International Journal of Control, Automation, and Systems (2011) 9(4):750-758 http://www.springer.com/12555
DOI 10.1007/s12555-011-0417-7

Anytime Synchronized-Biased-Greedy Rapidly-exploring Random Tree Path Planning in Two Dimensional Complex Environments

Kwangjin Yang

Abstract: A new synchronized biased-greedy RRT is proposed which leverages the strengths of the biased and greedy RRTs. It combines the advantage of the biased RRT that grows trees towards the goal location, with the ability of the greedy RRT that makes trees traverse the environment in a single iteration. The proposed method achieves performance improvements compared to other RRT variants, not only in computational time but also in the quality of the path. Two enhancements are made to the initial path to relax the sub-optimality of the RRT path; first a path pruning algorithm is executed to eliminate redundant nodes and an anytime strategy is adapted to continuously enhance the quality of the path within the deliberation time.

Keywords: Anytime strategy, obstacle avoidance, path planning, rapidly-exploring random tree.

1. INTRODUCTION

The field of robot motion planning has matured over the years since the original work in the 1970s. Various path planning techniques have been proposed for computing a collision free path amongst obstacles, however, autonomous real-time navigation in complex environments is still an active research topic. There are several considerations for the path planner to take, for example, optimality, completeness and computational complexity. There is a natural trade off between these elements [1]. For real-time applications, computational complexity is the most important requirement since path planning has to occur quickly as the change of the environment [2].

In recent years, sampling-based motion planning has been emergedas a promising approach for the real time implementation. Two traditional probabilistic search methods are the Probabilistic Roadmap (PRM) and the Rapidly-exploring Random Tree (RRT). The PRM planner consists of an off-line roadmap construction and an on-line graph search for path generation. It constructs a roadmap of the configuration space by sampling a random point and trying to link it from each nearby sample point using local planner. After constructing a roadmap network, graph search is applied online once the initial point and goal points are assigned. However, it is difficult to apply PRM to a dynamic and rapidly changing environment as building a roadmap a priori may not be feasible [3]. The RRT algorithm incrementally builds a tree on-line from the starting point to the goal point by adding a new vertex towards randomly selected points. Although the PRM method was intended to explore the configuration space exhaustively in the preprocessing phase, the RRT algorithm tends to achieve efficient single-query planning by exploring the environment as little as possible [3]. A good trait of RRT is that the tree grows towards unvisited regions of the space and never regresses into already explored areas which results in a rapid exploration of the space [4]. This comes from the Voronoi regions in RRT nodes. Larger Voronoi regions exist on the border of the tree. Therefore, the nearest neighbor node selection makes vertices with large Voronoi regions be selected for expansion [5]. The RRT has been demonstrated successfully in real-time applications [6-9].

Normal RRT is not an efficient way to find a path between start and goal configurations because the tree explores the whole space. By selecting the goal point with some probability, the growth of RRT is biased towards the goal instead of exploring the whole space. A greedy variation of the algorithm reduce the planning time by extending trees successively along the ray connecting selected node and random points until a collision occurs. The efficiency of planning algorithm is further enhanced by combining the biased and greedy RRT. Though the biased-greedy RRT algorithm shows better performance in terms of the computational time, the quality of computed path is severely suboptimal.

This paper proposes a new synchronized biased-greedy RRT which leverages the good properties of the biased and greedy RRT. This method can be interpreted as a gradient descent search with a designated probability and uniform exploration with the other's probability.

It should be noted that the RRT planner inherently sacrifices the optimality for the computational efficiency. In this work, the optimality is defined as the shortest path. In addition, the RRT planner does not have the ability to control the quality of the path. These characteristics of RRT naturally raise questions on how one can cope with

Manuscript received October 13, 2010; revised March 30, 2011 and April 5, 2011; accepted April 9, 2011. Recommended by Editorial Board member Youngjin Choi under the direction of Editor Jae-Bok Song.

Kwangjin Yang is with the Department of Aerospace and Mechanical Engineering, Korea Air Force Academy, P.O. Box 335-2 Ssangsoo Namil Cheongwon, Chungbuk 363-849, Korea (e-mail: ykj4957@gmail.com).

the sub-optimality of the path and how one can improve the path quality systematically. One way to relax the sub-optimality of the RRT is to run the RRT algorithm iteratively within the available time for planning. When the planning time has run out, the planner then chooses the best path among paths generated during the deliberation time. This method can reduce the probability of generating a severely sub-optimal path but it cannot guarantee the improvement of path quality.

The anytime planning algorithm is a well established method in the deterministic planning community which generates an initial path quickly and improves the quality of the path within the time allotted for planning. Therefore, the anytime planning strategy offers a built-in means of improving path quality progressively whilst deliberation time is permitted. The anytime planning strategy was firstly applied to RRT path planning in [10]. Path costs were incorporated explicitly into the planning. A discretization of the environment is a conventional way to build a cost map in deterministic planning methods such as A^* and D^*. This discretization approach also is used in [10]. A more exact cost of the path can be calculated by a minute tiling of the environment. However, the disadvantage of the discretization may outweigh the benefits.

Instead of discretization of the environment to evaluate a cost, the Euclidean path distance is used as a cost in this research. The distance measure is not an exact evaluation of the cost but it will be matched if the shortest path generation is the objective of planing. In addition, it removes the necessity of the environment discretization. Therefore, path length based anytime RRT can be operated within the RRT framework which only needs a collision detection function.

The paper is organized as follows: Section 2 presents the synchronized-biased-greedy RRT path planning, and a path pruning algorithm which removes extraneous nodes is described in Section 3. Section 4 presents an anytime RRT path planning. Simulation results are shown in Section 5 and conclusions are thereafter.

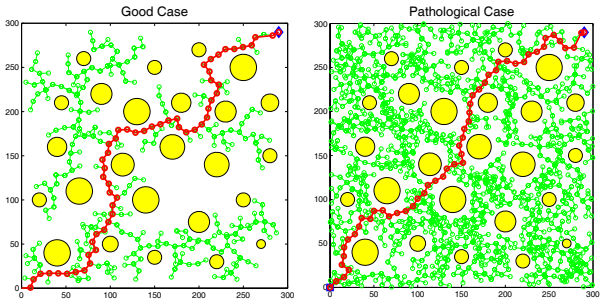
2. SYNCHRONIZED-BIASED-GREEDY RRT PATH PLANNING

2.1. Basic RRT

Algorithm 1: Baisic RRT Algorithm

1: **Build_RRT**(x_{init})
2: $T_{.init}$ (x_{init}))
3: **while Distance**(x_{goal}, x_{new}) > d_{lim} **do**
4: $x_{rand} \leftarrow$ **RandomState**()
5: $x_{near} \leftarrow$ **NearestNeighbor**(T, x_{rand})
6: $x_{new} \leftarrow$ **Extend**(T, x_{rand}, x_{near})
7: **If Distance**(x_{goal}, x_{new}) ≤ d_{lim} **then**
8: Return T
9: **endif**
10: **end while**

The RRT algorithm incrementally builds a tree from the starting point to the goal point adding a new vertex which is one of randomly selected points.



(a) Good case. (b) Pathological case.

Fig. 1. Basic RRT planning results.

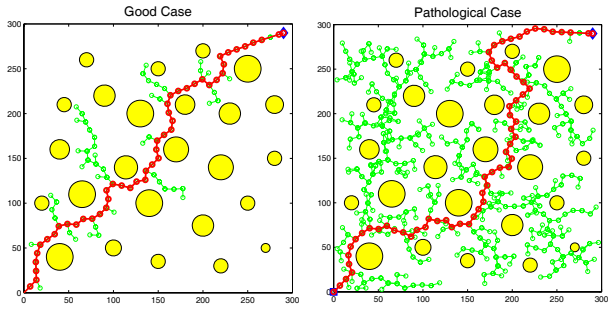
Algorithm 1 shows the basic RRT algorithm. First, RRT selects a random state x_{rand} (Line 4), and this state is compared with existing tree nodes to find the closest node in the tree, x_{near} (Line 5). A line is drawn connecting x_{near} to x_{rand}, and a new point x_{new} is generated along this ray at a fixed distance from x_{near}. If there is no collision on the interval between x_{near} and x_{new}, the latter is added to the tree (Line 6). The algorithm ends if the distance between x_{new} and x_{goal} is smaller than the threshold d_{limit} (Line 7).

Fig. 1 shows the basic RRT path planning results. The green lines denotes all RRT trees and the red lines denote the generated path. The starting point is (0,0) and the goal point is (290,290). In Fig. 1(a), RRT can find the collision free path quickly without exploring the whole space but this is not often the case. RRT tries to explore the whole space as shown in Fig. 1(b) for the most part. This comes from the fact that the basic RRT algorithm always selects the candidate point randomly which makes the tree growth uniform. It is not an efficient way to wander the whole space as the primary goal of path planning is to find a feasible path between start and goal configurations.

2.2. Biased RRT

A simple but efficient way to enhance the performance of the search is to bias the growth of RRT towards the goal instead of exploring the whole space. This can be achieved by selecting the goal point with some probability. A biased RRT selects the goal locations with probability p and random points are selected with probability 1– p. As p increases, the RRT behaves increasingly like the best first search [4]. Here the probability of goal selection is a design parameter. If the environment is densely cluttered with obstacles, it is better to reduce this probability but if the environment has sparsely located obstacles, higher probability can reduce the time for generating a collision free path. It is generally recognized that 10% of probability is a reasonable choice regardless of the density of obstacles.

Fig. 2 shows the biased RRT path planning results. This method enables a feasible path to be found within a short time and stumble less to unnecessary areas compared to the basic RRT case as can be seen in Fig. 2(a). Even though the total tree number is reduced greatly, trees still explore unnecessary areas of the environment as in Fig. 2(b).



(a) Good case. (b) Pathological case.

Fig. 2. Biased RRT planning results.

2.3. Greedy RRT

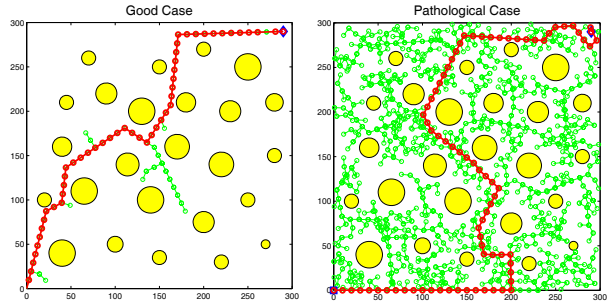
One of the most time consuming processes in the RRT algorithm is when finding the nearest neighbor node. The computational cost increases as the number of trees increase. K-d tree method can relieve this problem by efficiently maintaining a data structure, but the best way to speed up the process is to reduce the use of the nearest neighbor function. A greedy variation of the algorithm can reduce the use of this function by extending trees successively along the ray connecting selected node and random points until a collision occurs. This greedy RRT frequently shows a better performance since any relatively open and unobstructed regions are traversed in a single iteration [11].

Fig. 3 shows the greedy RRT path planning results. This method can find a feasible path quickly with less frequent use of the nearest neighbor calculation function as in Fig. 3(a). However, the drawback of this method is that the resulting path is severely suboptimal. Moreover, the planning time will be increased if the RRT nodes are extended in the wrong direction as can be seen in Fig. 3(b).

2.4. Biased-greedy RRT

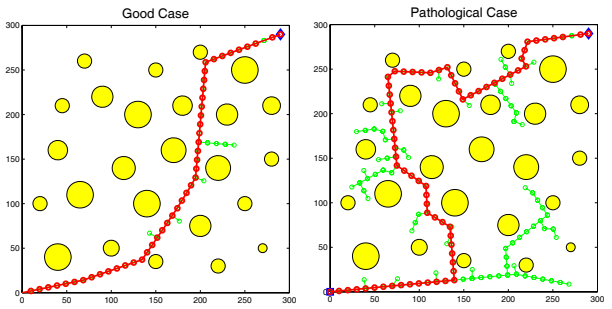
Biased RRT can make the tree grow toward the goal location and the efficiency will be enhanced by combining the biased and greedy RRT. It chooses a state using a biased sampling strategy and then grows trees as many times as possible before a collision is detected.

Fig. 4(a) shows the result of the biased-greedy RRT planning method. This method can reduce the planning time greatly. However, similar to the greedy RRT this method is sub-optimal as can be seen in Fig. 4(b).



(a) Good case. (b) Pathological case.

Fig. 3. Greedy RRT planning results.



(a) Good case. (b) Pathological case.

Fig. 4. Biased-greedy RRT planning results.

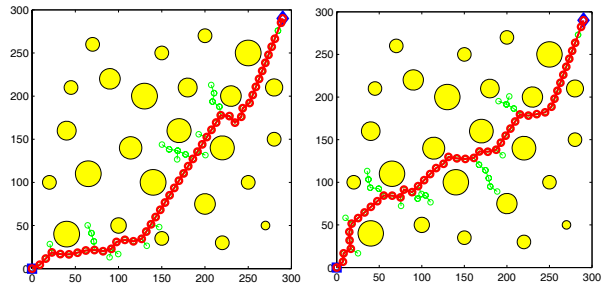
2.5. Synchronized biased-greedy RRT

The biased-greedy RRT algorithm shows better performance in terms of the computational time. However, this algorithm has the advantages and disadvantages of the biased RRT and the greedy RRT algorithms. The advantages of the biased RRT and greedy RRT are that it grows trees towards the goal location and it makes trees traverse the environment in a single iteration respectively. The disadvantage of the biased RRT algorithm is that it requires frequent use of the nearest neighbor function as extensions occur only once though potential direction is selected and the greedy RRT frequently explores unnecessary areas as it extends trees regardless of goodness in that direction. A synchronized biased-greedy RRT algorithm is proposed to improve the path planning performance which has only good properties of both methods. The synchronized biased-greedy RRT executes a greedy extension where the goal location is selected as a target point and extends just one time otherwise.

Algorithm 2: Synchronized Biased-Greedy RRT Algorithm

1: **Build_RRT**(x_{init})
2: $T_{.init}$ (x_{init}))
3: **while Distance**(x_{goal}, x_{new}) > d_{lim} **do**
4: x_{rand}← **BiasedState**()
5: x_{near}← **NearestNeighbor**(T, x_{rand})
6: x_{new}← **SyncGreedyExtend**(T, x_{rand}, x_{near})
7: **If Distance**(x_{goal}, x_{new}) ≤ d_{lim} **then**
8: Return T
9: **endif**
10: **end while**

Algorithm 2 shows the synchronized biased-greedy RRT algorithm. The synchronized biased-greedy RRT can be interpreted as a gradient descent search with a p probability and a uniform exploration with 1-p probability. This approach will not get trapped in a local minima due to the 1-p probability of random exploration and will find the shortest path with p probability due to the greedy extension. With this synchronized combination of the biasness and greediness, the method is effective in reducing the computational time in comparison with biased RRT. In addition, it generates a more optimal path compared to greedy RRT.



(a) Case 1. (b) Case 2.

Fig. 5. Synchronized biased-greedy RRT planning results.

Fig. 5 shows planning results of a synchronized biased-greedy RRT algorithm. There are two long greedy extensions in Fig. 5(a) case which can quickly extend trees toward the goal location. The resulting path is shorter and takes less time to generate compared to previous methods. There are a significant number of obstacle blockages in Fig. 5(b) case but it can easily get away from the local minima and find the shortest path quickly without exploring unnecessary areas. From Fig. 5 it can be seen that the greedy extension is stopped when a collision is detected but it can easily find the collision free path in that regions by uniformly searching the free space with 1-p probability. After escaping from an obstacle, it tries again to extend the tree in the direction of the gradient descent to the goal with p probability.

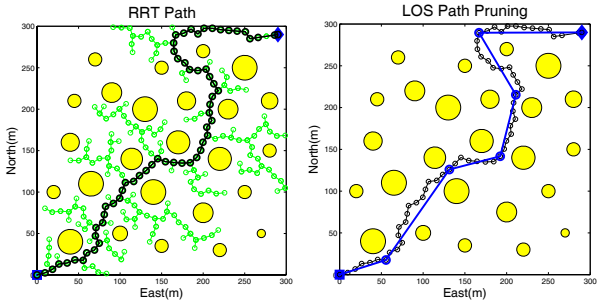
3. PATH PRUNING

Even though RRT is an effective and computationally efficient tool for complex online motion planning, the solution is far from optimal due to its random exploration of the space. There are a lot of redundant nodes and wavy motions which simply increase the path length. In this Section, a simple yet an efficient method which is able to eliminate most extraneous and wavy nodes within a short time is described. Two algorithms are proposed for this path pruning.

3.1. Line of sight path pruning

The first path pruning method uses local information about the path. Let the pruned path be initially an empty set. First, define the begin node (BNode) and end node (ENode) for collision checking. Initially, the BNode is assigned to the first node of RRT path and the ENode is the second node of the RRT path. Then check the line between the BNode and the ENode for a collision, stopping when a collision is detected between them. If there is a collision, the previous node from this ENode is added to the pruned path and reassigned to the BNode and ENode. This process is repeated until a complete path is generated. Finally, the final path is obtained adding the first and the last RRT path to the pruned path.

Fig. 6(b) shows the pruning result of the LOS path pruning algorithm applied to the initial RRT path in Fig. 6(a). The RRT path initially has 59 nodes between the start and goal point. However, this number reduces to only 5 after the redundant waypoints are pruned.



(a) RRT path before pruning. (b) RRT path after pruning.

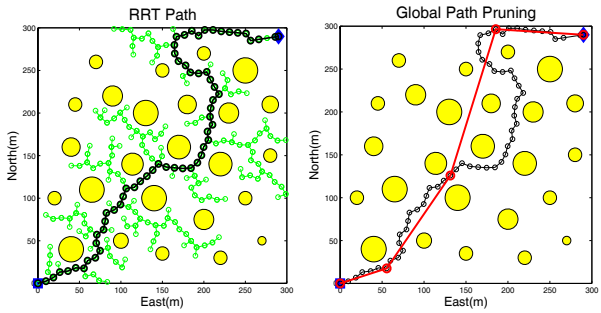
Fig. 6. Line of sight path pruning.

There are several benefits to this pruning algorithm. First, it reduces the path length by removing the wavy nodes. Second, path following is made easily since most of the path consists of straight lines. Finally, it is also beneficial to the vehicle actuators because a lot of jaggy motions are removed.

3.2. Global path pruning

As seen in the preceding Section, the LOS path pruning can efficiently remove the redundant waypoints in most cases. However, there still exists wavy motions due to short-sightedness of LOS. To provide better reasoning about the pruning process, a modification is made to the LOS path pruning. The LOS algorithm starts the pruning process from the next point of BNode and stops the process when a collision is detected. Although there is a collision in the j_{th} node, there can be no collisions between BNode and the $j + 1_{th}$ node. Based on this reasoning, the pruning process starts on the last node of the RRT path. It slightly increases the processing time but removes the myopic behavior of the LOS pruning. This method is called Global path pruning as it tries to prune the path considering the entire path nodes.

Fig. 7 shows the pruning result of the global path pruning which is the same path as seen in Fig. 6. After applying the global path pruning algorithm, nodes between the start and the goal point are reduced to 3 which is less than the LOS pruning case.



(a) RRT path before pruning. (b) RRT path after pruning.

Fig. 7. Global path pruning.

4. ANYTIME RRT

The second improvements to relax the sub-optimality of the RRT path is to apply anytime strategy. It generates an initial path quickly and improves the quality of the path within the allowable planning time. The anytime RRT operates as follows: First, the RRT algorithm generates an initial path without considering path cost and favoring quick generation of the initial path. After obtaining the initial path, the role of the planner is shifted from path generation to path optimization. Then, the planner progressively improves the quality of the path until the planning time runs out. Several modifications in the *state sampling*, *node selection* and *node expansion* functions are made to enable the RRT algorithm fit into an anytime planning strategy.

4.1. State sampling

It is possible to get information about the environment at each iteration from the previous successful solution because of the iterative path generation process. We would like to use the previous path information in the next iteration to enhance the path quality. However, it should be noted that all prior information is not always beneficial. If the preceding path is severely sub-optimal, the information of this path will mislead the current path. On the other hand, if the previous path is close to the optimal path, this information can greatly contribute to improve the path quality. Therefore, it is important to make a decision as to whether or not use the previous path information and how useful that information is in guiding the current path generation. With this insight, a path cost is incorporated for the judgment of the level of prior path information to be used in the generation of a new path.

The next question is how to utilize the previous path information. The waypoint cache method is used which is suggested in [12]. The nodes of the prior path are stored and these nodes will be chosen with some probability in the next path. This approach is modified to fit in the anytime RRT framework. First, as explained previously, the probability of choosing the prior waypoint cache is determined based on the quality of the path. Yet a fixed probability is used in [12]. Second, a small perturbations are added to the prior waypoint cache. This modification comes from the intuition that the prior path is not necessarily the optimal path. To find a better solution near the previous information, the points near the prior cache are chosen.

4.1.1 Adaptive state sampling

Algorithm 3 shows the state sampling strategy which is used in the anytime RRT.

Algorithm 3: Anytime RRT State Sampling

1: p=RandomValue([0,1])
2: **if** $0 \le p \le p_g$ **then**
3: return x_{goal}
4: **elseif** $p_g \le p \le p_c$ **then**
5: return x_{seed}
6: **else**
7: return x_{random}
8: **end if**

Line 2-3 is a biased sampling which chooses the goal point with some probability. Line 4-5 is a guided sampling which chooses previous path nodes with adaptive probability. This makes the planner reuse prior information. Line 6-7 is a random sampling which chooses any points within the environment. The combination of three sampling methods speeds up the search process by growing the tree towards a promising direction.

There are two parameters to be determined; the probability to choose the goal point (p_g) and the probability to choose the prior path nodes (p_c). The probability to choose a goal point is decided as p_g=0.1 because 10% goal biased sampling exhibits a good performance regardless of the environment. The prior path node selection probability (p_c) is not fixed but varied based on the prior path quality. The path length is used as a cost of the path. First the path length between the starting point to the goal point is calculated. This value is set as an optimal cost because the straight line between these points is the shortest path. Equation (1) shows the optimal cost calculation.

$$C_{opt} = L_{min} = \sqrt{(x_g - x_s)^2 + (y_g - y_s)^2 + (z_g - z_s)^2} \quad (1)$$

If a new path is generated, the length of this path is computed to evaluate the quality of the path. Then the probability p_c is decided adaptively based on this value as in (2).

$$p_c = \begin{cases} Q_p - 0.5, & \text{if} \quad Q_p \ge 0.5 \\ 0, & \text{if} \quad Q_p < 0.5, \end{cases} \quad (2)$$

where $Q_p = \frac{L_{min}}{L_{path}}$ is the ratio between the optimal path length and the previous path length which decides the path quality. As the quality of the previous path is improving, the probability to choose the waypoint cache increases. If Q_p is less than 0, the probability to choose the waypoint cache is set to zero.

4.2. Node selection

Once the state is sampled according to the adaptive state sampling method, the next task is to select a node to extend toward the sampled state. The second modification of the RRT for the application of the anytime RRT method is to incorporate a path cost into the node selection to extend. Similar to the state sampling, the weighting of the cost is varied according to the quality of the path instead of applying a fixed contribution.

4.2.1 Progressive weighting of cumulative cost

The distance metric when choosing a node is changed for the improvement of path quality. Once the feasible path is generated, the focus of the planning will be shifted from the feasible path generation to the optimal

path generation. The original algorithm chooses the nearest node from the existing node to the sample point. Instead of applying only the nearest node selection strategy, the cumulative cost of the existing node is also considered for optimization.

$$C_{sum} = C_{dist} + w \cdot C_{cum}. \quad (3)$$

Equation (3) shows the evaluation of the cost when choosing a node to extend. C_{dist} is a Euclidean distance from each node to the sampled point; C_{cum} is a cumulative cost from the starting point to the current node; w is a weighting of the cumulative cost; and C_{sum} is the cost of each node to extend toward the sampled state. The nodes are sorted in ascending order of C_{sum} when choosing the cheapest node.

The cumulative cost weighting w is also varied in a similar fashion to state sampling. Equation (4) demonstrates how to decide on the weighting w.

$$w = \begin{cases} N_{ite} \cdot k, & \text{if} \quad w \le w_{max} \\ w_{max}, & \text{otherwise,} \end{cases} \quad (4)$$

where N_{ite} is an iteration number of path planning, k is an incremental value of weighting, and w_{max} is a maximum weighting value.

As the planning time progresses, the cumulative cost becomes increasingly weighted. The subsequent increase of the cumulative cost weighting is aimed at making the new path gradually approach the optimal path. The node selection result is the same as the original RRT method if w = 0 in (3) and only the starting point will be selected if w = 1. Therefore, the w must be smaller than 1 to grow the tree. In our application, w_{max} and k are chosen as $w_{max} = 0.7$ and $k = 0.1$ respectively.

The transition from path generation to path optimization changes the growing characteristic of the RRT tree. The distance based nearest neighbor node selection makes the tree grow rapidly to the larger Voronoi region. The focus of this stage is to find a solution quickly. The inclusion of the cumulative cost hinders the pure expansion of the tree growth towards the Voronoi region. Instead, it drives the tree growth towards the direction associated with the shortest path. As a result, there is a natural trade-off the quick generation of the sub-optimal path and the better quality path generation with the increase in planning time. To accommodate this characteristic, the cumulative cost weighting gradually increases to progressively improve the path quality within the time available for planning.

4.3. Node expansion

After the node to extend is decided, the RRT extends the selected node to the sampled point. In anytime RRT, however, the chosen node will not be extended if it is not beneficial for the improvement of path quality. The rationale of this decision is that the poor node extension cannot contribute to the enhancement of the path quality. The purpose of the anytime RRT is not a feasible path generation but a better quality path generation than the previous path. Therefore, if the extension of the selected node cannot contribute to the path quality improvement, it does not need to extend. To achieve this, we need to estimate the expected total cost of the path

4.3.1 Cost-to-go estimation

The total cost of a path can be calculated by summing the accumulated cost from the starting point to the selected node (C_{cum}), and the cost from the selected node to the sampled point (C_{dist}), and the cost from the sampled point to the goal point (C_{ctg}).

$$C_{total} = C_{cum} + C_{dist} + C_{ctg} = C_{ctc} + C_{ctg}. \quad (5)$$

Equation (5) shows how to calculate the total cost of the path. The cumulative cost (C_{cum}) and the extension cost (C_{dist}) can be expressed as a cost-to-come cost (C_{ctc}). Therefore, the total cost consists of the cost-to-come cost (C_{ctc}) and cost-to-go cost (C_{ctg}). Only the cost-to-go needs to be calculated as the cost-to-come has already been calculated in Section 4.2.

It is difficult to calculate the exact cost-to-go. The complexity of finding the exact cost-to-go value is comparable to the original path planning problem. Instead of finding the exact cost-to-go, the Euclidean distance is used from the sampled point to the goal point as an estimate of the cost-to-go. Even though this Euclidean cost-to-go estimation is not an exact measure, it will offer the lower bound of cost-to-go as any path cannot be shorter than the straight line.

4.3.2 Decision for the node expansion

In anytime RRT, whenever a new path is generated the length of this path is set as the upper-bound-cost of the path. From (5), the expected cost of the path can be calculated after tree expansion. If this expected cost is higher than the upper-bound-cost, the node expansion will not occur. Equation (6) shows the node expansion condition.

$$C_{total} < C_{upper}. \quad (6)$$

According to the evaluation of the path cost before extending the tree, only the promising nodes are extended towards the promising state. This process reduces the planning time by not allowing the addition of unnecessary trees.

5. SIMULATION EXPERIMENTS

We conduct the simulation experiments to investigate the performance of synchronized biased-greedy RRT and anytime RRT.

5.1. Performance comparison of several RRT algorithms in randomly generated environment

To compare the performance of the proposed synchronized biased-greedy RRT with other RRT variants, Monte-Carlo simulations are conducted in randomly generated environment. The environment size is 500m×500m and the starting point is (0, 0) and the goal point is (500,500). At each run, the number of obstacle

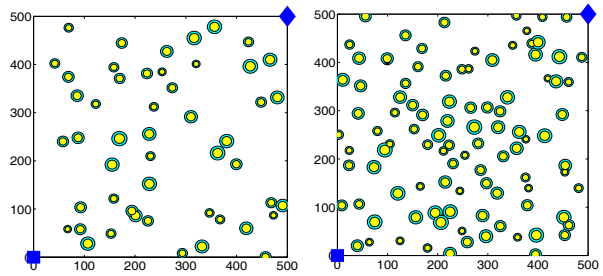


Fig. 8. Randomly generated environments: Monte-Carlo simulations are conducted in randomly generated environment to compare the performance of several RRTs. (a) Randomly generated environment when the obstacle number is 50. (b) Randomly generated environment when the obstacle number is 100.

varies from 50 to 100, and the radius of obstacle varies from 5m to 10m.

Fig. 8 shows two examples of the randomly generated environments. Fig. 8(a) shows randomly generated environment when the obstacle number is 50, and Fig. 8(b) shows randomly generated environment when the obstacle number is 100. The number and size of obstacle are changed at each run. The yellow circle denotes randomly generated obstacles and the cyan circle denotes the safety zone of the obstacle.

Table 1 shows mean values of run time, node number, and path length over 500 runs. The basic RRT takes the longest time to find a path. Even though the greedy RRT is able to find the solution faster than the basic RRT, the path length is much larger than the basic RRT. This comes from the blind extension of trees. The biased RRT reduces run time greatly in comparison with the basic and greedy RRT. In addition, the resulting path is shorter than the basic and greedy RRTs. The biased-greedy RRT can further reduce the run time but preserves the sub-optimal property of greedy RRT. The synchronized biased-greedy RRT generates the shortest path with the least time amongst them. If we compare the planning time of between the basic RRT and the synchronized biased-greedy RRT, the former longer than the latter about one hundred times.

Table 2 shows standard deviation values of the run time, node number, and path length over 500 runs. Similar to the mean values as Table 1, the variation of the run time, node number, and the path length for the proposed synchronized biased-greedy RRT algorithm is smaller than other methods.

Table 1. Comparison of several RRT algorithms in randomly generated environment: mean values over 500 runs.

	Run Time	Node	Length
Basic	60.82s	1880	884m
Greedy	23.64s	1161	914m
Biased	1.31s	311	820m
Biased-greedy	0.64s	233	869m
Synchronized	0.38s	145	747m

Table 2. Comparison of several RRT algorithms in randomly generated environment: standard deviation values over 500 runs.

	Run Time	Node	Length
Basic	47.21s	526	39m
Greedy	32.06s	634	96m
Biased	1.59s	121	30m
Biased-greedy	0.65s	134	91m
Synchronized	0.56s	108	34m

5.2. Performance comparison of several rrt algorithms in deadlock environment

In the second simulation, a deadlock environment is generated to investigate the RRT performance in local minima situation. The environment size is 100m×100m, and the starting point is (50,40) and the goal point is (50,95) as shown in Fig. 9. The starting point is encompassed by obstacles and only one side is free.

Fig. 9 shows the synchronized biased-greedy RRT planning result. The RRT planner must grow trees in opposite way from the start point to the goal point to escape the local minima and also pass the narrow passage in both sides.

Table 3 shows mean and standard deviation values of run time over 500 runs. The path length of several RRTs are almost same since the route from the staring point to the goal point is fixed and the narrow corridors of both sides restrict the tree expansion. For this reason, only planning time is used to compare the performance. Though the differences of the planning time among several RRTs are not big compared with the randomly generated environment case, the proposed synchronized biased-greedy RRT takes the least time and exhibits the smallest the standard deviation of the planning time.

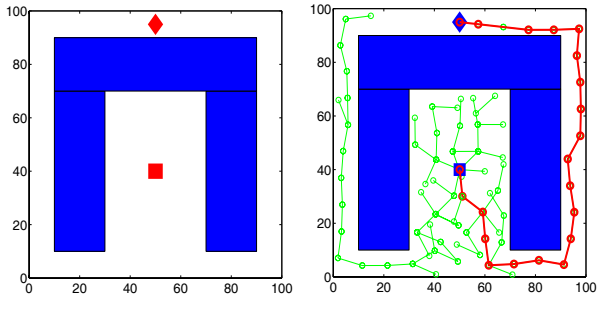


Fig. 9. RRT performance in deadlock environment. (a) Deadlock environment. (b) Synchronized Biased-Greedy RRT planning result.

Table 3. Comparison of several RRT algorithms in deadlock environment: mean and standard deviation values over 500 runs.

	Run Time(mean)	Run Time(STD)
Basic	0.31s	0.25s
Greedy	0.24s	o.19s
Biased	0.27s	0.18s
Biased-greedy	0.19s	0.13s
Synchronized	0.16s	0.08s

5.3. Performance of the anytime RRT

The performance of the anytime RRT is investigated in this simulation. The starting point is (0,0) and the goal point is (180,230).

Figs. 10 and 11 show the simulation results of the anytime RRT. The linear paths of the anytime RRT are shown from Fig. 10 and their smooth paths are shown from Fig. 11. A path smoothing algorithm in [13] is applied to the linear path to clearly compare the path quality among them. The previous path and the current path are plotted together in the linear path to show the improvement of the path quality.

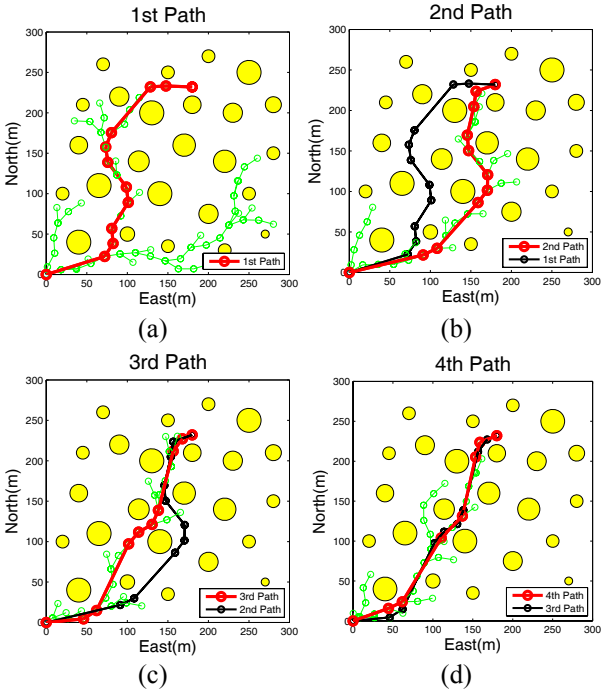


Fig. 10. The performance of the anytime RRT: Linear Path.

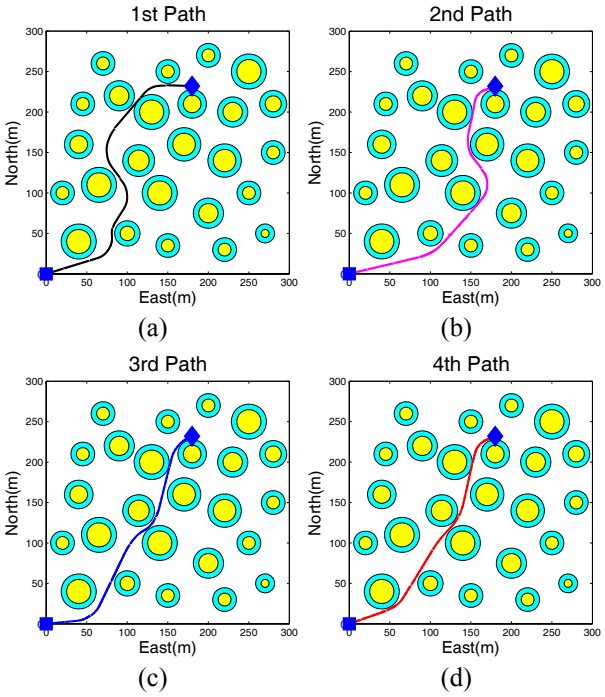


Fig. 11. The performance of the anytime RRT: Smooth Path.

Table 4. Path cost of the anytime RRT.

	1st	2nd	3rd	4th
Path length	377m	369m	328m	319m

Initially the RRT explores the space to find the solution quickly which results in a poor quality path as shown in Fig. 10(a). After finding the initial path, however, the anytime RRT can find the enhanced path from the second iteration as shown in Fig. 10(b). The big difference between the first and the second path is the tree expansion profile. The trees in the first path grow largest Voronoi region but the trees in the second path does not explore unnecessary regions.

This characteristic comes from the node expansion property of the anytime RRT which does not grow tree if it is not beneficial for the improvement of the path quality. The cost of path is greatly reduced at the third iteration by finding a best route as shown in Fig. 10(c). The advantage of anytime RRT is that it tries to find the better route to reduce the path cost as the increase of the iteration. Once it finds the best route, it tries to improve the path quality by finding a better path within the best route. Fig. 10(d) shows this characteristic. By drawing more samplings from the previous waypoint cache, and by selecting and expanding node within the best route with higher probability, the anytime RRT find a better path.

Table 4 shows the path cost of each RRT path. The cost of the path is the length of the path. The quality of the path is gradually improved and it becomes approaches to the optimal solution as the increase of the iterations.

6. CONCLUSIONS

This paper proposes a synchronized biased-greedy RRT which utilizes the good properties of the biased and greedy RRT. This method can be interpreted as a gradient descent search with a designated probability and uniform exploration with the other's probability. With the synchronized combination of the biasness and greediness, this method can reduce the computational time compared to biased RRT as well as generate a more optimal path compared to greedy RRT. Though RRT is a computationally efficient tool for complex online motion planning, the resulting path contains a lot of wavy redundant nodes. A pruning algorithm is proposed which is able to remove most of the extraneous and wavy nodes within a short time. After applying the pruning algorithm, the most path consists of straight lines and has less jaggy motion. Finally, an anytime strategy is applied to continually improve the quality of the path until deliberation time allows. Simulation results show that anytime RRT makes the path gradually closer to the optimal path as the increase of the iteration number.

REFERENCES

[1] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*, MIT Press, Cambridge, MA, 2005.

[2] E. Frazzoli, M. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *Proc. of American Control Conference*, Arlington, Virginia, June 2001.

[3] S. LaValle and J. Kuffner, "Randomized Kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378-400, May 2001.

[4] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with RRTs," *Proc. of IEEE International Conference on Robotics and Automation*, Orlando, Florida, May 2006.

[5] S. LaValle, "Rapidly-exploring random trees: a new tool for path planning," *TR 98-11*, Computer Science Dept, Iowa State University, 1998.

[6] J. Saunders, B. Call, A. Curtis, R. Beard, and T. McLain, "Static and dynamic obstacle avoidance in miniature air vehicles," *AIAA Infotech@ Aerospace*, Arlington, Virginia, September 2005.

[7] E. Koyuncu, N. Ure, and G. Inalhan, "Integration of path/maneuver planning in complex environments for agile maneuvering UCAVs," *Journal of Intelligent and Robotic Systems*, vol. 57, no. 1-4, pp. 143-170, February 2010.

[8] M. Wzorek and P. Doherty, "Reconfigurable path planning for an autonomous unmanned aerial vehicle," *Proc. of IEEE International Conference on Hybrid Information Technology*, Cheju, Korea, November 2006.

[9] N. Aminy, D. Boskovic, and K. Raman, "A fast and efficient approach to path planning for unmanned vehicle," *Proc. of AIAA Guidance, Navigation, and Control Conference*, Keystone, Colorado, August 2006.

[10] D. Ferguson and A. Stentz, "Anytime RRTs," *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, China, pp. 5369-5375, October 2006.

[11] M. Kalisiak and M. Panne, "RRT-blossom: RRT with a local flood-fill behavior," *Proc. of IEEE International Conference on Robotics and Automation*, Orlando, Florida, pp. 1237-1242, May 2006.

[12] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Lausanne, Switzerland, October 2002.

[13] K. Yang and S. Sukkarieh, "An analytical continuous curvature path smoothing algorithm," *IEEE Trans. on Robotics*, vol. 26, no. 3, pp. 561- 568, June 2010.

Kwangjin Yang received his B.S. degree in Mechanical Engineering from Korea Air Force Academy in 1996, an M.S degree in Mechanical Engineering from Pohang University of Science and Technology in 2002 and a Ph.D. degree in Aerospace, Mechanical and Mechatronic Engineering from the University of Sydney in 2010. His research interests include path planning, RUAV control, cooperative control.