map 6, Map 7 and Map 8 than the RRT-Connect algorithm. And it shows better performance for the Map1, Map 3 and Map 4 than the algorithm proposed by the RRT-Connect algorithm. However, most of the map there is no significant difference, such as showing a 2% difference from the map average. There are cases where the proposed algorithm is max 23% to min 20% better than the RRT-Connect algorithm, and there are cases where the RRT-Connect algorithm is also max 33% to min 1% better than the proposed algorithm.

6. Conclusion

In this paper, we proposed a Triangular Inequality based RRT-Connect algorithm using Triangular Inequality principles to overcome the limitation of the optimality of RRT-Connect algorithm. We verified the validity of the Triangular Rewiring method based on Triangular Inequality principle and applied it to the RRT-Connect algorithm to enhance the optimality. Also, to check the performance such as number of sampling for first path finding, optimal path distance and convergence time of the proposed algorithm, we compared between the RRT and RRT-Connect algorithms in a total of 8 environments through simulation. The proposed algorithm showed, on average, 20% better efficiency than RRT algorithm and 16% better efficiency than RRT-Connect algorithm in the optimality, and 47% better efficiency than the RRT algorithm in the convergence time, but 2% slightly worse efficiency than RRT-Connect algorithm. In conclusion, the proposed algorithm shows more optimal than RRT-Connect algorithm with similar number of sampling and convergence time.

Author Contributions: Idea and conceptualization: J.-G.K. , D.-W.L. and J.-W.J.; methodology: J.-G.K., D.-W.L. and J.-W.J.; software: J.-G.K., D.-W.L. and J.-W.J.; experiment: J.-G.K., D.-W.L., W.-J.J. and J.-W.J.; validation: J.-G.K., D.-W.L., Y.-S.C. and J.-W.J.; investigation: J.-G.K.; D.-W.L., Y.-S.C., W.-J.J. and J.-W.J.; resources: J.-G.K. and J.-W.J.; writing: J.-G.K., D.-W.L., Y.-S.C., W.-J.J. and J.-W.J.; visualization: J.-G.K. and J.-W.J.; project administration: J.-W.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. 2020R1F1A1074974), the KIAT(Korea Institute for Advancement of Technology) grant funded by the Korea Government(MOTIE : Ministry of Trade Industry and Energy). (No. N0001884, HRD program for Embedded Software R&D), the AURI(Korea Association of University, Research institute and Industry) grant funded by the Korea Government(MSS : Ministry of SMEs and Startups). (No.S2938281, HRD program for Enterprise linkages R&D) and the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2020-2020-0-01789) supervised by the IITP(Institute for Information & Communications Technology Planning & Evaluation).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Schwab, K. *The fourth industrial revolution*. Currency, 2017.
2. Sariff, N.; Buniyamin, N. An overview of autonomous mobile robot path planning algorithms. In Proceedings of the IEEE 4th Student Conference on Research and Development, Selangor, Malaysia, 28-29 June 2006; pp. 183-188.
3. Roy, D. Visibility Graph based Spatial Path Planning of Robots using Configuration Space Algorithms. *International Journal of Robotics and Automation*. **2009**, *24*, 1-9.
4. Katevas, N.I.; Tzafestas, S.G.; Pnevmatikatos, C.G. The Approximate Cell Decomposition with Local Node Refinement Global Path Planning Method: Path Nodes Refinement and Curve Parametric Interpolation. *Journal of Intelligent and Robotic Systems*. **1998**, *22(3-4)*, 289-314.
5. Warren, C. W. Global Path Planning using Artificial Potential Fields. In Proceedings of the International Conference on Robotics and Automation, Arizona, USA, 14-19 May 1989; Volume 1, pp. 316-321.
6. LaValle, S.M. Motion Planning Part II: Wild Frontiers. *IEEE Robotics Automation Magazine*. **2011**, *18(2)*, 108-118.
7. Mac, T.T.; Copot, C.; Tran, D.T.; De Keyser, R. Heuristic Approaches in Robot Path Planning: A survey. *Robotics and Autonomous Systems*. **2016**, *86*, 13-28.
8. Paden, B.; Čáp, M.; Yong, S.Z.; Yershov, D.; Frazzoli, E. A Survey of Motion Planning and Control Techniques for Self-driving Urban Vehicles. *IEEE Transactions on Intelligent Vehicles*. **2016**, *1(1)*, 33-55.
9. Karaman, S.; Frazzoli, E. Incremental Sampling based Algorithms for Optimal Motion Planning. *Robotics Science and Systems VI*. **2010**, *104(2)*.
10. Brunner, M.; Bruggemann, B.; Schulz, D. Hierarchical Rough Terrain Motion Planning using an Optimal Sampling based Method. In Proceedings of the IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, 6-10 May 2013; pp. 5539-5544.
11. Adiyatov, O.; Varol, H.A. Rapidly-exploring Random Tree Based Memory Efficient Motion Planning. In Proceedings of the IEEE International Conference on Mechatronics and Automation, Takamatsu, Japan, 4-7 August 2013; pp. 354–359.
12. LaValle, S.M.; Kuffner Jr, J.J. Randomized Kinodynamic Planning. *The International Journal of Robotics Research*. **2001**, *20(5)*. 378–400.
13. LaValle, S.M. Rapidly-exploring random trees: A new tool for path planning. **1998**.
14. Englot, B.; Hover, F.S. Sampling based coverage path planning for inspection of complex structures. **2012**.

15. Kuffner Jr, J.J.; LaValle, S.M. RRT-connect: An Efficient Approach to Single-query Path Planning. In Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, USA, 24-28 April 2000; Volume 2, pp. 995-1001.
16. Islam, F.; Nasir, J.; Malik, U.; Ayaz, Y.; Hasan, O. Rrt*-smart: Rapid convergence implementation of rrt* towards optimal solution. In Proceedings of the IEEE International Conference on Mechatronics and Automation, Chengdu, China, 5-8 August 2012; pp. 1651-1656.
17. Jeong, I.-B.; Lee, S.-J.; Kim, J.-H. Quick-RRT*: Triangular Inequality based Implementation of RRT* with Improved Initial Solution and Convergence rate. *Expert Systems with Applications*. **2019**, *123*, 82-90.
18. Karaman, S.; Frazzoli, E. Sampling based Algorithms for Optimal Motion Planning. *International Journal of Robotics Research*. **2011**, *30(7)*, 846-894.
19. Gammell, J.D.; Srinivasa, S.S.; Barfoot, T.D. Informed RRT*: Optimal Sampling based Path Planning Focused via Direct Sampling of an Admissible Ellipsoidal Heuristic. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Illinois, USA, 14-18 September 2014; pp. 2997-3004.
20. Klemm, S.; Oberländer, J.; Hermann, A.; Roennau, A.; Schamm, T.; Zollner, J.M.; Dillmann, R. RRT*-Connect: Faster, Asymptotically Optimal Motion Planning. In Proceedings of the IEEE International Conference on Robotics and Biomimetics, Zhuhai, China, 6-9 December 2015; pp. 1670-1677.
21. Choudhury, S.; Scherer, S.; Singh, S. RRT*-AR: Sampling based Alternate Routes Planning with Applications to Autonomous Emergency Landing of a Helicopter. In Proceedings of the IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, 6-10 May 2013; pp. 3947–3952.
22. da Silva Arantes, M.; Toledo, C.F.M.; Williams, B.C.; Ono, M. Collision-Free Encoding for Chance-Constrained Nonconvex Path Planning. *IEEE Transaction on Robotics*. **2019**, *35(2)*, 433-448.
23. Nazarahari, M.; Khanmirza, E.; Doostie, S. Multi-objective multi-robot path planning in continuous environment using an enhanced genetic algorithm. *Expert Systems With applications*. **2019**, *115*, 106-120.
24. Sung, I.; Choi, B.; Nielsen, P. On the training of a neural network for online path planning with offline path planning algorithms. *International Journal of Information Management*. **2020**, 102142.
25. Jeon, G.-Y.; Jung, J.-W. Water Sink Model for Robot Motion Planning. *Sensors*. **2019**, *19(6)*, 1269.
26. Han, J. Mobile robot path planning with surrounding point set and path improvement. *Applied Soft Computing*. **2017**, *57*, 35-47.
27. Jung, J.-W.; So, B.-C.; Kang, J.-G.; Lim, D.-W.; Son, Y. Expanded Douglas–Peucker Polygonal Approximation and Opposite Angle based Exact Cell Decomposition for Path Planning with Curvilinear Obstacles. *Applied Sciences*. **2019**, *9(4)*, 638.