Two-Sided RRT* Planner Considering Inter-Node Maximum Length Connection on Adversarial Workspaces for AGV

Anugrah K. Pamosoaji[1]
Department of Industrial Engineering, Fakultas Teknologi Industri Universitas Atma Jaya Yogyakarta
Jl. Babarsari 44, Depok, Sleman, Daerah Istimewa Yogyakarta, Indonesia
Email: [1]anugrah.pamosoaji@uajy.ac.id

Abstrak. *Paper ini mempresentasikan Two-SidedRapidly-Explored Random Tree (RRT*) yang merupakan varian dari algoritma perencanaan jalur RRT* yang memanfaatkan simpul awal dan simpul target sebagai basis untuk menghasilkan jalur. Keuntungan dari metode ini adalah kemampuan untuk membuat koneksi antara simpul awal dan simpul target di bawah ruang kerja yang berlawanan. Dalam jenis ruang kerja ini, masalah utama dalam RRT* adalah tingkat keberhasilan membangun rute yang lengkap dan optimal dan mengurangi waktu pemrosesan pembuatan jalur. Algoritma yang diusulkan bertujuan untuk meningkatkan tingkat keberhasilan pembuatan rute dan mengurangi waktu pemrosesan tersebut. Teknik ini terdiri dari dua hal: penerapan panjang maksimum jalur antar-simpul dan pembuatan simpul dua sisi, yaitu, dari simpul awal dan target. Hasil simulasi menyimpulkan bahwa penerapan panjang maksimum jalur antar-simpul yang besar dapat meningkatkan tingkat keberhasilan konstruksi rute yang lengkap.*
Keywords: *Automated guided vehicle (AGV), perencanaan jalur, rapidly explored random tree (RRT), Algoritma A**

Abstract. *This paper presents a two-sided Rapidly-Explored Random Tree (RRT*) which is a variant of RRT* path planning algorithm that utilizes a start and a target node as the bases for generating paths. The advantage of this method is in the capability to make a connection between start and target nodes under adversarial workspace. In this type of workspaces, the main problem in RRT* is the success rate of constructing a complete and optimized route and reducing path-generation processing time. The proposed algorithm is purposed to increase the route generation success rate and reduce the processing time. The technique consists of two-fold: the application of maximum length of inter-node path and two-sided node generation, i.e., from the start and target nodes. Simulation results conclude that the application of large maximum length of inter-node path can increase the success rate of complete route construction.*
Keywords: *Automated guided vehicle (AGV), path planning, rapidly explored random tree (RRT), A* algorithm*

1. Introduction

Manufacturing is an important component of modern industry. In modern industry, especially Industry 4.0, warehouses play a significant role in ensuring the continuity of the supply chain. If warehouse operations are disrupted, the entire supply chain can be affected. Contemporary warehouse systems are characterized by efficient material handling activities. In large-scale warehouse systems, the use of automated guided vehicles (AGVs) and robotic arms has become common, especially with the emergence of Industry 4.0 as a prevailing paradigm. Typical warehouse operations include loading and unloading of docks and pallets.

The operation of AGVs relies on two critical components: motion control and path planning. Motion control refers to the mechanism of driving the motion of AGVs from a predetermined starting node to a specific target node. Nevertheless, in adversarial environments like those in large warehouses, motion control itself is often insufficient to handle some dynamic conditions, such as the presence of unpredictable obstacles. Therefore, path planning is utilized to enhance the performance of motion control. The issue of path planning is urgent in the environment of AGVs, especially in the context of safety guarantees. Applying motion control

without exact path planning will potentially increase the safety risk. Therefore, designing a collision-free and rigorous path planning for AGVs is mandatory in smart manufacture.

One widely used path planning technique is the Rapidly-Exploring Random Tree (RRT) algorithm, and its more efficient variant, RRT*. Such algorithms are especially effective in complex or adversarial workspaces, which is highly possible to include obstacles and differential constraints. To ensure the efficiency and safety in operations, warehouse robotic systems require a generated plan—either a path or a trajectory. For each task, the robot must solve the problem of navigating from its initial position and orientation (or configuration) to a target configuration while avoiding collisions with obstacles.

Numerous RRT and RRT* algorithms have been developed over the past two decades. Recently, their applications span a variety of domains, including mobile robots [1]-[3], truck-trailer systems [4][5], unmanned surface vehicles [6], robotic arms [7][8], aerial vehicles [9]-[11], ships [12], and surgical robots [13]. Notwithstanding these advancements, a common issue remains: the efficiency of connecting the start and target nodes in the minimum time.

Most of the referenced works do not address the issue of the required time for generating feasible paths. Although nodes are affirmed to be randomly generated, some form of "directional randomness" is needed to ensure that the exploration does not bend into irrelevant or counterproductive directions, such as paths leading significantly away from the target node. Moreover, conventional RRT and RRT* algorithms tend to rely on short node-to-node connections, which often increases the number of iterations required to obtain an optimized path.

The modified RRT* algorithm in this paper addresses two key issues. First is the need to guide the randomness in node generation. Second is the use of an allowable maximum extension length. Directional randomness is defined as the reduced probability of generating nodes in areas that lie significantly outside the viable path corridor toward the target. In this paper, the relationship between the allowable maximum extension length and it is hypothesized that both features will influence the number of required iterations and overall processing time.

2. Literature Review

The Rapidly-Exploring Random Tree algorithms (RRT and RRT*) are well-known and very efficient class of path planning algorithms used in robotics. Such the algorithms provide a support for robots to discover the motion strategy for a very basic mission, i.e., moving from a start node to a target node in a far distance under complicated workspace.

The basic RRT operates under the following four steps: random sampling, tree expansion, bias toward exploration, and goal biasing. The random sampling step is a method that randomly samples configurations from a set of free-space in a given workspace. The dimension of the configurations may be various. It can be 2D, 3D, or higher dimensions.

Tree expansion is the core of the exploration of the RRT. In this method, new nodes are generated and connections between them and the existing closer nodes are established. Under a complex and adversarial workspace, the new node generation also calls the function of collision avoidance. Therefore, the resulted new nodes and their connections are collision-free. The randomly-generation method potentially plots the nodes into unpredictable unexplore areas. Then, some route alternatives can be performed. Finally, this method makes connections between some nodes that are close to the target configuration.

According to Zhu and Qiu (2025), the RRT algorithm was developed by considering kinematic model of mobile robots [1]. Here, the turning radius of the robot leads to the next generated nodes. Therefore, the generated edges (node-to-node link) have a shape of nonholonomic vehicles motion, which in turn is more realistic. In other developments, the combination of RRT and a training algorithm called Generative Adversarial Network-based Direct Trajectory Planning (GDTP) was proposed in [4].

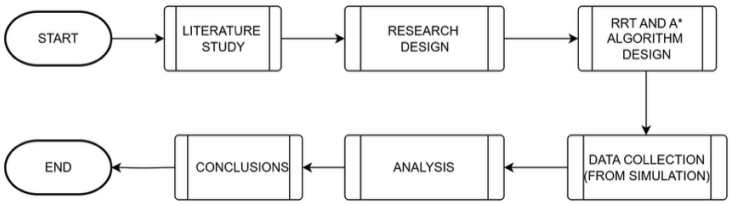


Figure 1. Research Workflow

Various techniques in more optimalized algorithm of RRT, namely, RRT* have been introduced. The use of dual-structure RRT* was proposed by Chen et al (2018). The proposed algorithm enhanced the traditional RRT* (first structure) with optimalization (second structure). Smoothing has been an interesting issue since, usually, the traditional results of RRT and RRT* mostly cannot accommodate the kinematics constraints of the tracking vehicles. Therefore, some smoothing algorithms were developed [6][9][14]. Here, the use of spline curves was proposed. As the work in [14], a generated RRT* can be smoothed by treating some waypoints in a particular route as control points for the curve by considering some geometrical constraints related to collision.

Based on the literature review, there is an issue that has not been considered, i.e., the factors that affect the computational time. Mostly, the literature focused on the success rate of connecting the start and the goal points without proposing a mechanism to reduce the computational time. In this paper, the maximum length of node-to-node paths and the application of two path sources are considered to reduce the computational time. Moreover, the ability of the proposed algorithm to generate paths on a narrow passage between obstacles is demonstrated.

3. Research Methodology

3.1. Research Workflow

The overall research workflow is illustrated in Figure 1. This research begins with a literature review aimed at tracking the development of path planning research—particularly Rapidly-Explored Random Trees (RRT)—identifying issues that have emerged in previous studies, and revealing research gaps. When the issue is identified, the activities are continued to designing a simulation scenario. This includes setting up various parameters such as the allowable maximum extension length of all node-to-node paths, and the number of nodes placed in free space.

The defined issues are then applied through algorithm design and followed by the development of a simulation program using MATLAB. This software is chosen for its robustlibrary support, which facilitates data visualization and efficient calculation. The proposed algorithms include: one method is dedicated for generating nodes in free space, one method for sorting nodes based on their distance to the target node, and another method is utilized for detecting obstacles.

The RRT simulation is executed according to the designed scenario. Visual results are collected for several parameter configurations. Three key outputs are analyzed: (1) the shape of the resulting path and whether a connection from the START to the GOAL node is successfully formed, (2) the processing time for each parameter setting, and (3) the measured distance to determine whether it is optimal. The results of these simulations are summarized in the conclusion.

3.2. Path Planning using Rapidly-Explored Random Tree* (RRT*)

3.2.1 Basic Principle

We introduce a path generation algorithm based on the Rapidly-Explored Random Tree (RRT*) method, utilizing a two-sided node generation approach, as presented in Algorithm 1. In this variant of RRT*, each node is generated by referencing both the start and target nodes, as

shown in Algorithm 2. The objective of the algorithm is to produce an optimal complete route connecting the start and target nodes within an adversarial workspace.

Algorithm 1. Two-Sided RRT* Planning

```
Define workspace layout and static obstacles
Determine Ps and PT
G ← Ps, PT
Determine Lmax, N
distDummy = 100000  (or any large number)
idx = 1
while idx < N:
   new_node = generateNewNode(side)
   closerNodes = findClosestNode(nodes)
   new_node = limitLength(new_node)
   collStatus = checkObstacleIntersection (β, γ, α)
   if collStatus == COLLISION
      continue
   else
      closerNode.neighbor ←new_node
      new_node.neighbor ←closerNode.neighbor
      G ← new_node
   end
   side = changeSide(side)
end
resultedPath = A*-search( )
```

Algorithm 2. generateNewNode(side)

```
if side == TARGET:
   new_node.x = rand( ) * HEIGHT + |xT| * rand(Δxmin,Δxmax] )
   new_node.y = rand( ) * WIDTH + |yT| * rand(Δymin,Δymax] )
else:
   new_node.x = rand( ) * HEIGHT + |xS| * rand(Δxmin,Δxmax] )
   new_node int.y = rand( ) * WIDTH + |yS| * rand(Δymin,Δymax] )
end
```

Algorithm 3. A-star-search( )

```
1: openList ← {}
2: closeList ← {}
3: openList ← START
4: current = openList(1)
5: while openList ∉ {}
6:   if current == TARGET
7:      resultedPath = FindRoute ( current, closeList)
8:      break
9:   end

10:  for each member m of closeList
11:     if current ∩ closeList = {}
12:        closeList ← updateCloseList()
13:        removeOpenList(current)
14:     else
15:        goto Line 6
17:     end
17:  end
18:     openList ← current.neighbor
19: end
```

Algorithm 4. FindRoute(current, closeList)

```
closeList ←current
m = length(closeList)
for each member m of closeList:
   m = m - 1
   parent = m-th member of closeList
   if m-th member of closeList == START
      resultedPath ← m-th member of closeList
      break
   end
   If parent == m-th member of closeList
      resultedPath ← parent
      parent = the parent of the m-th member of closeList
   end
end
return resultedPath
```

Let G be defined as a set of nodes on a Euclidean workspace. Let $\mathbf{p}_S(x_S, y_S)$ and $\mathbf{p}_T(x_T, y_T) \in$ G be start and target nodes, respectively. Let $L_{\max}$ be the maximum length of inter-node path; N be the number of nodes, and $L_{i,S}$ and $L_{i,T}$ be the length of the i-th node to the start and target nodes, respectively. A complete route is defined as the set of adjacent connected node pairs such that there is a way to go from the start to the target node.

Algorithm 1 shows the process of generating a complete route using the two-sided RRT* planning algorithm. At the beginning, the information of the workspace's layout, known obstacles are defined (line 1), and the position of $\mathbf{p}_S$ and $\mathbf{p}_T$ (line 2). Then, a graph G is provided to store the incoming generated nodes (line 3). The feature of this proposed algorithm is $L_{\max}$ and N determined in line 4. Lines 7-19 show the procedure for generating a new node. This procedure starts by creating a new node by calling `generateNewNode`() in line 8, followed by calling `findClosestNode`() in line 9, which connects the new node to the closest existing node. The process is continued by calling `limitLength` () which adjust the position of the generated node such that its distance to the closestNode is below $L_{\max}$. The last process is the collision-free verification by calling `checkObstacleIntersection()` in line 11. In this method, if the connection between the new node and its closest node intercepts any obstacle, then the process is repeated for a different side. If the new node is collision-free, then the new node is added as the neighbor of the closer node (lines 15-17). Otherwise, the exploration is repeated (line 8). Once the new node is created, the source of the node generated is switched to another side (line 19).

Algorithm 2, i.e., `generateNewNode()`, specifically explains the process of generating a new node. In this algorithm, the new node is generated by plotting the position using the formula in lines 2 and 3 (for target-side nodes) and lines 5-6 (for start-side nodes).

The process of finding the best path is shown in Algorithms 3 and 4. The A* algorithm is a well-known algorithm dedicated to optimal pathfinding over a graph, typically the closest one. This algorithm is efficient, complete, and optimal for shortest path finding between the start and goal nodes. The path is composed of a series of nodes, where each node has a minimized cost function formulated as shown in Equations (1)-(2).

$$f(n) = g(n) + h(n), \tag{1}$$

where

$$g(n) = g(n-1) + \|\mathbf{p}(n) - \mathbf{p}(n-1)\| . \tag{2}$$

This algorithm starts by declaring *`openList`* and *`closeList`* as empty sets (Lines 1-2). The *`openList`* is a list for nodes that will be checked, and the *`closeList`* is the list for all nodes that are finished checking. Then, the openList is filled with the start point (Line 3). A variable *`current`* is used to store the node that is checked. The term "check" in here means to explore the neighbors and identify the parent node.

A loop is run as long as the openList is not empty (Lines 5-19). Here, each node, starting from the start point, is checked. If the node is a target, then Algorithm 4 is run, and then the entire program is terminated (Lines 6-9). In Lines 10-17, a loop for each member of the process is described as follows. A method *`updateCloseList()`* is called. If the current node is not a member of *`closeList`*, then put the node as the new member of *`closeList`*. This step is followed by removing the `current` from the `openList`. Otherwise, if the value f of *`current`* is less than the m-th member of the `closeList`, then the member is updated with *`current`*, and the process starts again at Line 4.

Algorithm 5 describes the steps for arranging a route from the target point to the start point. Here, a complete route is constructed by arranging each scanned node with its parent. This process is recursively executed until the last parent node, i.e., the start point.

3.2.2 Obstacle-Free Nodes Plotting Algorithm

In a workspace with many adversarial objects, each new randomly generated node needs to be evaluated to determine whether it is located in the prohibited area or in its shadow area. If this node is located in the prohibited area, then the position of this node must be modified.

Defined $C_{\text{obs}}(x_{\text{obs}}, y_{\text{obs}})$ is the center node of an obstacle object with radius r; $\text{A}(x_{\text{A}}, y_{\text{A}})$ and $\text{B}(x_{\text{B}}, y_{\text{B}})$ are respectively the positions of the reference node (the closest candidate node to

the new random node) and the new random node to be evaluated; γ is the angle formed by the line connecting nodes A and C_{obs} and the positive X-axis; α is the angle formed by the line connecting nodes A and B and the positive X-axis; and d is the distance from node A to C_{obs}.

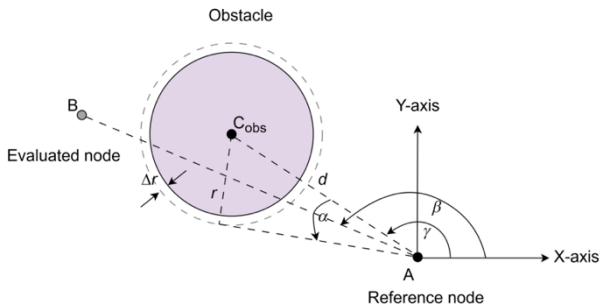


Figure 2. Model of randomly-generated node validation

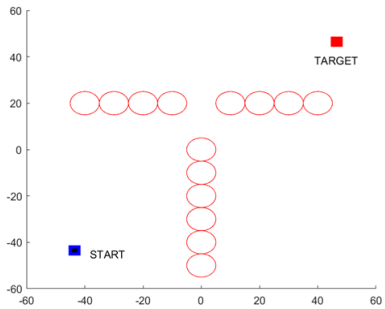


Figure 3. Workspace set up for simulations

According to Figure 2, it can be concluded that node B is said to be prohibited if: 1) its distance from the center of the obstacle object C_{obs} is smaller than the radius $r + \Delta r$, where Δr is the minimum distance between node B and the edge of the obstacle object; and 2) the angle γ is between α and γ.

From Figure 2, we can derive Equations (3)-(5):

$$\gamma = \arctan((y_{\text{obs}} - y_{\text{A}})/(x_{\text{obs}} - x_{\text{A}})), \tag{3}$$

$$\beta = \arctan((y_{\text{B}} - y_{\text{A}})/(x_{\text{B}} - x_{\text{A}})), \tag{4}$$

$$\alpha = \text{asin}\left(\frac{r+\Delta r}{d}\right) \tag{5}$$

where, $\alpha, \beta, \gamma \in (-\pi, \pi]$. Since α is limited by the reference node (namanya), obstacles, and the evaluated node, therefore the interval of α is: $\alpha \in [-\pi/2, \pi/2]$. Moreover, since α is calculated from the line connecting the center node C_{obs} and the tangent line of the obstacle then in general there exits two angles performed by the two tangential line, i.e., $(\gamma + \alpha)$ and $(\gamma - \alpha)$.

Another issue to be taken into account is that the intervals of $(\gamma + \alpha)$ and $(\gamma - \alpha)$ are $[-\pi/2, \pi/2]$. The problem appears because when γ is near π or $-\pi$ since $(\gamma + \alpha)$ may be larger than π for and $\gamma \geq 0$ and $(\gamma - \alpha)$ may be less than $-\pi$ for any $\gamma < 0$, which violates the boundary of α. In consequences, when $(\gamma + \alpha)$ or $(\gamma - \alpha)$ satisify such the conditions, their value must be converted such that $(\gamma + \alpha) < 0$ for all $\gamma \geq 0$ and $(\gamma - \alpha) > 0$ for $\gamma < 0$.

4. Results and Discussion

Simulations were established using MATLAB and were based on three parameters, i.e., the minimum distance to the obstacle $L_{\text{ob,min}}$, the number of generated nodes N, and the maximum length of the edges L_{max}. The layout of the simulation workspace is shown in Figure 3. The simulations are held for $L_{\text{ob,min}} = 2$ m. Two series of simulations were organized. The first series was those for $L_{\text{max}} = 15$ m and the second one was those for $L_{\text{max}} = 30$ m. Each series consisted of five simulations for N equals to 100, 200, 400, and 800 nodes. The output to be analyzed is the success rate of the algorithm to connect the start node to the target node and the success rate of the generated path to pass through the passage.

4.1. Simulation set for $L_{\text{max}} = 15$ m.

The results of the simulations for $L_{\text{max}} = 15$ m, is revealed in Table 1. Also, the sampled visualization of the performance is depicted in Figure 4. From Table 1, it can be concluded that for the small number of *N*, the success rate of connecting the start and the target nodes tends to be low. As the number of *N* increases, the success rate of route generation increases. The results can be explained as follows. The increase of *N* leads to the increase of the density of the node distribution (those generated from the start-side and from the target-side). Therefore, the possibility of connecting a node generated from the start-side and any particular node generated from the target-side increases.

Table 1. Simulation Results for L_{max}= 15m

L_{max} (m)	N	success rate of start-to-target connection	success rate of passing-through path
15	100	20%	0%
15	200	40%	0%
15	400	100%	20%
15	800	100%	60%

Another performance to analyze is the ability of the algorithm to perform a path that passes through the space between obstacles. From Table 1, the possibility of generating such kind of path is larger as *N* becomes larger. It is straighforward that as *N* larger, the possibility to place nodes in the inter-obstacle area is higher.

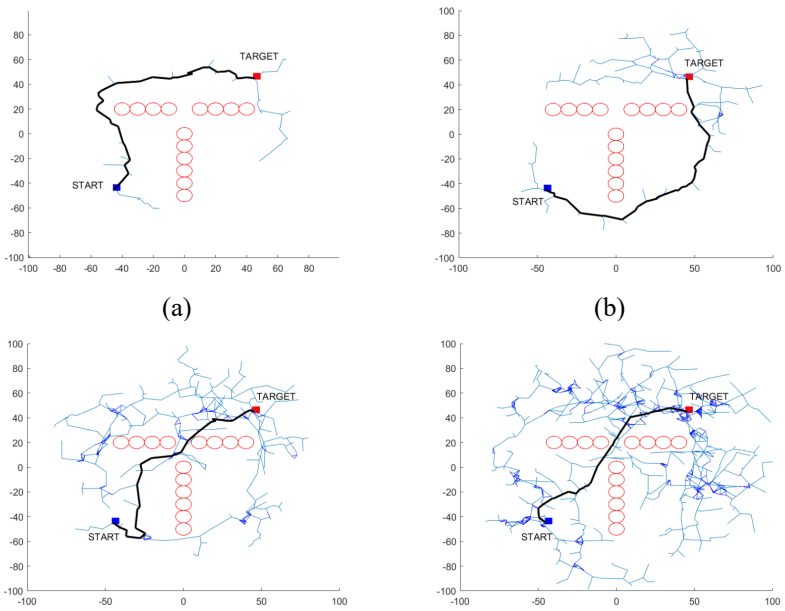


Figure 4. Simulation results for (a) $L_{\text{ob,min}}$ = 2, N = 100, and L_{max} = 15; (b) $L_{\text{ob,min}}$ = 2, N = 200, L_{max}= 15; (c) $L_{\text{ob,min}}$= 2, N = 400, and L_{max} = 15; and (d) $L_{\text{ob,min}}$= 2, N = 800, and L_{max}= 15

4.2. Simulation set for $L_{\max} = 30$ m

The results of the simulations for $L_{\max} = 30$ m are revealed in Table 2, and the sampled visualization of the performance of the algorithm is shown in Figure 5. According to Table 2, it can be concluded that for the small size of *N*, the success rate of connecting the start and the target nodes tends to be low. However, compared to the results of $L_{\max} = 15$ m, the possibility of success in this simulation set is larger. The results indicate that the larger $L_{\max}$ leads to performing a long edge. Consequently, the possibility of a node generated from the start-side to be closer to the target node (and vice versa) is higher than that of lower $L_{\max}$. One interesting issue in here that for $N = 800$, the success rate of start-to-target connection is lower than that for $N = 400$. It is possible because of the emptiness of the connection between the set of the start-based nodes and the target-based ones.

Table 1. Simulation Results for L_{max}= 30m

$L_{\max}$ (m)	N	success rate of start-to-target connection	success rate of passing-through path
30	100	20%	0%
30	200	80%	40%
30	400	100%	100%
30	800	80%	100%

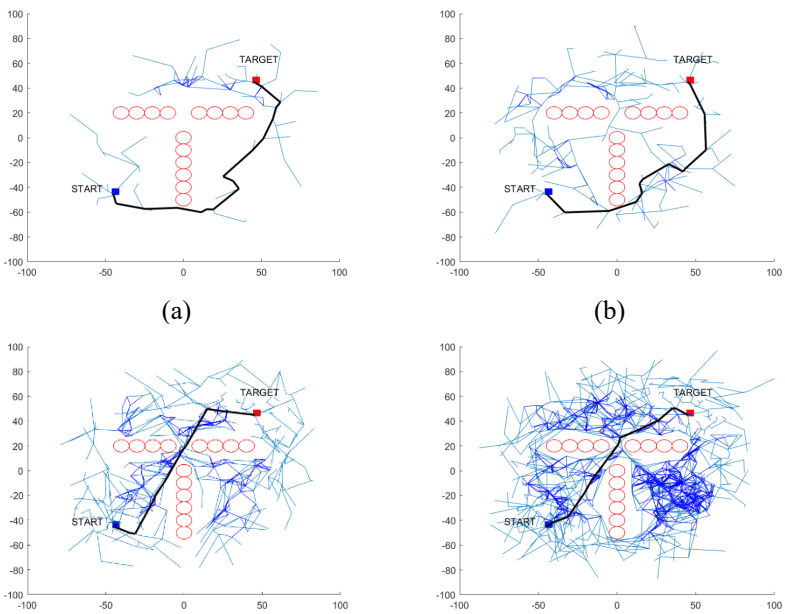
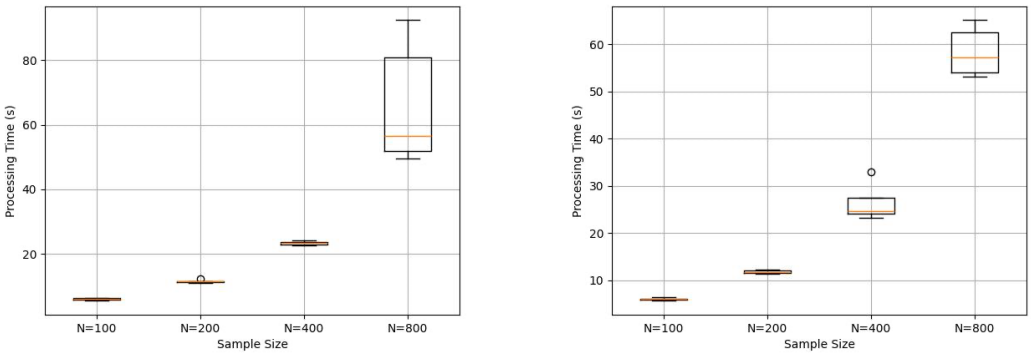


Figure 5. Simulation results for $L_{\max}$ = 30 m: (a) $L_{ob,min}$ = 2, N = 100, and; (b) $L_{ob,min}$ = 2, N = 200; (c) $L_{ob,min}$= 2, N = 400; and (d) $L_{ob,min}$= 2, N = 800.

For the analysis of the success rate of generating an inter-obstacle path, it can be seen from Table 2 that such a path has a larger possibility of success compared to the first simulation set. Figure 6 reveals the processing time of path generation using the proposed method for maximum inter-node distances of 15 meters and 30 meters. For each distance, the evaluation is focused on four sample sizes, i.e., 100, 200, 400, and 800 nodes. It is shown that the processing time at N equals 100 and 200 nodes is highly precise. For N equals 800 nodes in both distances, the precisions are low. This phenomenon is caused by the larger possibility of obtaining feasible paths as N gets larger. The comparison between the Figures. 6(a) and 6(b) conclude that for the same number of nodes, the processing time has no significant difference. Some outliers exist in

N = 200 (Figure 6(a)) and N = 400 (Figure 6(b)). It is because the searching in the A* algorithm may come to some dead nodes (nodes that have no neighbor other than its own parent node).



(a) (b)

Figure 6. Processing time for maximum inter-node distance equals to (a) 15 m; (b) 30 m.

5. Conclusions and Suggestions

A two-sided path planning algorithm, which is a modification of traditional RRT*, is proposed. This algorithm introduces the generation of path alternatives by using different formulas for nodes that are generated from the start-point side and from the target-point side. In addition, a maximum allowable length of inter-node connection is applied. The advantage of this approach lies in the increase in the success rate of route construction. Moreover, the number of maximum nodes can be decreased significantly compared to other methods in previous works. The success rate of the proposed algorithm is 100% for a maximum path length of 15 m and a number of nodes of 400-800, and 80-100% for a maximum path length of 30 m and a number of nodes of 200-800. Evaluating for the passing-through path generation success rate, the maximum success rate is achieved for number of nodes=800 for maximum path length 15 m, and 100% for maximum path length 30 m, and number of nodes = 400 and 800. One notable observation is that increasing the maximum nodes does not significantly reduce the processing time. Therefore, further research is suggested in the future to solve this problem.

6. Acknowledgement

This research is established by the research funding awarded by the Lembaga Penelitian dan Pengabdian pada Masyarakat (LPPM), Universitas Atma Jaya Yogyakarta. A very big gratitude is awarded to the Laboratory of Modeling and Optimization, Department of Industrial Engineering, Universitas Atma Jaya Yogyakarta, Indonesia, for all valuable support.

References

[1] W. Zhu and G. Qiu, “Path Planning of Intelligent Mobile Robots with an Improved RRT Algorithm”, *Applied Sciences*, vol. 15, no. 6, 3370, 2025. DOI: 10.3390/app15063370.

[2] M. Duan, Z. Wang, X. Shao, and G. Ren, “VGA*-RRT*: A Mobile Robot Path Planning Algorithm for Irregular and Complex Maps”, *IEEE Access*, vol. 13, 2025. DOI:10.1109/ACCESS.2025.3552785.

[3] H. Yang, X. Luo, C. Duan, P. Wang, K. Zhu, X. Deng, and W. Ren, “Research on Multi-Objective Point Path Planning for Mobile Inspection Robot Based on Multi-Informed-Rapidly Exploring Random Tree”, *Engineering Applications of Artificial Intelligence*, vol. 151, no. 110645, pp. 1-21, 2025. DOI:10.1016/j.engappai.2025.110645

[4] C. Zhao, Y. Zhu, Y. Du, F. Liao, and C.-Y. Chan, “A Novel Direct Trajectory Planning Approach Based on Generative Adversarial Networks and Rapidly-Exploring Random Tree”, *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 10, pp. 17910-17921, 2022. DOI:10.1109/TITS.2022.3164391.

[5] E. Aydemir and M. Unel, "Motion Planning and Path Following for Autonomous Navigation and Reversing of a Full-Scale Mining Truck and Trailer System", *International Journal of Automotive Technology*, vol. 26, no. 3, pp. 595-606, 2025. DOI:10.1007/s12239-024-00174-9.

[6] B. Z. Türkkol, N. Altuntas, and S. Ç. Yavuz, "A Smooth Global Path Planning Method for Unmanned Surface Vehicles Using A Novel Combination of Rapidly Exploring Random Tree and Bézier Curves", *Sensors*, vol. 24, no. 8145, pp. 1-17, 2024. DOI:10.3390/s24248145.

[7] X. Li, J. Yang, X. Wang, L. Fu, and S.Li, "Adaptive Step RRT*-Based Method for Path Planning of Tea-Picking Robotic Arm", *Sensors*, vol. 24, no. 7759, pp. 1-22, 2024. DOI:10.3390/s24237759.

[8] C. Urrea, P. Sari, J. Kern, and H. Torres, "Enhancing Adaptability and Autonomy in Cooperative Selective Compliance Assembly Robot Arm Robots: Implementation of Coordination and Rapidly Exploring Random Tree Algorithms for Safe and Efficient Manipulation Tasks", *Applied Sciences*, vol. 14, no. 6804, pp. 1-28, 2024. DOI:10.3390/app14156804.

[9] T. Xu, J. Ma, P. Qin, and Q. Hu, "An Improved RRT* Algorithm Based on Adaptive Informed Sample Strategy for Coastal Ship Path Planning", *Ocean Engineering*, vol. 333, no. 121511, pp. 1-17, 2025. DOI:10.1016/j.oceaneng.2025.121511.

[10] H. Zhong, Y. Du., D. Liu, M. Wang, M. Cong, and X. Tian, "A Fruit Fly-Inspired Path Planning Algorithm for Unmanned Aerial Vehicle in Underground Environments Based on Low-Discrepancy Sequences", *Engineering Applications of Artificial Intelligence*, vol. 156, no. 111250, pp. 1-15, 2025. DOI: 10.1016/j.engappai.2025.111250.

[11] X. Xu, P. Li, J. Zhou, and W. Deng, "Path Planning for Quadrupedal Robot Based on Improved RRT-Connect Algorithm", *Sensors*, vol. 25, no. 8., pp. 1-18, 2025. DOI:10.3390/s25082558.

[12] C. Liu, F. Xiao, Y. Ma, H. Chen, Y. Wu, Z. Li, and L. Guo, "An Enhanced RRT* Algorithm with Biased Sampling and Dynamic Stepsize Strategy for Ship Route Path Planning in the High-Risk Areas", *Ocean Engineering*, vol 332, no. 121 466, pp. 1-11, 2025. DOI:10.1016/j.oceaneng.2025.121466.

[13] Z. Liu, M. Li, R. Zhang, G. Zhang, S. Han, and K. Chen, "Preoperative Path Planning of Craniotomy Surgical Robot Based on Improved MDP-LQR-RRT* Algorithm", *Biomedical Signal Processing and Control*, vol. 105, no. 107647, pp. 1-16, 2025. DOI:10.1016/j.bspc.2025.107647.

[14] P. Upadhyay and R. Singh, "Advanced Hybrid Path Planning: Integrated Modified IRRT* with Bezier Curve Enhancements", *Robotic Intelligence and Automation,* vol. 45, no. 3, pp. 434-462, 2025. DOI:10.1108/RIA-12-2024-0285.