J Inf Process Syst, Vol.16, No.6, pp.1324~1342, December 2020
https://doi.org/10.3745/JIPS.04.0196

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

Boundary-RRT* Algorithm for Drone Collision Avoidance and Interleaved Path Re-planning

Je-Kwan Park* and Tai-Myoung Chung**

Abstract

Various modified algorithms of rapidly-exploring random tree (RRT) have been previously proposed. However, compared to the RRT algorithm for collision avoidance with global and static obstacles, it is not easy to find a collision avoidance and local path re-planning algorithm for dynamic obstacles based on the RRT algorithm. In this study, we propose boundary-RRT*, a novel-algorithm that can be applied to aerial vehicles for collision avoidance and path re-planning in a three-dimensional environment. The algorithm not only bounds the configuration space, but it also includes an implicit bias for the bounded configuration space. Therefore, it can create a path with a natural curvature without defining a bias function. Furthermore, the exploring space is reduced to a half-torus by combining it with simple right-of-way rules. When defining the distance as a cost, the proposed algorithm through numerical analysis shows that the standard deviation (σ) approaches 0 as the number of samples per unit time increases and the length of epsilon ε (maximum length of an edge in the tree) decreases. This means that a stable waypoint list can be generated using the proposed algorithm. Therefore, by increasing real-time performance through simple calculation and the boundary of the configuration space, the algorithm proved to be suitable for collision avoidance of aerial vehicles and re-planning of local paths.

Keywords

Collision Avoidance, Drone, Global Path Planning, Local Path Planning, Planner, Rapidly-exploring Random Tree (RRT), RRT* (RRT star), Torus, Unmanned Aerial Vehicle (UAV), Velocity Obstacle

1. Introduction

Recently, drones have been used not only in the military and environmental fields, but they are also widely utilized in various applications involving, for example, logistics, movies, and disaster relief. In the near future, drones will also be applied to transportation field so that people can board and safely move from one destination to another. As the field of application and the number of increases, drones will fly in the sky at low altitudes and at high traffic densities. The most important aspect of this high traffic density is the collision detection technique and autonomous avoidance of collision. The greater the traffic density, the more collisions will be detected, and the more complex the techniques for collision avoidance.

Collision avoidance for commonly known static obstacles is considered in global path planning, and dynamic or unexpected obstacles are considered in local path planning. In particular, many studies apply

Manuscript received May 28, 2020; first revision September 4, 2020; second revision October 21 2020; accepted October 22, 2020.
Corresponding Author: Tai-Myoung Chung (tmchung@skku.edu)
* Dept. of Electrical and Computer Engineering, Sungkyunkwan University, Suwon, Korea (jkpark68@skku.edu)
** College of Information and Communication Engineering, Sungkyunkwan University, Suwon, Korea (tmchung@skku.edu)

the theory of velocity obstacles to avoid local collisions. The collision avoidance maneuver is completed by selecting a velocity outside of velocity obstacle (VO) [1-4], which is expressed as a set related to the velocity of ownship or intruder. Within the defined look-ahead time Δt, the velocity at which collision avoidance is possible is selected from the VO-derived velocity set, and the optimal velocity is selected using a heuristic algorithm. The definition and calculation of time-related parameters for collision detection and collision avoidance are critical to the algorithm's performance and reliability. In particular, once the collision potential is identified, the new paths must be immediately explored and prepared for maneuvers for obstacle avoidance. The proposed boundary-RRT* algorithm does not require a complete and optimized new path plan to be completed before starting the collision avoidance maneuver. Based on the asymptotic optimality of rapidly-exploring random tree (RRT) [5-13], we can continuously search for and optimize the optimal path during flight. The boundary-RRT* algorithm is performed in two steps. The first is assignment, which is a global planner when performing matching between drones and destinations using the relatively simple Hungarian assignment algorithm (HAA) [14] and generating waypoints. The second is a local planner that detects collisions with dynamic or static obstacles during actual flight and re-plans paths to avoid collisions.

The goal of the boundary-RRT* algorithm is to rapidly detect collisions, pay a low cost in airspace and distance, and plan the optimal path by applying the advantages of VO based on a time-varying environment and RRT* based on spatial-configuration environments. The boundary-RRT* algorithm assumes that the sampling region of the configuration space is a subset of the velocity space. According to [1], the reachable avoidance velocity (RAV) derived from the VO over time t can be instantly mapped into a series of geometric shapes for collision avoidance manipulation in the velocity space. This subset can be shaped into a torus-like region according to time t, and the RRT* convergence speed can be significantly improved by limiting exploration to the torus. VO simply detects collisions, calculates the position for collision avoidance maneuvering through simple geometry-based Euclidean distance calculations, and re-plan the optimal path by the boundary-RRT* planner by exploring the region.

The contributions of the proposed algorithm to the literature are from three perspectives. First, the algorithm is a three-dimensional (3D) algorithm that can be applied to both dynamic and static obstacles. Second, the algorithm does not define speed as a constant to avoid collisions; that is, if the right-of-way rules are flexibly extended, collisions with unpredictable dynamic obstacles with variable speeds can be avoided. Third, by simplifying the calculation, we can reduce the execution time of the algorithm.

The remainder of this paper is organized as follows. Section 2 reviews the existing literature related to collision detection and avoidance, as well as other relevant studies. Section 3 first describes the global planner HAA and explains how to calculate the various parameters for the boundary-RRT* algorithm, which is a local planner for collision detection and avoidance, and how to use these parameters to limit the exploration space of the boundary-RRT* algorithm. Section 4 defines and describes the pseudo-code of the boundary-RRT* algorithm. Section 5 explains the simulation and results using MATLAB. Section 6 compares and evaluates several-path planning algorithms and the boundary-RRT* algorithm. Finally, Section 7 provides the conclusions of the study and future research directions.

2. Related Work

Static obstacles can be recognized in advance and reflected in the path plan whereas dynamic obstacles are not recognized in advance. Therefore the path needs to be re-planned depending on the situation for

collision avoidance. The entire path, from the origin to the destination, can be viewed as a global path plan. In the latter case, as a local path plan for which the new path is re-planned by avoiding obstacles, we need to delete the existing collision path and interleave the re-planned path with the existing global path. Traditional algorithms for global path planning include the A* and Dijkstra algorithms, while RRT [16,17] has recently gained popularity. Local path planning and methods for avoiding collisions include VO and collision cone (CC), which are based on geometry. In addition, there are artificial potential field [18] and probabilistic RoadMap algorithm [19]. Shi and Ng [15] proposed a collision-free path planning algorithm based on a high-level A* algorithm with a heuristic function added with waiting time.

There are quite a few examples of studies on RRT [17] and modified RRT. Karaman and Frazzoli [13] proposed "Rewiring" to overcome the limitation that the path generated by RRT does not guarantee convergence with the optimal path. RRT*-Smart [7] increases the convergence speed of RRT* by applying a path optimization using triangular inequality conditions and a biased sampling technique that performs sampling within a specific area. The application area of RRT is surprising; it also applies to the concept of software engineering. Park et al. [5] proposed a black-box-based test case generation method for a Simulink/Stateflow model utilizing the RRT algorithm which is a method used to efficiently solve path planning for complicated systems. It has been proven that the RRT algorithm is probabilistically complete, but the main drawback is that it does not focus on the quality of the solution [6]. In particular, it does not always reach the destination optimally within a predefined finite time, and the convergence speed slows down depending on the application environment [7]. To address this drawback, many modified RRT algorithms have been proposed and compared. Urmson and Simmons [8] proposed a method to bias the growth of RRT based on the costs determined by spatial exploration. Informed-RRT* [9] induces heuristic bias sampling to increase the sampling probability inside the heuristic sampling domain while reducing the sampling probability from the outside. Xu et al. [10] proposed a biased sampling potentially guided intelligent bidirectional RRT (BPIB-RRT*) algorithm to address the RRT issue of convergence speed. Naderi et al. [11] proposed a real-time RRT*(RT-RRT*), a modified version of the informed-RRT* for a real-time planning path in a dynamic environment such as a game. Yao et al. [12] proposed a hybrid strategy based on an interfered fluid dynamical system (IFDS) and an improved rapidly-exploring random tree (IRRT) for unmanned aerial vehicle (UAV) path planning problems in a complex 3D environment. Veras et al. [20] classified RRT-based algorithms into goal-biased, obstacle-biased, region-biased, path-biased, and narrow passage-biased. However, it is not easy to classify RRT-based algorithms used in various application areas. For example, in UAV applications, RRT runs in a given exploring area to avoid obstacles and find a path to reach the goal. In other words, it includes goal-biased, path-biased, and region-biased approaches. Table 1 compares several RRT-based algorithms mentioned in this paper.

Table 1. Compares RRT-based algorithm

Algorithm	Sampling method	Region	Planning mode	Probabilistic completeness
RRT [17]	Nearest	Global	Offline	Yes (non-optimal)
RRT* [13]	Rewiring	Global	Offline	Yes (optimal)
RRT*-Smart [7]	Triangular inequality	Global	Offline	Yes (optimal)
Informed RRT* [9]	Prolate hyperspheroid	Local	Offline	Yes (optimal)
RT-RRT* [11]	Base on informed RRT*	Local	Online	Yes (optimal)

There are two major areas of collision avoidance, which are studies based on velocity. The first area deals with prevention of collisions by dividing the spatial environment into a constant grid cell and adjusting the time the agent stays in the grid cells (adjustment is performed through acceleration and deceleration of velocity: Velocity Planning) [21,22]. The second area recognizes one of the two objects as an obstacle first, and allows natural definition of a moving object by defining the obstacle as a velocity vector after recognizing it only as an obstacle without changing the moving properties [1,23]. Alejo et al. [21] proposed a method to determine whether there is a possibility of collision when the trajectory of the UAV is calculated, and to resolve the potential collision by changing the residence time in the cell of the UAV if there is a possibility of collision. This is solved by assigning a velocity profile to the UAV. Rebollo et al. [22] is proposed a meta-heuristic method based on a combination of search tree algorithm (STA) and tabu search algorithm (TS) to determine the time the UAV stays in each cell of the trajectory. These two heuristic algorithms are used to create a new velocity profile to find a solution to the collision avoidance problem. The study of Fiorini and Shiller [1] is the most representative study that defined VO and the authors conducted research on avoiding collisions between agents based on this. Van den Berg et al. [23] proposed a new theory "Reciprocal Velocity Obstacle" (RVO) theory for real-time multi-agent navigation. Velocity planning (VP) and VO both find commonality as they coordinate velocity and apply appropriate tree algorithms and other heuristics based on a set of velocities. The difference between VP and VO is that VP divides space into grid cells and defines trajectories as cell sequences, but VO computes a set of velocities by mapping space into vectors of time, velocity, and direction.

3. Modeling Collision Avoidance in a 3D Environment

For UAVs, such as drones to fly safely and autonomously, a module that senses the environment, a control module for flight control, and a path planning module are required. As shown in Fig. 1, the path planning module requires a global planner for global path planning and a local planner for local path planning. This study focuses on local path planners, collision avoidance and re-planning of local paths. Therefore, the global path method was simplified to allow the path to be planned based on the distance to each drone and the destination point.

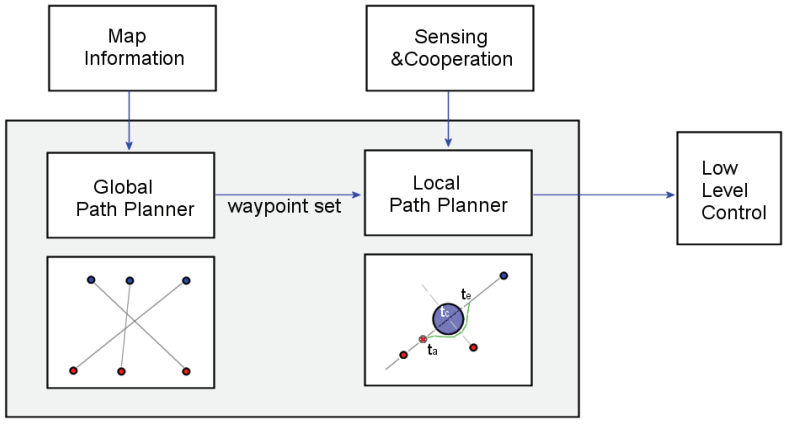


Fig. 1. Path planning framework.

3.1 Global Path Planning

In general, the flight path of a UAV is designed considering the mission to be performed. When the path design is completed, a technology that can track the path is applied to automatically guide the UAV. The most basic and simple path planning method is waypoint planning, which creates a path by setting a waypoint to pass through and connects the path points in a straight line. When the path point planning is completed, a tracking method for deriving a UAV in a direction is typically applied to reduce errors by calculating the error with the path point at every moment during flight. Global path planning is performed in advance, and planned paths take the form of passing to each UAV. If a single algorithm planner is used, the algorithm should be designed to find collision-free paths in the global planner. However, in this study, collision detection and collision avoidance were performed by the local planner. Thus only the optimization of the global path cost is considered in global path planning. Prior to achieving collision avoidance, the global path plan in the configuration space with M drones and N destinations is first described. At this time, it is assumed that there are no obstacles in the configuration space, and only the distance cost function is used to assign the destination to each drone based on the minimum distance cost function. From this distance, the destination allocation matrix can be obtained using the HAA [14]. The HAA is an algorithm that can optimally allocate each drone and destination within the polynomial time $O(N^3)$ and is based on bipartite graph modeling. The global planner aims to find the smallest value of the total sum of matched pairs between each drone and destination.

$$\min Z = \sum_{i,j \in M} C_{i,j} \tag{1}$$

The HAA makes a match between the drone and the destination and creates a waypoints list according to the time interval Δt between the matched drone and the destination, as shown Table 2.

Table 2. Waypoints list

Pair	t_0	t_1	…	t_n	…
V_1,U_2	$(x_{1,0}, y_{1,0}, z_{1,0})$	$(x_{1,1}, y_{1,1}, z_{1,1})$	…	$(x_{1,n}, y_{1,n}, z_{1,n})$	…
V_2,U_4	$(x_{2,0}, y_{2,0}, z_{2,0})$	$(x_{2,1}, y_{2,1}, z_{2,1})$	…	$(x_{2,n}, y_{2,n}, z_{2,n})$	…
V_3,U_1	$(x_{2,0}, y_{3,0}, z_{3,0})$	$(x_{3,1}, y_{3,1}, z_{3,1})$	…	$(x_{3,n}, y_{3,n}, z_{3,n})$	…
V_4,U_4	$(x_{2,0}, y_{4,0}, z_{4,0})$	$(x_{4,1}, y_{4,1}, z_{4,1})$	…	$(x_{4,n}, y_{4,n}, z_{4,n})$	…

3.2 Collision Avoidance and Local Path Re-planning

3.2.1 Time sequence of the algorithm

The algorithm avoids static and moving obstacles as the goal of this study is avoiding obstacles associated with a given time-varying environment. In time-varying environments, collision avoidance depends on the dynamics of the agent. Thus constraints should include appropriate dynamic constraints (minimum radius, maximum velocity limit, etc.). Fig. 2 shows the time sequence of the boundary-RRT* algorithm. The variable t_0 indicates the time when a static or dynamic obstacle is detected, after which the collision is determined for time t_d.

If collisions are detected in a significantly short period of time and a collision is to be expected, the

algorithm must have a path re-planned for collision avoidance, even if it is not a fully optimized plan until time t_a. It is important to optimize the point of evasion rather than the distant target. Therefore, the location of time t_a is the root node of the boundary-RRT* algorithm, which will cause the tree to grow from this location. The RRT* algorithm is optimally convergent as the number of sampling nodes increases, but it is difficult to optimize in real time as the computation time increases. Therefore, the goal is to optimize real-time and fast paths by bounding the region of the exploration space. The boundary-RRT* algorithm further narrows the space of exploration by naturally converting the vector set of velocity defined by the VO into a position vector, and applying the right-of-way rules to the transformed set. Based on the planned path, the algorithm starts performing collision avoidance until it reaches the intermediate target node for time t_m, and the algorithm continues to find the path for quality improvement and optimizes while performing the collision avoidance maneuver.

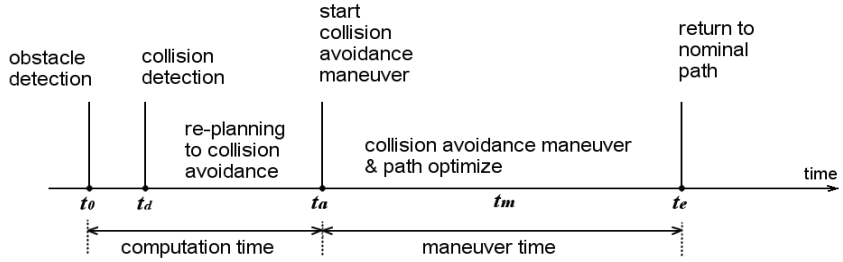


Fig. 2. Time sequence of the boundary-RRT* algorithm.

3.2.2 Collision detection

As shown in the boundary-RRT* algorithm time sequence in Fig. 2, the algorithm checks for collisions with obstacles within a distance D at time t_d. Distance D is defined as a constant and should be long enough to consider the agent's speed of movement and allow the algorithm to plan the initial path. When defining the separation distance or protection area of the UAV for collision avoidance, the protection area is generally defined as a sphere or cylinder. In this study, a spherical protection zone is defined rather than a cylindrical one for drones with relatively flexible trajectories in a 3D space. There are two methods for detecting collisions using velocity. One is using a relative CC, and the other is uses an absolute CC (i.e., VO). To use the relative CC, when r_A is the radius of drone *A's* protection sphere and r_B is the radius of drone *B's* protection sphere, a relative CC can be created by adding r_A to drone *B's* protection sphere r_B. The relative CC is bounded to form the tangent lines of the enlarged protection sphere to the mass point V_A.

$$CC = r_A + r_B \tag{2}$$

The relative velocity vector V_{AB} indicates how drone A moves in relation to drone B and transforms a dynamic collision avoidance problem into a static problem. Drone B is now considered a static obstacle. If the direction of the V_{AB} vector passes through the enlarged protection zone of obstacle B, the two drones will collide in the near future. In Fig. 3(a), we can see the λ_{AB}, an extension of V_{AB}.

$$V_{AB} = V_A - V_B \tag{3}$$

To use the absolute CC, add the velocity of obstacle B to the relative collision. This absolute CC is called the velocity obstacle, VO [3].

$$\text{VO} = \text{CC} \oplus V_B \tag{4}$$

where $\oplus$ is the Minkowski vector sum.

To detect a collision using a CC, when the distance to the obstacle is within the sensing distance D, d_{AB}, the semi-angle θ_{AB} based on the center axis of the CC can be calculated as follows:

$$\theta_{AB} = sin^{-1}\left(\frac{r_A + r_B}{d_{AB}}\right) \tag{5}$$

where d_{AB} is the Euclidean distance. Therefore, the angle between the relative vectors V_{AB} and d_{AB} can be calculated using Eq. (6).

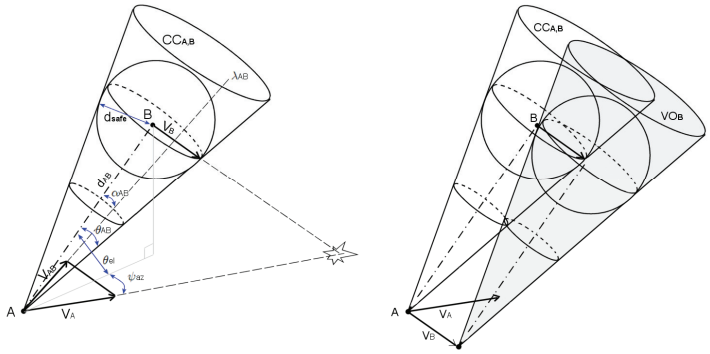
$$cos\,\alpha_{AB} = \frac{V_{AB,x}(x_B - x_A) + V_{AB,y}(y_B - y_A) + V_{AB,z}(z_B - z_A)}{\|V_{AB}\|\|d_{safe}\|} \tag{6}$$

The derived d_{AB} and α_{AB} can be used to determine whether they collide.

$$V_{AB} \in \text{CC} \quad \text{if } \alpha_{AB} \leq \theta_{AB} \tag{7}$$

$$V_{AB} \notin \text{CC} \quad \text{if } \alpha_{AB} > \theta_{AB} \tag{8}$$

The condition in (7) causes a collision, where the condition of (8) does not. Therefore, if condition (7) is established, the algorithm starts re-planning for collision avoidance.



(a) (b)

Fig. 3. Configuration of (a) collision cone (CC) and (b) velocity obstacle (VO).

3.2.3 Collision avoidance maneuver

The algorithm applies to both static and dynamic obstacles. Important parameters in the calculation of the avoidance maneuver include the drone minimum turning radius ρ_{min} and the distance of r_A+r_B. This quantification is significant for determining the avoidance times. When d_{safe} is the radius of the safety protection sphere of the obstacle at the relative position P_B, it is the same as Eq. (2) in Section 3.1. To calculate a safer trajectory, Eq. (9) is defined by adding clearance δ.

$$d_{safe} = r_A + r_B + \delta \tag{9}$$

It is assumed that if the drone does not carry out an avoidance maneuver there will be a collision, and we can predict the collision time as follows (t_c denotes the collision time):

$$t_c = \frac{d_{AB}\cos\alpha_{AB} - \sqrt{(d_{safe})^2 - (d_{AB}\sin\alpha_{AB})^2}}{||V_{AB}||} \tag{10}$$

Similarly, the collision avoidance start time t_a can be calculated using Eq. (17). The variable t_a uses the parameter, ρ, which is the radius of turning. To calculate the radius of the drone's turning, two parameters were used. The first is the angle of bank (ϕ), which is the angle at which the aircraft should lie down in the direction of rotation when the drone is turning. The second is the speed ν of the drone's turning. Therefore, when the increased speed during the drone's turning speed is Δv, the turning radius ρ is equal to Eq. (11) as follows :

$$\rho = \frac{1{,}091 \cdot tan(\phi)}{\Delta v} \tag{11}$$

where constant 1,091 is the aircraft's speed unit conversion value. The ρ can be determined by taking the constant 1,091, multiplying it by the tangent of the bank angle, and dividing it by the airspeed in knots [24]. Using Eq. (11) as a parameter, the instantaneous time t_a of collision avoidance can be calculated using Eq. (12).

$$t_a = \frac{d_{AB}\cos\alpha_{AB} - \sqrt{(\rho+d_{safe})^2 - (\rho+d_{AB}\sin\alpha_{AB})^2}}{||V_{AB}||} \tag{12}$$

The time sequence is $t_0 < t_d < t_a < t_c$. Because t_d completes the detection of collisions, a collision avoidance maneuver can be started between times t_d and t_a, where the velocities of each point of mass are constant between time sequences. Thus, the position of the points of mass on time $t = t_0 + t_a$ can easily be determined, where t_0 is the initial time. Hence, the relative position P_{Aoid} of drone A to collision avoidance in 3D space can be defined by the following set Eq. (13),

$$\begin{aligned} P_{Avoid,x} &= x_A = x_A^0 + V_{A,x} \cdot t_a \\ P_{Avoid,y} &= y_A = y_A^0 + V_{A,y} \cdot t_a \\ P_{Avoid,z} &= z_A = z_A^0 + V_{A,z} \cdot t_a \end{aligned} \tag{13}$$

Similarly, the collision position P_{Col} of drone A with drone B in 3D space is predicted by Eq. (14).

$$\begin{aligned} P_{Col,x} &= x_A = x_A^0 + V_{A,x} \cdot t_c \\ P_{Col,y} &= y_A = y_A^0 + V_{A,y} \cdot t_c \\ P_{Col,z} &= z_A = z_A^0 + V_{A,z} \cdot t_c \end{aligned} \tag{14}$$

3.2.4 Determination of maneuvers for collision avoidance

A drone's travel distance is relatively limited, except for special models. In particular, its radius of movement is considerably shorter than the radius of movement of a fixed-wing drone, and the radius of turning is much smaller. Thus, maintaining speed consistently is difficult because of its light weight. Further, exploration to avoid collisions from moving obstacles that are difficult to detect at a constant

speed can be conducted vertically, not horizontally. This can remove the precondition that the obstacles must remain constant without changing the speed. To define these rules, this study refers to and expands the Federal Aviation Administration 91.133 [25,26] right-of-way rule. Between each drone that is expected to collide, priorities are defined by the situation in each drone's airspace. That is, defining which drones will perform a collision avoidance maneuver is prioritized. Static obstacles have the lowest priority order, so moving drones dodge unconditionally. Dynamic obstacles are prioritized according to predefined rules that simplify the avoidance problem and are commonly used in UAVs. The right-of-way rule also provides additional predictions regarding how other UAVs move. The boundary-RRT* algorithm defines four rules, which define the direction of the search area, that is, the half-torus area. The right-of-way rules are defined as follows.

(1) The right drone has the right-of-way in close encounters.
(2) In a head-on encounter, both drones must avoid moving to the right.
(3) Overtaken drones have the right-of-way.
(4) If detecting the constant speed of the other drone is difficult, move upwards to avoid a collision.

3.2.5 Boundary region definition

Boundary-RRT* is an algorithm aimed at local path planning and re-planning. Therefore, the tree does not need to grow throughout the configuration space. Bounding the size of the exploring space is an efficient method for reducing computation time in real time. Bounding the exploration space is defined by the size of the obstacle's protection sphere, the collision avoidance time t_a, and the predicted collision time t_c of drone A. The sliced torus (i.e., the half-torus) can be defined using the starting and target (nominal waypoint: t_e) positions in the XY and XZ planes, respectively, as shown in Fig. 4.

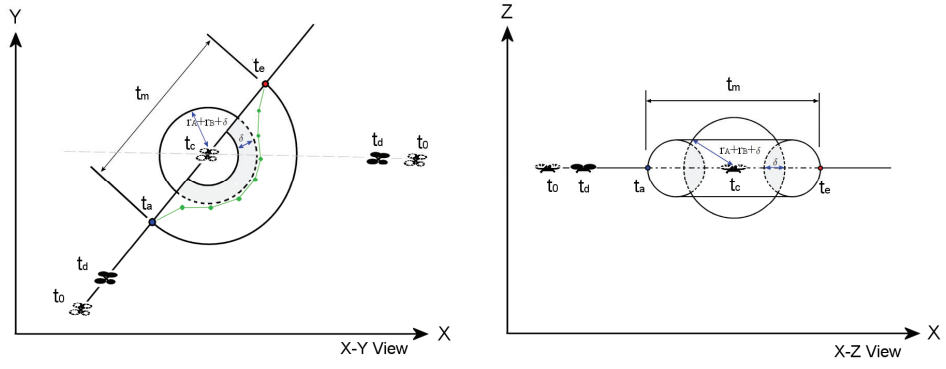
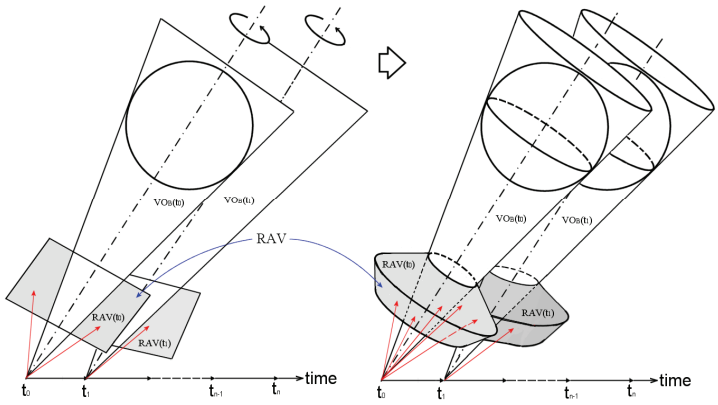


Fig. 4. Half-torus (sliced-torus) bounded region and expected path plan: (a) XY plan and (b) XZ plan.

Before mathematically modeling the half-torus region, it can be defined as a subset of the union of each RAV generated according to the look-ahead time Δt. Therefore, the theorem is defined as follows.

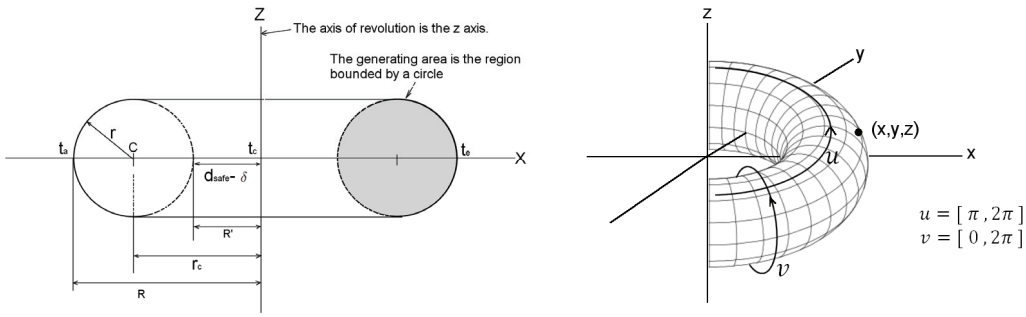
THEOREM. RAV is a set of reachable avoidance velocity spaces, and instantly maps collision avoidance maneuvers in the velocity space to a series of geometric shapes. Therefore, the half-torus region in Fig. 3.4 is a subset of RAV($t + \Delta t$), which is the union of the calculated RAV according to instant time t, as shown in Fig. 5(b).

Proof. When defining RAV as a set of collision avoidance velocities and directions (velocity space), maneuver avoidance for drone *B*, which is considered an obstacle, is made possible by selecting any velocity in the RAV. In Fig. 5(a), RAV is represented by two separate subsets (left and, right). However, the 3D RAV is a set of spaces connected at the instant time t, as shown in Fig. 5(b). That is, the collision avoidance direction and velocity can be selected in all directions. In addition, if RAV ($t + \Delta t$) is defined, each discrete space defined according to the discrete time t can be defined as a continuous space according to time t. At this time, if the direction and the region for the collision avoidance maneuver are defined, the avoidance region can be bounded as a subset in total spaces. Therefore, the half-torus is a subset of the entire continuous space RAV ($t + \Delta t$).



(a) (b)

Fig. 5. (a) Two-dimensional and (b) three-dimensional comparison of reachable avoidance velocities (RAV) sets.



(a) (b)

Fig. 6. (a) Computation of the region of the half-torus and (b) parametrization of a half-torus by coordinates.

form bounds, the region size of the torus sliced in half (half-torus) must be determined. The half-torus can be calculated by applying the theory of Pappus–Guldinus [27]. Fig. 6(a) shows the parameters for obtaining the region of a half-torus. Fig. 6(b) shows the parameterization for the circumference of a circle with radius *r*, and the circumference of a half-circle with radius R. The complete torus prior to calculating the half-torus region can be parameterized by Eq. (20).

$$
\begin{aligned}
x(u,v) &= (R + r \cdot \cos v) \cos u \\
y(u,v) &= (R + r \cdot \cos v) \sin u \\
z(u,v) &= r \cdot \sin v
\end{aligned} \tag{20}
$$

In order to slice the torus, it is necessary to calculate the current position of the drone and the direction angle to the target point. This is the same as Eq. (21) if the direction angle is defined as Φ.

$$
\begin{aligned}
deltaP &= goal.position - current.position \\
\angle\Phi &= tan^{-1}(deltaP(x), deltaP(y), deltaP(z))
\end{aligned} \tag{21}
$$

The turning section of the half-torus is defined based on the calculated direction angle Φ. The variable range of the complete torus section for u is [0, 2π], but the half-torus is [π, 2π], as expressed in Eq. (22).

$$
\begin{aligned}
u &= [\, \Phi + \pi\,,\ \Phi + 2\pi\,] \\
v &= [\, 0\,, 2\pi\,]
\end{aligned} \tag{22}
$$

Finally, to center the calculated collision position, the half-torus region is calculated by adding the collision position derived from Eq. (19), and the search region of the algorithm is defined in Eq. (23). The tree grows within this defined region.

$$
\begin{aligned}
x' &= P_{col,x} + x \\
y' &= P_{col,y} + y \\
z' &= P_{col,z} + z
\end{aligned} \tag{23}
$$

If the z axis is the axis of rotation, the collision position is the origin, the radius is R, the radius of the half-torus is r, and the parametric equation of the half-torus reflecting the direction is defined by Eq. (24).

$$
T\,R,r \begin{cases} x' = P_{Col,x} + (R + r \cdot \cos v) \cos u \\ y' = P_{Col,y} + (R + r \cdot \cos v) \sin u \\ z' = P_{Col,z} + r \cdot \sin v \end{cases} \tag{24}
$$

At this time, u and v are limited as in Eq. (22).

4. Boundary-RRT* Algorithm

Table 3 indicates the overall pseudo code of the boundary-RRT * algorithm, Table 4 defines the collision avoidance maneuver, and Table 5 defines the half-torus region.

The algorithm detects information regarding obstacles within a certain distance D. The detection method is assumed to be perfectly detected by a suitable method, such as receiving information from a sensor, airborne or mutual cooperation between drones. *CollisionManeuverCheck*(α_{AB}, θ_{AB}) is a function that detects a collision and determines which drone will be involved in the collision avoidance maneuver. If a collision is predicted, but the return value is FALSE, the star of collision avoidance is not performed. This follows the right-of-way rules (line 3, in the algorithm). Epsilon ε is maximum length of an edge in the tree and *nSample* indicates the number of sampling nodes per Δt (line 5 and line 6, respectively). Then, when the collision is predicted, the collision position t_c is calculated using Eq. (10), and the critical time t_a to avoid the collision is calculated using Eq. (12) (lines 7–8). It then searches for waypoint away from t_c by the distance between t_c and t_a in the waypoints list. This list of waypoints is the path list that

the global planner has already planned, and it searches for waypoints that will return to their nominal positions from this waypoint list after the collision avoidance maneuver is completed (line 9). Next, define the exploring space in the configuration space is defined by applying the defined and calculated parameters, as well as Eq. (24) with *ComputeHalfTorusRegion*(X_{free}). The exploring space is in a half-torus shape and is bounded to the X_{torus} region in the configuration space X_{free} (line 10).

The root of the initial tree becomes position t_a (line 11). Within the bounded exploring space, the *RegionRandomSample*(X_{torus}) function samples randomly (line 14). The following procedure performs the same operation as the RRT* algorithm. The difference is that if the current time t has passed time t_a, the drone enters the bounded half-torus region.

Table 3. Pseudo-code of the boundary-RRT* algorithm

	Boundary-RRT* Algorithm
1:	**if** detect obstacle with in detect distance D
2:	**if** a collision is predicted
3:	Do_avoid ← *CollisionManeuverCheck*(α_{AB} , θ_{AB})
4:	**if** Do_avoid is TRUE
5:	ε ←Epsilon_Length
6:	*nSample* ← Number of Sampling node per Δt
7:	calculate t_c , calculate t_a
8:	WP ← *FindNominalWaypoint()*
9:	X_{torus} ← *ComputeHalfTorusRegion*(X_{free})
10:	X_{base} ← t_a , X_{goal} ← WP
11:	T ← *init*(X_{base})
12:	**while** $P_{current}$ ≅ WP **do**
13:	x_{rand} ← *RegionRandomSample* (X_{torus})
14:	$x_{nearest}$ ← (T , x_{rand})
15:	x_{new} ← Steer($x_{nearest}$, x_{rand})
16:	T(V,E) ← AddVertex (T , x_{new})
17:	$x_{neighbor}$ ← FindNeighborVertex (T , x_{new})
18:	x_{min} ← BestParent ($x_{neighbor}$, $x_{nearest}$, x_{new})
19:	T(V,E) ← ReWire (T , $x_{neighbor}$, x_{new})
20:	**end while**
21:	**end if**
22:	**end if**
23:	**end if**

Table 4. Pseudo-code of *CollisionManeuverCheck*

	***CollisionManeuverCheck*(α_{AB} , θ_{AB})**
1:	**if** $\alpha_{AB} < \theta_{AB}$
2:	Collision = TRUE
3:	**else**
4:	Collision = FALSE
5:	**end if**
6:	Maneuver ← Determination of collision avoidance maneuver
7:	**return** Collision ∨ Maneuver

Table 5. Pseudo-code ComputeHalfTorusRegion

	ComputeHalfTorusRegion(X_{free})
1:	, $\Phi \leftarrow$ Calculate HeadingAngle
2:	Half-Torus $\leftarrow$ *Bound(X_{free} , Predicted Collision Position , Φ)*
3:	$X_{torus} \leftarrow$ Half-Torus

5. Simulation

A simulation was performed to evaluate the performance of the proposed algorithm in this study. The specifications of the computer that performed the simulation are Intel Core i5-5200U CPU 2.20 GHz, 4GB RAM, and Windows 10 Education. MATLAB R2020a 64 bit was used to run the simulation. For the evaluation of the algorithm, the starting and arrival points are defined in advance. After the collision is detected, the interleave path is generated based on the boundary-RRT* algorithm. For the evaluation of the algorithm, the z-axis is assumed to be 0 for convenience. Drone A is the target position from [20 1 0] to [1 20 0], and drone B is directed to the destination [1 1 0] from the starting position [20 20 0]. The Euclidean distance between the two points based on the coordinates is 268.701 m. It is assumed that Δt = 1 second and the flight speed is 10m/Δt. Therefore, if there is no obstacle to the destination, the drone arrives at the target point in approximately 26.8 seconds. Fig. 7 shows the virtually configured configuration space for the simulation.

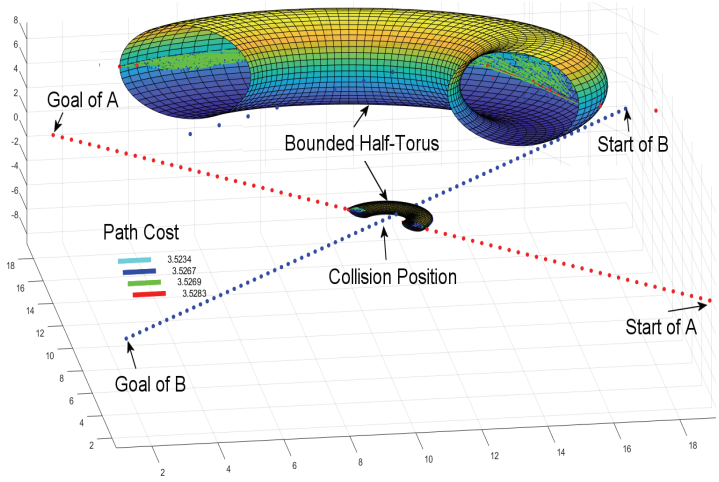


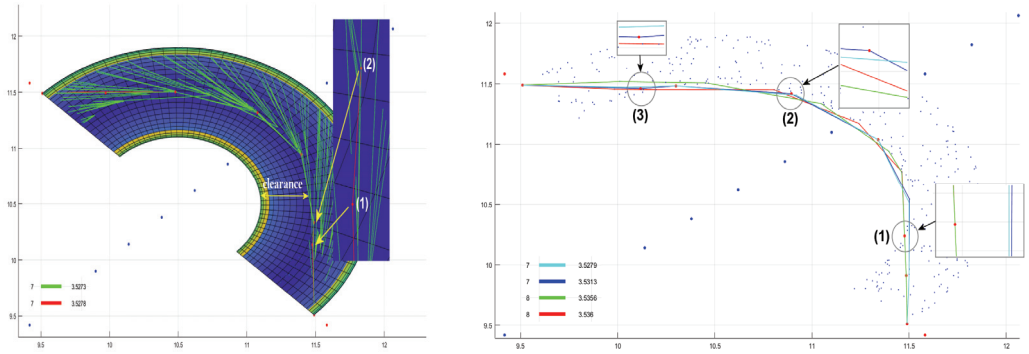
Fig. 7. Simulation environments and half-torus.

Waypoints after the two drones started from the starting point and avoided the collision point were expressed in dot form. As shown in Fig. 7, the expansion of the tree only grows within the bounded half-torus space. In particular, if the configuration space is defined in the form of curvature, the defined space naturally has an implicit bias. Therefore, it does not need to design a separate bias function. These properties can be found in the simulation results. Because the algorithm has already defined a dynamic obstacle as a static obstacle, the algorithm finds an avoidance path even if it is not perfect from the defined t_d to the time t_a calculated by Eq. (18). In this simulation, the collision detection area was set to 70 m and

t_a was calculated to be 20 m. The inside of the area defined by the half-torus region is shown in Fig. 8(a). To ensure collision avoidance, the tree is designed to grow beyond an area equal to the safety clearance δ in the inner area. In Fig. 8, two optimal paths were updated during the collision avoidance maneuver. The red and green paths are the optimal paths that the drone finds before entering the half-torus region, and the blue paths are the optimal paths found after entering the half-torus region. Therefore, the drone's next waypoint can be observed to change from (1) to (2).

A detailed explanation of the simulation result after removing the half-torus region and the edges of the tree can be observed in Fig. 8(b).

The algorithm has one optimal path update between t_d and t_a for collision avoidance maneuvering, and three optimal path updates until the collision avoidance maneuver is completed. Therefore, the total distance cost has been changed four times. If the red colored areas of the optimal path calculated before entering the half-torus region are removed, the green area is the optimal path when entering the half-torus region. Finally, exit can be observed exiting via by the cyan colored path. Therefore, the reflection of the ideal waypoint should follow the path from (1) to (2) and (3). However, in the simulation, the cyan path was already set as the next waypoint and moving to the set waypoint could not select the newly updated waypoint path. As a result, the travel path leaves the half-torus region through (1) and (2).



(a) (b)

Fig. 8. Top views of the waypoint: (a) with a half-torus view and (b) without a half-torus view.

Next, we conducted a comparative evaluation of the changes in epsilon ε which is the edge maximum length (distance) of RRT, and the number of sampling nodes (*nSample*) per Δt (Table 1, lines 5–6). In Fig. 9(a) and 9(b), the distance of ε was limited to 5 m in both cases. In the case of Figs. 9(a), the number of sampling nodes per unit time is set to 20, and in the case of Fig. 9(b), the number of sampling nodes per unit time is set to 30. Fig. 9(c) and 9(d) show the results of running 20 and 30 sampling nodes, respectively, per Δt with the epsilon ε distance limited to 2 m. Apparently, there is a difference in the number of evaluated edges depending on the number of sampling nodes. In Fig. 9, when comparing only Fig. 9(a) and 9(b), the result shows that as the number of sampling nodes increases, a better path is found. That is, the path show in Fig. 9(b) was found to have a lower distance cost than that show in Fig. 9(a). Similarly, in Fig. 9(c) and 9(d), a better path was found in Fig. 9(d) with a larger number of sampling nodes. In the simulation results, unexpected results were obtained for the path length (distance cost).When the algorithm was proposed and designed, it was expected that the smaller the epsilon ε, the smaller the distance length (distance cost). In contrast, as ε became smaller, the number of waypoints increased and the distance cost slightly increased.

The results for this can be found in Fig. 10(a) and 10(b). In the case of Fig. 10(a), the number of sampling nodes is set to 20 and epsilon ε is set to 2, 3, 4, and 5 m. This is a graph of the simulation results, 100 times in each case. The red graph has epsilon ε set to 2 m, and the green graph has epsilon ε set to 5 m.

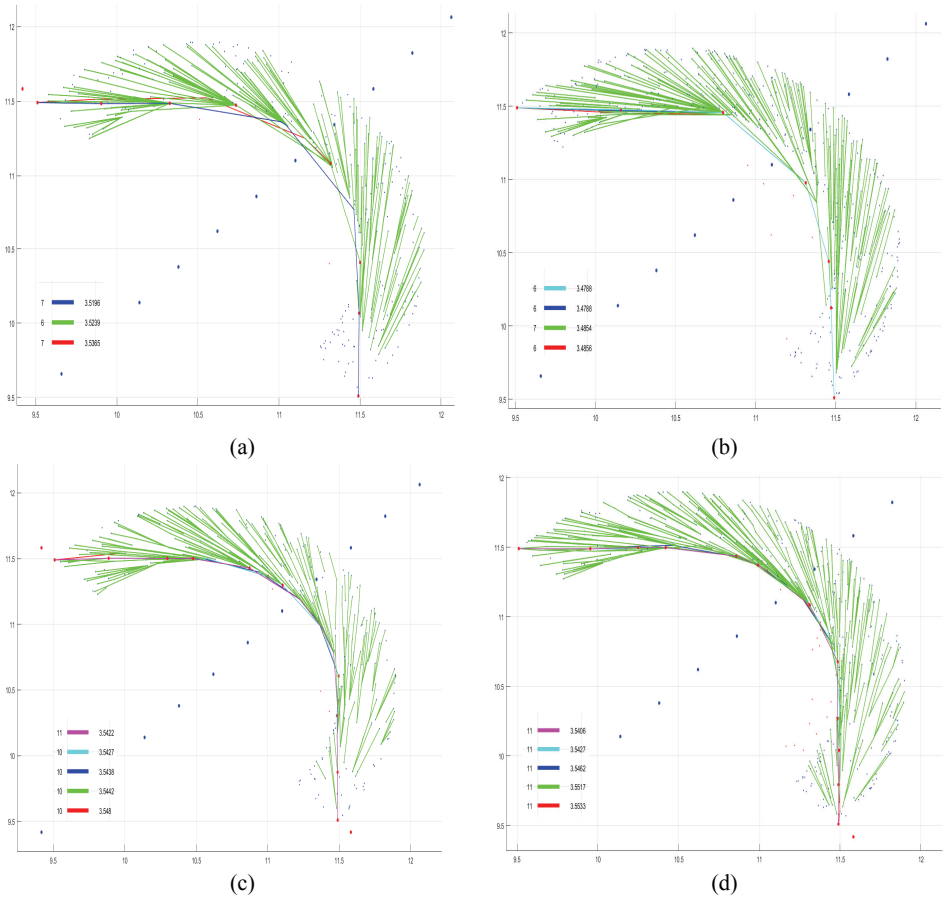


Fig. 9. Simulation results with change in epsilon ε and number of sampling nodes: (a) ε=0.5, sampling=20/Δ***t***, (b) ε=0.5, sampling=30/Δ***t***, (c) ε=0.2, sampling=20/Δ***t***, and (d) ε=0.2, sampling=30/Δ***t***.

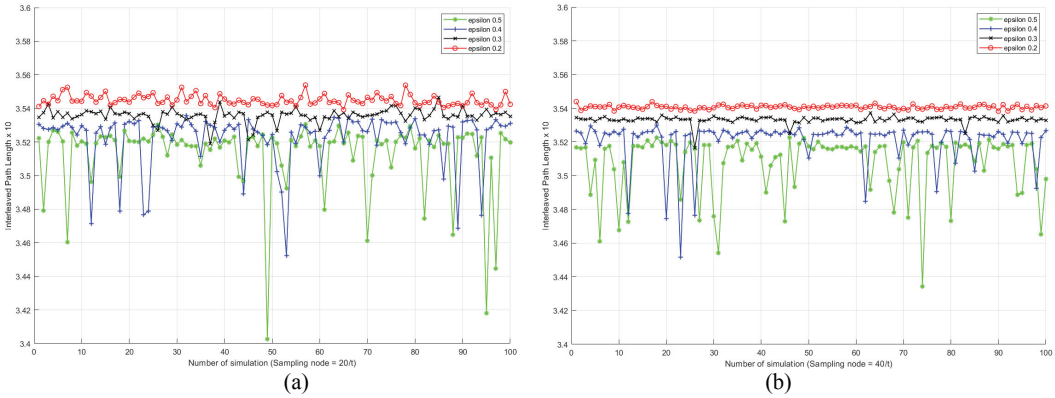


Fig. 10. Results of 100 experiments: (a) sampling=20/Δ***t*** and (b) sampling=40/Δ***t***.

This is a slight difference, but the red portion has more path length. As can be observed in the graph, the smaller the epsilon ε, the larger is the path length. However, the extent of displacement of the increase or decrease in the simulation collision avoidance path is much more stable as ε becomes smaller. These results are clearly observed in Fig. 10(b). When evaluating the algorithm with the results of Fig. 10(a) and 10(b), it was concluded that the smaller ε and the greater the number of samples, the smoother the avoidance path.

Table 6 shows the numerical analysis results derived from experimental data. When defining the distance as a cost, the table shows that the standard deviation approaches 0 as the number of samplings per unit time increases and the distance of the epsilon ε decreases. This means that a stable waypoint list can be generated using the proposed algorithm.

Table 6. Numerical analysis of the experimental results (unit: ×10 m)

	Sampling=20/Δ*t*			**Sampling=40/Δ*t***		
	Total cost	**Average**	**SD (σ)**	**Total cost**	**Average**	**SD (σ)**
ε=0.5	351.316	3.513	0.0215	350.805	3.508	0.0178
ε=0.4	352.245	3.522	0.0164	352.091	3.521	0.0129
ε=0.3	353.586	3.536	0.0040	353.318	3.533	0.0023
ε=0.2	354.494	3.545	0.0031	354.071	3.541	0.0012

6. Discussion

Until recently, most of the existing path planning algorithms for UAVs have not provided a thorough insight into explore space limitations in terms of trade-offs between optimality and speed in large and complex 3D environments. Additionally, most path-planning algorithms have focused on building better heuristics at the expense of memory overhead. This can be understood to mean that the algorithm sacrifices efficiency for convergence speed. Almost all algorithms focus only on the shortest path search, so there is a risk of time performance degradation if the search space is not adequately considered. In the worst case, the calculation time increases exponentially and extensive pre- and post-processing is required to obtain the final path. Therefore to effectively resolve the trade-off between optimality and speed, it is considered that the path search should be limited to the exploring region with high priority during the path plan. To better reflect these requirements, a boundary-RRT* algorithm, suitable for local path planning, was proposed. In Table 7, the proposed boundary-RRT* algorithm and several path planning algorithms were compared and evaluated.

The proposed boundary-RRT* algorithm can reduce the number of sampling nodes by bounding the exploration space. Furthermore, the algorithm can prevent unnecessary tree expansion by removing nodes that have already passed from the current location. For flexible expansion, the half-torus region can be varied to allow expansion and contraction depending on the size of the static obstacle and speed of the moving object. In addition, the proposed algorithm can create a path with natural curvature without defining a bias function. Therefore, the boundary-RRT* algorithm can compensate for the problem of convergence speed, which is a disadvantage of RRT*, and can ensure optimal convergence of RRT*, which is an advantage.

However, a part that needs to be supplemented has emerged in this study. The waypoint lists generated by the algorithm did not reflect the kinematic properties of the drone. This means that the optimal path

generated by the boundary-RRT* algorithm can explore unacceptable, or unreachable, paths given the drone's kinematic properties. Fig. 11, illustrates this problem.

Table 7. Comparison and evaluation of several path planning algorithms and boundary-RRT*.

Algorithm	Taxonomy	Disadvantages	Advantages
Boundary-RRT*	Sampling based	Difficulty in expressing various types of obstacles. Needs post-processing to smoothness.	Includes advantages of RRT. Static and dynamic obstacle. Curvature path (implicit bias). Variable size exploration area.
RRT [17]	Sampling based	Single path. Non-optimal. Static obstacle only.	Probabilistically complete. Relatively simple. Heavily biased.
Artificial potential [18,28]	Sampling based	Local minima or oscillations.	Fast convergence. Simple and elegance.
A* [15]	Node based	High time complexity. Non smoothness. Static obstacle only.	Fast searching Complete and Optimal.
Ant colony optimization [29]	Bio-inspired based	High time complexity. Local optimum.	Dynamic application Rapid discovery through positive feedback.
Mixed integer-linear programming (MILP) [30]	Mathematical model based	High time complexity. Complex formulation. Computational burden.	Solution is guaranteed. Optimal solution.

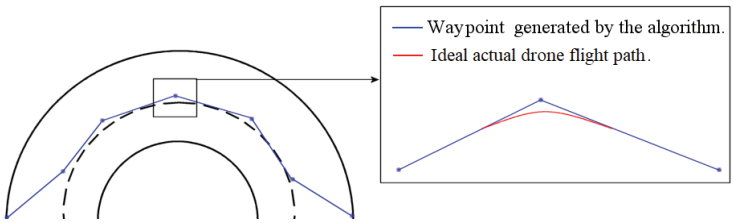


Fig. 11. Difference between the generated waypoint and the ideal actual flight path.

7. Conclusion and Future Work

When the initial algorithm was proposed and the study was initiated, the path length was expected to be shortened as the number of sampling nodes increased, and the epsilon ε would grow smaller. Contrary to our expectations, the path length became slightly longer and the number of waypoints increased. It was concluded that the search region naturally finds the optimal path along the curved surface within the bounded space to avoid collision because the half-torus-bounded region takes on an elliptical shape rather than a polygonal shape. In addition, the simulation confirmed that the RRT (or RRT*) algorithm, which is difficult to apply to a local path plan and collision avoidance in real-time, can be applied to a fast path plan in real-time by bounding the local region.

Although the kinematic properties are not reflected in this study, the proposed algorithm is expected to be applied to reduce the computational load as much as possible and to determine the optimal path for collision avoidance in a short time. In a future study, a method to create a smoother path for re-planning and collision avoidance by reflecting the kinematic characteristics of the drone will be proposed. In addition, the simulation environment will be expanded to illustrate that a large number of drones can move in airspace avoiding collisions based on the proposed algorithm.

References

[1] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *The International Journal of Robotics Research*, vol. 17, no. 7, pp. 760-772, 1998.

[2] J. A. Douthwaite, S. Zhao, and S. Mihaylova, "Velocity obstacle approaches for multi-agent collision avoidance," *Unmanned Systems*, vol. 7, no. 1, pp. 55-64, 2019.

[3] Y. I. Jenie, E. J. van Kampen, C. C. de Visser, J. Ellerbroek, and J. M. Hoekstra, "Three-dimensional velocity obstacle method for uncoordinated avoidance maneuvers of unmanned aerial vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 39, no. 10, pp. 2312-2323, 2016.

[4] X. Yang, Y. Zhang, and W. Zhou, "Obstacle avoidance method of three-dimensional obstacle spherical cap," *Journal of Systems Engineering and Electronics*, vol. 29, no. 5, pp. 1058-1068, 2018.

[5] H. S. Park, K. H. Choi, and K. H. Chung, "Test-case generation for Simulink/Stateflow Model using a separated RRT space," *KIPS Transactions on Software and Data Engineering*, vol. 2, no. 7, pp. 471-478, 2013.

[6] M. Cap, P. Novak, J. Vokrinek, and M. Pechoucek, "Multi-agent RRT: sampling-based cooperative pathfinding," in *Proceedings of International conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Saint Paul, MN, 2013, pp. 1263-1264.

[7] J. Nasir, F. Islam, U. Malik, Y. Ayaz, O. Hasan, M. Khan, and M. S. Muhammad, "RRT*-SMART: a rapid convergence implementation of RRT," *International Journal of Advanced Robotic Systems*, vol. 10, no. 7, article no. 299, 2013.

[8] C. Urmson and R. Simmons, "Approaches for heuristically biasing RRT growth," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No. 03CH37453)*, Las Vegas, NV, 2003, pp. 1178-1183.

[9] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed RRT*: optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *Proceedings of 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Chicago, IL, 2014, pp. 2997-3004.

[10] X. Wu, L. Xu, R. Zhen, and X. Wu, "Biased sampling potentially guided intelligent bidirectional RRT∗ algorithm for UAV path planning in 3D environment," *Mathematical Problems in Engineering*, vol. 2019, article no. 5157403, 2019.

[11] K. Naderi, J. Rajamaki, and P. Hamalainen, "RT-RRT* a real-time path planning algorithm based on RRT," in *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games*, Paris, France, 2015, pp. 113-118.

[12] P. Yao, H. Wang, and Z. Su, "Hybrid UAV path planning based on interfered fluid dynamical system and improved RRT," in *Proceedings of the 41st Annual Conference of the IEEE Industrial Electronics Society* (IECON), Yokohama, Japan, 2015, pp. 000829-000834.

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

[14] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the Society for Industrial and Applied Mathematics*, vol. 5, no. 1, pp. 32-38, 1957.

[15] Z. Shi and W. K. Ng, "A collision-free path planning algorithm for unmanned aerial vehicle delivery," in *Proceedings of 2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, Dallas, TX, 2018, pp. 358-362.

[16] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," in *Proceedings of 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, Detroit, MI, 1999, pp. 473-479.

[17] S. M. LaValle, "Rapidly-exploring random trees: a new tool for path planning," Department of Computer Science, Iowa State University, Ames, IA, 1998.

[18] Y. B. Chen, G. C. Luo, Y. S. Mei, J. Q. Yu, and X. L. Su, "UAV path planning using artificial potential field method updated by optimal control theory," *International Journal of Systems Science*, vol. 47, no. 6, pp. 1407-1420, 2016.

[19] M. U. Farooq, Z. Zhang, and M. Ejaz, "Quadrotor UAVs flying formation reconfiguration with collision avoidance using probabilistic roadmap algorithm," in *Proceedings of 2017 International Conference on Computer Systems, Electronics and Control (ICCSEC)*, Dalian, China, 2017, pp. 866-870.

[20] L. G. D. Veras, F. L. Medeiros, and L. N. Guimaraes, "systematic literature review of sampling process in rapidly-exploring random trees," *IEEE Access*, vol. 7, pp. 50933-50953, 2019.

[21] D. Alejo, J. M. Diaz-Banez, J. A. Cobano, P. Perez-Lantero, and A. Ollero, "The velocity assignment problem for conflict resolution with multiple aerial vehicles sharing airspace," *Journal of Intelligent & Robotic Systems*, vol. 69, no. 1-4, pp. 331-346, 2013.

[22] J. J. Rebollo, A. Ollero, and I. Maza, "Collision avoidance among multiple aerial robots and other non-cooperative aircraft based on velocity planning," in *Proceedings of the 7th Conference on Mobile Robots and Competitions,* Paderne, Portugal, 2007.

[23] J. Van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *Proceedings of 2008 IEEE International Conference on Robotics and Automation*, Pasadena, CA, 2008, pp. 1928-1935.

[24] Federal Aviation Administration, "Aerodynamics of flight," in *Pilot's Handbook of Aeronautical Knowledge*. Washington, DC: Federal Aviation Administration, 2016, pp. 5-38.

[25] Federal Aviation Administration, Department of Transportation, "Section 91.113: Right-of-way rules: except water operations," 2004 [Online]. Available: https://www.law.cornell.edu/cfr/text/14/91.113.

[26] Y. I. Jenie, E. J. van Kampen, C. C. de Visser, J. Ellerbroek, and J. M. Hoekstra, "Selective velocity obstacle method for deconflicting maneuvers applied to unmanned aerial vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 38, no. 6, pp. 1140-1146, 2015.

[27] J. Farmer, "The volume of a torus using cylindrical and spherical coordinates," *Australian Senior Mathematics Journal*, vol. 19, no. 2, pp. 49-58, 2005.

[28] L. Liu, R. Shi, S. Li, and J. Wu, "Path planning for UAVS based on improved artificial potential field method through changing the repulsive potential function," in *Proceedings of 2016 IEEE Chinese Guidance, Navigation and Control Conference (CGNCC)*, Nanjing, China, 2016, pp. 2011-2015.

[29] L. Wang, J. Kan, J. Guo, and C. Wang, "Improved ant colony optimization for ground robot 3D path planning," in *Proceedings of 2018 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, Zhengzhou, China, 2018, pp. 106-1066.

[30] M. Radmanesh and M. Kumar, "Flight formation of UAVs in presence of moving obstacles using fast-dynamic mixed integer linear programming," *Aerospace Science and Technology*, vol. 50, pp. 149-160, 2016.

Je-Kwan Park **https://orcid.org/0000-0001-6640-7629**

He received his M.S. degrees in computer science and engineering from Kyung Hee University, Korea, in 1994. Since March 1998, he is pursuing his Ph.D. degree in the Department of Electrical and Computer Engineering of Sungkyunkwan University. Since October 2002, he is the CEO of NAVI Co. Ltd., in Korea. His research interests are autonomous mobile robot communication and collision avoidance scheduling among agents.

Tai-Myoung Chung **https://orcid.org/0000-0002-3154-1868**

He received his M.S. degree in computer engineering from the University of Illinois in 1987, and his Ph.D. degree in computer engineering from Purdue University in 1995. He is currently a Professor of computer science & engineering department at Sungkyunkwan University, Korea. His research interests are in information security, information management, digital therapeutics, and protocols on the next-generation networks such as active networks, and mobile networks.