IJE TRANSACTIONS A: Basics Vol. 36 No. 04, (April 2023) 797-806

International Journal of Engineering

Journal Homepage: www.ije.ir

Efficient Sampling-based for Mobile Robot Path Planning in a Dynamic Environment Based on the Rapidly-exploring Random Tree and a Rule-template Sets

M. A. R. Pohan*, J. Utama

Electrical Engineering Department, Universitas Komputer Indonesia, Jl. Dipatiukur 102-116, Bandung 40132, Indonesia

P A P E R I N F O

Paper history:
Received 12 December 2022
Received in revised form 30 December 2022
Accepted 16 January 2023

Keywords:
Efficient Sampling
Path Planning
Dynamic Environment
Rapidly-exploring Random Tree
Rule-template Sets

A B S T R A C T

This study presents an efficient path planning method for mobile robots in a dynamic environment. The method is based on the rapidly-exploring random tree (RRT) algorithm. The two primary processes in mobile robot path planning in a dynamic environment are initial path planning and path re-planning. In order to generate a feasible initial path with fast convergence speed, we used a hybridization of rapidly-exploring random tree star and ant colony systems (RRT-ACS). When an obstacle obstructs the initial path, the path re-planner must be executed. In addition to the RRT-ACS algorithm, we proposed using a rule-template set based on the mobile robot in dynamic environment scenes during the path re-planner process. This novel algorithm is called RRT-ACS with Rule-Template Sets (RRT-ACS+RT). We conducted many benchmark simulations to validate the proposed method in a real dynamic environment. The performance of the proposed method is compared to the state-of-the-art path planning algorithms: RRT*FND and MOD-RRT*. Numerous experimental results demonstrate that the proposed method outperforms other comparison algorithms. The results show that the proposed method is suitable for the use on robots that need to navigate in a dynamic environment, such as self-driving cars.

***doi**: 10.5829/ije.2023.36.04a.16*

1. INTRODUCTION

Path planning is one of the most researched problems in robotics [1-3]. Any path planning algorithm's primary goal is to provide a collision-free path from a start state to an end state within the robot's configuration space [4, 5]. The Rapidly Exploring Random Tree (RRT) is a simple and fast algorithm that generates a tree in the configuration space incrementally until the goal is reached [6]. The RRT is one of the most widely used probabilistic planning algorithms [7-9]. On the other hand, the RRT has never converged to an asymptotically optimal solution [10, 11]. As a result, Karaman developed the RRT* [12]. Since the introduction of the RRT* algorithm, research has expanded to discover new ways to improve the RRT* algorithm in the path planning application [13-16]. However, the operational environment of path planning is dynamic in many scenarios [17, 18]. The path from a single query is frequently obstructed during execution. As a result, the topic of dynamic and re-planning is critical to robotic path planning [19].

Many methods for dynamic path planning algorithms have been presented in the literature. Sampling-based planners, such as RRT-related methods, are widely used among existing mobile robot path planning in a dynamic environment [6, 17]. Meng et al. [20] proposed an improved two-way RRT algorithm to solve the path re-planning problem for Unmanned Aerial Vehicles (UAV) in dynamic environments. Before the UAV takes off, offline path planning is performed. If a pop-up threat is detected during the flight, the affected nodes will be removed while the remaining tree structure will be preserved. The improved two-way RRT is then used for path re-planning. Chen and Wang [21] present a novel RRT-based path planning algorithm that allows UAVs to fly safely in dynamic threat environments. Chen and Wang [21] proposed a pruning-reconnecting mechanism to avoid collisions with dynamic threats and repair the path when new obstacles appear. Adiyatov and Varol

*Corresponding Author Institutional Email: muhammad.aria@email.unikom.ac.id (M. A. R. Pohan)

Please cite this article as: M. A. R. Pohan, J. Utama, Efficient Sampling-based for Mobile Robot Path Planning in a Dynamic Environment Based on the Rapidly-exploring Random Tree and a Rule-template Sets, ***International Journal of Engineering, Transactions A: Basics***, Vol. 36, No. 04, (2023), 797-806

[22] proposed a novel RRT-based algorithm for dynamic motion planning and named it RRT*FND. Adiyatov and Varol [22] extends the memory-efficient RRT*FN algorithm to dynamic scenarios. Adiyatov and Varol [22] then used two greedy heuristics to repair the solution rather than restarting the entire motion planning process. Qi et al. [23] present an RRT algorithm suitable for robot navigation in an undetermined dynamic environment. The algorithm includes two procedures: initial path planning and re-planning. To determine the initial path, a modified RRT* is employed. If the current path is not feasible, a different method is intended to reroute it. This algorithm is called MOD-RRT*. Wei et al. [24] proposed a Bi-RRT* dynamic path planning approach based on an improved exploring function with a goal direction to deal with re-planning paths for a robotic manipulator to avoid dynamic obstacles. A multi-step expansion strategy with greedy heuristics is also used in the Bi-RRT* proposal. Meng and Dai [25] updated the current path with the advanced RRT algorithm for local path planning to avoid obstacles based on the obstacle map. Meng's method solved the problem of avoiding dynamic obstacles during real-time autonomous robot navigation. However, the RRT algorithm is still used in the methods described above.

Numerous techniques have been developed to enhance the performance of the RRT* algorithm. Klemm et al. [13] introduced the RRT*-connect algorithm, a dual-tree variant of RRT*. RRT*-connect could find the solution faster than RRT, especially in a narrow passage environment where the planner must traverse to find solutions. However, to optimize their paths, RRT*-connect must search all states. Gammel et al. [14] introduced the informed RRT* algorithm, which employs informed sampling on RRT* once the first solution has been determined. The informed RRT* algorithm can find the optimal solution 1.2–10.3 times faster than the RRT* algorithm. However, informed RRT* encounters difficulty when the target node is concealed behind a narrow passage. The informed RRT*-connect algorithm was developed by Mashayekhi et al. [15], which combines RRT*-connect with informed RRT*. Therefore, informed RRT*-connect can find the optimal solution faster than informed RRT* in a narrow passage environment. Pohan et al. [16] created the RRT-ACS algorithm, a hybrid of the RRT algorithm and the Ant Colony System (ACS). In Friedman's nonparametric test, Pohan et al. [16] reported that the RRT-ACS algorithm outperformed the informed RRT*, informed RRT*-connect, RRT*-connect, and RRT* algorithms. To the authors' best of knowledge, no research has used the RRT-ACS algorithm in a path planning mobile robot in a dynamic environment.

This study presents an effective mobile robot path planning method in a dynamic environment. The method is based on the RRT algorithm. Initial path generation and path re-planning are the two primary processes in path planning for mobile robots in dynamic environments. In order to generate a feasible initial path with fast convergence speed, we used the RRT-ACS algorithm. The path re-planner must be executed when an obstruction blocks the initial path. During the path re-planner process, we propose using a rule-template set based on the mobile robot in dynamic environment scenes. A rule-template set consists of multiple rule templates generated offline based on the context of an environment scene. At the start of the algorithm, the rule-template set is loaded, and a suitable rule template can be chosen automatically based on the scene of the environment. If some nodes in the rule template are not collision-free, these nodes will be discarded, and the remaining nodes will be added to the root. Thus, the tree likely possesses a portion of branches and leaves at this time. If this tree has reached the goal state, then random searching is no longer necessary, and this strategy will reduce the time needed for re-planning. However, if the goal state has yet to be reached, the RRT-ACS algorithm will be implemented to accelerate the growth rate of the search tree in order to reach the goal state. RRT-ACS with Rule-Template Sets (RRT-ACS+RT) is the name of this proposed algorithm. We ran numerous benchmark simulations to validate the proposed method in a dynamic environment. The performance of the proposed method is compared to RRT*FND and MOD-RRT*, the state-of-the-art path planners in a dynamic environment. Numerous experimental results indicate that the proposed method outperforms other comparitive algorithms. The results show that the proposed method is suitable for use on robots that need to navigate in a dynamic environment, such as self-driving cars.

The remaining sections of this paper are structured as follows. The RRT-ACS+RT algorithm is described in section II. In section III, the performance of the RRT-ACS+RT algorithm in comparison to RRT*FND and MOD-RRT*, the state-of-the-art path planners for a dynamic environment, is evaluated and discussed. Finally, section IV presents some conclusions.

2. PROPOSED ALGORITHM: RRT-ACS+RT

2. 1. Proposed Algorithm This section details the proposed algorithm. Algorithm 1 depicts the RRT-ACS+RT algorithm. Compared with RRT-ACS, the improvement is the addition of rule templates. By introducing path rule templates, more vertices and edges will be derived from rule templates than constructed online during the re-planning phase when obstacles block the initial path. The concept of using rule templates to speed up the re-planning process was inspired by the research of Ma et al. [26]. Ma et al. use rule templates for cases of autonomous on-road driving.

A set of rule templates contains multiple rule templates, which are generated offline according to the context of the environment. Figure 1 shows an example of a rule template created offline for a straight-line initial path blocked by an obstacle.

At the start of the proposed algorithm, the rule template set T is loaded (line 2). This template is then parsed and saved as a tree-like structure. If some of the vertices and path edges in the rule template are not collision-free with obstacles or road boundaries, these sections are discarded, and the remainder in the rule template is preserved and added to the root (line 3). Therefore, at this point, the tree may already have some branches and leaves before searching randomly. If the tree from this template can immediately plot a path to the current destination state without any obstructions, then the algorithm stops and returns the tree. If the function fails, then the algorithm starts the following iteration step.

In the iteration step process, a random variable value will be generated at each iteration to determine whether the exploration or exploitation process will be used (line 8). If the exploration process is chosen, the random sample will determine the random state $q_{rndstate}$ (line 14). If the exploitation process is selected, the random state will be determined using pheromone distribution data (lines 10-12).

After obtaining the value of the random state, the following process is to look for nodes in T that are less than a threshold value to the random state. The nodes are named Q_{near} (line 16). Then the node in Q_{near} that is closest to $q_{rndstate}$ is selected. The node with the shortest distance is named $q_{nearest}$ (line 17). Next, a new branch will be created from the $q_{nearest}$ node that goes to $q_{rndstate}$ with a distance of Δq from $q_{nearest}$. The new node is named q_{new} (line 18). If there are no obstacles between $q_{nearest}$ and q_{new} (line 19), then the q_{new} node will be entered into T (lines 20-22). Next, it will be checked whether there are nodes in the current T close enough to the destination node or q_{goal} (line 23). If a node is close enough to the destination node, an X^{bs} route will be built (line 24). This process will repeat until the stop condition is met. The looping process can have several conditions that vary to stop, for example, if the maximum number of iterations has been met.

The RRT-ACS+RT algorithm determines the new path using pheromone information to ensure the quality of the final result. This motivation is based on the principle of learning by doing. A procedure for updating pheromone data has been implemented (lines 31-33). Only ants with the best path will contribute to providing additional pheromones at each iteration. This restriction is intended to provide additional reinforcement only to the good paths. The function LocalSearch (line 30) will optimize RRT's existing branches to improve the quality of found paths.

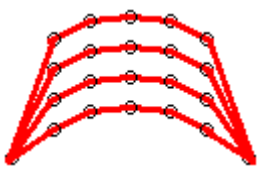


Figure 1. Example of a rule template for a straight line initial path blocked by an obstacle

Algorithm 1 : $X^{bs} \leftarrow$RRT-ACS+RT(map)

1: **% ========== Initialization**
2: $T \leftarrow Load\ Template()$
3: $T \leftarrow TrimTree(T)$
4: $s \leftarrow 0$
5: **while** *termination condition not met* **do**
6: **for** $k = 1$ to m **do**
7: **while** $s = 0$ **do**
8: $q \leftarrow randvar[0,1]$ **and** $\tau \neq \emptyset$
9: **if** $q \leq q_o$
10: $q_{samp} \leftarrow SamplingeFrom\ 10LastNode(T)$
11: $\tau_{near} \leftarrow Near(\tau, q_{samp})$
12: $q_{rndstate} \leftarrow RouletteWhell(\tau_{near})$
13: **else**
14: $q_{rndstate} \leftarrow RandomSample(k)$
15: **end if**
16: $Q_{near} \leftarrow Near(T, q_{rndstate})$
17: $q_{nearest} \leftarrow NearestNeighboor\ (q_{rndstate}.Q_{near}.T)$
18: $q_{new} \leftarrow GrowTree\ (q_{nearest}, q_{rndstate}, \Delta q)$
19: **if** $Obstaclefree(q_{nearest}, q_{new})$ **then**
20: $Q_{near} \leftarrow Near(T, q_{new})$
21: $q_{min} \leftarrow ChooseParent\ (q_{new}.Q_{near}.q_{nearest})$
22: $T \leftarrow InsertNode(q_{min}.q_{new}.T)$
23: **if** $CanConnected(q_{goal}, T)$ **then**
24: $X^{bs} \leftarrow UpdateBestPath(T)$
25: **end if**
26: **end if**
27: **end while**
28: $X(k) \leftarrow MakePath$ from T
29: **end for**
30: $X^{bs} \leftarrow LocalSearch(X^{bs}.map)$
31: $X^{w} \leftarrow$ Select w best $Path(X)$
32: $\tau \leftarrow$ Add $Pheromone\ Node$ on the w best $Path(X^{w}, \tau)$
33: $\tau \leftarrow$ Evaporate $Pheromone\ Node(\tau)$
34: **end while**

The RRT-ACS+RT algorithm was designed to deal with dynamic obstacles. Algorithms 2 - 4 depict the RRT-ACS+RT-based dynamic path planning algorithm. In order to generate a feasible initial path with fast convergence speed, the RRT-ACS algorithm, as described by Pohan et al. [16], is used (line 2 in algorithm 2). After that, the robot will set its velocity (line 4 in algorithm 2). The robot's position is updated based on the

speed. This process is described in algorithm 3. When the vertex is reached, the robot changes the velocity vector to move toward the next node (line 2-5 in algorithm 2).

Algorithm 2 : $ExecutePath()$

```
1:  map ← SetRobotEnvironment()
2:  X^bs ←RRT-ACS(map)
3:  RobotDestinaton ← NextWaypnt(X^bs)
4:  Set RobotVelocity()
5:  SetObstacleDestination(NumberObstacles)
6:  SetObstacleVelocities(NumberObstacles)
7:  while RobotLocation ! = GoalLocation do
8:    UpdateRobotLoc(X^bs)
9:    UpdateObstaclesLoc(NumObstacles)
10: end while
```

Algorithm 3 : $UpdateRobotLoc(X^{bs})$

```
1:  RobotLocation = RobotLocation +
                    RobotVelocity
2:  If RobotLocation == RobotDestination
                    then
3:    RobotDestinaton ← NextWaypnt(X^bs)
4:    Set RobotVelocity()
5:  end
6:  ObsDistance ← GetDistance(
                    RobotLocation,
                    ObstacleLoc)
7:  if ObsDistance < RobotRange then
8:    if ObstacleLoc blocks the path do
9:      X^bs ←DoReplan(map)
10:   end if
11: end if
```

Algorithm 4 : X^{bs} ←DoReplan(map)

```
1:  T ← InvalidateNodes()
2:  map ← SetReplanStartLocation()
3:  map ← SetReplanGoalLocation()
4:  X^replan ←RRT-ACS+RT(map)
5:  while RobotLocation ! =
          ReplanGoalLocation do
6:    UpdateRobotLoc(X^replan)
7:    UpdateObstaclesLoc(NumObstacles)
8:  end while
```

When the simulation begins, the moving obstacles choose several random adjacent vertices as the destination points for movement (lines 5 and 6 in algorithm 2). During the simulation, the obstacle will move towards several random adjacent vertexes (line 9 of algorithm 2).

As the robot approaches the RobotDestination, it will determine if there are any dynamic obstacles within the robot's detection range. Furthermore, the dynamic obstacle movement must be observed in at least two steps to predict the dynamic obstacle's speed and direction. If the dynamic obstacle is predicted to block the path that the vehicle should take, DoReplan will be executed (lines 7 - 9 in algorithm 3).

The DoReplan algorithm works on the principle of determining the current vehicle position node as ReplanStartLocation (line 2 in algorithm 4). Following that, the DoReplan algorithm will determine the next position on the global path that is not blocked by dynamic obstacles (line 3 in algorithm 4). ReplanGoalLocation is the name of this position. The RRT-ACS+RT algorithm will then modify and expand the nodes surrounding the dynamic obstacle that connects ReplanStartLocation and ReplanGoalLocation (line 4 in algorithm 4). In principle, a path will be found that will circle the dynamic obstacle to ReplanGoalLocation. Any nodes that collide with a random moving obstacle are invalidated rather than deleted. After that, the robot will follow the re-planning result path until it reaches the ReplanGoalLocation (lines 5-8 in algorithm 4). Once the ReplanGoalLocation is reached (and the obstacle has been overcome), the robot will retrace the initial path determined previously.

2. 2. The Illustration Figures 2 – 10 depict illustrations of the proposed algorithm. This illustration mimics the scenario proposed by Connell et al. [6], Jin et al. [27], and Wei and Ren [28]. The illustration shows a robot path planning process in a maze environment with a dynamic object. Figure 2 shows the scenario of the maze environment used and a unicycle robot model (big grey circle) moving from the starting location (red dot) to the destination location (blue dot). Robots are currently treated as particles. Furthermore, certain safe distances are set aside to form obstacles. The barriers are expanded to ensure the robot's safety during operation, as shown in Figure 3.

The next step is to perform path planning using the RRT-ACS algorithm, with the results displayed in Figure 4. In Figure 4, it is evident that the path generated by the RRT-ACS algorithm is close to the optimal path, as the path's corner points are close to the corners of the obstacle. Figure 5 depicts the results of path planning on the original map (without obstacle expansion). Based on Figure 5, the resulting path is safe for the robot to traverse, as it will not cause it to collide with the wall. The illustration of a robot moving through a given path is shown in Figure 6.

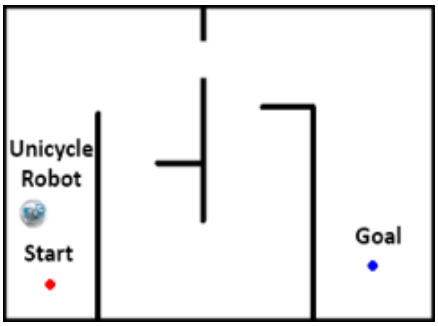


Figure 2. Maze environment scenario used for testing

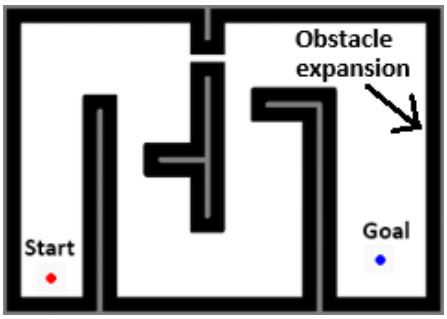


Figure 3. Obstacle expansion

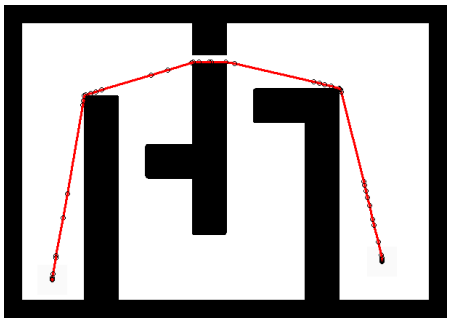


Figure 4. The results of path planning using the RRT-ACS algorithm

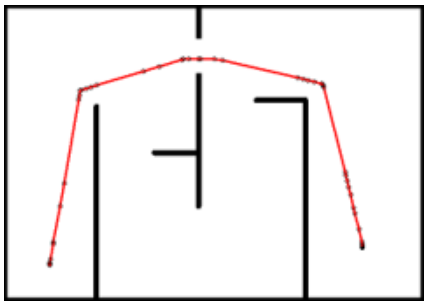


Figure 5. The results of the path planning using the RRT-ACS algorithm on the initial map

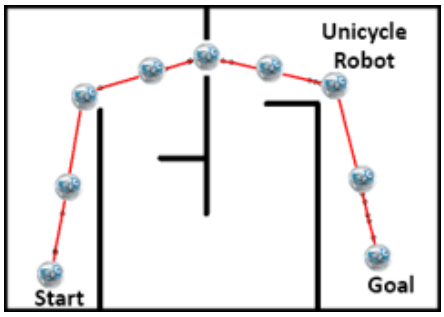


Figure 6. Illustration of a robot moving through a given path

Furthermore, a dynamic obstacle in the form of a basketball is given, as shown in Figure 7. It can be seen that the dynamic obstacle blocks the path that the robot must pass. The robot will crash into these dynamic obstacles if re-planning is not carried out. So that in the proposed algorithm, if an obstacle is detected that blocks the robot, it will be detected by the sensor. The algorithm will update the environmental map, as shown in Figure 8.

Next, the RRT-ACS+RT algorithm is run again to produce a re-planning algorithm to avoid dynamic constraints. The first stage of the RRT-ACS+RT algorithm for the re-planning process is to use the tree of rule templates, as shown in Figure 9. If the tree of rule templates can be connected to the destination node, then the re-planning process is complete. If not, the RRT-ACS algorithm will start iterating to connect the tree with the destination node. The resulting re-planning path is shown in Figure 10.

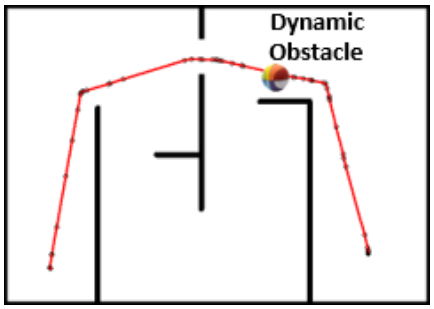


Figure 7. Given an obstacle on the path traversed by the robot

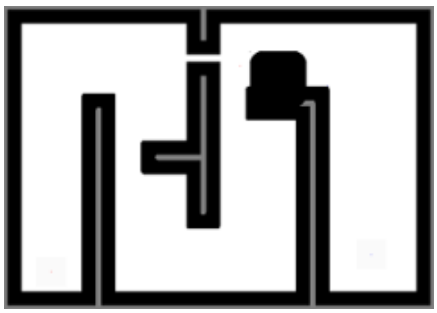


Figure 8. Update the environmental map after the robot detects a dynamic obstacle

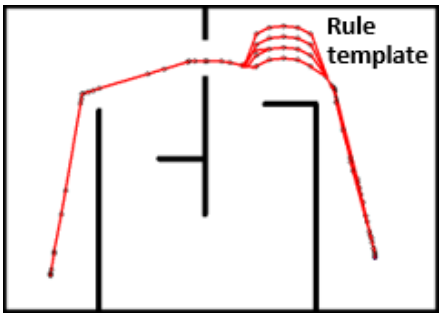


Figure 9. Use of rule template to speed up the re-planning process

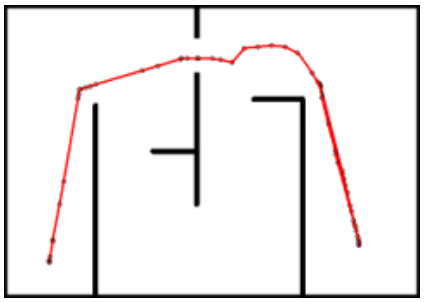


Figure 10. Path re-planning results to avoid dynamic obstacle

3. RESULTS AND DISCUSSION

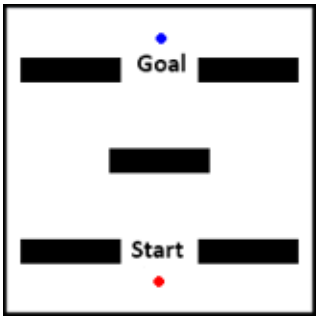
To validate the performance of the proposed path planning mobile robot in a dynamic environment using the RRT-ACS+RT algorithm, we ran it through a three-dynamic environment simulation scenario (as shown in Figures 11 – 13). The three scenarios replicate the test scenario proposed by Wang et al. [29]. In each scenario, the robot's initial location is indicated by a red dot, while a blue dot indicates the robot's goal location. The simulation tests were carried out on a PC with a Core i5 3.20 GHz CPU and 4 GB RAM running Windows 10 64-bit, with LabVIEW 7.1 as the compilation environment.

The first scenario is shown in Figure 11(a). In this scenario, there are five static obstacles (black box) and one dynamic obstacle (in the form of a basketball). First, the robot will create an initial path, as shown in Figure 11(b). Then, when the robot moves, a dynamic obstacle will be placed that will block the initial path, as shown in Figure 11(c) or Figure 11(d). The robot must then re-plan in order to find a new path to the goal state that avoids the newly-placed dynamic obstacles. Figure 11(c) shows an example of the re-planning path generated by the proposed algorithm. It can be seen that the robot simply circles around the dynamic obstacle to avoid colliding with it. As for the case of Figure 11(d), the dynamic obstacle covers the narrow path that the robot wants to pass (making the robot unable to pass through the narrow path). The RRT-ACS+RT algorithm then searches for a new route that takes a right turn to reach the goal state while avoiding collisions with dynamic obstacles.

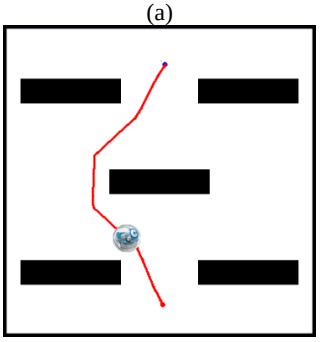
The second scenario is shown in Figure 12(a). In this scenario, there are two static obstacles and two dynamic obstacles. First, the robot will create an initial path, as shown in Figure 12(b). Then, when the robot begins to move, it will place the first dynamic obstacle that will block the initial path, as shown in Figure 12(c). The robot must then re-plan to find a new path to the goal state without being impeded by the newly placed dynamic obstacles. Figure 12(c) shows an example of the re-planning path generated by the proposed algorithm. It can be seen that the robot simply circles around the dynamic obstacle to avoid a collision.

Furthermore, a second dynamic obstacle will be placed when the robot moves again to block the initial path on the other side, as shown in Figure 12(d). Then the robot must re-do the re-planning to find a new path to reach the goal state and not be hindered by the second dynamic obstacle. Figure 12(d) shows an example of the second re-planning path generated by the RRT-ACS+RT algorithm.

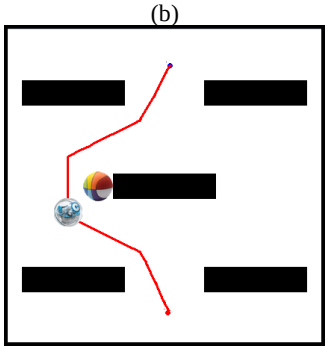
Meanwhile, the third scenario is shown in Figure 13(a). The initial path generated by the proposed algorithm is also shown in Figure 13(a). In this scenario, there are four static obstacles and three dynamic obstacles. The three dynamic obstacles will be placed in turn to block the initial path of the robot, as shown in Figure 13(b) – 13(d). It can be seen in the three figures that every time a dynamic obstacle blocks the robot's path; the robot will re-plan to find a new path to reach the goal state and not be hindered by dynamic obstacles. Figures 13(b) – 13(d) show an example of the re-planning path generated by the RRT-ACS+RT algorithm every time the robot detects a dynamic object.



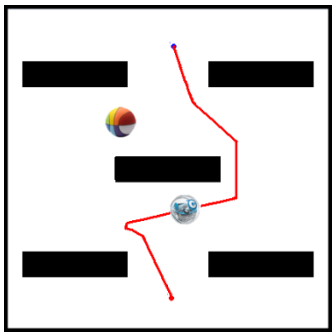
(a)



(b)

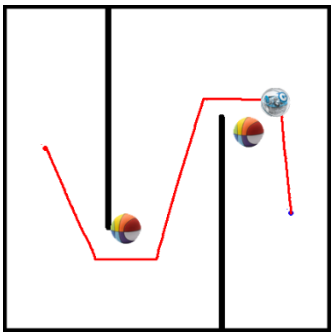


(c)



(d)

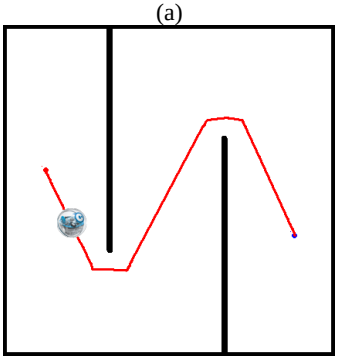
Figure 11. First test scenario: (a) test scenario, initial location and goal location of the robot, (b) an example of the initial path generated by the proposed algorithm, (c) and (d) dynamic obstacle is placed that will block the initial path and the robot must re-planning the path to reach the goal location while avoiding collisions with dynamic obstacles



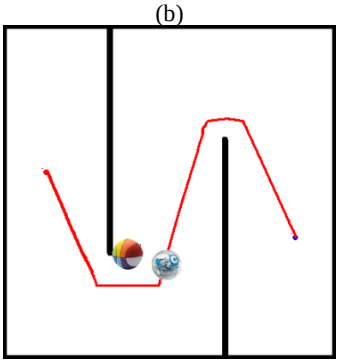
(d)

Figure 12. Second test scenario: (a) test scenario, initial location and goal location of the robot, (b) an example of the initial path generated by the proposed algorithm, (c) and (d) the first and the second dynamic obstacle is placed that will block the initial path and the robot must re-planning the path to reach the goal location while avoiding collisions with dynamic obstacles

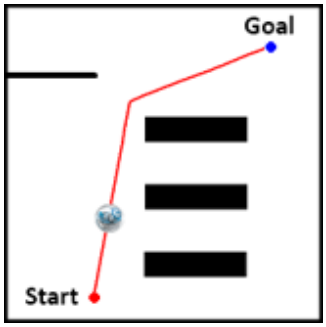
(a)



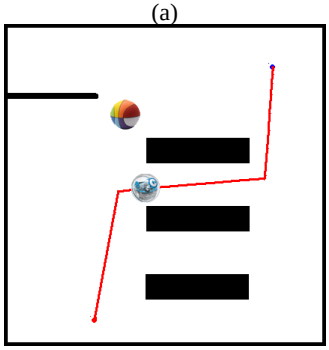
(b)



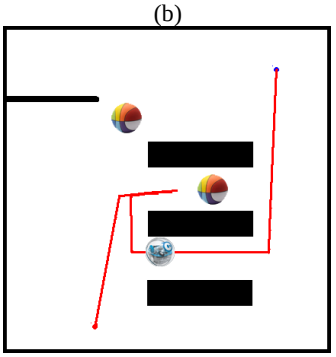
(c)



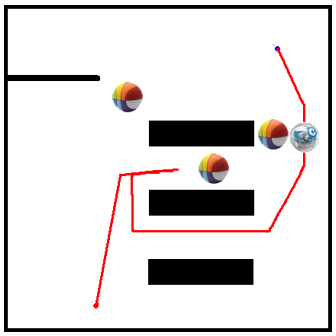
(a)



(b)



(c)



(d)

Figure 13. Third test scenario: (a) test scenario, initial location, goal location, and the initial path generated by the proposed algorithm, (b) - (d) the first, the second, and the third dynamic obstacle is placed that will block the initial path and the robot must re-planning the path to reach the goal state while avoiding collisions with dynamic obstacles

The performance of the re-planning process of the mobile robot in a dynamic environment using the RRT-ACS+RT algorithm is compared to the RRT*FND, MOD-RRT*, and RRT* algorithms. Testing was carried out using the three scenarios that have been described. The results are summarized in Tables 1-3. Performance measurements include the best, the worst, and the mean computation time of each algorithm.

Based on the data in Tables 1-3, we will compare how fast the RRT-ACS+RT, RRT*FND, and MOD-RRT* algorithms perform the re-planning process compared to the RRT* algorithm. The results of this comparison are stated in Table 4.

Based on the datareported in Table 4, it can be seen that the average time for the RRT-ACS+RT algorithm to carry out the re-planning process is 65.5% faster than the RRT* algorithm. These results are consistent with the RRT-ACS algorithm performance measurement results reported by Pohan et al. [16]. They have also reported that the speed of the RRT-ACS algorithm was 59.4%

TABLE 1. Comparison of the time required for each algorithm to re-planning in the first scenario. The time is in milliseconds

Computation time performance	RRT-ACS+RT	RRT* FND	MOD-RRT*	RRT*
The dynamic obstacle's location as shown in Figure 11(c)				
Best	478.9	1216.2	1577.0	1770
Mean	565.3	1451.0	2617.8	3113
Worst	1158.1	1935.9	4050.3	5296
The dynamic obstacle's location as shown in Figure 11(d)				
Best	763.8	1385.5	1649.7	1875
Mean	936.5	2340.9	2549.5	2932.2
Worst	2824.5	19670.1	4289.6	4965

TABLE 2. Comparison of the time required for each algorithm to re-planning in the second scenario. The time is in milliseconds

Computation time performance	RRT-ACS+RT	RRT* FND	MOD-RRT*	RRT*
The first dynamic obstacle is placed as in Figure 12(c)				
Best	615.5	1003.8	1028.2	1154
Mean	921.7	1086.6	1696.1	2017
Worst	943.1	1329.1	2207.2	2886
The second dynamic obstacle is placed as in Figure 12(d)				
Best	266.8	360.0	1525.7	1734
Mean	1427.7	1861.0	2327.6	2677
Worst	2593.5	4060.7	3627.8	4199

TABLE 3. Comparison of the time required for each algorithm to re-planning in the third scenario. The time is in milliseconds

Computation time performance	RRT-ACS+RT	RRT* FND	MOD-RRT*	RRT*
The first dynamic obstacle is placed as in Figure 13(b)				
Best	286.2	794.3	798.3	896
Mean	312.8	980.0	1709.7	2033.2
Worst	441.6	1254.3	3501.2	4578
The second dynamic obstacle is placed as in Figure 13(c)				
Best	869.8	1577.7	1343.5	1527
Mean	1245.8	2092.6	3123.2	3592
Worst	1147.2	6019.3	4884.9	5654
The third dynamic obstacle is placed as in Figure 13(d)				
Best	482.1	801.3	805.4	904
Mean	753.7	1038.2	1256.2	1493.8
Worst	922.6	2069.1	1804.9	2360

TABLE 4. Comparison of the average time for each algorithm executes the re-planning process relative to the RRT* algorithm

Dynamic obstacle position	RRT-ACS+RT	RRT* FND	MOD-RRT*
First scenario, Figure 11(c)	81.8%	53.4%	15.9%
First scenario, Figure 11(d)	68.1%	20.2%	13.1%
Second scenario, Figure 12(c)	54.3%	46.1%	15.9%
Second scenario, Figure 12(d)	46.7%	30.5%	13.1%
Third scenario, Figure 13(b)	84.6%	51.8%	15.9%
Third scenario, Figure 13(c)	73.7%	41.7%	13.1%
Third scenario, Figure 13(d)	49.5%	30.5%	15.9%
Average percentage	**65.5%**	**39.2%**	**14.7%**

faster than the RRT* algorithm in reaching the optimal path. Based on the data in Table 4, it can be apparent that the average time for the RRT*FND algorithm to carry out the re-planning process is 39.2% faster than the RRT* algorithm. These results are consistent with the performance measurement results of the RRT*FND algorithm reported by Adiyatov et al. [22]. Adiyatov et al. [22] also reported that the speed of the RRT*FND algorithm in reaching the optimal path was 43.5% faster than the RRT* algorithm. Based on Tables 1 - 4, it is clear that the RRT-ACS+RT algorithm outperforms other algorithms in terms of best path length, worst path length, and mean path length.

4. CONCLUSION

This study implements a path planning mobile robot based on the RRT-ACS+RT algorithm in a dynamic environment. The simulation was carried out to validate the proposed method in a real dynamic environment. The proposed method's performance is compared to the RRT*FND and MOD-RRT algorithms. The test results demonstrate that the proposed method outperforms other comparison algorithms. The findings indicate that the proposed method is appropriate for use on robots navigating in a dynamic environment, such as self-driving cars.

5. ACKNOWLEDGMENTS

This research was funded by Universitas Komputer Indonesia through the UNIKOM Internal Research program in 2023.

6. REFERENCES

1. Ganeshmurthy, M. S., and Suresh, G. R. "Path planning algorithm for autonomous mobile robot in dynamic environment." In 2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN), (2015), 1-6. https://doi.org/10.1109/icscn.2015.7219901
2. Ab Wahab, M. N., Lee, C. M., Akbar, M. F., and Hassan, F. H. "Path planning for mobile robot navigation in unknown indoor environments using hybrid PSOFS algorithm." ***IEEE Access***, Vol. 8, (2020), 161805-161815. https://doi.org/10.1109/access.2020.3021605
3. Nascimento, L. B., Morais, D. S., Barrios-Aranibar, D., Santos, V. G., Pereira, D. S., Alsina, P. J., and Medeiros, A. A. "A Multi-Robot Path Planning Approach Based on Probabilistic Foam." In 2019 Latin American Robotics Symposium (LARS), 2019 Brazilian Symposium on Robotics (SBR) and 2019 Workshop on Robotics in Education (WRE), (2019), 329-334. https://doi.org/10.1109/lars-sbr-wre48964.2019.00064
4. Aria, M. "Optimal Path Planning using Informed Probabilistic Road Map Algorithm." ***Journal of Engineering Research-ASSEEE Special Issue***, (2021), 1-15. https://doi.org/10.36909/jer.asseee.16105
5. Teshnizi, M. M., Kosari, A., Goliaei, S., and Shakhesi, S. "Centralized Path Planning for Multi-aircraft in the Presence of Static and Moving Obstacles." ***International Journal of Engineering, Transactions B: Applications***, Vol. 33, No. 5, (2020) 923-933. https://doi.org/10.5829/ije.2020.33.05b.25
6. Connell, D., and Manh La, H. "Extended rapidly exploring random tree–based dynamic path planning and replanning for mobile robots." ***International Journal of Advanced Robotic Systems***, Vol. 15, No. 3 (2018), 1729881418773874. https://doi.org/10.1177/1729881418773874
7. Zhang, H., Wang, Y., Zheng, J., and Yu, J. "Path planning of industrial robot based on improved RRT algorithm in complex environments." ***IEEE Access***, Vol. 6, (2018) 53296-53306. https://doi.org/10.1109/access.2018.2871222
8. Li, X., Jiang, H., Shi, W., Chen, S., and Wang, Y. "Path planning of multipoint region attraction RRT* algorithm in complex environment." In 2019 Chinese Control Conference (CCC), (2019) 4409-4414. https://doi.org/10.23919/chicc.2019.8865834
9. Taheri, E. "Any-time randomized kinodynamic path planning algorithm in dynamic environments with application to quadrotor." ***International Journal of Engineering, Transactions A: Basics***, Vol. 34, No. 10, (2021) 2360-2370. https://doi.org/10.5829/ije.2021.34.10a.17
10. Wang, J., Li, X., and Meng, M. Q. H. "An improved rrt algorithm incorporating obstacle boundary information." In 2016 IEEE International Conference on Robotics and Biomimetics (ROBIO), (2016) 625-630. https://doi.org/10.1109/robio.2016.7866392
11. Zhang, Y., Wang, R., Song, C., and Xu, J. "An Improved Dynamic Step Size RRT Algorithm in Complex Environments" In 2021 33rd Chinese Control and Decision Conference (CCDC), (2021) 3835-3840. https://doi.org/10.1109/ccdc52312.2021.9602069
12. Solovey, K., Janson, L., Schmerling, E., Frazzoli, E., and Pavone, M. "Revisiting the asymptotic optimality of RRT." In 2020 IEEE International Conference on Robotics and Automation (ICRA), (2020) 2189-2195. https://doi.org/10.1109/icra40945.2020.9196553
13. Klemm, S., Oberländer, J., Hermann, A., Roennau, A., Schamm, T., Zollner, J. M., and Dillmann, R. "RRT-connect: Faster, asymptotically optimal motion planning." In 2015 IEEE International conference on robotics and biomimetics (ROBIO), (2015) 1670-1677. https://doi.org/10.1109/robio.2015.7419012
14. Gammell, J. D., Barfoot, T. D., and Srinivasa, S. S. "Informed asymptotically optimal anytime search." in ***International Journal of Robotics Research***, (2017) 1-27. https://doi.org/10.1177/0278364919890396
15. Mashayekhi, R., Idris, M. Y. I., Anisi, M. H., Ahmedy, I., and Ali, I. "Informed RRT*-connect: An asymptotically optimal single-query path planning method." ***IEEE Access***, Vol. 8, (2020) 19842-19852. https://doi.org/10.1109/access.2020.2969316
16. Pohan, M. A. R., Trilaksono, B. R., Santosa, S. P., and Rohman, A. S. "Path Planning Algorithm Using the Hybridization of the Rapidly-Exploring Random Tree and Ant Colony Systems." ***IEEE Access***, Vol. 9, (2021) 153599-153615. https://doi.org/10.1109/access.2021.3127635
17. Connell, D., and La, H. M. "Dynamic path planning and replanning for mobile robots using RRT." In 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), (2017) 1429-1434. https://doi.org/10.1109/smc.2017.8122814
18. Yaghmaee, F., and Koohi, H. "Dynamic obstacle avoidance by distributed algorithm based on reinforcement learning." ***International Journal of Engineering, Transactions B:***

Applications, Vol. 28, No. 2, (2015) 198-204. https://doi.org/10.5829/idosi.ije.2015.28.02b.05

19. Kala, R., Shukla, A., and Tiwari, R. "Robot path planning using dynamic programming with accelerating nodes." ***Paladyn***, Vol. 3, No. 1, (2012) 23-34. https://doi.org/10.2478/s13230-012-0013-4
20. Meng, L., Qing, S., and Jun, Z. Q. "UAV path re-planning based on improved bidirectional RRT algorithm in dynamic environment." In 2017 3rd International Conference on Control, Automation and Robotics (ICCAR), (2017) 658-661. https://doi.org/10.1109/iccar.2017.7942779
21. Chen, Y., and Wang, L. "Adaptively Dynamic RRT*-Connect: Path Planning for UAVs Against Dynamic Obstacles." In 2022 7th International Conference on Automation, Control and Robotics Engineering (CACRE), (2022) 1-7. https://doi.org/10.1109/cacre54574.2022.9834188
22. Adiyatov, O., and Varol, H. A. "A novel RRT*-based algorithm for motion planning in Dynamic environments." In 2017 IEEE International Conference on Mechatronics and Automation (ICMA), (2017) 1416-1421. https://doi.org/10.1109/icma.2017.8016024
23. Qi, J., Yang, H., and Sun, H. "MOD-RRT*: A sampling-based algorithm for robot path planning in dynamic environment." ***IEEE Transactions on Industrial Electronics***, Vol. 68, No. 8, (2020) 7244-7251. https://doi.org/10.1109/tie.2020.2998740
24. Wei, K., Chu, Y., and Gan, H. "An improved Rapidly-exploring Random Tree Approach for Robotic Dynamic Path Planning." In 2021 11th International Conference on Intelligent Control and Information Processing (ICICIP), (2021) 181-187. https://doi.org/10.1109/icicip53388.2021.9642182
25. Meng, C., and Dai, H. "An Obstacle Avoidance Method Based on Advanced Rapidly-exploring Random Tree for Autonomous Navigation." In 2021 IEEE International Conference on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom), (2021) 1118-1125. IEEE. https://doi.org/10.1109/ispa-bdcloud-socialcom-sustaincom52081.2021.00154
26. Ma, L., Xue, J., Kawabata, K., Zhu, J., Ma, C., and Zheng, N. "Efficient sampling-based motion planning for on-road autonomous driving." ***IEEE Transactions on Intelligent Transportation Systems***, Vol. 16, No. 4, (2015) 1961-1976. https://doi.org/10.1109/tits.2015.2389215
27. Jin, Q., Tang, C., and Cai, W. "Research on Dynamic Path Planning Based on the Fusion Algorithm of Improved Ant Colony Optimization and Rolling Window Method." ***IEEE Access***, Vol. 10, (2021) 28322-28332. https://doi.org/10.1109/access.2021.3064831
28. Wei, K., and Ren, B. "A method on dynamic path planning for robotic manipulator autonomous obstacle avoidance based on an improved RRT algorithm." ***Sensors***, Vol. 18, No. 2 (2018), 571. https://doi.org/10.3390/s18020571
29. Wang J., Meng M. Q. -H., and Khatib, O. "EB-RRT: Optimal Motion Planning for Mobile Robots," in ***IEEE Transactions on Automation Science and Engineering***, Vol. 17, No. 4 (2020), 2063-2073, https://doi.org/10.1109/TASE.2020.2987397

Persian Abstract

چکیده

این مطالعه یک روش برنامه ریزی مسیر کارآمد برای ربات های متحرک در یک محیط پویا ارائه می دهد. این روش بر اساس الگوریتم درخت تصادفی در حال کاوش سریع (RRT) است. دو فرآیند اصلی در برنامه ریزی مسیر ربات متحرک در یک محیط پویا، برنامه ریزی مسیر اولیه و برنامه ریزی مجدد مسیر است. به منظور ایجاد یک مسیر اولیه امکان‌پذیر با سرعت همگرایی سریع، ما از ترکیبی از سیستم‌های مستعمره‌ای از درختان تصادفی که به سرعت در حال کاوش هستند (RRT-ACS) استفاده کردیم. هنگامی که مانعی مسیر اولیه را مسدود می کند، برنامه ریزی مجدد مسیر باید اجرا شود. علاوه بر الگوریتم RRT-ACS، ما استفاده از یک مجموعه الگو–قانون مبتنی بر ربات متحرک را در صحنه‌های محیط پویا در طول فرآیند برنامه‌ریزی مجدد مسیر پیشنهاد کردیم. این الگوریتم جدید RRT-ACS با مجموعه‌های الگو–قانون (RRT-ACS+RT) نامیده می‌شود. ما شبیه‌سازی‌های معیار زیادی را برای اعتبارسنجی روش پیشنهادی در یک محیط پویا واقعی انجام دادیم. عملکرد روش پیشنهادی با الگوریتم‌های پیشرفته برنامه‌ریزی مسیر مقایسه می‌شود: RRT*FND و MOD-RRT*. نتایج تجربی متعدد نشان می‌دهد که روش پیشنهادی بهتر از سایر الگوریتم‌های مقایسه عمل می‌کند. نتایج نشان می‌دهد که روش پیشنهادی برای استفاده در روبات‌هایی که نیاز به جهت‌یابی در یک محیط پویا، مانند اتومبیل‌های خودران دارند، مناسب است.