Receding Horizon–based RRT* Algorithm for a UAV Real-time Path Planner

Hanseob Lee[1], Dasol Lee[1], and David Hyunchul Shim[2]
Korea Advanced Institute of Science and Technology, Daejeon, South Korea, 34141

This paper describes the real-time application of the rapidly exploring random tree* (RRT*) algorithm, which is a type of sample-based path planning. The RRT* algorithm can generate an optimized path depending on the number of nodes. However, as the number of nodes increases, the computational speed slows down because of the scanning procedure to find the best nodes, which is why the RRT* algorithm is not normally suitable for real-time applications. Many nodes need to be considered to optimize a path through the entire workspace. If the optimization process can be performed sequentially in the receding horizon area, the computational load can be reduced because fewer nodes are considered. This paper presents the receding horizon–based RRT* (RH-RRT*) algorithm, which uses a biased random sample and node removal method to solve this problem. The algorithm was then applied to a quadrotor simulation as a demonstration.

Nomenclature

$\boldsymbol{T}$	=	set of trees
$\boldsymbol{Z}$	=	set of nodes
$\boldsymbol{V}$	=	set of vertices
$\boldsymbol{E}$	=	set of edges
$\boldsymbol{P}$	=	local path points
z_{goal}	=	goal point
z_{rand}	=	random sample node
$z_{nearest}$	=	node nearest to z_{rand}
z_{new}	=	new node added to the set of vertices
$\boldsymbol{Z}_{rand}$	=	set of nodes near z_{new}

I. Introduction

Recently, the unmanned aerial vehicle (UAV) has been used in various fields, such as reconnaissance and surveillance in the military and for delivery and taking aerial pictures in the civil sector. In order to perform a mission with a UAV, a flight path should be determined so that the UAV can go to a specific place autonomously. Such an algorithm is called path planning. There are two types of path planning methods: grid-based[1] and sample-based.[2] Although each has benefits depending on the environment, the sample-based algorithm is preferable to solve more complicated environments. We used the rapidly exploring random tree* (RRT*) algorithm to generate a path in a random environment.

The RRT algorithm was developed to search high-dimensional spaces efficiently. It solves problems with obstacles and differential constraints easily, so it is widely used in autonomous robot path planning. Multiple studies have considered applying the RRT algorithm in real-time, such as path planning for the autonomous driving of unmanned cars[3] and kinodynamic motion planning to generate a high-dimensional randomized path with a nonlinear dynamic robot.[4,5]

[1] Graduate Student, Dept. of Aerospace Engineering, 291 Daehak-ro, Daejeon, South Korea, PO 34141
[2] Associate Professor, Dept. of Aerospace Engineering, 291 Daehak-ro, Daejeon, South Korea, PO 34141, Senior Member.

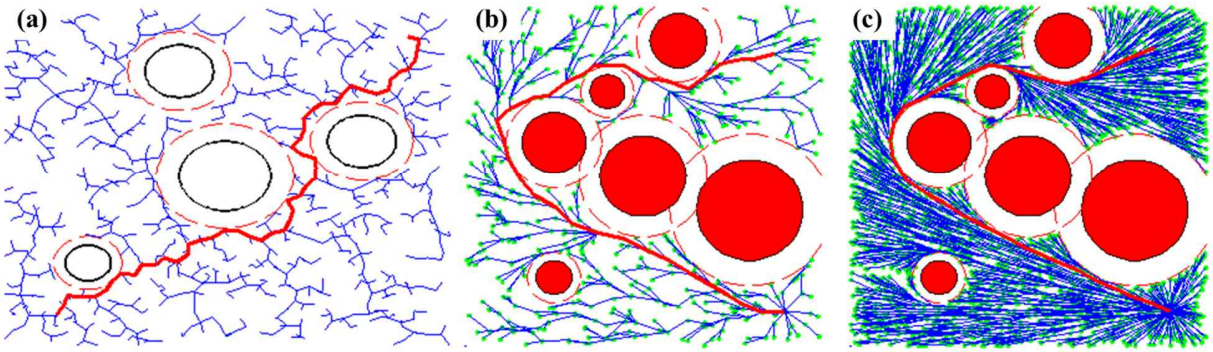

Figure 1. RRT and RRT* path planning around obstacles.

II. Problem Definition

The RRT algorithm is designed to efficiently search nonconvex high-dimensional spaces by randomly building a space-filling tree as shown in Fig. 1(a). It can easily handle problems with obstacles and differential constraints (nonholonomic and kinodynamic) and has been widely used in autonomous robotic path planning. The RRT* algorithm was developed from the RRT algorithm with an added procedure to find the best parent node and rewire the tree to generate a suboptimal path for the given environment.[6, 7] Fig. 1(b) is the figure about the middle of the process of the RRT* algorithm and Fig. 1(c) is the optimized solution by recursive process.

However, the computation time increases with the number of samples. Thus, optimizing the path over the entire area in real-time is difficult. For example, when a random sample is generated, the RRT algorithm performs a process to determine if this sample is near the current tree as shown in Fig. 2(a). If it is close to the tree, the sample is added to the set of the current tree. This requires the distances between the random sample and all of the current nodes to be calculated. Thus, the computation time is dependent on the number of nodes in the tree. And the complexity of obstacle in the workspace is the other factor to make it slow down as shown in Fig. 2(b).

This means that a more optimal flight path is generated as the number of sample nodes increases. However, increasing the number of sample nodes during the RRT* operation slows the computational speed. Hence, we suggest the receding horizon–based (RH-RRT*) algorithm, for which the key concept is optimizing the path sequentially in the receding horizon area rather than the entire path at once. The RH-RRT* algorithm requires procedures to move the mean point of a random sample (i.e., biased random sample) and remove trees connected to the previous flight path that do not affect the next local path (i.e., node removal). We applied the proposed algorithm to the simulation of a six-degree-of-freedom (6DOF) quadrotor to demonstrate its validity in a complicated environment.

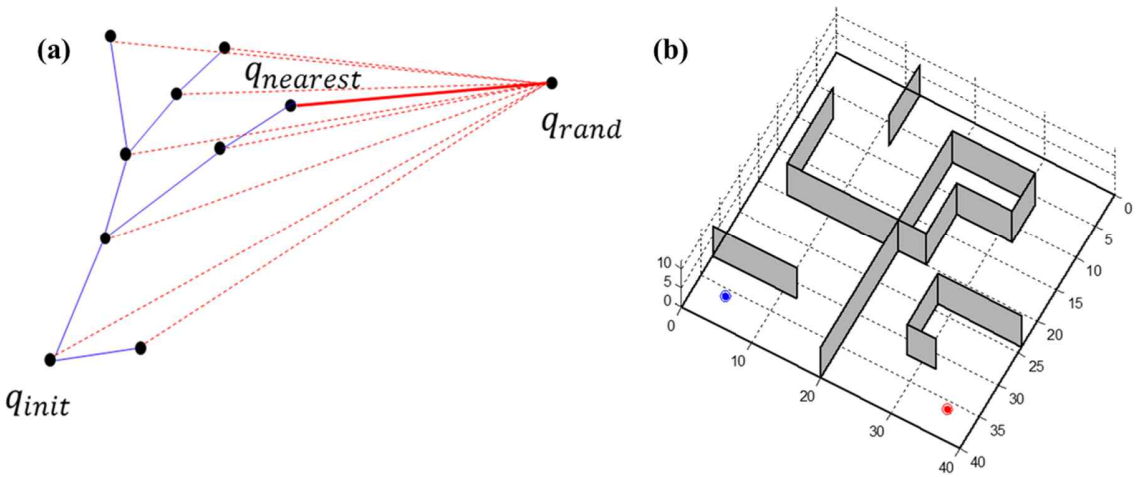

Figure 2. Real-time path planning interference factors: (a) a searching algorithm with many nodes; (b) a complex environment.

III. RH-RRT* Algorithm

The purpose of the RRT* algorithm is to find the global optimal path in a given workspace. The computation time to optimize the path is dependent upon the size of the workspace, resolution of the path, and complexity of the obstacles. In order to apply the RRT* algorithm in real-time, the UAV needs to concentrate on the local area that will pass after a few seconds. Thus, we proposed the RH-RRT* algorithm.

The path that the UAV follows right away is not near the goal point. That means that the optimal path planning algorithm does not need to optimize the path far from the UAV at first. The RH-RRT* algorithm develops the quasi-optimal path sequentially from the receding horizon area. In order to optimize the path in the receding horizon area, a random sample is generated according to the local path. Sequential optimization is performed along the local path, which is called the "biased random sample" method. While the UAV is flying along the local path, the RRT* tree continues to expand, and even a tree connected to the path at a point that has already been passed is expanded as well. As noted before, having to consider many nodes increases the computation time. Thus, nodes that are connected to the passed path need to be removed. This is called the "node removal" process. The biased random sample is considered first.

A. Biased Random Sample

The RRT* algorithm has a process to generate a random sample to extend the random tree. Many studies have considered which random sample generation method is an effective way to find the solution. We previously noted that the key point of an algorithm for real-time searching is the local area. In the case of random sample generation, the mean value, searching radius, variance, etc. are all affected in the same manner.

In order to perform random sample generation in a local area, the local path should be assigned first. We can select the local path among the generated trees containing the nodes by calculating the cost according to the distance information. The maximum cost node within the tree is called *EndVertexInd*. Each node has its parents' node information. Thus, the path which the end vertex of *EndVertexInd* can be found. That is assigned as the local path used for the biased random sample method.

The biased random sample method makes many nodes surrounding the receding-horizon local path. The biased point is assigned as a constant distance away from the current point. Biased Gaussian distribution sampling is used not only to make the quasi-optimal path but also to generate the random tree for the next local path list. It generates many random samples surrounding the biased point, as shown in Fig. 3(b), to make the path more optimal with high probability.

A random sample can be as shown in Fig. 3(b). The workspace has a goal point, which the UAV already knows, and we can obtain the direction angle toward the goal point from the current point. Then, we can formulate the generation of a random sample as follows:

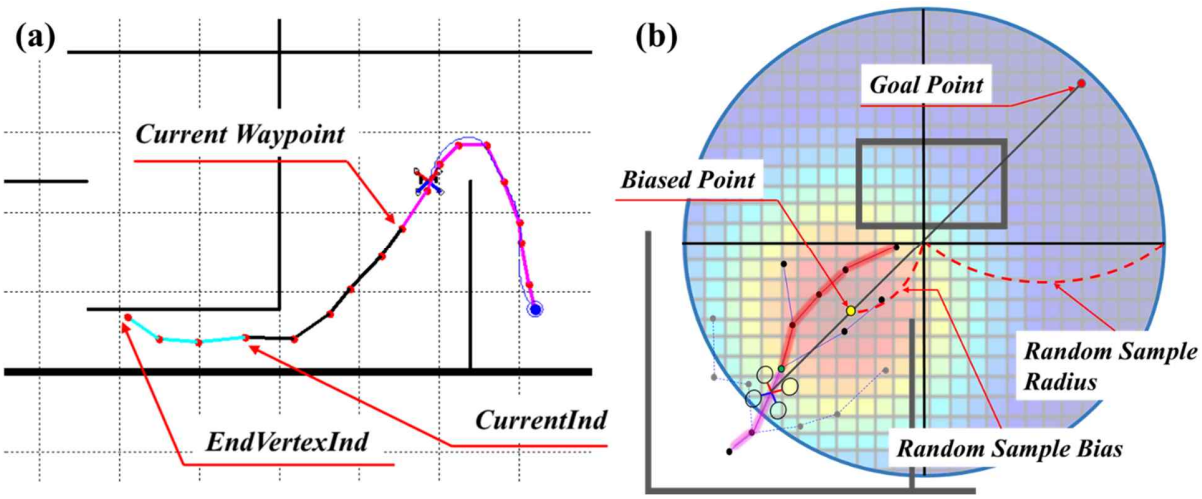

Figure 3. RH-RRT* point definitions: (a) RH-RRT* algorithm path with the waypoint, current index, and maximum cost point; (b) biased random sample method with a biased Gaussian distribution to optimize the receding horizon path and find the open path to the goal point by avoiding obstacles.

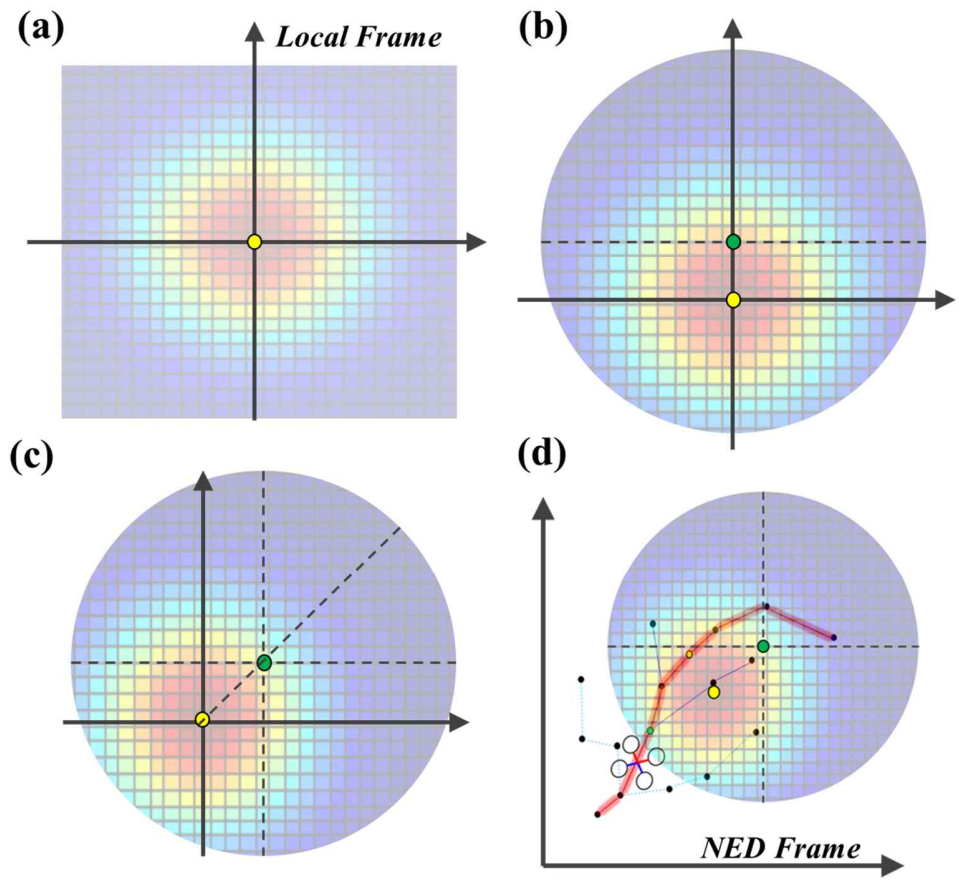

Figure 4. Biased random sample process: (a) zero mean random samples with a Gaussian distribution in a local frame; (b) biased boundary condition; (c) rotation toward the goal point; (d) translation according to the position of the UAV in the NED frame.

$$[R_{bx} \quad R_{by}]^T = \left[\frac{a}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{x^2}{2\sigma^2}\right) \frac{a}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{y^2}{2\sigma^2}\right)\right]^T \tag{1}$$

$$(R_{bx} - b)^2 + {R_{by}}^2 \leq a^2 \tag{2}$$

where a is the random sample radius and b is the random sample bias.

First, we obtain the random values from a Gaussian distribution with a zero mean, as shown in Fig. 4(a). Then, we multiply them by the scalar a, as shown in Eq. (1). Equation (2) indicates that random values are generated with the biased boundary condition, as shown in Fig. 4(b). Then, the directional angle is calculated as shown in Eq. (3) and Fig. 4(c) so that the random samples can be rotated. Finally, Eq. (4) shows that the random values in local coordinates are rotated and moved to north east down (NED) coordinates.

$$\psi_{goal} = tan^{-1}\left(\frac{y_{goal}-y_{aur}}{x_{goal}-x_{aur}}\right) \tag{3}$$

$$\begin{bmatrix} R_{Nx} \\ R_{Ny} \end{bmatrix} = \begin{bmatrix} x_{aur} \\ y_{aur} \end{bmatrix} + \begin{bmatrix} \cos(\psi_{goal}) & -\sin(\psi_{goal}) \\ \sin(\psi_{goal}) & \cos(\psi_{goal}) \end{bmatrix} \begin{bmatrix} R_{bx} + b \\ R_{by} \end{bmatrix} \tag{4}$$

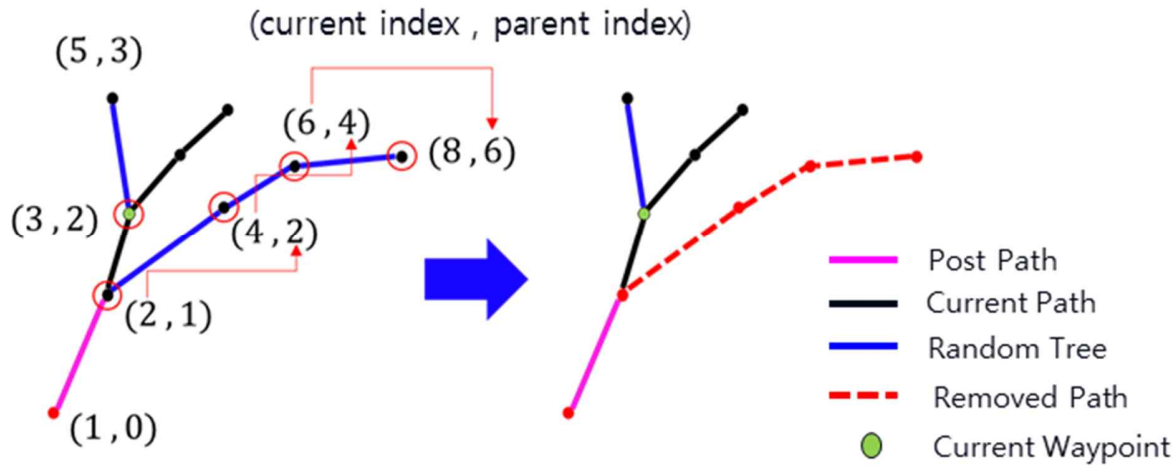

Figure 5. RH-RRT* node removal method: process of finding nodes to remove from the path.

B. Node Removal

In the RRT* algorithm, the main factors that affect the computation time are the number of nodes in the vertex set and the number of obstacles in the workspace. Because an obstacle in the workspace is an element that cannot be changed during the operation of the algorithm, we have to adjust the number of nodes in the vertex set to realize a real-time algorithm.

The node removal method eliminates the nodes that are connected to the previous waypoint node. A tree connected to the previous waypoint no longer affects the path, so these nodes become unnecessary. If these nodes still remain during the algorithm process, they will increase the computation time as explained above. And the previous waypoints are removed as well, because there might be a new generated node connecting the previous waypoint, so they can be expanded at useless area.

In computer science, a tree is a widely used type of abstract data that simulates a hierarchical tree structure with a root value and subtree of children from a parent node; this is represented as a set of linked nodes. Thus, the tree structure can express the relation between two nodes as the child and parent. A node with a child is called a parent node, and every node in the tree has one parent at most. In order to remove a node connected to the previous waypoint, we find those nodes among the vertex set.

The previous waypoint is removed from the vertex set and the parent index is saved in the temporary memory. In the next procedure, all of the nodes with the index of the previous waypoint are found in the vertex set and removed. The node removal method repeats this process until there is no node with the previous parent index. Fig. 6 shows how the node removal method removes nodes during the algorithm process.

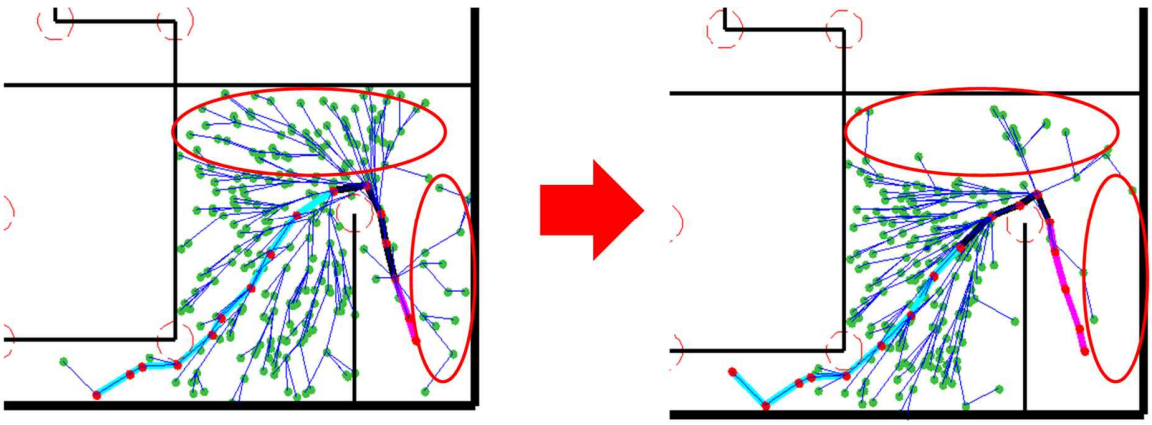

Figure 6. Node removal when the waypoint is changed for the path.

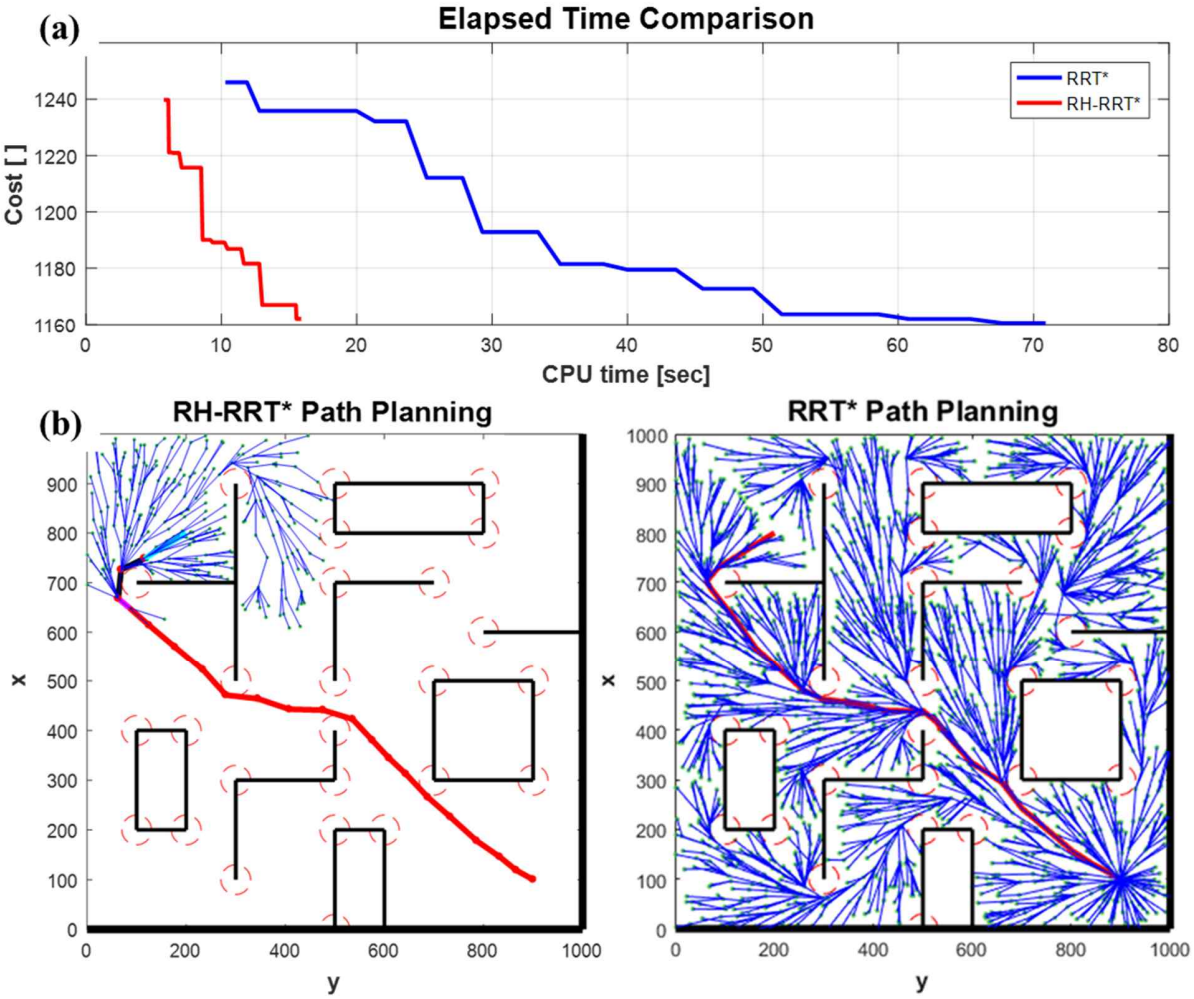

Figure 7. Elapsed time analysis: (a) elapsed times of the RRT* and RH-RRT* algorithms; (b) resulting paths of the RRT* and RH-RRT* algorithms.

IV. Elapsed Time Comparison

The main purpose of this study was to develop an algorithm that generates a random sample-based path in real-time. For this reason, the operating time of the algorithm needed to be analyzed. In this section, we present an analysis of the paths obtained with the RRT* algorithm and proposed RH-RRT* algorithm. The algorithms were built in MATLAB R2016b.

The process time was measured according to the central processing unit (CPU) time. We began to record data when the complete path from the start to the goal point was generated. The cost of the path was calculated by measuring the distance to evaluate the optimization performance. Figure 7(a) graphs the elapsed time and cost of the path with the RRT* and RH-RRT* algorithms. The starting points of the two paths were similar, but the time for convergence was about 70 s for the RRT* algorithm and about 16 s for the RH-RRT* algorithm. The right graph in Fig. 7(b) shows why the computation time was slow with the RRT* algorithm. This algorithm extended and kept the tree throughout the workspace even when optimizing the selected path. In addition, because new nodes were generated outside the path even during optimization, the process slowed over time.

With the RH-RRT* algorithm, however, there were multiple trees near the goal point. In other words, the tree of the RH-RRT* algorithm generated new nodes with a high probability near the path to be optimized right away and removed trees that were connected to the previously passed path, so the number of nodes in the workspace was limited. The entire process of the RH-RRT* algorithm is shown in Fig. 8.

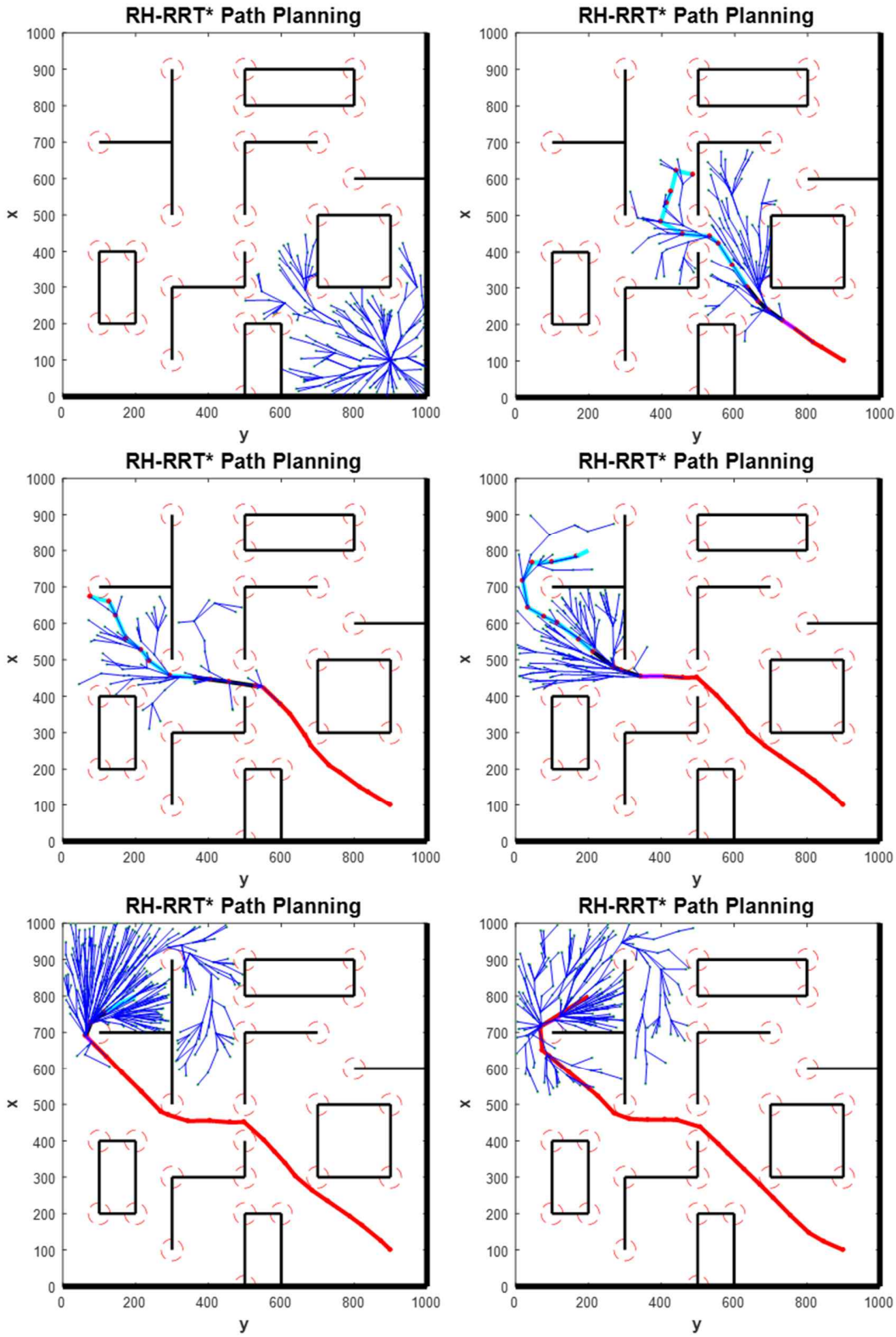

Figure 8. RH-RRT* algorithm process.

Table 1. RH-RRT* Pseudo-code

Algorithm 1. Real-time RRT*

$\boldsymbol{T} \leftarrow$ InitialSetting( )
while(norm $(\boldsymbol{x}_{goal} - \boldsymbol{x}_{uav}) > \eta$)
  $z_{rand} \leftarrow$ BiasRandomSample$(\boldsymbol{x}_{uav})$
  $z_{nearest} \leftarrow$ Nearest$(\boldsymbol{T}, z_{rand})$
  $z_{new} \leftarrow$ Steer$(\boldsymbol{T}, z_{rand}, z_{nearest})$
  if CollisionCheck$(z_{new}, z_{nearest})$
  $\boldsymbol{T}$(V, E) $\leftarrow$ AddVertex$(\boldsymbol{T}, z_{new})$
  $\boldsymbol{Z}_{near} \leftarrow$ FindNearVertex$(\boldsymbol{T}, z_{new})$
  $z_{min} \leftarrow$ BestParent$(\boldsymbol{Z}_{near}, z_{nearest}, z_{new})$
  $\boldsymbol{T}$(V, E) $\leftarrow$ ReWire$(\boldsymbol{T}, \boldsymbol{Z}_{near}, z_{min}, z_{new})$
    if WaypointTranslation(*Waypoint, PostWaypoint)*
      [flag_goal , *EndVertexInd*] $\leftarrow$ GetLocalMax ($\boldsymbol{T}$)
      if flag_goal = 1
    EndVertexInd =GoalResignVertexInd
    end
      $\boldsymbol{P} \leftarrow$ GetLocalPath$(\boldsymbol{T}, EndVertexInd)$
    CurrentInd $\leftarrow$ SampleBiasUpdate$(\boldsymbol{P}, Waypoint)$
    $\boldsymbol{T}$(V, E) $\leftarrow$ RemoveNode$(\boldsymbol{T}, \boldsymbol{P}, Waypoint)$
  end
end
$t = t + \Delta t$
$x_{uav} \leftarrow$ 6DofEquation$(x_{uav}, \boldsymbol{P})$
end

Algorithm 2. $z_{rand} \leftarrow$BiasRandomSample$(\boldsymbol{x}_{uav})$

flag = 1;
flag_workspace = 1;
while (flag_workspace)
  while (flag)
  $N_{rand} \leftarrow$ RandomDist $\times$ GaussianRandom$(0, \sigma_x)$
    dist =norm(N_{rand}, N_{Biased})
  if (dist < RandomDist)
    $\boldsymbol{R}_{temp} = \boldsymbol{N}_{rand}$
    flag = 0
  end
end
ψ_{uav} = atan$(\boldsymbol{x}_{goal} - \boldsymbol{x}_{uav})$
$\boldsymbol{z}_{rand} = \boldsymbol{x}_{uav} + \boldsymbol{Rot}(\psi_{uav}) \times \boldsymbol{R}_{temp}$
if $(\boldsymbol{z}_{rand} < Workspace)$
  flag_workspace = 0
else
  flag = 1
end
end
Return $\boldsymbol{z}_{rand}$

Algorithm 3. $\boldsymbol{T}$(V, E) $\leftarrow$ RemoveNode$(\boldsymbol{T}, \boldsymbol{P}, Waypoint)$

for i = 1 to i = *Waypoint* **do**
  $\boldsymbol{R}(i) \leftarrow$ FindChildNode$(\boldsymbol{P}, \boldsymbol{T})$
end for
$\boldsymbol{R} \leftarrow$ RemovePathVertex$(\boldsymbol{R}, \boldsymbol{P})$
$\boldsymbol{T} \leftarrow$ IndexRemoveNode$(\boldsymbol{R}, \boldsymbol{P})$
while (flag)
  $\boldsymbol{R}' \leftarrow$ FindChildNode$(\boldsymbol{R}, \boldsymbol{T})$
  $\boldsymbol{T} \leftarrow$ IndexRemoveNode$(\boldsymbol{R}', \boldsymbol{P})$
  $\boldsymbol{R} = \boldsymbol{R}'$
  if $\boldsymbol{R}$ = empty
    break;
end while
$\boldsymbol{T} \leftarrow$ RemoveNode$(\boldsymbol{T})$
return $\boldsymbol{T}$

V. Pseudo-code

Before we verified the algorithm with the simulation, we created a pseudo-code as presented in Table 1 to implement RH-RRT* for real-time path planning.

This pseudocode contains additional algorithms, including the existing RRT* algorithm. This algorithm is performed when the UAV reaches the current waypoint on the path and moves to the next waypoint. When the process begins, the algorithm searches for the local maximum cost path that extends the farthest among the random trees and then updates *CurrentInd* as described in the previous section and the *P* set with the local path. An algorithm is then executed to remove the trees connected to the path points that follow. At the end of this part, the UAV is moved by the 6DOF motion equation, and the whole algorithm is repeated.

Algorithm 2 shows the pseudo-code for generating a biased random sample with the RH-RRT* algorithm. It generates a random sample that satisfies the boundary and direction conditions. The direction condition is determined by the current position of the UAV and the goal point. Random samples are generated in unit coordinates, and these are translated by the position of the UAV and rotated by the direction toward the goal point.

Algorithm 3 removes the vertices and edges from the connected trees. The algorithm finds points with the passed path points as the parent node and then finds the points with the parent nodes again as the points found above. This process is repeated until the end point is reached; then, the algorithm becomes an empty matrix, and the found nodes are removed from the *T* set.

With the algorithms described above, the UAV flies until it finds the final target point while the optimized path is generated in real-time.

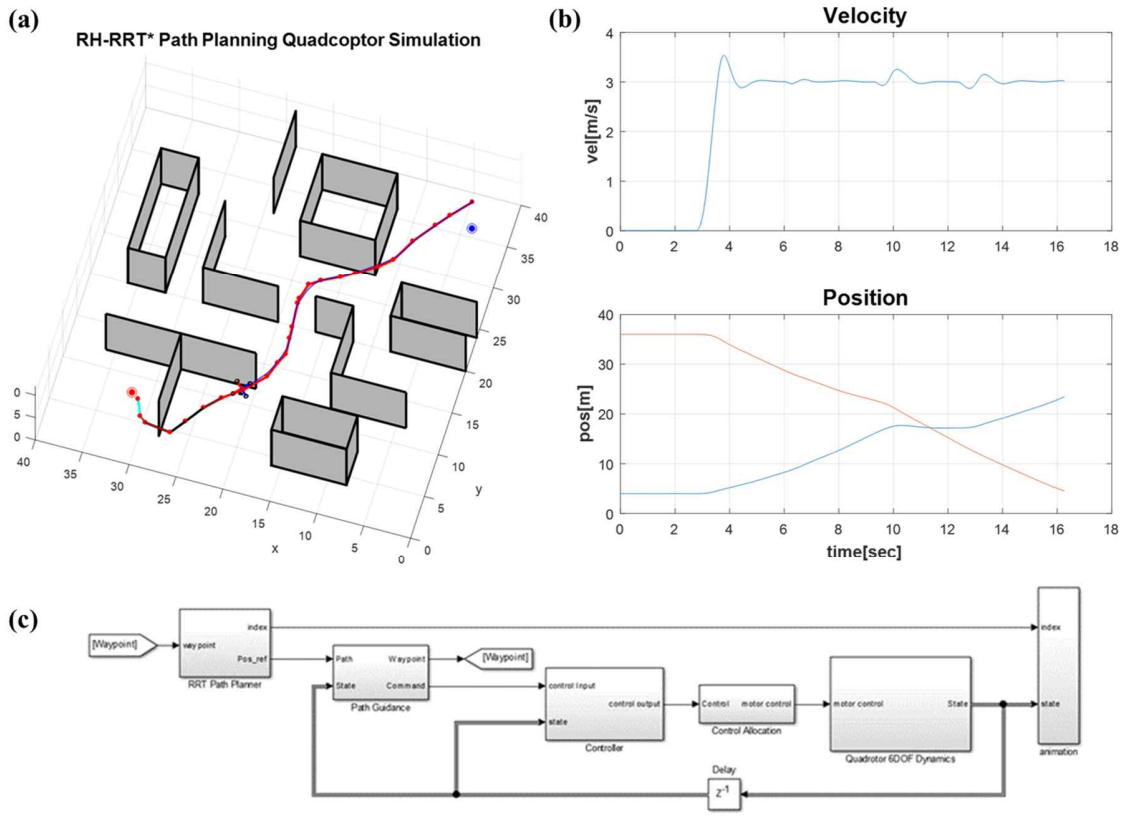

Figure 9. RH-RRT* simulation results: (a) three-dimensional graph of the quadrotor simulation; (b) simulation position and velocity graph; (c) Simulink diagram.

VI. Simulation Results

For the simulation, we used a 6DOF quadrotor model and Simulink-based environment. Figure 9(a) shows how the RH-RRT* algorithm was used in the simulation environment to generate a two-dimensional path in real-time in an area with obstacles such as a wall to reach the goal point. The red line represents the previous waypoint path of the UAV, and the blue line represents the actual path of the UAV. The velocity command was set to 3 m/s.

The simulation diagram in Fig. 9(c) consists of the path planner, guidance, controller, and dynamics parts. The path planner block diagram includes an s-function that executes the RH-RRT* algorithm. It receives the current waypoint index and outputs the command waypoint generated by the RH-RRT* algorithm. The guidance block diagram inputs the waypoint command and generates the velocity command for the UAV to go to the designated waypoint while referring to the current flight state. The controller block diagram consists of the attitude controller and feedforward controller to provide attitude commands depending on the velocity command. The dynamics block diagram includes a 6DOF equation of motion for a quadrotor. The attitude and thrust commands generated by the controller converts the force and torque required for the equation of motion.

VII. Conclusion

In this paper, we raised the problem about not applying in real time. There are numbers of nodes needed in order to optimize the global path, and the more the set of vertex have the nodes, the more it takes the time to optimize. The complexity of obstacle in the workspace is the other reason not to apply in real time as well.

we developed a real-time path planning method based on the RH-RRT* algorithm. In order to overcome the disadvantage of RRT*, for which the computation time slows according to the number of nodes, our algorithm continuously performs node removal and updates the biased random sample to a point in the receding horizon area for effective sampling. The algorithm was verified by simulating the flight of a 6DOF quadrotor with MATLAB/Simulink.

Acknowledgments

This research was part of the project "Development of a core technology and infra technology for the operation of USV with high reliability" funded by the Ministry of Oceans and Fisheries, Korea, and was supported by the BK21+ Program.

References

[1]Lee, D., and Shim, D., "Optimal Path Considering Real Terrain for Fixed-Wing UAVs," *Journal of Institute of Control, Robotics and Systems*, Vol. 20, No. 12, 2014, pp. 1272~1277

[2]Ferguson, D., Likhachev, M. and Stentz, A., "A guide to heuristic-based path planning," *International Conference on Automated Planning and Scheduling (ICAPS) Proceedings of the Workshop on Planning under Uncertainty for Autonomous Systems*, 2005.

[3]Kuwata, Y., Teo, J., Fiore, G., Karaman, S., Frazzoli, E., and How, J., "Real-time Motion Planning with Applications to Autonomous Urban Driving," *IEEE Transactions on Control Systems*, Vol. 17, No. 5, 2009, pp. 1105–1118.

[4]LaValle, S.M., and Kufe, J.J., "Randomized Kinodynamic Planning," *International Journal of Robotics Research*, Vol. 20, No. 5, 2001, pp. 378–400.

[5]Karaman, S. and Frazzoli. E., "Optimal Kinodynamic Motion Planning Using Incremental Sampling-based Methods," *Proceedings of the 49th IEEE Conference on Decision and Control*, IEEE, New York, 2010.

[6]Karaman, S., and Frazzoli, E., "Incremental Sampling-based Algorithms for Optimal Motion Planning," *Proceedings of Robotics: Science and Systems VI*, P33, Robotics Science and Systems Foundation, 2010.

[7]Sertac, K., Matthew, R., Alejandro, P., Emilio, F. and Seth, T., "Any Motion Planning using RRT*," *IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 1478~1483