Article

Improved RRT-Connect Algorithm based on Triangular Inequality for Robot Path Planning

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 Rapidly exploring Random Tree (RRT)-Connect robot path-planning algorithm that guarantees the planning time compared to the RRT algorithm, to bring it closer to the optimum. To check the proposed algorithm's performance, this paper compared the RRT and RRT-Connect algorithms in various environments through simulation. From these experimental results, the proposed algorithm shows both quicker planning time and shorter path length than the RRT algorithm and shorter path length than the RRT-Connect algorithm with a similar number of samples and planning time.

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

1. Introduction

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

Optimality means always ensuring the optimal path. ***Clearance*** indicates a lower probability of collision between obstacles and the robot. ***Completeness*** means that if a path exists, it can always be found. Optimality, clearance and completeness are considered important in these classical algorithms and have been the main focus of study [6]. Particularly if completeness is not guaranteed by the robot path-planning algorithm, there is a problem that the path may not be found in finite time. This is a fatal problem in robot path planning.

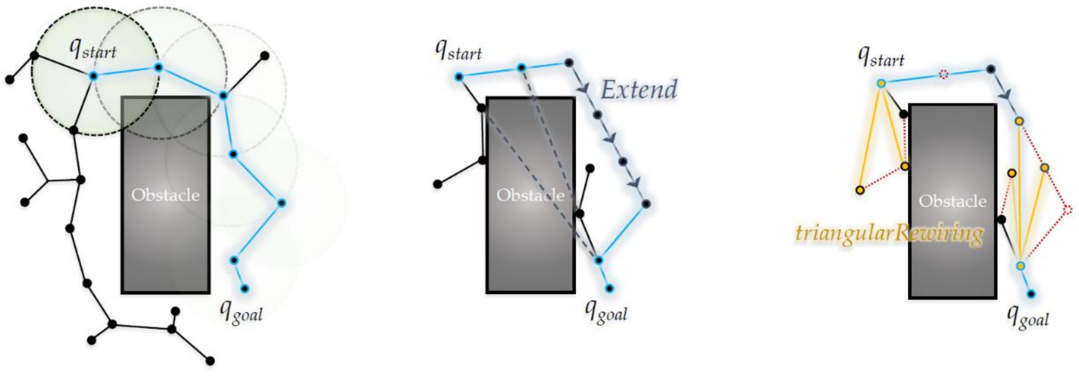
Recently, sampling-based path-planning algorithms [7–12] such as Rapidly Exploring Random Tree (RRT) [13], which is quicker and less computationally intensive than classical algorithms, have been attracting attention. The main purpose of sampling-based algorithms is to find a path that can reach the goal as quickly as possible using randomly extracted sample points (random sampling). Unlike classical algorithms, sampling-based algorithms have difficulty fully reflecting the optimality and completeness. Therefore, most sampling-based algorithms claim ***Probabilistic Completeness***, which explains that they can be probabilistically close to complete when random sampling is repeated infinitely [14]. This means that it is difficult to guarantee the ***Planning time*** *(First path finding time)*, which refers to how quickly the path can be planned from the start point to the goal point, and the ***Convergence rate***, which means iterative sampling to bring the path closer to the optimum after the first path has been found. If the situation does not allow enough time to plan the path, it can create a path that is more different from the optimal path. Even so, the sampling-based algorithm is mainly used in dynamic environments because it enables quicker path planning with very little planning time compared to classical algorithms.

To overcome these limitations of planning time and convergence rate, many studies are being conducted to expand the RRT algorithm. The RRT-Connect [15] algorithm finds a connected path more quickly than the RRT algorithm by setting the start point and goal point as the roots of separate trees and expanding both trees alternately. In addition, there are algorithms that optimize paths 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 to overcome the limitations of sampling-based methods but they are still not perfect. Their limitations include being unable to derive the optimal length and there is room for improvement in terms of the number of operations and time. For example, the RRT* algorithm has rewiring(search for the parent node as a via point nearby a newly inserted node, where the addition of path length from the start point to the via point and path length from the via point to the newly inserted node in the tree is the optimized, and change the neighboring nodes to optimize the path length) and neighbor search(search for nodes nearby the node to be newly inserted in the tree) processes to obtain shorter path lengths than the RRT algorithm [18]. However, there is an efficiency trade-off in this process. In other words, while the convergence rate has improved, the planning time has significantly increased [22]. Therefore, the RRT* algorithm cannot be said to be better than the RRT algorithm in all performance metrics and it can be said that the RRT* algorithm gets closer to the optimum at the expense of planning time.

To overcome the limitation of getting closer to the optimum at the expense of planning time, this paper proposes a triangular inequality-based RRT-Connect algorithm that finds an ancestor node as a via point, where the addition of path length from the start point to the via point and path length from the via point to the newly inserted node is the most optimized, based on the principle of triangular inequality and RRT-Connect. The proposed algorithm shortens the planning time while also pursuing optimization through rewiring. In addition, we will verify the efficiency by comparing the RRT and RRT-Connect algorithms from previous studies through simulation experiments. As a result, this paper shows that the proposed algorithm has a shorter path length than the RRT and RRT-Connect algorithms without sacrificing other performance measures such as the number of sample or planning time.

The scope of the research we will cover is how much more quickly it can find the path and how much shorter the path is. This is because in a dynamic environment, it is more important to find a navigable path. In a dynamic environment, there may not be enough time for convergence. In other words, the purpose of our proposed algorithm is to improve the RRT-Connect algorithm so that it can find a shorter path over the same planning time(computation time before convergence or computation time for first path finding).



(a) (b) (c)

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

Figure 1 shows an overview of the three main algorithms covered in this paper: RRT, RRT-Connect, and the proposed algorithm. In this figure, the start q_{start} and goal points are q_{goal}, respectively. 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 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 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 is proposed in Chapter 4. In detail, Section 4.1 shows the pseudocode of the proposed rewiring 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, Sections 4.3 and 4.4 show the pseudocode of each method of the RRT-Connect algorithm applying the proposed rewiring method, and Section 4.5 shows the path-planning process for the proposed algorithm that applies the proposed rewire method to the RRT-Connect algorithm. Chapter 5 shows the experimental environment and results to check the performance of the proposed algorithm and Chapter 6 presents the conclusion.

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 plans a path by gradually expanding a tree with a root node at the start point 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 nodes constituting the tree with the step length. If tree connections are possible, nodes are added to create an extended tree.

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

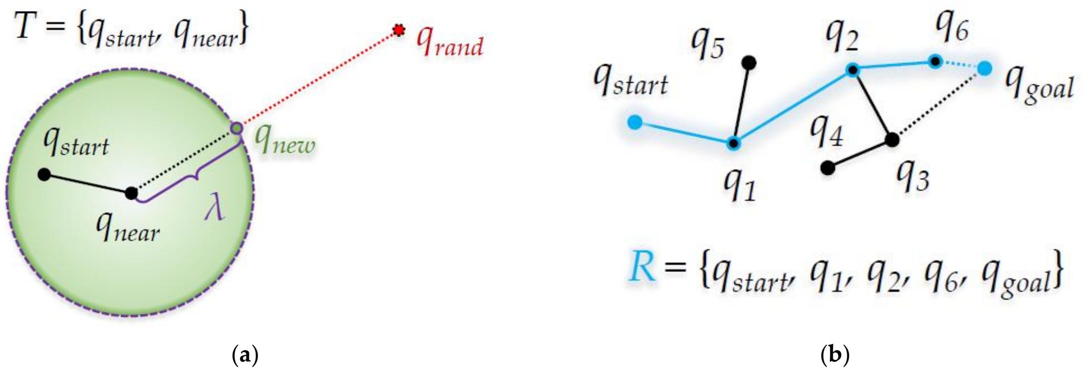


Figure 2. The RRT algorithm: (**a**) Process when q_{new} is created; (**b**) After the random sampling has ended.

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 resultant path R among several candidate paths to the start position q_{start} and the goal position q_{goal}.

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

Path planning through the RRT-Connect algorithm can find a path quicker than the RRT algorithm, but the *Extend* method does not work properly in complex environments with narrow paths and many obstacles and it can be difficult. In addition, the path planned using the RRT-Connect algorithm is far from the optimal length, so it does not properly reflect optimality.

3.1. Pseudocode of the RRT-Connect Algorithm

This section shows the pseudocode of the RRT-Connect algorithm used in the experiment in this paper that was designed based on [15] in which the RRT-Connect algorithm was proposed. The RRT-Connect algorithm can be represented by a main algorithm (A1) and two main methods (A2 and 3).

Algorithm 1. Pseudocode of the RRT-Connect Algorithm.

```
Input:
$q_{start}$ ← Start Point Position
$q_{goal}$ ← Goal Point Position
$\lambda$ ← 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
$d_{shorter}$ ← 0

Begin RRT-Connect Procedure
1    $T_a$ ← Insert Root Node<$q_{start}$> to $T_a$
2    $T_b$ ← Insert Root Node<$q_{goal}$> to $T_b$
3    While 1 ← $n$ to $N$ do
4        Generate $n$-th Random Sample
5        $q_{rand}$ ← Position of $n$-th Random Sample
6        If Not Extend($T_a$, $T_b$, $q_{newB}$ ← Null, $q_{rand}$, $\lambda$, $C$) then
7            If Connect($P_{reach}$ ← Null Path, $T_a$, $T_b$, $q_{newB}$, $\lambda$) then
8                $d_{reach}$ ← Distance of $P_{reach}$
9                If $d_{shorter} = 0$ or $d_{shorter} > d_{reach}$ then
```

```
10                  $R \leftarrow P_{reach}$
11                  $d_{shorter} \leftarrow d_{reach}$
12          $Swap(T_a, T_b)$
End RRT-Connect Procedure
```

Algorithm 1 shows the pseudocode of RRT-Connect algorithm. Both of the two initial trees T_a and T_b have q_{start} and q_{goal} as root nodes and these two trees randomly sample N times and aim to reach each other during their expansion. Unlike RRT, the RRT-Connect algorithm is divided into two methods: *Extend* and *Connect*. The ***Extend*** method (A2) 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 (A3) determines whether the two trees T_a and T_b have reached each other; 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.

When a path is created by the Connect method, the distance d_{reach} is calculated for the path P_{reach} from q_{start} to q_{goal}. At this time, if d_{reach} is smaller than $d_{shorter}$(the shortest path length until now) or P_{reach} is the first path found (i.e., $d_{shorter} = 0$), the resultant path R becomes P_{reach}, and $d_{shorter}$ 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.

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

This section introduces the *Extend* method used in pseudocode (A1) of the RRT-Connect algorithm in Section 3.1.

Algorithm 2. Pseudocode of the original *Extend* method from the RRT-Connect Algorithm.

```
Input:
$T_a \leftarrow$ Tree $T_a$ from RRT-Connect
$T_b \leftarrow$ Tree $T_b$ from RRT-Connect
$q_{newB} \leftarrow$ Position $q_{newB}$ from RRT-Connect
$q_{rand} \leftarrow$ Position $q_{rand}$ from RRT-Connect
$\lambda \leftarrow$ Step Length $\lambda$ from RRT-Connect
$C \leftarrow$ Position Set $C$ from RRT-Connect
Output:
$f_{trap} \leftarrow$ Result of Boolean $f_{trap}$
$T_a \leftarrow$ Result of Tree $T_a$        // Return by Reference
$T_b \leftarrow$ Result of Tree $T_b$        // Return by Reference
$q_{newB} \leftarrow$ Result of Position $q_{newB}$        // Return by Reference
Initialize:
$f_{trap} \leftarrow$ False

Begin Extend Procedure from RRT-Connect
1    $q_{near} \leftarrow$ Find Position of Nearest Node in $T_a$ from $q_{rand}$
2    If Not $isInside(q_{near}, q_{rand}, \lambda)$ then
3        $q_{newA} \leftarrow$ Position of the Intersection Point between the Line Segment connecting $q_{rand}$ and $q_{near}$ and
             a Circle with Radius $\lambda$ centered at $q_{near}$        // 2D: Circle, 3D: Sphere, …
4    Else
5        $q_{newA} \leftarrow q_{rand}$
6    If $isTrapped(q_{newA}, q_{near}, C)$ then
7        $f_{trap} \leftarrow$ 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($q_{near}$, $q_{newA}$, $\lambda$) 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 $\lambda$ 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($q_{newA}$, $q_{newB}$, $\lambda$) then
18                   $q_{near}$ ← $q_{newB}$
19                   $q_{newB}$ ← Position of Intersection Point between     // 2D: Circle, 3D: Sphere, …
                         Line Segment connecting $q_{newA}$ and $q_{near}$, and Circle with Radius $\lambda$ centered at $q_{near}$
20               Else
21                   Break
End Extend Procedure from RRT-Connect
```

Algorithm 2 shows the pseudocode of the *Extend* method in the RRT-Connect algorithm. The *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 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. Otherwise, it proceeds with the remaining process and returns *False* (f_{trap}) when the process ends.

This is the process of making T_a and T_b reach each other: First, the node T_b nearest to q_{newA} becomes the new q_{near}. At this time, using 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 the following process is repeated until it can determine whether there is an obstacle between q_{newB} and $\underline{q_{near}}$ by the *isTrapped* function and if there is the obstacle between them (*True*) or if q_{newB} reaches q_{newA} by the *isInside* function.

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

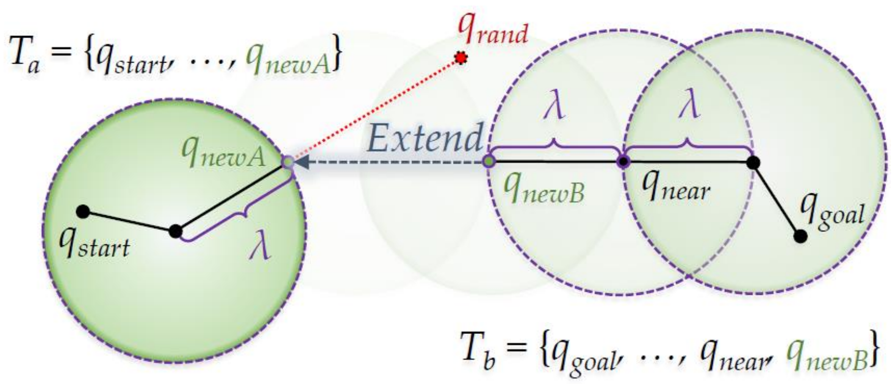


Figure 3. The *Extend* method from RRT-Connect algorithm.

Figure 3 shows the *Extend* method in the RRT-Connect algorithm. In detail, it shows that the first q_{newA} is created, and q_{newB} is created with radius of length λ in the direction of q_{newA} from the q_{near} position in the figure. Clearly, 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 (A1) of the RRT-Connect algorithm in Section 3.1.

Algorithm 3. 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
$\lambda$ ← Step Length $\lambda$ 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($q_{newA}$, $q_{newB}$, $\lambda$) 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 3 shows the pseudocode of the *Connect* method in the RRT-Connect algorithm. Here, T_a, T_b, and q_{newB} are from the *Extend* method (A2).

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 a 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 completing planning the path P_{merged} from q_{start} to q_{goal}. After this, it returns *True* (f_{trap}), and the *Connect* method ends.

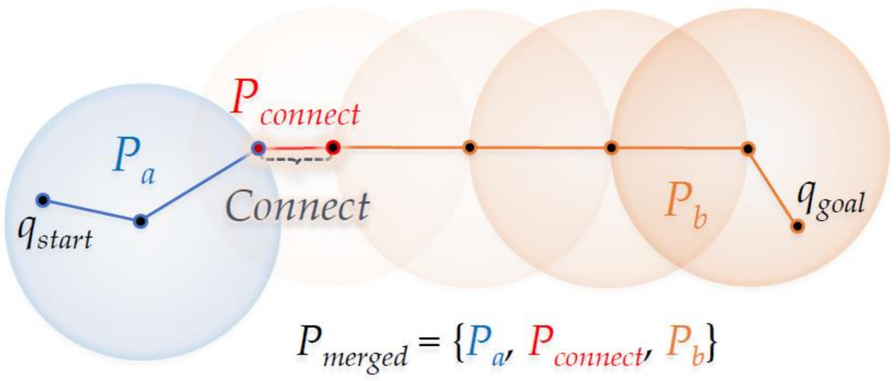


Figure 4. The *Connect* method from the RRT-Connect algorithm.

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} has expanded in the direction of q_{newA} where T_a enters the λ radius 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 $P_{connect}$ goes from position q_{newA} to position q_{newB} and the path P_b goes from position q_{newB} to position q_{goal} in T_b. The merged path P_{merged} goes from q_{start} to q_{goal}.

4. Proposed Triangular Inequality-based RRT-Connect Algorithm

The **proposed triangular inequality-based RRT-Connect** algorithm is a rewire based on the principle of triangular inequality between nodes on a path planned in the RRT-Connect algorithm, so it is closer to the optimal compared to the RRT-Connect. This is like the RRT*-Smart algorithm [16] and Quick-RRT* [17] algorithms, which shorten their paths using the triangular inequality principle for the RRT algorithm. In this paper, the rewire part based on the triangular inequality principle is called the ***Triangular-Rewiring*** method.

The proposed triangular inequality-based RRT-Connect algorithm requires the following assumptions.

[*Assumptions*]

1. There is only one start point and one goal point even though the goal point may be changed incrementally as time goes on.
2. The robot is capable of omnidirectional motion.

Therefore, this chapter introduces the proposed *Triangular-Rewiring* method for the RRT-Connect algorithm, and performs mathematical modeling to confirm the validity that the proposed *Triangular-Rewiring* method is always shorter 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 (A2) and *Connect* method (A3), 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, this chapter introduces the *Extend* and *Connect* methods to which the proposed *Triangular-Rewiring* method is applied.

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

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

Algorithm 4. 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 4 shows the pseudocode of the *Triangular-Rewiring* method applicable in the *Extend* (A2) and *Connect* (A3) methods of the RRT-Connect algorithm. When inserting a new node and edge in T_a or T_b in the *Extend* method (A5), when a tree T_{merged} (P_{merged}) in which T_a and T_b trees are merged in the *Connect* method is created (A6), rewiring is performed on the tree T.

In the *Extend* and *Connect* methods, q_{new} (or q_{newA} or q_{newB}) is inserted as a q_{child} and q_{near} is inserted as a candidate for the node's parent node. From q_{parent}, the node's parent node (a second ancestor node candidate based on q_{child}) is called $q_{ancestor}$. Next, it is determined whether an obstacle exists between $q_{ancestor}$ and q_{child} (using the *isTrapped* function). 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 such that the contents of the *Extend* and *Connect* methods 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 q_{parent}. In other words, it disconnects the existing q_{parent} and q_{child} and prepares to connect q_{child} to $q_{ancestor}$, the candidate parent node of q_{child}. Again, q_{parent} becomes its parent node $q_{ancestor}$ and $q_{ancestor}$ becomes the parent node of $q_{ancestor}$. Then, as previously done, determine whether an obstacle exists between $q_{ancestor}$ and q_{child} (using the *isTrapped* function). This iterative process continues until no $q_{ancestor}$ exists (When no parent node exists for the previous $q_{ancestor}$, i.e., when $q_{ancestor}$ is q_{start}) or an obstacle exists between q_{child} and $q_{ancestor}$. Then, in tree T, the last created q_{parent} is inserted as the parent node of q_{child}.

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

This section introduces the mathematical modeling of the proposed triangular inequality-based RRT-Connect algorithm. The results show that the proposed algorithm is more efficient in terms of path length than the RRT-Connect algorithm. For reference, this mathematical modeling is based on a two-dimensional Euclidean space.

Equations 1 and 2 define the path length $\mathbb{d}_n(q_i)$ between an arbitrary node q_i and its parent node in the RRT algorithm:

$$D(q_i, \xi(q_i)) = \sqrt{(\xi(q_i).x - q_i.x)^2 + (\xi(q_i).y - q_i.y)^2}, \tag{1}$$

$$\therefore \mathbb{d}_n(q_i) = D\big(\xi^n(q_i), \xi^{n+1}(q_i)\big). \tag{2}$$

Here, q_i refers to the i-th inserted arbitrary node and takes 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. Equation 1 obtains the distance between an arbitrary node q_i and its parent node, which can be summarized as a function $\mathbb{d}_n$ as in Equation 2. Here, n is the distance between the ancestor node and its parent node, based on an arbitrary node. That is, the ξ function to the power of n ($n \geq 0$) can be represented as $\xi^n(q_i) := (\overbrace{\xi \circ \xi \circ \ldots \circ \xi}^{n})(q_i)$; when n is 0, $\xi^0(q_i) := q_i$ holds.

In addition, consider 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 $D\big(\xi^n(q_i), \xi^{n+1}(q_i)\big)$.

Equations 3 and 4 show the path length $\mathbb{D}_R$ from the start position q_{start} to the goal position q_{goal} by the RRT algorithm:

$$\xi^{\delta+1}(q_{goal}) = q_{start}, \tag{3}$$

$$\therefore \mathbb{D}_R = \textstyle\sum_{n=0}^{\delta} \mathbb{d}_n(q_{goal}). \tag{4}$$

Equation 3 shows when the (δ + 1)-th ancestor node from q_{goal} is q_{start}, where δ is the upper limit of $\sum_{n=0}^{\delta} \mathbb{d}_n(q_{goal})$ for obtaining the path length $\mathbb{D}_R$ in Equation 4. In other words, Equation 4 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}).

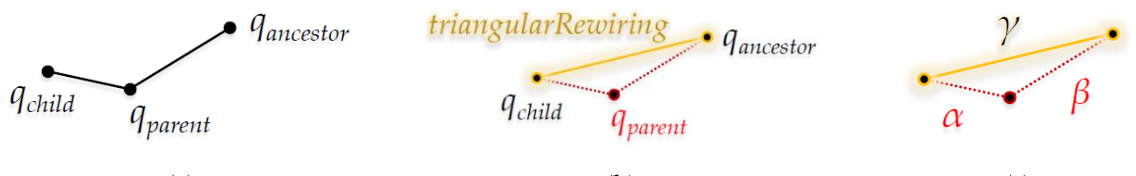


Figure 5. Abstract process of the *Triangular-Rewiring* method: (**a**) Example tree; (**b**) After rewiring between q_{child} and $q_{ancestor}$; (**c**) At this time, α is the distance between q_{child} and q_{parent}, β is the distance between q_{parent} and $q_{ancestor}$, and γ is the distance between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

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

$$q_{ancestor} = \xi\big(q_{parent}\big) = \xi^2(q_{child}). \tag{5}$$

If the distances between the edges 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 $q_{ancestor}$ is as shown in Figure 5 (c), this can be represented as Equation 6 using the principle of the triangular inequality:

$$\alpha + \beta \geq \gamma. \tag{6}$$

Equations 7 and 8 show the distance relationship between the ancestor nodes of q_{child}:

$$D(q_{child}, \xi(q_{child})) = \alpha,\ D(\xi(q_{child}), \xi^2(q_{child})) = \beta,\ D(q_{child}, \xi^2(q_{child})) = \gamma, \tag{7}$$

$$\therefore D\big(q_{child}, \xi(q_{child})\big) + D\big(\xi(q_{child}), \xi^2(q_{child})\big) \geq D(q_{child}, \xi^2(q_{child})), \tag{8}$$

Equation 7 can be summarized as Equation 8 by substituting Equation 5, which represents the relationship between the n-th ancestor nodes of q_{child}, with the distance as Equation 1 in Equation 6, which represents the distance between each node as a triangular inequality.

Equations 9–15 show that the path of the RRT algorithm applying the *Triangular-Rewiring* method is always shorter or equal to that planned by the original RRT algorithm. Equation 9 shows the sequence index k_j to compare the distance $\mathbb{u}$ when applying the Triangular-Rewiring method with distance $\mathbb{d}$ when this method is not applied:

$$k_j = \tau_j + k'_j,\ k'_j = \begin{cases} 0,\ j = 0 \\ k_{j-1} + 1,\ j \geq 1' \end{cases} \tag{9}$$

Here, j is a sequence index for $\mathbb{u}$. That is, k_j can be considered a sequence index for $\mathbb{d}$. Currently, τ_j is the number of times that rewiring occurs in the j-th.

If this is summarized by Equation 1 for a distance based on an arbitrary node q_i, it is as Equation 10. For example, as shown in Figure 5, if j is 0 and 1 a rewire occurs ($\tau_0 = 1$), it can be represented in combination with the distance relationship of Equation 8 for q_{child}, as in Equation 11:

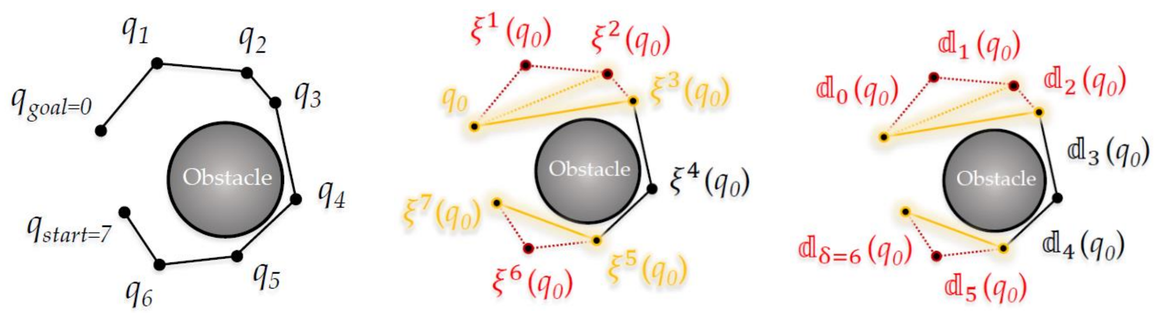
$$\mathbb{u}_{k_j}(q_i) = D\big(\xi^{k'_j}(q_i), \xi^{k_j+1}(q_i)\big), \tag{10}$$

$$\mathbb{d}_0(q_{child}) + \mathbb{d}_1(q_{child}) = \textstyle\sum_{n=0}^{1} \mathbb{d}_n(q_{child}) \geq \mathbb{u}_{k_0=1}(q_{child}). \tag{11}$$

The result of Equation 11 can be generalized as shown in Equation 12:

$$\therefore \textstyle\sum_{n=0}^{k_j} \mathbb{d}_n(q_i) \geq \mathbb{u}_{k_j}(q_i). \tag{12}$$

For $\mathbb{d}$ based on an arbitrary node q_i, the path length $\sum_{n=j}^{k_j} \mathbb{d}_n$ 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 Triangular-Rewiring method is at least 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 than $\mathbb{d}$ when not rewired.



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

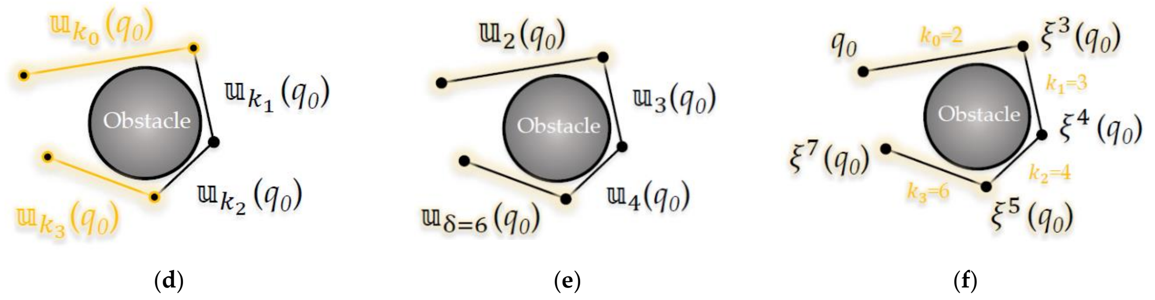


Figure 6. Detailed process of the *Triangular-Rewiring* method: (**a**) Each node q for index i (at this time, q_{start} is same as q_7 and q_{goal} is same as q_0); (**b**) Represent each node using the n-th ancestor ξ^n of q_0; (**c**) Each distance $\mathbb{d}_n$ between the n-th and (n + 1)-th ancestor nodes of q_0; (**d**) When the *Triangular-Rewiring* method is applied and rewired by distance $\mathbb{u}_{k_j}$; (**e**) Represent as the value of k_j; (**f**) Represent each node by the 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 Equations 5–12 (at this time, it is assumed that the node of the path shown in the figure is not positioned in a straight line). As shown in Figure 6 (b), a total of two rewires occurred ($\tau_0 = 2$) between q_0 and q_3 ($\xi^3(q_0)$), and a total of one 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 according to Equation 9.

Comparing Figures 6 (c) and 6 (e), according to Equation 7, the rewired distance $\mathbb{u}_2(q_0)$ is shorter than the path length $\sum_{n=0}^{2}\mathbb{d}_n(q_0)$ from $\mathbb{d}_0$ to $\mathbb{d}_2$ and the rewired distance $\mathbb{u}_6(q_0)$ is shorter than the path length $\sum_{n=5}^{6}\mathbb{d}_n(q_0)$ 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), the path afterward looks shorter.

Equations 13 and 14 show the path length $\mathbb{D}_R$ when the *Triangular-Rewiring* method is not applied and the path length $\mathbb{U}_R$ when the method has been applied for an arbitrary path (start position: q_{start}, goal position: q_{goal}), as shown in Figure 6:

$$k_\varphi = \delta, \tag{13}$$

$$\mathbb{D}_R = \sum_{n=0}^{\delta}\mathbb{d}_n(q_{goal}) = \sum_{j=0}^{\varphi}\sum_{n=k'_j}^{k_j}\mathbb{d}_n(q_{goal}),\ \ \mathbb{U}_R = \sum_{j=0}^{\varphi}\mathbb{u}_{k_j}(q_{goal}), \tag{14}$$

Equation 13 shows the upper limit when the index n of d is δ in Equation 3; when this is substituted into the sequence index k_j, if k_j is δ, j becomes φ. In that case, as in Equation 14, $\mathbb{D}_R$ is used to compare the $\sum_{n=0}^{\delta}\mathbb{d}_n(q_{goal})$ shown in Equation 4 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 $\sum_{j=0}^{\varphi}\mathbb{u}_{k_j}(q_{goal})$.

Equation 15 shows when the equation summarized in Equation 14 is substituted into Equation 12:

$$\therefore \mathbb{D}_R \geq \mathbb{U}_R. \tag{15}$$

Finally, as can be confirmed using Equation 15, $\mathbb{U}_R$ as a 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 a result, this method is not applied.

Equations 16–18 show the path length $\mathbb{D}_A$ of the path from the start position (root node) of T_a to the last (inserted node) position q_{newA} and the path length $\mathbb{U}_A$ when the *Triangular-Rewiring* method has been applied to the path. In addition, it shows that $\mathbb{U}_A$ is at least equal to or always shorter than $\mathbb{D}_A$:

$$\xi^{\delta_A+1}(q_{newA}) = q_{start},\ k_{\varphi_A} = \delta_A, \tag{16}$$

$$\mathbb{D}_A = \sum_{j=0}^{\varphi_A} \sum_{n=k'_j}^{k_j} \mathbb{d}_n(q_{newA}),\ \mathbb{U}_A = \sum_{j=0}^{\varphi_A} \mathbb{u}_{k_j}(q_{newA}), \tag{17}$$

$$\therefore \mathbb{D}_A \geq \mathbb{U}_A. \tag{18}$$

Equations 19–21 show the path length $\mathbb{D}_B$ of the path from the start position (root node) of T_b to the last (inserted node) position q_{newB} and the path length $\mathbb{U}_B$ when the *Triangular-Rewiring* method has been applied to the path. In addition, it shows that $\mathbb{U}_B$ is at least equal to or always shorter than $\mathbb{D}_B$:

$$\xi^{\delta_B+1}(q_{newB}) = q_{goal},\ k_{\varphi_B} = \delta_B, \tag{19}$$

$$\mathbb{D}_B = \sum_{j=0}^{\varphi_B} \sum_{n=k'_j}^{k_j} \mathbb{d}_n(q_{newB}),\ \mathbb{U}_B = \sum_{j=0}^{\varphi_B} \mathbb{u}_{k_j}(q_{newB}), \tag{20}$$

$$\therefore \mathbb{D}_B \geq \mathbb{U}_B. \tag{21}$$

Therefore, Equations 16 and 19 can be derived from Equations 3 and 13, Equations 17 and 20 from Equation 14, and Equations 18 and 21 from Equation 15.

As a result, Equations 22 and 23 show that RRT-Connect with the proposed *Triangular-Rewiring* method is at least the same or better in terms of path length than the RRT-Connect algorithm without the method:

$$\mathbb{D}_R = \mathbb{D}_A + \mathbb{D}_B + D(q_{newA}, q_{newB}),\ \mathbb{U}_R \leq \mathbb{U}_A + \mathbb{U}_B + D(q_{newA}, q_{newB}), \tag{22}$$

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

$\mathbb{D}_R$ (Eq. 4), which refers to the path length of the RRT-Connect algorithm path without the *Triangular-Rewiring* method, is represented by the sum of the distance $\mathbb{D}_A$ of the partial path P_a (Eq. 17), the distance $\mathbb{D}_B$ of the partial path P_b (Eq. 20), and the distance $D(q_{newA}, q_{newB})$ between q_{newA} and q_{newB} as shown in Equation 22.

$\mathbb{U}_R$ (Eq. 14), which refers to the path length of the RRT-Connect algorithm path with the *Triangular-Rewiring* method, is equal to or shorter than the sum of the distance $\mathbb{U}_A$ of the partial path P_a for the RRT-Connect (Eq. 17), the distance $\mathbb{U}_B$ of the partial path P_b (Eq. 20), and the distance $D(q_{newA}, q_{newB})$ between q_{newA} and q_{newB} as shown in Equation 22.

Here, Equation 23 shows that $\mathbb{U}_R$ is at least equal to or shorter than $\mathbb{D}_R$ in the RRT algorithm summarized in Equation 15, and it is used efficiently in the RRT-Connect algorithm.

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

This section introduces the *Extend* method in the proposed triangular inequality-based RRT-Connect algorithm. This proposed *Extend* method (A5) replaces the *Extend* method (A3) in the pseudocode of the RRT-Connect algorithm (A2).

Algorithm 5. 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
$\lambda$ ← Step Length $\lambda$ 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($q_{near}$, $q_{rand}$, $\lambda$) then
3       $q_{newA}$ ← Position of Intersection Point between Line Segment connecting $q_{rand}$ and $q_{near}$, and
                Circle with Radius $\lambda$ 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($q_{near}$, $q_{newA}$, $\lambda$) 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 $\lambda$ 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($q_{newA}$, $q_{newB}$, $\lambda$) then
18                  $q_{near}$ ← $q_{newB}$
19                  $q_{newB}$ ← Position of Intersection Point between    // 2D: Circle, 3D: Sphere, …
                        Line Segment connecting $q_{newA}$ and $q_{near}$, and Circle with Radius $\lambda$ centered at $q_{near}$
20              Else
21                  Break
End Extend Procedure from RRT-Connect
```

Algorithm 5 is the application of the *Triangular-Rewiring* method (A4) to the original *Extend* method (A2) of the RRT-Connect algorithm. Compared to the original *Extend* method, the part where a node is newly inserted in the tree in lines 9 and 16 is inserted through the *Triangular-Rewiring* method. Other than that, the contents are the same as the original *Extend* method.

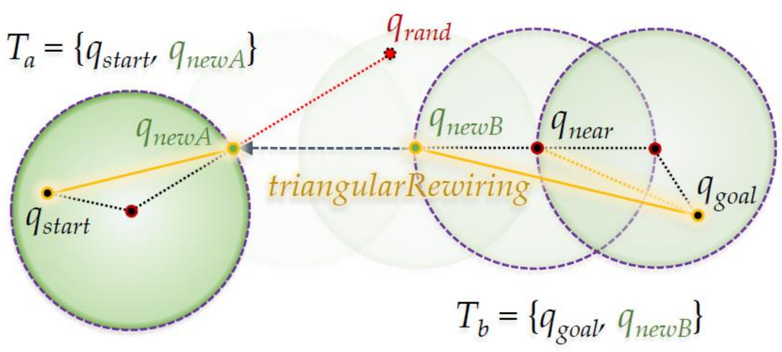


Figure 7. Proposed *Extend* method for the RRT-Connect algorithm.

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 the 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 (A6) replaces the *Connect* method (A4) in the pseudocode of the RRT-Connect algorithm (A2).

Algorithm 6. 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
$\lambda$ ← Step Length $\lambda$ 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($q_{newA}$, $q_{newB}$, $\lambda$) 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 6 is an application of the *Triangular-Rewiring* method (A4) to the *Connect* method (A3) of the RRT-Connect algorithm. Compared to the original *Connect* method, it has been changed to apply the method to the merged tree by considering the *Triangular-Rewiring* method when merging the path, which is in lines 5–10. Other than that, the contents are the same as the original *Connect* method.

When paths P_a and P_b merge 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. In addition, the last inserted node of T_{merged} becomes q_{goal}.

Then, the *Triangular-Rewiring* method is applied to this T_{merged}. Since it is applied to the tree itself, it determines whether rewiring is possible for all nodes inserted in the tree, and rewires and updates the tree if possible. 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}. Here, if q_{newA} is the i-th inserted node, the first node pair to be determined is the $(i - 1)$-th node q_{new} (as q_{child}) and i-th node q_{near} (as q_{parent}). When all nodes inserted in T_{merged} have been determined, the tree structure T_{merged} is converted into the path P_{merged} and the method terminates (*True*).

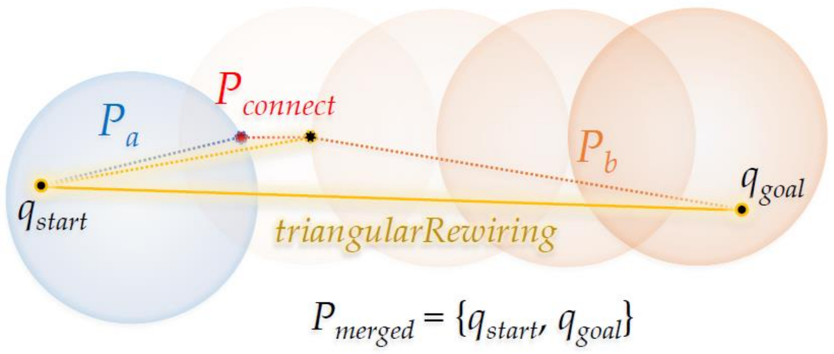
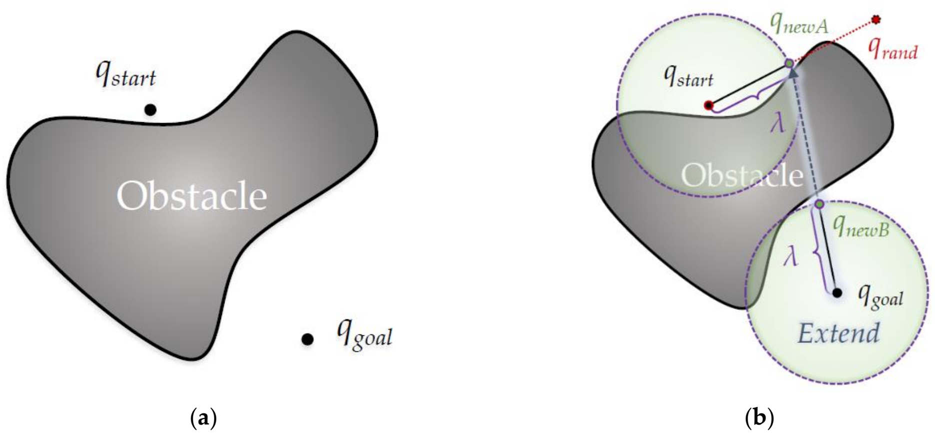


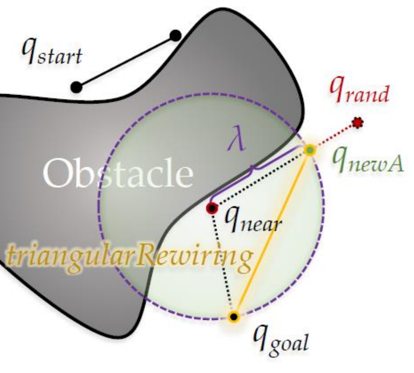
Figure 8. Proposed *Connect* method for the RRT-Connect algorithm.

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 has been applied (assuming there is no obstacle between q_{start} and q_{goal}), the result is P_{merged} in which q_{start} and q_{goal} are connected with a straight line.

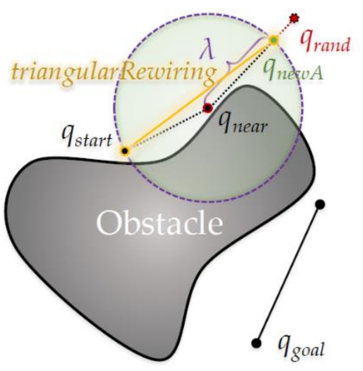
4.5. Process of the Proposed Triangular Inequality-based RRT-Connect Algorithm

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.

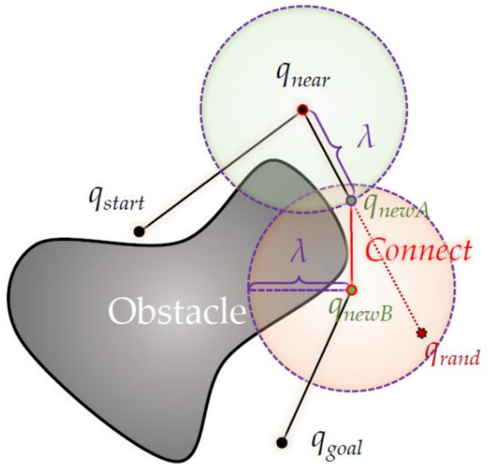




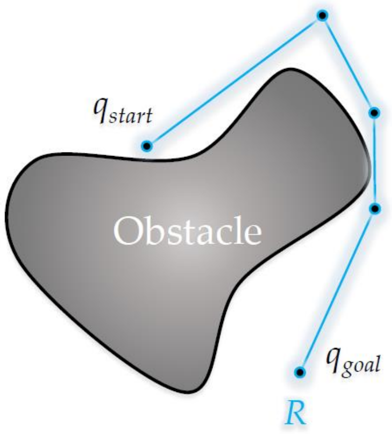
(c)



(d)



(e)



(f)

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 the second random sampling position q_{rand} and rewire between q_{newA} and q_{goal} the ancestor of the q_{newA}; (**d**) Create a new q_{newA} from q_{near} nearest to T_a from the third random sampling position q_{rand} and rewire between q_{newA} and q_{start} with the ancestor of q_{newA}; (**e**) Create new q_{newA} from q_{near} nearest to T_a from the fifth random sampling position q_{rand} and connect between q_{newA} and q_{newB} nearest to T_b from q_{newA}; (**f**) Result of Path R from q_{start} to q_{goal}.

Figure 9 shows planning 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 position q_{rand} 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. In addition, 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 separated by the length of λ in the position direction, at the node q_{near} that is nearest among 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}. Here, since it also collides with an obstacle to extend in the direction of 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} is also 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 when updating that q_{newA} merges into one tree through the *Connect* process because q_{newA} is within range of 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 the q_{newA} generation process is skipped. 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 in various environment maps shown in the experimental environment through the simulator.

Each algorithm was implemented based on the pseudocode (A1–9) shown in Chapters 3 and 4 (For the RRT algorithm, refer to the pseudocode (AS1) in Appendix A), and the performance measures used for comparison of various algorithms are ***Number of sampling*** (samples), ***Path length*** (pixels), and ***Planning time*** (milliseconds). And each performance measure is experimented with 50 trials from the same start point to the same goal point until the first path has been found). Among the performance measures, as the number of samples decreases, the cost of recalculation in a dynamic environment also decreases, and the path length is a measure of the optimality of the path-planning algorithm. In addition, the Step length (λ) is 30 pixels.

5.1. Experimental Environment

This section introduces the environment map used in the simulation and the simulator used in the simulation with the computer's performance.

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

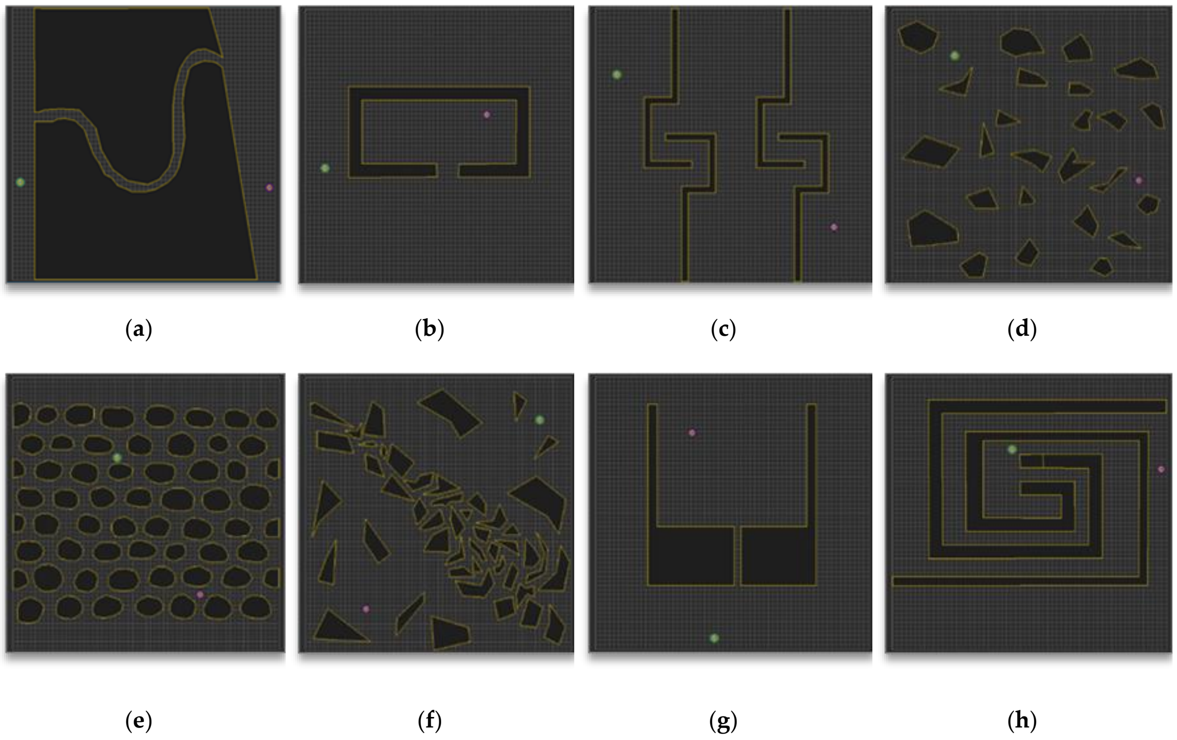


Figure 10. Maps for the experiment: (**a**) Map 1; (**b**) Map 2; (**c**) Map 3; (**d**) Map 4; (**e**) Map 5; (**f**) Map 6; (**g**) Map 7; (**h**) Map 8.

Many environmental maps were considered and used to verify the performance of various path-planning algorithms including the RRT algorithm, [23–26]. Which environment map to use is important because the expected performance measure varies depending on the obstacles' placement and shape among other properties.

In this paper, to check the proposed algorithm's performance, the eight maps shown in Figure 10 were benchmarked in the experimental environment of the paper [27] 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 in which it is easy to verify the completeness of the path-planning algorithm. Map 2 in Figure 10 (b) seems to be an environment in which it is also easy to verify the completeness of the path-planning algorithm, and the environment is mainly used to show the solution for the Local Minima problem [28] in the artificial potential field algorithm [26]. Map 3 in Figure 10 (c) seems to be an environment in which it 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 in Figure 10 (d) seems to be an environment in which it is easy to verify the optimality and the planning time for the path-planning algorithm, and the Cell Decomposition algorithm, which increases the computation cost as the angle of obstacle increases, is an unfavorable environment [29]. Map 5 in Figure 10 (e) seems to be an environment in which it is also easy to verify the optimality and planning time of the path-planning algorithm; 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 in which it is easy to verify the optimality, completeness, and planning 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 in which it is easy to verify the completeness and optimality of the path-planning algorithm, and for the same reason as Map 2, it is the environment used in the Artificial Potential Field algorithm. Lastly, Map 8 in Figure 10 (h) seems to be an environment in which it is easy to verify the completeness and planning time of the path-planning algorithm and yet is 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 samples and the planning time are extremely increased as long as there are narrow or fewer entrances for directions to the goal.

Table 1. Computer performance for simulation.

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

Table 1 shows the specifications 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 was used for the calculation. Differences in planning time may occur depending on the computer's performance capability.

5.2. Experimental Results and Analysis for Each Map

This section checks the experimental results (on average, the number of samples, path length, and planning time) of each algorithm: RRT, RRT-Connect, the proposed algorithm in the eight environment maps (Fig. 10) presented in the experimental environment. Each map shows a figure of the path-planning result (of one trial) for each algorithm and the experimental results for the performance measure are shown numerically in a table (The figure for each algorithm is for one trial rather than the average of repeated trials and it may differ from the performance measure both visually and by the average numerical performance measure of the repeated trials shown in the table. In particular, the number of samples differs greatly).

The values shown in Tables 2–9 can be expressed as Equations 24 and 25 as follows:

$$A_{cmp}(i) = \sum_{k=0}^{T} a_{cmp_k}(i)\,/T, \tag{24}$$

Here, $A_{cmp}(i)$ refers to the performance value of each algorithm shown in Tables 2–9, *cmp* is the algorithm to be compared, *i* is the index of the environment map (X-axis in Figures 19–21 (b)), *k* is the repeat index, and *T* is the number of repeats ($a_{cmp_k}(i)$ is the value of the performance measure *a* for the *k*-th implementation of the *cmp* algorithm in Map *i*). Fifty repetitions are performed for the experiment in this paper. That is, Equation A shows the average value of the performance when it is repeated *T* times to check the performance of a certain algorithm in Map *i*,

$$\therefore x_{cmp}(i) = A_{cmp}(i)/A_{RRT}(i), \tag{25}$$

Here, $x_{cmp}(i)$ refers to the Y-axis in Figures 19–21 (a) and *A* is the value of the corresponding performance measure of the algorithm to be compared (A_{RRT} is the value of the RRT algorithm).

In each path-planning result figure, the white circles indicate nodes on the path and the yellow line segments indicate edges between nodes. The gray circles and segments are paths (trees) that have been 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.

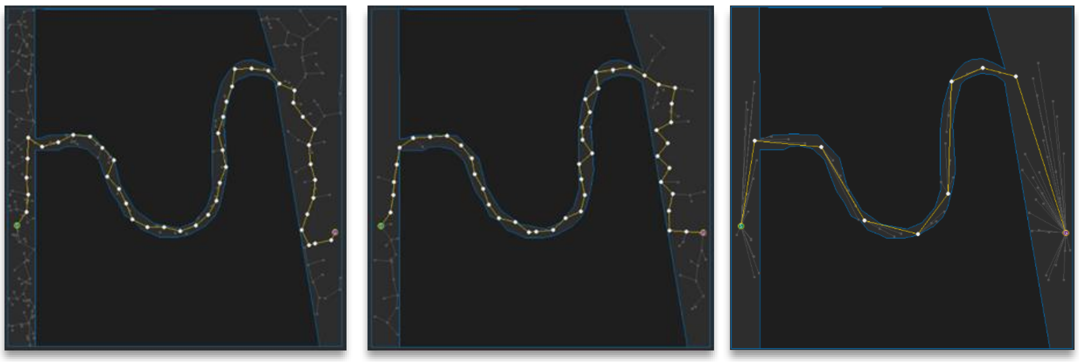

(**a**) (**b**) (**c**)

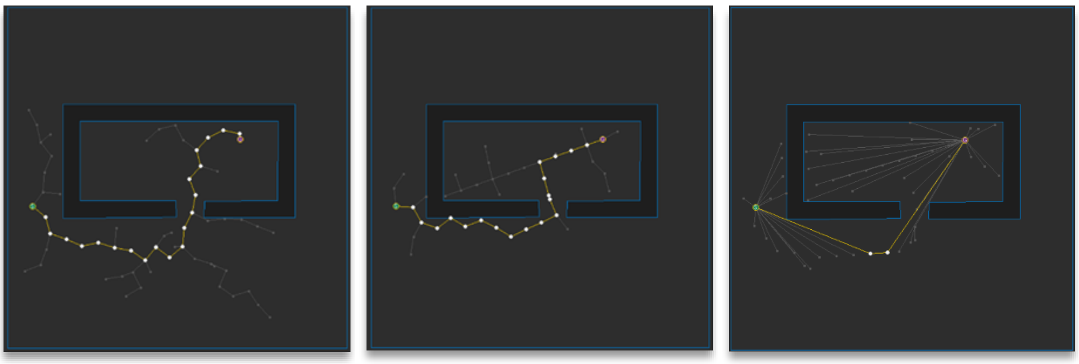
Figure 11. Experimental result of Map 1: (**a**) RRT; (**b**) RRT-Connect; (**c**) the proposed algorithm.

Figure 11 shows the path-planning results of Map 1 among the environmental maps for each algorithm. Visually, the number of samples looks similar to the RRT-Connect algorithm in Figure 11 (b) and the proposed algorithm in Figure 11 (c) is comparable to the RRT algorithm in Figure 11 (a), and the path length 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% ($x_{cmp}(1)$)).

Performance ($\boldsymbol{A_{cmp}(1)}$)	**RRT**	**RRT-Connect**	**Proposed Algorithm**
Avg. Num. of Samples [samples]	1,216 *(100)*	729 *(60)*	823 *(68)*
Avg. Path length [px]	1,341 *(100)*	1,343 *(100)*	1,200 *(89)*
Avg. Planning time [ms]	12 *(100)*	7 *(58)*	10 *(83)*

Table 2 shows the path-planning results (after repeating the trial 50 times) in Map 1 for each algorithm. The average number of samples is the smallest in RRT-Connect algorithm at 60%, and the proposed algorithm is 68% compared to the RRT algorithm, which is 8% less efficient than the RRT algorithm compared to the RRT-Connect algorithm. The average path length is 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 planning time is the shortest for the RRT-Connect algorithm at 58% compared to the RRT algorithm, and the proposed algorithm is 83% compared to the RRT algorithm, i.e., 15% less efficient than the RRT algorithm.



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

Figure 12. Experimental results 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. Visually, the number of samples 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 path length looks shortest for the proposed algorithm.

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

Performance ($\boldsymbol{A_{cmp}(2)}$)	**RRT**	**RRT-Connect**	**Proposed Algorithm**
Avg. Num. of Samples [samples]	271 *(100)*	101 *(37)*	113 *(42)*
Avg. Path length [px]	598 *(100)*	613 *(98)*	484 *(81)*
Avg. Planning time [ms]	6 *(100)*	3 *(50)*	3 *(50)*

Table 3 shows the path-planning result (after repeating the trials 50 times) in Map 2 for each algorithm. The average number of samples is smallest in the RRT-Connect algorithm at 37%, and the proposed algorithm is 42% compared to the RRT algorithm, which is 5% less efficient than RRT algorithm compared to the RRT-Connect algorithm. The average path length of the proposed algorithm is the shortest at 81% compared to the RRT algorithm, while the RRT-Connect algorithm is 98%, which is 17% less efficient than the RRT algorithm compared to the proposed algorithm. The average planning time for the proposed algorithm and the RRT-Connect shows the same performance as the RRT algorithm.

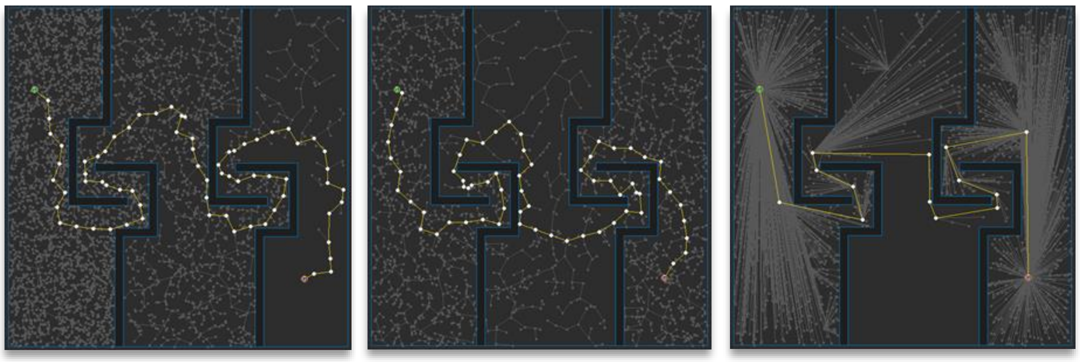

(a) (b) (c)

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. Visually, the number of samples 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 path length looks shortest for the proposed algorithm.

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

Performance ($A_{cmp}(3)$)	**RRT**	**RRT-Connect**	**Proposed Algorithm**
Avg. Num. of Samples [samples]	6,106 *(100)*	4,574 *(75)*	4,679 *(77)*
Avg. Path length [px]	1,934 *(100)*	1,871 *(97)*	1,489 *(77)*
Avg. Planning time [ms]	866 *(100)*	299 *(35)*	313 *(36)*

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

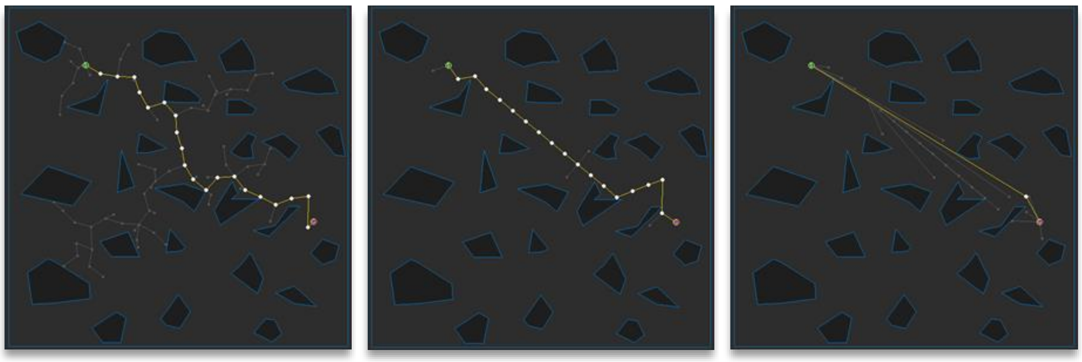

(a) (b) (c)

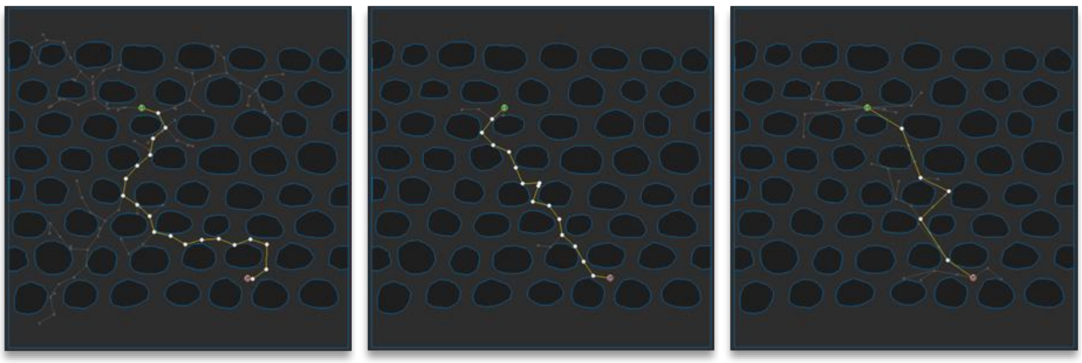
Figure 14. Experimental result of Map 4: (**a**) RRT; (**b**) RRT-Connect; (**c**) the proposed algorithm.

Figure 14 shows the path planning results of Map 4 among the environmental maps for each algorithm. Visually, the number of samples looks smallest for the RRT-Connect algorithm in Figure 14 (b) compared to the others and the path length 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% ($x_{cmp}(4)$)).

Performance ($A_{cmp}(4)$)	**RRT**	**RRT-Connect**	**Proposed Algorithm**
Avg. Num. of Samples [samples]	290 *(100)*	28 *(10)*	32 *(11)*
Avg. Path length [px]	711 *(100)*	588 *(83)*	534 *(75)*
Avg. Planning time [ms]	3 *(100)*	3 *(100)*	4 *(133)*

Table 5 shows the result (after repeating the trial 50 times) of path planning in Map 4 for each algorithm. The average number of samples is smallest in the RRT-Connect algorithm at 10%, and the proposed algorithm is 11% compared to the RRT algorithm, which is 1% less efficient than the RRT algorithm compared to the RRT-Connect algorithm. The average path length of the proposed algorithm is the shortest at 75% compared to the RRT algorithm and the RRT-Connect algorithm is 83%, which is 8% less efficient than the RRT algorithm compared to the proposed algorithm. The average planning time is not different by 100% compared to the RRT algorithm, and the proposed algorithm is 133% compared to the RRT algorithm, i.e., 33% less efficient than the 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. Visually, the number of samples 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 path length looks similar for the RRT-Connect algorithm and the proposed algorithm.

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

Performance ($A_{cmp}(5)$)	**RRT**	**RRT-Connect**	**Proposed Algorithm**
Avg. Num. of Samples [samples]	371 *(100)*	68 *(18)*	74 *(20)*
Avg. Path length [px]	554 *(100)*	588 *(106)*	465 *(84)*
Avg. Planning time [ms]	13 *(100)*	2 *(15)*	2 *(15)*

Table 6 shows the results (after repeating the trial 50 times) of path planning in Map 5 for each algorithm. The average number of samples is smallest in RRT-Connect algorithm at 18%, and the proposed algorithm is 20% compared to the RRT algorithm, which is 9% less efficient than the RRT algorithm compared to the RRT-Connect algorithm. The average path length 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 planning time for the proposed algorithm and the RRT-Connect algorithm is 15% over the RRT algorithm, showing the same performance.

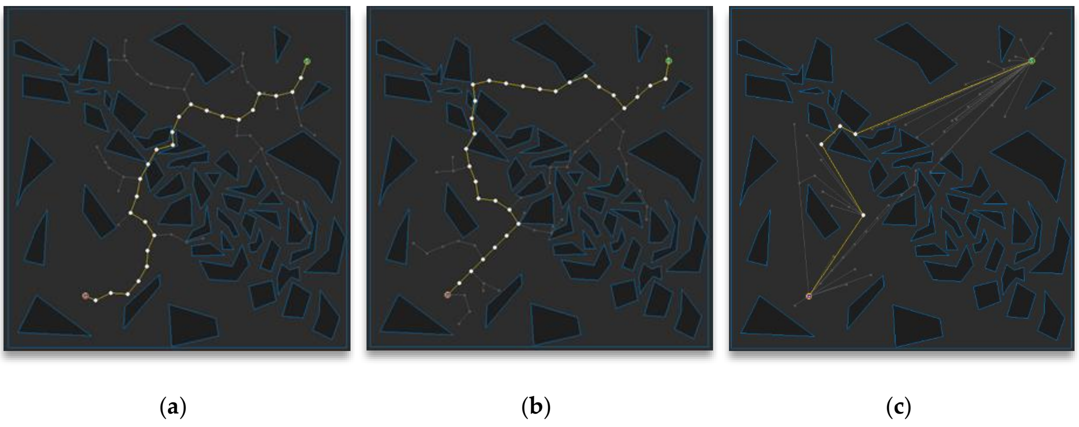


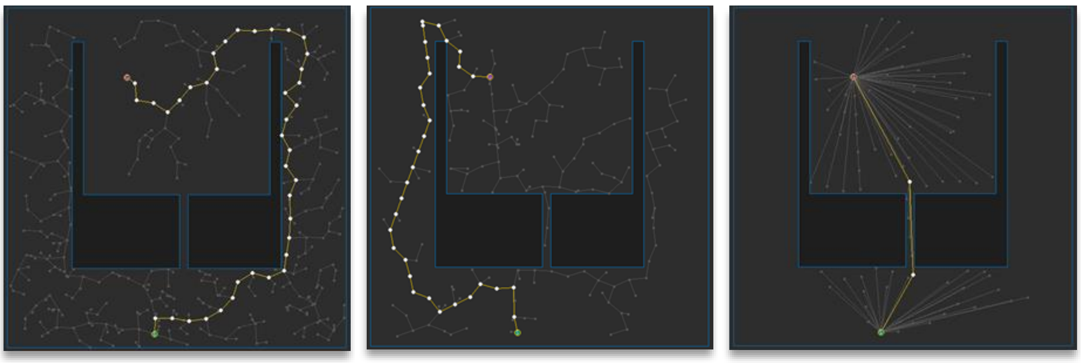
Figure 16. Experimental result of Map 6: (**a**) RRT; (**b**) RRT-Connect; (**c**) the proposed algorithm.

Figure 16 shows the path-planning results of Map 6 among the environmental maps for each algorithm. Visually, the number of samples looks smallest for the proposed algorithm in Figure 16 (c) compared to others, and the path length looks shortest for the proposed algorithm.

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

Performance ($A_{cmp}(6)$)	**RRT**	**RRT-Connect**	**Proposed Algorithm**
Avg. Num. of Samples [samples]	541 *(100)*	184 *(34)*	140 *(26)*
Avg. Path length [px]	886 *(100)*	778 *(88)*	668 *(75)*
Avg. Planning time [ms]	9 *(100)*	6 *(67)*	4 *(44)*

Table 7 shows the result (after repeating the trial 50 times) of path planning in Map 6 for each algorithm. The average number of samples is smallest in the proposed algorithm at 26% and the RRT-Connect algorithm is 34% compared to the RRT algorithm, which is 8% less efficient than RRT algorithm compared to the proposed algorithm. The average path length 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 than the proposed algorithm. The average planning time is smallest in the proposed algorithm at 44%, and the RRT-Connect is 67% compared to the RRT algorithm, which is 23% less efficient than the RRT algorithm compared to the proposed algorithm.



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

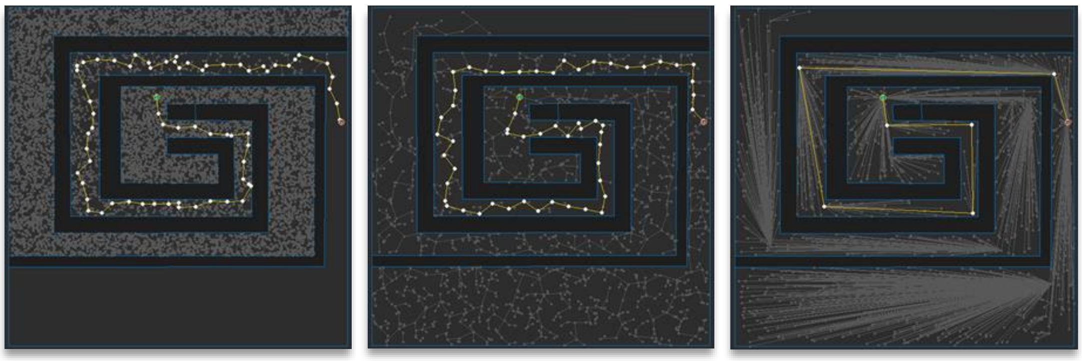
Figure 17. Experimental result of Map 7: (**a**) RRT; (**b**) RRT-Connect; (**c**) the proposed algorithm.

Figure 17 shows the path planning results of Map 7 among the environmental maps for each algorithm. Visually, the number of samples looks smallest for the proposed algorithm in Figure 17 (c) compared to others, and the path length 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% ($x_{cmp}(7)$)).

Performance ($\boldsymbol{A_{cmp}(7)}$)	**RRT**	**RRT-Connect**	**Proposed Algorithm**
Avg. Num. of Samples [samples]	436 *(100)*	235 *(54)*	244 *(56)*
Avg. Path length [px]	898 *(100)*	862 *(96)*	674 *(75)*
Avg. Planning time [ms]	5 *(100)*	4 *(80)*	3 *(60)*

Table 8 shows the result (after repeating the trial 50 times) of path planning in Map 7 for each algorithm. The average number of samples is smallest in RRT-Connect algorithm at 54%, and the proposed algorithm is 56% compared to the RRT algorithm, which is 2% less efficient than RRT algorithm compared to the RRT-Connect algorithm. The average path length of the proposed algorithm is 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 planning time is smallest in the proposed algorithm at 60%, and RRT-Connect is 80% compared to the RRT algorithm, which makes it 20% less efficient than the RRT algorithm compared to the 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. Visually, the number of samples 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 path length looks shortest for the proposed algorithm.

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

Performance ($\boldsymbol{A_{cmp}(8)}$)	**RRT**	**RRT-Connect**	**Proposed Algorithm**
Avg. Num. of Samples [samples]	17,033 *(100)*	3,031 *(18)*	2,954 *(17)*
Avg. Path length [px]	1,611 *(100)*	1,576 *(98)*	1,358 *(84)*

Avg. Planning time [ms]	4,501 *(100)*	119 *(3)*	125 *(3)*

Table 9 shows the result (after repeating the trial 50 times) of path planning in Map 8 for each algorithm. The average number of samples is smallest in the proposed algorithm at 17%, and the RRT-Connect algorithm is 18% compared to the RRT algorithm, which is 1% less efficient than RRT algorithm compared to the proposed algorithm. The average path length 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 planning time of the proposed algorithm and the RRT-Connect algorithm is 3% over the RRT algorithm, showing the same performance.

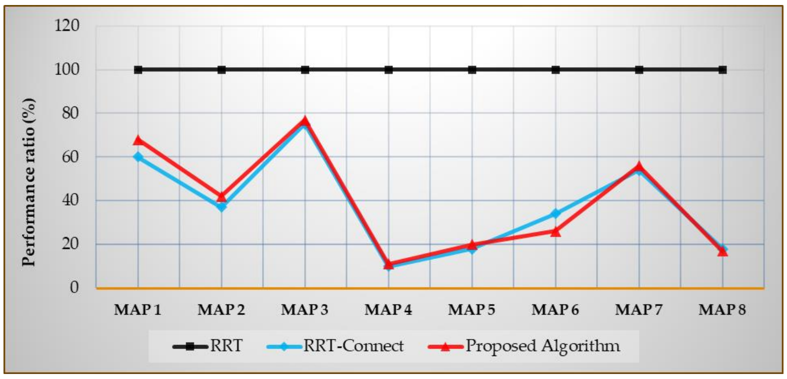
5.3. Experimental Results and Analysis in Total

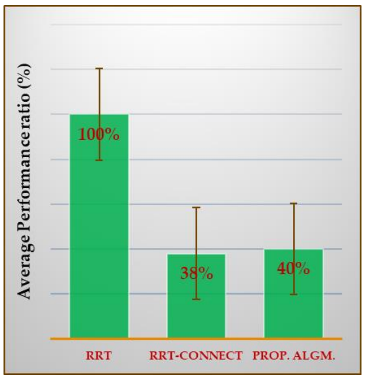
This section comprehensively presents the experimental results (on average, number of samples, path length, and planning time) for each algorithm: RRT, RRT-Connect, and the proposed triangular inequality-based RRT-Connect algorithm, in the eight environmental maps (Fig. 10) shown in Section 5.2.

Figures 19 (a), 20 (a), and 21 (a) show the performances of the RRT-Connect algorithm and the proposed algorithm when the RRT algorithm's performance is set to 100% for each environment map. The (b) of each figure shows the performance average of all environment maps for each algorithm. The values shown in (a) of Figures 19–21 can be expressed as in Equations 24 and 25 and the values shown in (b) can be expressed as Equation 26 below:

$$X_{cmp} = \sum_{i=0}^{M} x(i)_{cmp} / M, \tag{26}$$

Here, X_{cmp} refers to the Y-axis in (b) of Figures 19–21 and M is the number of environment maps used in the experiment. The experiment in this paper includes eight maps. That is, Equation 26 shows the average value of i for all maps in Equation 25.





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

Figure 19. Experimental results in total for the average number of samples(for first path finding): (**a**) result of each map compared with the RRT algorithm ($x_{cmp}(i)$); (**b**) average result compared with the RRT algorithm (X_{cmp}).

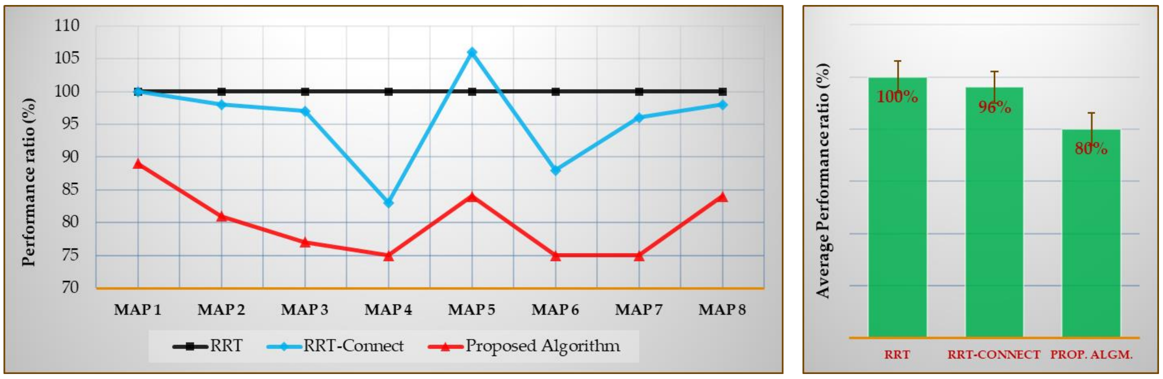
Figure 19 shows the average number of samples [%] compared with the RRT algorithm for Maps 1–8 (after repeating the trial 50 times) and the average number of samples [%] compared with the average result of each algorithm for each map (after repeating the trials 50 times) when the result of RRT algorithm is considered 100%.

As shown in Figure 19 (b), the average number of samples 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. Experimental results in total for the average number of samples(for first path finding) [%].

Algorithm (*cmp*)	Performance ratio based on RRT ($\boldsymbol{x_{cmp}(i)}$)								Avg. ($\boldsymbol{X_{cmp}}$)
	Map 1	**Map 2**	**Map 3**	**Map 4**	**Map 5**	**Map 6**	**Map 7**	**Map 8**	
RRT	100	100	100	100	100	100	100	100	100
RRT-Connect	60	37	75	10	18	34	54	18	38
Proposed	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 than the RRT-Connect algorithm for Maps 6 and 8 and the RRT-Connect algorithm shows better performance than the proposed algorithm in Maps 1–5 and 7. However, the difference is not significant for most of the maps, such as showing a 2% difference from the map average. There are cases in which the proposed algorithm is 1–8% better than the RRT-Connect algorithm and there are cases in which the RRT-Connect algorithm is 1–8% better than the proposed algorithm.



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

Figure 20. Experimental results in total for the average path length: (**a**) result of each map compared with the RRT algorithm ($x_{cmp}(i)$); (**b**) average result compared with the RRT algorithm (X_{cmp}).

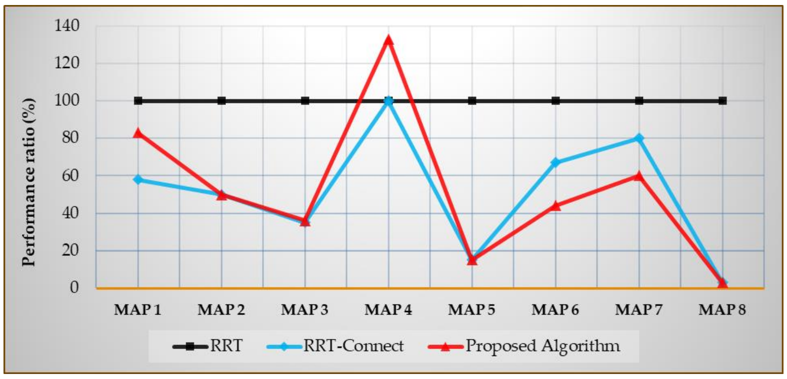
Figure 20 shows the average path length [%] compared to the RRT algorithm for Maps 1–8 (after repeating the trials 50 times), and the average path length [%] compared with the average result of each algorithm for each map (again after repeating the trials 50 times) where the result of the RRT algorithm was considered as 100%.

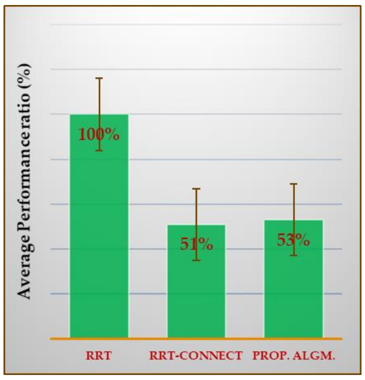
As shown in Figure 20 (b), the average path length for all environment maps was 96% less in the RRT-Connect algorithm and 80% less in the proposed algorithm compared to the RRT algorithm. The proposed algorithm is 16% more efficient than the RRT-Connect algorithm.

Table 11. Experimental results in total on the average path length [%].

Algorithm (*cmp*)	Performance ratio based on RRT ($\boldsymbol{x_{cmp}(i)}$)								Avg. ($\boldsymbol{X_{cmp}}$)
	Map 1	**Map 2**	**Map 3**	**Map 4**	**Map 5**	**Map 6**	**Map 7**	**Map 8**	
RRT	100	100	100	100	100	100	100	100	100
RRT-Connect	100	98	97	83	106	88	96	98	96
Proposed	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 than the RRT-Connect algorithm for all maps. The proposed algorithm is 8–21% better than the RRT-Connect algorithm.





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

Figure 21. Experimental results in total on the average planning time: (**a**) result of each map compared with the RRT algorithm ($x_{cmp}(i)$); (**b**) average result compared to the RRT algorithm (X_{cmp}).

Figure 21 shows the average planning time [%] compared to the RRT algorithm for each map (after repeating the trials 50 times), and the average planning time [%] compared with the average result of each algorithm for each map (after repeating the trials 50 times) when the result of RRT algorithm is considered 100%.

As shown in Figure 21 (b), the average planning 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 was 2% less efficient than the RRT-Connect algorithm.

Table 12. Experimental results in total for the average planning time [%].

Algorithm (*cmp*)	Performance ratio based on RRT ($\boldsymbol{x_{cmp}(i)}$)								Avg. ($\boldsymbol{X_{cmp}}$)
	Map 1	**Map 2**	**Map 3**	**Map 4**	**Map 5**	**Map 6**	**Map 7**	**Map 8**	
RRT	100	100	100	100	100	100	100	100	100
RRT-Connect	58	50	35	100	15	67	80	3	51
Proposed	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 Maps 2 and 5–8 than the RRT-Connect algorithm. It shows worse performance for Maps 1, 3 and 4 than the RRT-Connect algorithm. However, most of the maps show no significant difference, such as showing a 2% difference from the map average. There are cases in which the proposed algorithm is 20–23% better than the RRT-Connect algorithm and there are cases where the RRT-Connect algorithm is 1–33% 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 limitations in the optimality of the RRT-Connect algorithm. We verified the validity of the *Triangular-Rewiring* method based on the triangular inequality principle and applied it to the RRT-Connect algorithm to bring it closer to the optimum. In addition, to check performance indicators such as the number of samples for finding the first path, path length, and planning time of the proposed algorithm, we compared between the RRT and RRT-Connect algorithms across a total of eight environments through simulation. On average, the proposed algorithm showed 20% better efficiency than the RRT algorithm and 16% better efficiency than the RRT-Connect algorithm in path length and 47% better efficiency than the RRT algorithm in planning time but 2% worse efficiency than the RRT-Connect algorithm. In conclusion, the proposed algorithm showed shorter paths than the RRT-Connect algorithm with a similar number of samples and planning time.

However, one of the limitations of the proposed algorithm is the Kinodynamic planning problem [17]. When the intermediate node disappears by Triangular-Rewiring method, a non-differentiable piecewise linear section with sharp corner may occurs, which cause a problem related with the kinematic constraint of the robot.

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.

Appendix A. Details 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 (AS1) and two additional functions (AS2 and AS3).

A.1. Pseudocode of the RRT Algorithm

This section shows the pseudocode of main algorithm (AS1) of the RRT algorithm used in the experiment of this paper, designed based on the paper [13] in which the RRT algorithm was proposed.

Algorithm S1. Pseudocode of the RRT Algorithm.

```
Input:
$q_{start}$ ← Position of Start Point
$q_{goal}$ ← Position of Goal Point
$\lambda$ ← 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
$d_{shorter}$ ← 0

Begin RRT Procedure
1   $T$ ← Insert Root Node<$q_{start}$> to $T$
2   While 1 ← $n$ to $N$ do
3       Generate $n$-th Random Sample
4       $q_{rand}$ ← Position of $n$-th Random Sample
5       $q_{near}$ ← Find Position of Nearest Node in $T$ from $q_{rand}$
6       If Not $isInside(q_{near}, q_{rand}, \lambda)$ then
```

```
7               $q_{new}$ ← Position of Intersection Point between Line Segment connecting $q_{rand}$ and $q_{near}$,
                    and Circle with Radius $\lambda$ centered at $q_{near}$        // 2D: Circle, 3D: Sphere, …
8           Else
9               $q_{new}$ ← $q_{rand}$
10          If Not isTrapped($q_{new}$, $q_{near}$, $C$) then
11              $T$ ← Insert Node<$q_{new}$> and Edge<$q_{new}$, $q_{near}$> to $T$
12              If isInside($q_{new}$, $q_{goal}$, $\lambda$) then
13                  $T$ ← Insert Node<$q_{goal}$> and Edge<$q_{new}$, $q_{goal}$> to $T$
14                  $P_{reach}$ ← Path from Last Inserted Node [$q_{goal}$] to Root Node [$q_{start}$] in $T$
15                  $d_{reach}$ ← Distance of $P_{reach}$
16                  If $d_{shorter}$ = 0 or $d_{shorter}$ > $d_{reach}$ then
17                      $R$ ← $P_{reach}$
18                      $d_{shorter}$ ← $d_{reach}$
19                  $T$ ← Delete Node<$q_{goal}$> and Edge<$q_{new}$, $q_{goal}$> from $T$
End RRT Procedure
```

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, as required until the tree is expanded. q_{rand} is position of generated random sample, in which the 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* (AS2), 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 the intersection point between the line segment connecting q_{rand} and q_{near} and the 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* (AS3) (in the *isTrapped* function, C refers to the set of obstacles).

If q_{goal} is 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 path length $d_{shorter}$ or reached first ($d_{shorter}$ = 0), the result path R becomes P_{reach}, and $d_{shorter}$ becomes d_{reach}. At the end of the next N sampling, R becomes the final planned path.

If the number of random samples 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.2. Pseudocode of the functions used in the RRT Algorithm

This section introduces additional functions used in pseudocode of the RRT algorithm (AS1) in Section A.1. The *isTrapped* (AS2) function determines whether an obstacle collides, and the *isInside* function (AS3) determines if the point exists inside the radius.

Algorithm S2. 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
```

```
$\lambda$ ← Step Length $\lambda$ 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
```

AS2 shows the *isInside* function among RRT pseudocodes shown in AS1. In the 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 S3. 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 S3 shows the *isTrapped* function among RRT algorithm pseudocodes shown in Algorithm S1. In the 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, then 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. Currently, if any intersect occurs, AS3 returns *True* and stops the procedure immediately. Otherwise, it determines all the line segments that can be l_c and returns *False*.

A.3. Basic Mathematical Modeling of the RRT Algorithm

This chapter introduces basic mathematical modeling in the RRT algorithm. The following Equations, S1–S5, show that the coordinate value of q_{new} is calculated from the coordinate value of q_{rand} in the RRT algorithm:

$$d = \sqrt{(q_{rand}.\mathrm{x} - q_{near}.\mathrm{x})^2 + (q_{rand}.\mathrm{y} - q_{near}.\mathrm{y})^2}, \tag{S1}$$

$$|q_{rand}.\mathrm{x} - q_{near}.\mathrm{x}| : |q_{new}.\mathrm{x} - q_{near}.\mathrm{x}| = d : \lambda, \tag{S2}$$

$$|q_{rand}.\mathrm{y} - q_{near}.\mathrm{y}| : |q_{new}.\mathrm{y} - q_{near}.\mathrm{y}| = d : \lambda. \tag{S3}$$

Equations S2-S3 shows the relationship between d and λ in Equation S1 through the similarity ratios from q_{near} to q_{rand} and from q_{near} to q_{new}:

$$\therefore q_{new}.\mathrm{x} = \begin{cases} q_{rand}.\mathrm{x}, & d \le \lambda \\ |(\lambda|q_{rand}.\mathrm{x} - q_{near}.\mathrm{x}|/d) + q_{near}.\mathrm{x}|, & d > \lambda' \end{cases} \tag{S4}$$

$$\therefore q_{new}.\mathrm{y} = \begin{cases} q_{rand}.\mathrm{y}, & d \le \lambda \\ |(\lambda|q_{rand}.\mathrm{y} - q_{near}.\mathrm{y}|/d) + q_{near}.\mathrm{y}|, & d > \lambda' \end{cases} \tag{S5}$$

Through these equations, the x and y coordinate values of q_{new} can be derived as shown in Equations S4 and S5. In this case, $d \le \lambda$ refers to a case where q_{rand} exists in a position inside the λ radius.

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. Springer: London, UK, 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. Noreen, I.; Amna K.; Zulfiqar H. A comparison of RRT, RRT* and RRT*-smart path planning algorithms. *International Journal of Computer Science and Network Security*. **2016**, *16*(10), 20.
23. 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.
24. 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.

25. 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.
26. Jeon, G.-Y.; Jung, J.-W. Water sink model for robot motion planning. *Sensors*. **2019**, *19*(6), 1269.
27. Han, J. Mobile robot path planning with surrounding point set and path improvement. *Applied Soft Computing*. **2017**, *57*, 35-47.
28. Yoon, H.U.; Lee, D.-W. Subplanner algorithm to escape from local minima for artificial potential function based robotic path planning. *International Journal of Fuzzy Logic and Intelligent Systems*. **2018**, *18*(4), 263-275.
29. 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.