SINERGI Vol. 29, No. 2, June 2025: 459-472
http://publikasi.mercubuana.ac.id/index.php/sinergi
http://doi.org/10.22441/sinergi.2025.2.016

Design of path planning robot simulator by applying sampling based method

Heru Suwoyo*, Julpri Andika, Andi Adriansyah
Department of Electrical Engineering, Faculty of Engineering, Universitas Mercu Buana, Indonesia

Abstract
This research aims to create a simulator for solving the global path planning of mobile robots. Various sampling-based methods such as Rapidly-exploring Random Tree (RRT), RRT, and Fast-RRT, along with other derivative algorithms, have been widely used to solve path-planning problems in mobile robots. The level of computational efficiency, path optimality, and the ability to adapt to variant environments are some of the issues that still arise, although these techniques have shown good results in many cases. Although the existing solutions are innovative, comparison between the existing methods is still difficult due to significant differences in convergence speed, implementation complexity, and quality of the resulting paths. This makes choosing the most suitable method for a particular application difficult. The simulator uses sampling-based path planning algorithms such as RRT*, Fast RRT*, RRT*-Smart, informed-RRT*, and Honey Bee Mating Optimization-based Fast-RRT*. With this simulator, users can easily compare the performance of each algorithm and see the characteristics and efficiency of each algorithm in various situations. By running all methods through this simulator, the user can easily compare the methods based on convergence speed and optimality. Therefore, it will effectively help users understand robot navigation, improve the quality of learning, and promote the development of path-planning technology for mobile robots.*

Keywords:
Fast-RRT;
Global Path Planning;
HMBO-Integrated Fast-RRT;*
Mobile Robot;
Simulator;

Article History:
Received: September 30, 2024
Revised: December 1, 2024
Accepted: January 11, 2025
Published: May 15, 2025

Corresponding Author:
Heru Suwoyo
Electrical Engineering Department, Faculty of Engineering, Universitas Mercu Buana, Indonesia
Email:
heru.suwoyo@mercubuana.ac.id

INTRODUCTION

As the need for robotics for various applications, such as manufacturing, environmental exploration, and autonomous transportation, increases, mobile robot navigation becomes increasingly important. Planning the best path for the robot to reach the goal while avoiding obstacles is the main challenge in this navigation. Randomly exploring Random Tree (RRT)[1][2] and its variants (RRT* [3], Fast RRT [4], RRT*-Smart [5], informed-RRT* [6]) have been widely used due to their ability to explore large and complex search spaces. However, each method has its drawbacks, either in terms of computational efficiency, path optimality, or the ability to adapt to changing environments.

Although these methods are very popular, users often have difficulty understanding the differences in characteristics between one algorithm and another and choosing the most appropriate method for a particular situation. Some algorithms may excel in computational speed, while others may be more optimal in the path quality generated. Due to the lack of tools that allow direct comparison between these methods, the learning process and the advancement of robotic navigation technology are increasingly difficult.

Given the need for tools that can accelerate the learning process and assess various path-planning techniques, these simulators are essential. With this simulator, users can test algorithms directly and learn the advantages and disadvantages of each method in an applied and real-time context. In addition, this simulator is equipped with related references, so that users can not only test the performance of each method but also learn the theoretical concepts and mechanisms behind the implementation of the algorithm. These references provide a foundation for better algorithm design. MATLAB was chosen for this simulator because of its ability to visualize data and support various numerical operations quickly and accurately. In addition, its GUI program helps users interact with various path-planning algorithms. This interface allows users to select algorithms, change parameters, and view simulation results visually. Simulation results can be viewed in the form of paths created or in the form of comparisons of performance metrics such as path optimality and computation time. As a result, this simulator not only facilitates algorithm testing but also makes it easier for users to analyze the performance of each method.

MATERIAL AND METHOD

This section explains some of the methods used in the simulator. The method used is sampling-based path planning. Sampling-based methods are path-planning approaches that rely on random sampling to explore the configuration space without requiring a full representation of the space. This method is often used in situations where full mapping of the space would be too difficult or inefficient. This method allows the search for valid paths between known points in the configuration space. This simulator is equipped with references to related methods so that it can provide a comprehensive explanation. Complete and varied references show the development of sampling-based path-planning methods. There are two types of path-planning problems, namely local and global. Global path planning plans a path from the starting point to the destination point by considering the entire environment or available maps, without considering changes in conditions in the field.

Meanwhile, local path planning focuses on path planning based on real-time surrounding conditions, adjusting to changes in the surrounding environment to avoid obstacles or other dynamic conditions. Thus, global path planning is more holistic, while local path planning focuses on adjusting paths in local areas. This simulator is only limited to the use of the global path planning method. This simulator can help researchers compare one with another sampling method. Thus, the development of the method will be very possible. In addition, the presence of this scientific article will further complete the references that can be used in the form of an appropriate review. Before, all the methods are presented, the problem statement of global path planning [7] is explained.

Problem Statements

Let $X \in R^n$ be the representation of state space for a path planning problem, with $n \in N$ as space dimension, thus $X = \{X_{obs}, X_{free}\}$ is state space with $X_{obs} \in X$ refers to obstacle coordinates and $X_{free} \in X$refers to the free space. Moreover, if the start node $x_{init} \in X_{free}$ and goal node $x_{goal} \in X_{free}$are given, then referring to X_{obs}, the path planning algorithm has to find the ideal path from to those nodes, denoted as $\sigma = [0, T] \rightarrow X_{free}$with $\sigma(0) = x_{init}$and $\sigma(T) = x_{goal}$ where $X_{goal} = \{x \in X | x - x_{goal}| < r\}$ for r is the radius around x_{goal}.

Rapidly Exploring Random Tree (RRT)

For robots moving in large and complex spaces, Rapidly Exploring Random Trees (RRT) is one of the most popular path-planning algorithms. Steven M. LaValle first used this algorithm in 1998 to explore the search space quickly and efficiently, especially in environments with many obstacles [2, 8, 9, 10]. After reaching a target or meeting certain conditions, RRT gradually builds a tree by randomly expanding nodes in the search space. Although RRT does not guarantee an ideal path, it is very effective for solving path-planning problems in large and complex environments [11, 12, 13, 14]. The algorithm works by selecting a random point in the search space and then expanding the tree from the nearest node to that point. This method makes RRT very useful for applications where rapid exploration of the search space is required, even though the path quality is not always optimal [15].

As shown in Figure 1, RRT starts its operation by initializing a tree (or trees) with the root node at the starting position. Then, this process is repeated several times until it reaches a set maximum number of iterations. The algorithm takes a random point in the search space at each iteration and searches for the nearest node in the tree. After finding the nearest node, the algorithm extends from that node towards the random point to generate a new node. Before adding a new node to the tree, the algorithm checks whether the path between the

nearest node and the new node is collision-free. If not, the new node is anchored. In addition, the algorithm also checks whether the distance between the new node and the destination position is smaller than a certain threshold. If so, the path from the starting position to the destination can be returned. The algorithm returns to failure status if the path is not found after all iterations. In this way, RRT explores the search space and finds a route that can be used by the navigating robot.

RRT*

RRT* (RRT Star) is an evolution of the Randomly Exploring Random Tree (RRT) algorithm that aims to improve the optimality of the generated paths. While RRT focuses on rapidly exploring the search space, RRT* adds an optimization component by updating the generated paths as the tree grows [11, 16, 17, 18]. In RRT*, the algorithm examines existing nodes to see if a better path can be formed by connecting the new node to other nearby nodes in the tree. In this way, RRT* attempts to improve the quality of the solution and reduce the path length. RRT* also has a convergence guarantee, meaning that it will find a path that is close to the ideal in a complex search space as it iterates. In situations where path optimality is critical, such as in robotic settings that require high efficiency and minimal energy consumption, the application of RRT* is very beneficial.

Referring to Figure 2, the RRT* algorithm starts by initializing a tree (or trees) with the root node at the starting position. This process performs several iterations until it reaches a specified maximum number of iterations. The algorithm takes a random point in the search space at each iteration and searches for the nearest node in the tree from that random point. The algorithm generates a new node by extending from the nearest node to the random point. Then, if the path between the new and nearest nodes does not collide, the new node is inserted into the tree and connected to the nearest node. At this stage, RRT* also functions to optimize the path that has been found by checking the nodes that are within the range of the new node to determine if a better path can be created. If found, the relationships between the nodes are updated to create a more efficient path. In addition, the algorithm checks whether the distance between the new node and the destination position is smaller than a certain threshold. If so, the path from the starting position to the destination can be returned.

Algorithm 1 – RRT

```
RRT(start, goal, max_iterations, step_size)
Initialize tree T with root node at start position
for i = 1 to max_iterations do
    x_rand = SampleRandomPoint()
    x_nearest = NearestNode(T, x_rand)
    x_new = Steer(x_nearest, x_rand, step_size)
    if CollisionFree(x_nearest, x_new) then
        AddNode(T, x_new)
        AddEdge(T, x_nearest, x_new)
        if Distance(x_new, goal) < threshold then
            return PathFromStartToGoal(T, goal)
end for
return failure
```

Figure 1. Pseudocode of RRT

Algorithm 2 – RRT*

```
RRT*(start, goal, max_iterations, step_size)
Initialize tree T with root node at start position
for i = 1 to max_iterations do
    x_rand = SampleRandomPoint()
    x_nearest = NearestNode(T, x_rand)
    x_new = Steer(x_nearest, x_rand, step_size)
    if CollisionFree(x_nearest, x_new) then   n
        AddNode(T, x_new)
        AddEdge(T, x_nearest, x_new)
        for each node in Neighbors(T, x_new) do
            if CollisionFree(node, x_new) then
                new_cost = Cost(node) + Distance(node, x_new)
                if new_cost < Cost(x_new) then
                    UpdateParent(x_new, node)
        end for
    if Distance(x_new, goal) < threshold then
        return PathFromStartToGoal(T, goal)
end for
return failure
```

Figure 2. Pseudocode RRT*

The algorithm will return to failure status if the path is not found after all iterations. These steps allow RRT* to find valid paths and optimize them to improve navigation efficiency.

Fast-RRT

To improve the speed and efficiency of path planning, Fast RRT is a variant of the Rapidly exploring Random Tree (RRT) algorithm that focuses on reducing the computation time required to find a path while maintaining effective space exploration capabilities. Fast RRT implements a more efficient sampling strategy and reduces the complexity of the path-finding process [4, 7, 10]. Biased sampling is a way that the Fast RRT algorithm more often selects random points around the target area or previously discovered paths. In this way, the algorithm can quickly direct

the growth of the tree towards the desired goal. In addition, Fast RRT also implements constraint sampling, which means that the algorithm considers physical constraints and environmental obstacles when sampling. Keeping the path within the allowed limits helps prevent contention and improves the quality of the generated paths.

As seen in Figure 3, Fast RRT has two stages, namely improved RRT and fast optimal, both of which can be seen in Figure 4 and Figure 7. Referring to Figure 4, Fast Optimal consists of two methods, namely Fast Sampling and RandomSteer, which can be seen in Figure 5 and Figure 6. Meanwhile, for RandomSteer it can be seen in Figure 6.

```
Algorithm 3 – Fast RRT(start, goal, max_iterations, step_size, Map)
for i = 1 to max_iterations do
    Tinit = ImprovedRRT(start, goal, max_iterations, step_size, Map)
    if Tinit != empty then
        Toptimal = FastOptimal(Toptimal, Tinit)
```

Figure 3. Pseudocode of Fast RRT

```
Algorithm 4 – ImprovedRRT (start, goal, max_iterations, step_size, Map)
Initialize tree T with root node at start position
for i = 1 to max_iterations do
    x_rand = FastSample(start,goal,Map)
    x_nearest = NearestNode(T, x_rand)
    x_new = RandomSteer(x_nearest, x_rand, step_size)
    if CollisionFree(x_nearest, x_new) then
        AddNode(T, x_new)
        AddEdge(T, x_nearest, x_new)
        if Distance(x_new, goal) < threshold then
            return PathFromStartToGoal(T, goal)
end for
return failure
```

Figure 4. Pseudocode of Improve RRT

```
Algorithm 5 – FastSample(start,goal,Map)
x_rand = UniformSample(start,goal,Map)
while x_rand ∈ Xexplored do
    x_rand = UniformSample(start,goal,Map)
```

Figure 5. Pseudocode of FastSample

```
Algorithm 6 – RandomSteer(x_nearest, x_rand, step_size)
if CollisionFree(x_nearest, x_new) then
    return x_new
else
    θ = rand()
    x_new = Expand(x_new,θ)
```

Figure 6. Pseudocode of RandomSteer

```
Algorithm 7 – FastOptimal(Toptimal, Tinit)
for each point_A of Tinit do
    for each point_B of Toptimal do
        if d = ||point_A, point_B|| < threshold do
            intersection append (point_A.index(), point_B.index())
for k=1:size(joints) - 1 do
    (start_A, start_B)=intersection(k)
    (end_A, endt_B)=intersection(k)
    Toptimal=subpath(Toptimal, start_A, start_B)
    Tinit=subpath(Tinit, start_A, start_B)
    if cost(Toptimal)<cost (Tinit)
        Toptimal[end_A, end_B]=Tinit[start_B, end_B]
Toptimal=OptimizePath(Toptimal,intersection, Map)
```

Figure 7. Pseudocode of FastOptimal

```
Algorithm 8 – OptimizePath(Toptimal,intersection, Map)
Path=[ ]
for k=1:size(intersection)-1 do
    idx1=near_list[k]
    point_1=Toptimal(idx1)
    idx2=near_list[k+1]
    point_2=Toptimal(idx2)
    if CollisionFree(point_1, point_2) then
        subpath=generate(point_1, point_2)
    else
        subpath=P(idx1:idx2)
        Path append subpath
```

Figure 8. Pseudocode of OptimizePath

Unlike other methods, Fast RRT utilizes fusion and path optimization strategies to improve the initial path solution. It can be seen in Figure 7. Where OptimizePath is also used in all methods used in this simulator as seen in Figure 8.

RRT*-Smart

RRT-Smart* is a variant of the RRT* algorithm that aims to improve path quality and efficiency by incorporating more intelligent data processing techniques. The working process of RRT*-Smart is the same as standard RRT*: the tree is initialized with the root node at the starting position. The algorithm searches for the nearest node in the tree from a random point in the search space at each iteration. However, RRT*-Smart performs further optimization by using a more directed sampling approach, which considers random points and directs the sampling process to regions that are more relevant to the final goal. The algorithm creates new nodes by extending to random points after finding the nearest node. Then, it checks the paths between the new and the nearest nodes to ensure that the paths do not collide. RRT*-Smart not only adds new nodes to the tree but also evaluates other nodes within the range of the new node to determine the most ideal path. This process increases productivity because the time required to find better quality paths is reduced [1, 13, 19]. One of the main advantages of RRT*-Smart is the ability to generate smoother and more ideal paths faster than traditional RRT*. With the combination of intelligent sampling and path optimization, RRT*-Smart becomes a better choice for robotics applications that require fast and efficient navigation in obstacle-filled spaces. The algorithm can better adapt to complex environments and reduce overall computation time thanks to this smarter processing technology. The Pseudocode of RRT*-Smart can be seen in Figure 9.

Informed-RRT*

Informed-RRT* is a modified RRT* algorithm. Its goal is to improve the efficiency of pathfinding by using additional information about

the search space. The algorithm starts by initializing a tree (or trees) with the root node at the initial position. Then, based on the initial position and the goal, a bounding box is determined. The iteration process continues until certain conditions are met, such as reaching a maximum number of iterations or a set time. The algorithm uses the SampleWithinBoundingBox function to pick a random point from within the bounding box at each iteration. This makes sampling more efficient by limiting the search area. Next, the algorithm identifies the nearest node x_nearest in the tree and extends from that node towards the random point to generate a new node x_new. Then, the algorithm evaluates whether the path from the nearest node to the new node is not interrupted by collisions. The new node is embedded into the tree and connected to the nearest node if the path is not broken. Its ability to perform path optimization through rewiring is one of the advantages of Informed-RRT* [6, 11, 18, 20].

The algorithm will examine neighboring nodes near the new node to find a more optimal path. If a better path is found, the connections between the nodes will be updated to produce a more efficient path. In addition, the algorithm checks whether the distance between the new node and the goal position is smaller than a certain threshold. If it is true, the path from the starting position to the goal can be recovered. The algorithm will return a failure status if the path is not found after all iterations.

```
Algorithm 9 – RRT*-Smart (start, goal, max_iterations, step_size, Map)
Initialize tree T with root node at start position
for i = 1 to max_iterations do
  if i=n+b, n+2b, n+3b... then
    x_rand=SmartSample(start,goal,Map, x_beacon)
  else
    x_rand = UniformSample(start,goal,Map)
    x_nearest = NearestNode(T, x_rand)
    x_new = Steer(x_nearest, x_rand, step_size)
    if CollisionFree(x_nearest, x_new) then
        AddNode(T, x_new)
        AddEdge(T, x_nearest, x_new)
        for each node in Neighbors(T, x_new) do
            if CollisionFree(node, x_new) then
                new_cost = Cost(node) + Distance(node, x_new)
            if new_cost < Cost(x_new) then
                UpdateParent(x_new, node)
        end for
    if Distance(x_new, goal) < threshold then
      initialPathFoud=true;
      return T
    if initialPathFoud then
      n=i
      Tprev=T
      T=OptimizePath(T,start,goal)
      if cost(T)<cost(Tprev) then
        x_beacon=allPoint(T)
return failure
```

Figure 9. Pseudocode RRT*-Smart

```
Algorithm 10 – informed-RRT* (start, goal, max_iterations, step_size, Map)
Initialize tree T with root node at start position
for i = 1 to max_iterations do
    cbest =min_xsoln ∈ Xsoln {Cost (xsoln)}
    x_rand = InformedSample(start, goal, Map, cbest)
    x_nearest = NearestNode(T, x_rand)
    x_new = Steer(x_nearest, x_rand, step_size)
    if CollisionFree(x_nearest, x_new) then
        AddNode(T, x_new)
        AddEdge(T, x_nearest, x_new)
        for each node in Neighbors(T, x_new) do
            if CollisionFree(node, x_new) then
                new_cost = Cost(node) + Distance(node, x_new)
            if new_cost < Cost(x_new) then
                UpdateParent(x_new, node)
        end for
    if Distance(x_new, goal) < threshold then
      initialPathFoud=true;
      return Xsoln
```

Figure 10. Pseudocode of informed-RRT*

```
Algorithm 11 – informedSample (start, goal, Map, cbest)
if cbest < ∞ then
  cmin=Distance(start, goal)
  xcentre=(start+goal)/2
  C=RotationToWorldFrame(start,goal)
  r1=cbest/2
  {r_i}_{i=2,...,n} = (√(cbest² − cmin²))/2
  L=diag{r_1,2,...,r_n}
  x_ball= SampleUnitNBall
  x_rand= (CLx_ball+xcentre)∩X
else
  x_rand=∪ (X)
return x_rand
```

Figure 11. Pseudocode of informed Sample

These steps allow Informed-RRT* to find valid paths more efficiently and optimally; this makes it very useful for applications that require path planning in complex spaces. Informed-RRT* has a different way to do sampling when the initial path is found. It can be represented visually in Figure 10 and Figure 11.

HBMO-Integrated Fast-RRT*

HBMO-integrated Fast-RRT* [21, 22, 23, 24, 25] utilizes fast sampling and random steering in Fast RRT. In addition, there is a rewiring process that supports this process not only focusing on the speed of finding the initial path but also better optimality on the initial path found. As explained in Fast-RRT, by utilizing fast sampling which applies the concept of constraint sampling, exploration is not repeated in areas that have been touched. In addition, random steering which shows the work of bias sampling also allows the method to be relevant in solving problems in a narrow environment. Furthermore, by not using the fast optimal in Fast-RRT, optimization is carried out by applying batch beacon-based optimization utilizing the Honey Bee Mating Optimization procedure. From the initial path obtained, all waypoints are called beacons, a term commonly used in RRT*-Smart. From several

beacons, they will be selected randomly to then be repositioned with a focus on reducing path costs. This reduction involves HBMO by assuming that candidate solutions are nodes located around the selected beacon. The fitness function in this optimization is not only the cost path but also the feasibility of the path (freedom from obstacles). This approach replaces the fusion process that previously existed in Fast-RRT. The main reason is that fusion involves two different paths, which rarely obtain new paths that vary by simply adding samples. Another reason is that the process of merging paths takes a long time. Therefore, this is contrary to the initial motivation. Fast sampling, is done to increase the convergence rate of Fast RRT. It can be represented visually in Figure 12.

SIMULATOR DESIGN

The process of designing this path-planning simulator is carried out systematically and consists of several stages. The purpose of this process is to produce a tool that can help people learn and experiment with various path-planning algorithms. The process of developing this simulator is systematically structured as follows:

Algorithm 12 – HBMO-FastRRT*(start, goal, max_iterations, step_size)
```
Initialize tree T with root node at start position
for i = 1 to max_iterations do
    x_rand = FastSample( )
    x_nearest = NearestNode(T, x_rand)
    x_new = Steer(x_nearest, x_rand, step_size)
    if CollisionFree(x_nearest, x_new) then
      AddNode(T, x_new)
      AddEdge(T, x_nearest, x_new)
      for each node in Neighbors(T, x_new) do
          if CollisionFree(node, x_new) then
            new_cost = Cost(node) + Distance(node, x_new)
            if new_cost < Cost(x_new) then
              UpdateParent(x_new, node)
      end for
    if Distance(x_new, goal) < threshold then
      T=getPath(T)
      T=OptimizePath(T)
      T=HBMO(T)
      return T
end for
return failure
```

Figure 12. Pseudocode of HBMO-Integrated Fast-RRT*

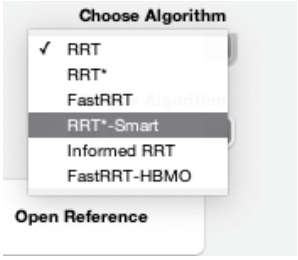


Figure 13. Select the available Algorithms from the popup menu in the Simulator

Determination of Objectives and Scope

The main purpose of the simulator is to provide an interactive tool that allows users to compare and understand various sampling-based path planning algorithms such as RRT, RRT*, Fast RRT, RRT*-Smart, informed-RRT*, and HBMO-based Fast-RRT*. In scenarios involving environments with obstacles, the scope of the simulator is designed to support robotic navigation.

Architecture and User Interface (GUI) Design

MATLAB is used to design the Architecture and User Interface (GUI) simulator. The GUI provides an easy-to-use interface where the user can select an algorithm to be tested, enter parameters such as starting position and destination, and view the resulting path visually. The user interface components should include:
In this simulator, the user can directly select the method they want to operate based on the popup menu provided in Figure 13.

Parameters such as the number of iterations, step size, and minimum distance to the destination can be set. For the number of iterations, the user can directly fill in the number of samples on the number of nodes. The step size has been determined in it, which requires the user to edit the master code if the setting is needed. While for the destination, it can be set by determining the x and y coordinates at the goal point in Figure 14. Before running, the determination of the starting point is also provided with the same setting steps.

In a simple way, users can determine the test environment used for testing. There are 21 environments modeled with varying levels of difficulty in Figure 15. Users only need to select one of the environments on the Choose Map popup menu.

Visualization of the path generated by the algorithm in the form of a two-dimensional graph can be generated by clicking run and the text box will show the cost of the path after this operation is complete. While at the bottom of the graph is written the number of nodes needed, in finding the initial path.

The Open Reference button is provided to ease the user seeing the related reference by only clicking it. The Reset button is also given, to cancel any setting is needed.

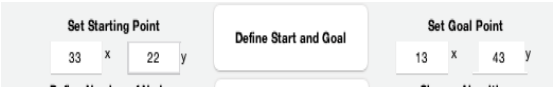


Figure 14. Set Start and Goal Point

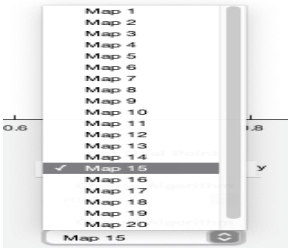


Figure 15. Select the Map Used for Simulation

Figure 16. Simulator Appearance

Implementation of Some Algorithms

Path Planning Algorithm Implementation MATLAB implements selected path planning algorithms, such as RRT, RRT*, Fast RRT, RRT*-Smart, informed RRT*, and HBMO-integrated Fast-RRT*. Each algorithm is written in the form of a function that can process GUI input and produce an optimal path with given parameters. All algorithms are implemented modularly so that it is easy for users if there is a desire for modification, and integration with other algorithms, or even its development. Some important features/modules contained in the method function include:

1. Sampling Method (different methods have their concept of sampling and exploration way)
2. Detection of collisions in the test environment
3. Rewiring process (in several algorithms such as RRT*, RRT*-Smart, informed-RRT*, and HBMO-integrated Fast-RRT*)
4. Sampling bias method and sampling area limitation in Fast-RRT and HBMO-integrated Fast-RRT*.

The ability to compare path planning algorithms is a key feature of the simulator. In addition to having a visual display, the quality of the path can also be viewed and compared based on the path costs available in the text box. Users can also record any changes that occur when operating a particular algorithm and perform analysis based on the knowledge built through theoretical learning through references.

Simulator Testing and Validation

After the implementation is complete, the simulator is tested to ensure that each algorithm runs as expected. To test the algorithm's ability to find the optimal path, trials are carried out with various environmental scenarios, ranging from simple to complex, as a sample represented in Figure 16. There are 21 different synthesis environments with different levels of difficulty in this simulator. By testing all these environments, users can get a variety of comparisons as a basis for further development. In addition, the testing process ensures an easy-to-understand user interface and a clear display of comparison results between algorithms.

Reference Integration

The simulator has references related to the algorithms used to help users understand them. This documentation includes a brief description of how the algorithm works, suitable applications, and its advantages and disadvantages. This documentation can be accessed directly from the GUI, making it easier for users to understand the method in more depth.

Code and performance optimization

The final stage of code and performance optimization involves improving the performance of the simulator, especially in terms of execution speed and memory usage. This is important to ensure that the simulator is real-time and responsive because path-planning algorithms often require a lot of computation time. Some steps in this optimization include reducing the computational complexity of certain algorithms and using memo management techniques.

RESULT and DISCUSSION

Regarding the result shown in Figure 17, through the simulator, the user can conclude that in this first environment, HBMO-Fast RRT* has the best performance. Not only is it fast, but the resulting path is also the most optimal. To ease observation, the user can observe the value below the visual result that shows the number of nodes needed to obtain the initial path and observe the path cost in the text box to observe the optimality of the path. To facilitate the analysis and comparison, Table 1 is provided.

Table 1. Comparative Results of Cost Path and Required Nodes to Find Initial Path

Method	Path Cost	Number of Sampling
RRT	79.42	115
RRT*	79.42	124
Fast RRT	76.93	106
RRT*-Smart	73.49	121
Informed-RRT*	72.43	127
HBMO-Fast RRT*	70.83	102

In the first test, all algorithms are operated to solve the path planning problem in test environment 1. With a low level of difficulty. In this test, the start and goal points are placed at (10,40) and (70,40), respectively. The test is carried out by limiting the number of samples to only 1000 repetitions. The results of this test can be seen in Figure 17.

In a simple case like this, the characteristics of the exploration process of each method are difficult to compare. This is shown in the same number of samplings between one method and another. Therefore, users can use other more complex environments such as environment 18, environment 19, environment 20, and environment 16 as depicted in Figure 18.

It can be seen in Figure 20 that the RRT*-Smart method focuses on generating more dominant random nodes around the beacon. This is as mentioned in the reference, with the aim of optimizing the rewiring process in line with reducing the cost path.

Meanwhile, referring to Figure 18 (b), informed-RRT* can effectively improve the optimality of the path by centering the distribution of nodes with elliptical restrictions, which are determined based on the initial cost path information, and direct cost from start to goal. Thus, not only theoretically, but users can also observe the optimality directly and prove that the underlying theory is correct. As a note, the results in the form of a straight path given by the two tested methods (Figure 19) are the result of the path optimization that applies triangular inequality. In this simulator, all methods have been equipped with the ability to shorten the final path by applying the method.

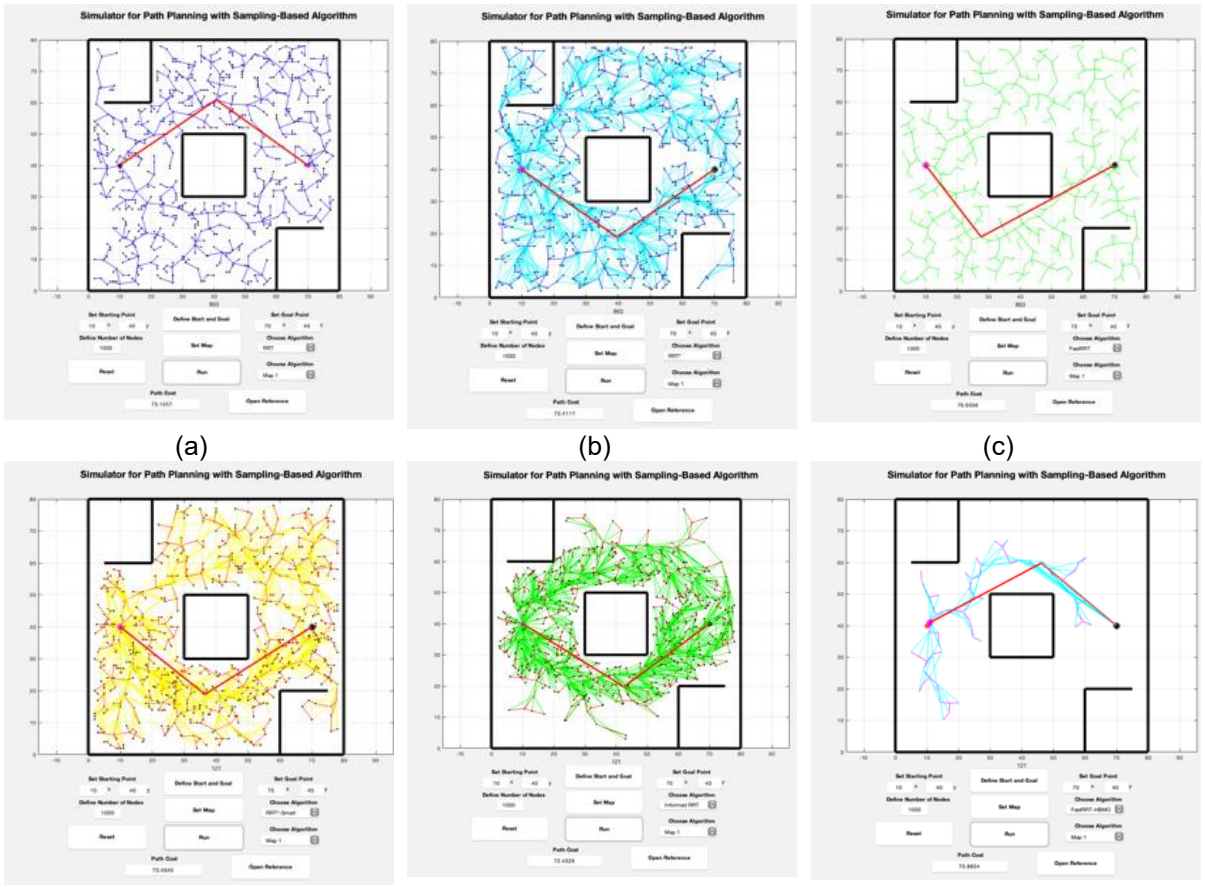
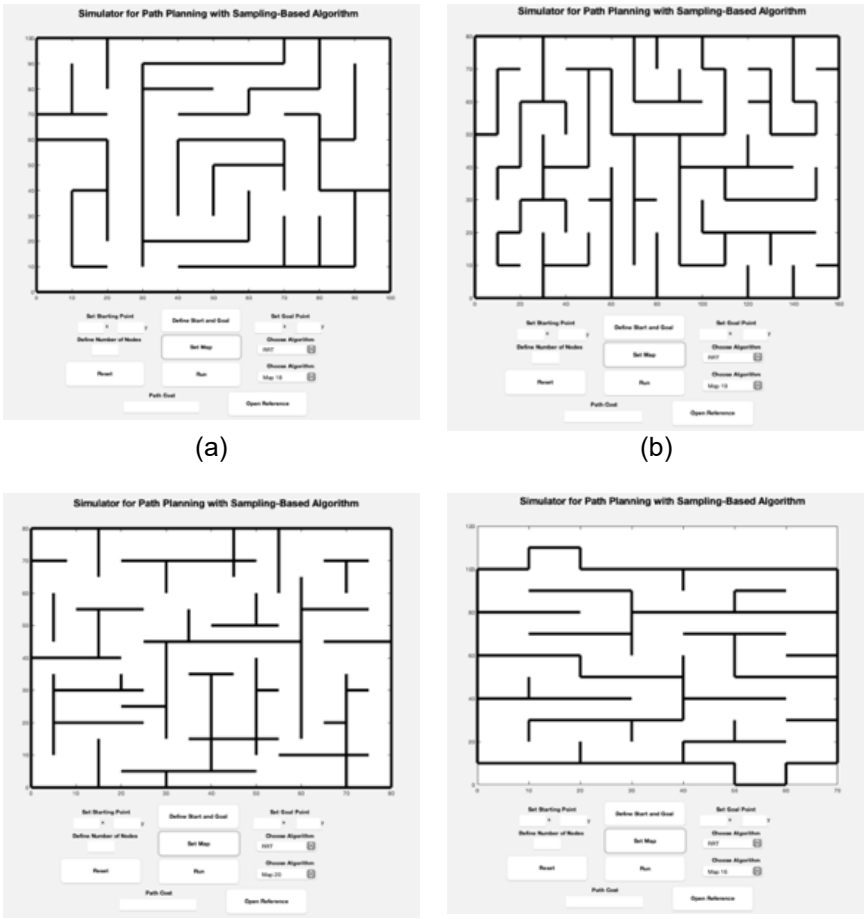


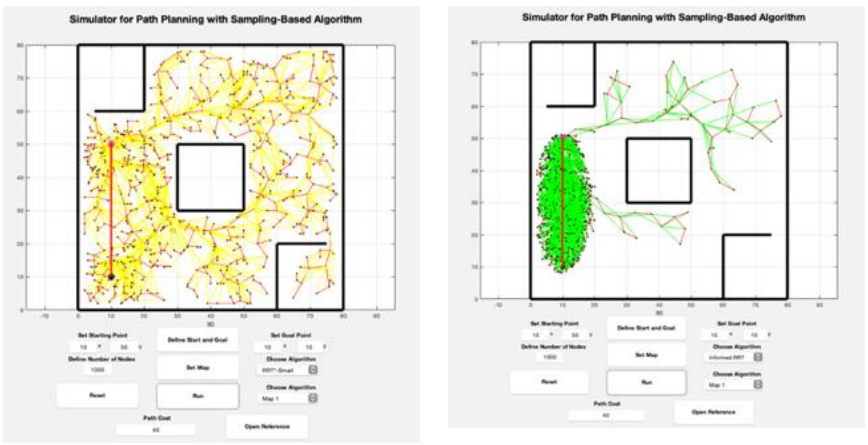
Figure 17. Simulator testing for all algorithms on test environment 1 (a) RRT (b) RRT* (c) Fast-RRT (d) RRT*-Smart (e) Informed-RRT* (f) HBMO-Fast-RRT*



(a) (b)

(c) (d)

Figure 18. Test Environment (a) environment 18, (b) environment 19, (c) environment 20, (d) environment 16



(a) (b)

Figure 19. Comparison of Path Optimization Technique (a) RRT*-Smart (b) Informed-RRT* and HBMO-Integrated Fast-RRT*

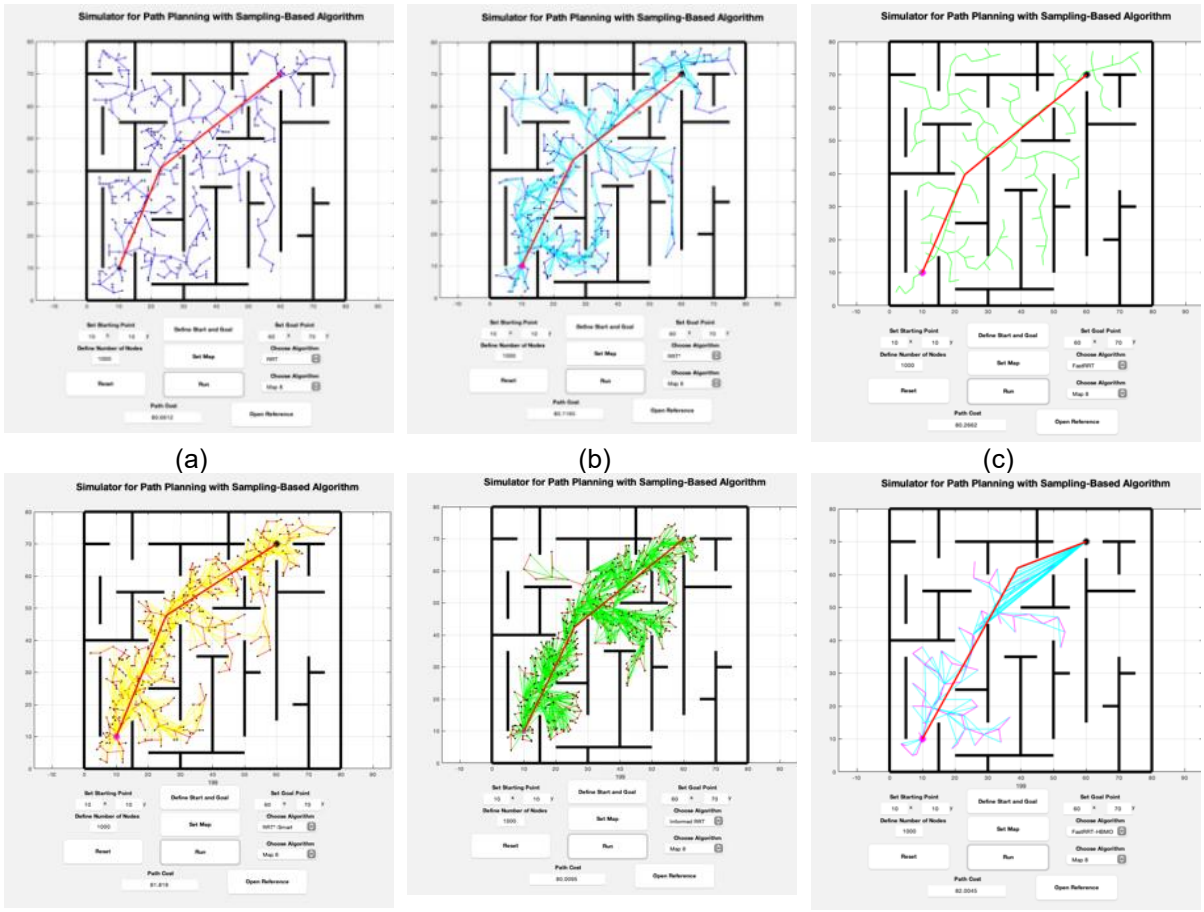


Figure 20. Performance of each method in solving complex problems (a) RRT (b) RRT* (c) Fast RRT (d) RRT*-Smart (e) informed-RRT* (f) HBMO-Integrated Fast-RRT*

Furthermore, to be able to determine the performance of each method in solving more complex problems, simulations were carried out in test environment 8. In this test, the start and goal points were positioned at (10,10) and (70,50), respectively. With a sampling restriction of 2000, the results are visually shown in Figure 19. As seen in Figure 19, with a limited number of nodes and a wider exploration scope, only the method with the expansion method can effectively solve the problem. The effectiveness is seen from the speed of finding the initial path, and also the path cost on the optimization path. To present the detailed results of this experiment, Table 2 is given.

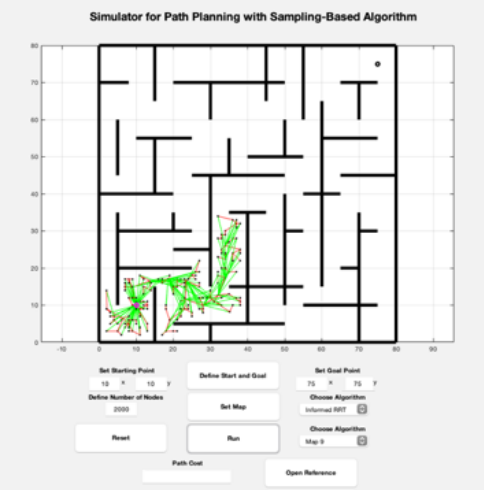
By observing Table 2, the fast ability to provide the initial path greatly determines the path optimization process. A method with good speed in determining the initial path will have a great chance of optimizing the optimization process. As shown in Figure 20, RRT*-Smart is not better than informed-RRT* while the optimization method and concept in it show very good performance referring to Figure 19.

Table 2. Comparison of Cost Path and Required Nodes to Find Initial Path

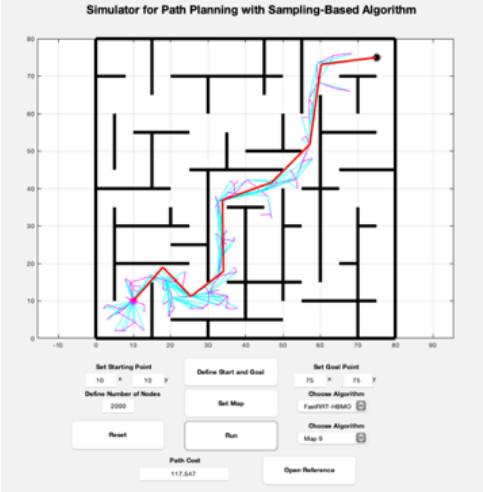
Method	Path Cost	Number of Sampling
RRT	80.6443	345
RRT*	80.1165	319
Fast RRT	80.2662	209
RRT*-Smart	81.8190	321
Informed-RRT*	80.0095	223
HBMO-Fast RRT*	80.0045	205

This is not because of the uncertainty of the optimization method but rather the different optimization time adequacy. In RRT*-Smart, the duration of the initial path is longer than that of informed-RRT*. So the effectiveness of optimization, the adequacy of time, is very much determined by the method in the algorithm.

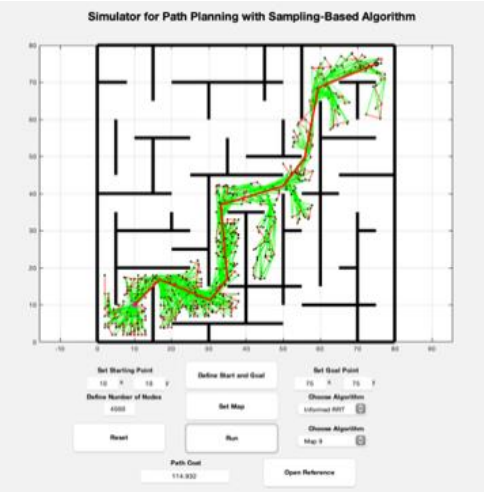
Furthermore, in informed-RRT* and RRT*-Smart, the path expansion method is carried out by maintaining the method in RRT* which tends to be slow, while in HBMO-Integrated Fast-RRT* there is fast sampling (as proven in Figure 20) because it adopts a method of determining the initial path that is almost the same as Fast RRT.


Simulator for Path Planning with Sampling-Based Algorithm
Set Starting Point
Define Start and Goal
Set Goal Point
Define Number of Nodes
Set Map
Choose Algorithm
Reset
Run
Choose Algorithm
Path Cost
Open Reference

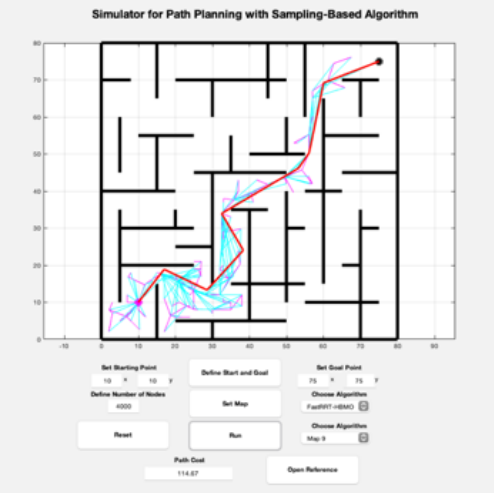
(a)


Simulator for Path Planning with Sampling-Based Algorithm
Set Starting Point
Define Start and Goal
Set Goal Point
Define Number of Nodes
Set Map
Choose Algorithm
Reset
Run
Choose Algorithm
Path Cost
Open Reference

(b)


Simulator for Path Planning with Sampling-Based Algorithm
Set Starting Point
Define Start and Goal
Set Goal Point
Define Number of Nodes
Set Map
Choose Algorithm
Reset
Run
Choose Algorithm
Path Cost
Open Reference

(c)



(d)

Figure 21. Comparison of Informed-RRT* and HBMO-Integrated Fast-RRT* for Solving Path Planning in Map 9

So that it is possible for HBMO-integrated Fast-RRT* to have sufficient time to perform optimization. This statement can be strengthened by the test results in environment 9 for the informed-RRT* and HBMO-integrated Fast-RRT* methods with the number of allowed samplings limited to only 2000 values, and the start and goal points spanning positions (10,10) and (75,75), which are presented in Figure 21. Then, with the addition of the number of samplings by 4000, informed-RRT* can effectively produce paths with relatively the same path cost as HBMO-integrated Fast-RRT*, namely 114.7 and 114.9, respectively.

By comparing Figure 20 (b) and Figure 20 (d), with a limited number of samplings, HBMO-Integrated Fast-RRT* can still perform its task well even though it is not more optimal if the number of allowed samplings is large, such as 4000 (in this case).

CONCLUSION

The developed path planning simulator enables the understanding and analysis of various sampling-based path planning algorithms, such as RRT, RRT*, Fast RRT, and other variations. It helps in determining the advantages and disadvantages of each method in a realistic and applicable context by providing a platform that allows users to directly test and compare the characteristics of each algorithm. Furthermore, thanks to the integration of in-depth references and explanations, the simulator enhances the user's understanding of the concepts and how each algorithm works. The simulator is highly relevant for education and research in the field of robotics and automation systems because it can accelerate the learning process and provide a more accurate evaluation of path-planning techniques. As a result, the simulator serves not only as a learning aid but also as a useful tool for developers and researchers to create more effective navigation solutions.

ACKNOWLEDGMENT

This research was funded by the Directorate of Research, Technology, and Community Service, Directorate General of Higher Education, Research and Technology, Ministry of Education, Culture, Research, and Technology, in the Fundamental-Regular Research Scheme, 2024, and supported by Universitas Mercu Buana.

REFERENCES

[1] Y. Huang and C. Jin, "Path Planning Based on Improved RRT Algorithm," *2023 2nd International Symposium on Control Engineering and Robotics (ISCER)*, Hangzhou, China, 2023, pp. 136-140, doi: 10.1109/ISCER58777.2023.00030.

[2] S. Ganesan, B. Ramalingam, and R. E. Mohan, “A hybrid sampling-based RRT* path planning algorithm for autonomous mobile robot navigation,” *Expert Systems with Applications*, vol. 258, pp. 125206–125206, Aug. 2024, doi: 10.1016/j.eswa.2024.125206.

[3] Z. Chen, X. Zhang, L. Wang and Y. Xia, "A Fast Path Planning Method Based on RRT Star Algorithm," *2023 3rd International Conference on Consumer Electronics and Computer Engineering (ICCECE)*, Guangzhou, China, 2023, pp. 258-262, doi: 10.1109/ICCECE58074.2023.10135365.

[4] Z. Wu, Z. Meng, W. Zhao, and Z. Wu, “Fast-

RRT: A RRT-based optimal path finding method," *Applied Sciences*, vol. 11, no. 24, 2021, doi: 10.3390/app112411777.

[5] S. Li and C. Zhu, "Improved RRT*-Smart Algorithm for UGV Path Planning in Emergency Rescue Scenarios," *2024 7th International Conference on Robotics, Control and Automation Engineering (RCAE)*, Wuhu, China, 2024, pp. 212-218, doi: 10.1109/RCAE62637.2024.10833983.

[6] Y. Zhao, Y. Liu, H. Gao and S. Yan, "Research on Informed-RRT* with Improved Initial Solution," *2022 4th International Conference on Intelligent Control, Measurement and Signal Processing (ICMSP)*, Hangzhou, China, 2022, pp. 977-981, doi: 10.1109/ICMSP55950.2022.9859229.

[7] H. Suwoyo, A. Adriansyah, J. Andika, A. Ubaidillah, and M. F. Zakaria, "An Integrated RRT*SMART-A* Algorithm for solving the Global Path Planning Problem in a Static Environment," *IIUM Engineering Journal*, vol. 24, no. 1, pp. 269–284, Jan. 2023, doi: 10.31436/iiumej.v24i1.2529.

[8] D. K. Muhsen, F. A. Raheem, and A. T. Sadiq, "A Systematic Review of Rapidly Exploring Random Tree RRT Algorithm for Single and Multiple Robots," *Cybernetics and Information Technologies*, vol. 24, no. 3, pp. 78–101, Sep. 2024, doi: 10.2478/cait-2024-0026.

[9] D. Muriyatmoko, A. Djunaidy, and A. Muklason, "Heuristics and Metaheuristics for Solving Capacitated Vehicle Routing Problem: An Algorithm Comparison," *Procedia Computer Science*, vol. 234, pp. 494–501, 2024, doi: 10.1016/j.procs.2024.03.032.

[10] F. Martinez, E. Jacinto, and H. Montiel, "Rapidly Exploring Random Trees for Autonomous Navigation in Observable and Uncertain Environments," *International Journal of Advanced Computer Science and Applications*, vol. 14, no. 3, Jan. 2023, doi: 10.14569/ijacsa.2023.0140399.

[11] W. Wu, C. Kong, Z. Xiao, Q. Huang, M. Yu, and Z. Ren, "Multi-Indicator Heuristic Evaluation-Based Rapidly Exploring Random Tree Algorithm for Robot Path Planning in Complex Environments," *Machines*, vol. 13, no. 4, p. 274, Mar. 2025, doi: 10.3390/machines13040274.

[12] Y. Li, W. Wei, Y. Gao, D. Wang, and Z. Fan, "PQ-RRT*: An improved path planning algorithm for mobile robots," *Expert Systems with Applications*, vol. 152, pp. 113425–113425, Apr. 2020, doi: 10.1016/j.eswa.2020.113425.

[13] B. Liao, F. Wan, Y. Hua, R. Ma, S. Zhu, and X. Qing, "F-RRT*: An improved path planning algorithm with improved initial solution and convergence rate," *Expert Systems with Applications*, vol. 184, pp. 115457–115457, Jun. 2021, doi: 10.1016/j.eswa.2021.115457.

[14] Z. Ma and J. Chen, "Adaptive path planning method for UAVs in complex environments," *International Journal of Applied Earth Observation and Geoinformation*, vol. 115, p. 103133, Dec. 2022, doi: 10.1016/j.jag.2022.103133.

[15] M. i. Hossain, M. -U. Alam, M. K. Rahat, M. A. Rahman, A. Shufian and M. S. R. Zishan, "Autonomous Parking Valet System: Path Planning and Control in Complex Environments," *2025 2nd International Conference on Advanced Innovations in Smart Cities (ICAISC)*, Jeddah, Saudi Arabia, 2025, pp. 1-6, doi: 10.1109/ICAISC64594.2025.10959602.

[16] J. Tian, T. Chao, M. Yang, J. Zhu and S. Wang, "A path planning algorithm based on improved RRT* for UAVs," *2022 IEEE International Conference on Unmanned Systems (ICUS)*, Guangzhou, China, 2022, pp. 1-6, doi: 10.1109/ICUS55513.2022.9986963.

[17] Z. Yu and L. Xiang, "NPQ-RRT $*$: An Improved RRT $*$ Approach to Hybrid Path Planning," *Complexity*, vol. 2021, no. 1, Jan. 2021, doi: 10.1155/2021/6633878.

[18] C. Li, C. Wang, J. Wang, Y. Shen and M. Q. . -H. Meng, "Sliding-Window Informed RRT*: A Method for Speeding Up the Optimization and Path Smoothing," *2021 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, Xining, China, 2021, pp. 141-146, doi: 10.1109/RCAR52367.2021.9517672

[19] H. Tao, Z. Yi and Z. Xiang, "Research On Path Planning of Mobile Robot Based On Improved RRT* Algorithm," *2022 IEEE 6th Information Technology and Mechatronics Engineering Conference (ITOEC)*, Chongqing, China, 2022, pp. 666-670, doi: 10.1109/ITOEC53115.2022.9734505.

[20] D. Wu, L. Wei, G. Wang, L. Tian, and G. Dai, "APF-IRRT*: An Improved Informed Rapidly-Exploring Random Trees-Star Algorithm by Introducing Artificial Potential Field Method for Mobile Robot Path Planning," *Applied Sciences*, vol. 12, no. 21, p. 10905, Oct.

2022, doi: 10.3390/app122110905.
[21] H. Suwoyo, Y. Tian, A. Adriansyah, and J. Andika, “A HBMO-based batch beacon adjustment for improving the Fast-RRT,” *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 38, no. 1, pp. 107–107, Jan. 2025, doi: 10.11591/ijeecs.v38.i1.pp107-119.
[22] W. Wang, H. Gao, Q. Yi, K. Zheng and T. Gu, "An Improved RRT* Path Planning Algorithm for Service Robot," *2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, Chongqing, China, 2020, pp. 1824-1828, doi: 10.1109/ITNEC48623.2020.9085226.
[23] Q. Li, J. Wang, H. Li, B. Wang, and C. Feng, “Fast-RRT*: An Improved Motion Planner for Mobile Robot in Two-Dimensional Space,” *IEEJ Transactions on Electrical and Electronic Engineering*, vol. 17, no. 2, pp. 200–208, Oct. 2021, doi: 10.1002/tee.23502.
[24] Z. Iklima and T. M. Kadarina, “Distributed Path Planning Classification with Web-based 3D Visualization using Deep Neural Network for Internet of Robotic Things," *Journal of Science and Technology*, vol. 13, no. 2, pp. 47–53, Dec. 2021.
[25] H. Suwoyo, A. Burhanudin, Y. Tian, and J. Andika, “Problem solving path planning and path tracking in a 3 DOF hexapod robot using the RRT* algorithm with path optimization and Pose-to-Pose,” *SINERGI*, vol. 28, no. 2, p. 265, Apr. 2024, doi: 10.22441/sinergi.2024.2.007