SINERGI Vol. 28, No. 2, June 2024: 265-276
http://publikasi.mercubuana.ac.id/index.php/sinergi
http://doi.org/10.22441/sinergi.2024.2.007

Problem solving path planning and path tracking in a 3 DOF hexapod robot using the RRT* algorithm with path optimization and Pose-to-Pose

Heru Suwoyo[1], Achmad Burhanudin[1]*, Yingzhong Tian[2], Julpri Andika[1]
[1]Department of Electrical Engineering, Faculty of Engineering, Universitas Mercu Buana, Indonesia
[2]School of Mechatronic Engineering and Automation, Shanghai University, China

Abstract
Path planning is one of the most fundamental problems that must be solved before a robot can navigate and explore autonomously. Path planning needs to be integrated with path tracking to be applied to autonomous robots. This makes path tracking also important for autonomous robot navigation which cannot be separated from path planning. There are two path planning methods, the first is search-based method, the second is sampling-based method. Both have their own advantages, but the popular and commonly used sampling-based algorithm due to its fast convergence is preferred in path planning. The RRT algorithm was developed. This improvement initiated a major civilization in sampling-based algorithms, namely parent node selection and rewiring in RRT. Although there has been an improvement in optimality, RRT* still doesn't provide the distance optimality value as expected, due to its character that is still adopted from RRT. The resulting path is still suboptimal and not smooth (jagged). On the other side, Path tracking has several methods, however, these path tracking methods are difficult to apply to autonomous robots and need to be adapted to the robot used. Based on the description above, there are still problems with path planning, namely paths that are still less than optimal and convergence that is still slow. This research will add a way to shorten the distance in the RRT* algorithm with the triangular inequality method. Meanwhile, for path tracking, we will apply the pose-to-pose method, which follows the waypoint that has been made by path planning.*

Keywords:
Hexapod;
Optimized-RRT;*
Path Planning;
Path Tracking;
Pose-to-Pose;

Article History:
Received: August 5, 2023
Revised: October 20, 2023
Accepted: December 4, 2023
Published: June 2, 2024

Corresponding Author:
Achmad Burhanudin
Department of Electrical Engineering, Universitas Mercu Buana, Indonesia
Email:
achmadburhan41@gmail.com

INTRODUCTION

A navigation system is a technology to determine the path and guide the autonomous robot to move along the path from the initial location to the destination location and independently. Choosing a navigation system is important to be appropriate and suitable for the type of robot and the tasks and objectives to be performed by the robot [1]. Navigation is important in robotics especially autonomous robots [2]. The most common problem for autonomous robot navigation is path planning [2]. This is supported by research [3], path planning is one of the very important topics in the development of mobile robot autonomous navigation. And then research [4], path planning is one of the most fundamental problems that must be solved before robots can navigate and explore autonomously. Path planning needs to be integrated with path tracking to be applied to an autonomous robot. This makes path tracking also important for autonomous robot navigation which is inseparable from path planning. This is supported by research [5], mobile robot tracking control is another important field of study in recent years.

There are two path planning methods, the first is searching based method, such as algorithm A* [4][6], algorithm D* [7] and algorithm Djiakstra [8]. The second is sampling based method, including algorithm RRT [9, 10, 11], algorithm RRT-Connect and similarly [12][13], algorithm RRT* [14][15]. Both have advantages, but the popular and commonly used is the sampling-based algorithm because of the fast convergence that is preferred in path planning. However, the RRT algorithm does not guarantee the optimality of the path according to [9]. The proposed RRT-Connect algorithm [16] then has two new ideas as methods to compensate for the weakness of the RRT algorithm. In short, RRT-Connect finds a path from a start point and an end point that then converge at a point. Path planning through the RRT-Connect algorithm can find a path faster than the RRT algorithm, but the 'Extend' method does not work well in complex environments with narrow paths and many obstacles and can be difficult. In addition, the path planned using the RRT-Connect algorithm is far from the optimal length, so it does not represent the optimality path.

Because of this, a more optimal RRT was developed by [17] called RRT*. This improvement initiated a major civilization in sampling-based algorithms, that is, parent node selection and rewiring in RRT. Although there has been an optimality improvement, RRT* still does not provide the distance optimality value as expected, because its character is still adopted from RRT. It is still slow and the resulting path is still suboptimal and still not smooth (jagged) [18]. On the other side, Path tracking has several methods, such as Pure Pursuit [19], MPC (Model Predictive Control) [20] and Stanley Controller [21]. However, these path tracking methods are difficult to apply to autonomous robots and need to be adapted to the robot used.

Based on the description above, there are still problems in path planning, which is a path that is still suboptimal and convergence is still slow, this is supported by research [18][22] and [23]. So there needs to be an approach that gives effect to the optimality of distance. As for path tracking, the existing method is still complex to apply to hexapod robots, so there needs to be a simpler approach that is easily applied to hexapod robots.

This research will be adding a way to be able to shorten the distance in the RRT* algorithm with the triangular inequality method. With the addition of approaches like this, it is believed that there will be an increase in path optimality, measured from the distance of the path formed and its convergence time. As for path tracking, it will apply the pose-to-pose method, which is following the waypoints created by path planning. However, it can still follow the path accurately.

The next of this paper is organized as follows; Methods and Materials are described in section 2, Experimental Results are described and discussed in Section 3, and the conclusion is presented in section 4.

METHOD

A hexapod robot with 3 DOF is used to apply the proposed method. Specifications of the hexapod robot legs; the joint on this hexapod robot is a revolute joint which means that the angle of rotation is a variable value with certain limitations. The length of coxa = 25 mm, femur = 50 mm, and tibia = 70 mm. The robot is designed with a rigid body made of acrylic and some parts are 3D printed from PLA. Several components, such as OpenRB-150 microcontroller, Dynamixel XL430 servo motor with a power source of 11.1 Volt Li-Po 3s battery are placed in the slot on the robot body. OpenRB-150 is used as a controller for the rotation of the servo motor. The size of the robot is 28 cm wide, 24 cm long and 10 cm high.

The robot can move translationally and rotationally. Rotational movement is the movement of the robot's perspective by pivoting (without displacing center of body robot's). Translational movement is the movement of the robot from one position to another without changing the robot's facing direction (straight motion). The appearance of this robot can be seen in Figure 1.

Algoritma RRT

RRT was introduced by Steven M. LaValle and James J. Kuffner in 1998, the RRT algorithm is one of the most popular path planning algorithms to the modern time. The key idea is that RRT builds trees using random sampling in the search space.

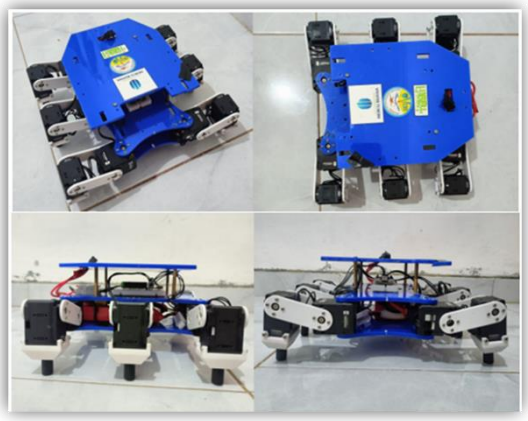

Figure 1. Hexapod Robot

The latest research on the application of RRT is in research [10] which applies the RRT algorithm to unmanned ship.

RRT builds a tree using random sampling in the search space. The RRT tree starts from an initial state let's say q.init and expands to find a path to a goal state let's say q.goal. The tree gradually expands as iterations continue. During each iteration, q.rand is randomly selected in the search space. If the random sample rand z lies in a barrier-free region, then the nearest node (q.near) is based on its euclidean or metric distance. If appropriate, then q.rand is connected to q.near. Otherwise, it generates a new vertex (q.new) that aligns with q.rand and connects it to q.near, as in Figure 2. Then a collision checking process is performed to ensure a collision-free path between the new vertex and the nearest vertex. This process continues until it reaches the specified number of iterations, and or the specified time has expired, and or has reached the destination node or the desired goal is achieved.

Pseduocode RRT Algoritm [18]:

T = (V, E) ← **RRT** (*z init)*
1 *T*← InitializeTree();
2 *T*← InsertNode(Θ, z init, *T);*
3 ***for*** *i=0 to i=N* **do**
4 *z rand* ← Sample*(i);*
5 *z nearest* ← Nearest*(T, z rand);*
6 *(z new , U new)* ← Steer *(Z near, Z rand);*
7 *If* ObstacleFree *(z new)* **then**
8 *T* ← InsertNode *(z min, z new, T);*
9 return *T*

Further detail of some major functions is described as the following:

1. Sample: This function generates a random zrand position from the search space in the free Z obstacle-free region.
2. Nearest: This function returns the nearest node from T = (V, E) to zrand according to the cost function.
3. Steer: This function gives a control input u [0, T] which moves the system from z(0) = z.rand to z(T)= z.near the path z:[0,T] → Z giving znew at a distance Δq from znear to zrand where Δq is the incremental distance.
4. CollisionCheck: This function is used to check for collision detection of tree branches and returns true if it is located in a barrier-free region, i.e., whether the path z: [0, T] lies in the Zfree region for all t=0 to t=T.
5. Near: This function returns the nearest node in the tree.
6. InsertNode: This function adds node znew to V in the tree T = (V, E) to connect node zmin as its parent.

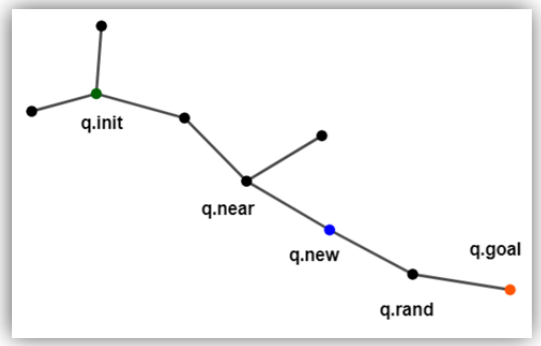

Figure 2. RRT Tree Expansion Process

Algoritma RRT-Connect

The RRT-Connect algorithm that has been suggested [24] incorporates two novel concepts to make up for the RRT algorithm's shortcoming. The first is that the destination and beginning point are both extended in each direction sequentially and included as root nodes. A flaw in the RRT method is that the two trees that extend from the start point and the destination point grow as though they are pulling on one another, preventing trees from growing in the direction that is not determined by the destination. This lengthens the amount of planning time needed to identify a route. The second is the idea of "Extend," which, should there be no collisions with impediments while the tree stretches, keeps going to the other side. This allows the path to be planned more quickly because, in contrast to the RRT algorithm, which increases the maximum extension length as samples are generated and entered into the tree, the tree continues to extend towards the goal if there are no collisions with barriers. way planning using the RRT-Connect algorithm can locate a way more quickly than the RRT algorithm; however, the 'Extend' technique can be challenging and less effective in complicated situations with plenty of barriers and limited paths. Furthermore, the RRT-Connect algorithm-planned path deviates significantly from the ideal length, failing to represent optimality.

Pseduocode RRT-Connect [24]:

CONNECT(*T, z*)
1 repeat
2 *S* ← EXTEND (*T, z*);
3 until not (*S* = *Advanced*);
4 Return *S;*

RRT_CONNECT(*z init, z goal*)
1 *Ta* init (*z init); Tb* init (*z goal*);
2 for *k* = 1 **to** *K* **do**
3 *z rand* ← RANDOM_CONFIG();
4 **if not** (EXTEND(*Ta, z rand*) = *Trupped*) **then**

5 **if** (CONNECT(*Tb, z new*) = *Reached*) **then**
6 Return PATH(*Ta, Tb*);
7 SWAP(*Ta, Tb*);
8 Return Failure

Algoritma RRT*

The RRT* algorithm is a variation of the RRT algorithm developed to improve its performance and ability to find optimal paths [25]. The RRT* algorithm was first proposed by Karaman and Frazolli [17] This algorithm aims to overcome the shortcomings of the RRT algorithm which is not optimal value convergence.

The RRT* algorithm combines sampling techniques with optimization techniques to improve performance and the ability to find optimal paths. The basic principle of RRT* is the same as RRT, but there are two additional steps in the RRT* algorithm that make it better than RRT [17]

Initially, RRT* measures the separation between each vertex and its parent vertex. The node cost is the name given to this. The neighbourhood of vertices within a set radius of the newly discovered vertex is inspected after the closest vertex has been located. The less expensive node replaces the proximal node if one is identified with a lower node cost. The addition of fan-shaped branches to the tree structure illustrates the impact of this feature. RRT's cubic structure is left out.

The rewiring of the tree is the second change that RRT* makes. The neighbouring node is examined once again once a node is linked to the one that is closest in price. It is determined whether reconnecting the nearby node to the recently added node will lower its cost. The neighbour gets reconnected to the newly added node if the cost is actually lower. The path is smoothed out by this feature. The most recent study on ship path planning using RRT* is found in [14]. Figure 3 shows a RRT* Tree Expansion process.

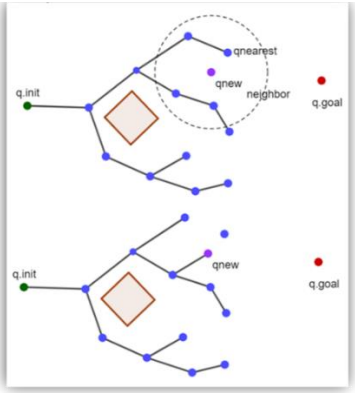

Figure 3. RRT* Tree Expansion Process

The functions in the RRT* algorithm are the same as the previous RRT, but with the addition of the following functions:
1. Rewire: This function checks if the cost to znear incoming nodes is less through znew compared to their old cost, then the parent is changed to z.new.
2. ChooseParent: This function chooses the best parent z.new from nearby nodes.

Pseduocode RRT* [18] :

T = (V, E) ← **RRT*** (*z init)*
1 *T*← InitializeTree();
2 *T*← InsertNode(Θ, z init, *T);*
3 ***for*** *i=0 to i=N* **do**
4 *z rand* ← Sample(*i);*
5 *z nearest* ← Nearest*(T, z rand);*
6 *(z new , U new)* ← Steer *(Z near, Z rand);*
7 *If* ObstacleFree *(z new)* **then**
8 *z near* ← Near*(T, z new, |V|);*
9 *z min* ← Chooseparent *(z near, z nearest, z new);*
10 *T* ← InsertNode *(z min, z new, T);*
11 return *T*

Pose-to-Pose

The principle of Pose-to-Pose is that the results of Path Planning in the form of points or waypoints will be used as input. This involves a series of points or waypoints that must be followed, so that the robot's movement control follows the resulting point by point path planning until it gets to the very end point (Goal or destination) of the path [26]. Waypoint is used to store and/or remember a position-based point from a location on the map [27, 28, 29].

In this research, Pose-to-Pose functions as a translator of the results of Path Planning in the form of Waypoints so that they can be programmed and used to control robot motion. To simplify the orientation, the Pose-to-Pose representation is made on the XY axis. A Robot's representation of the XY axis is represented in Figure 4.

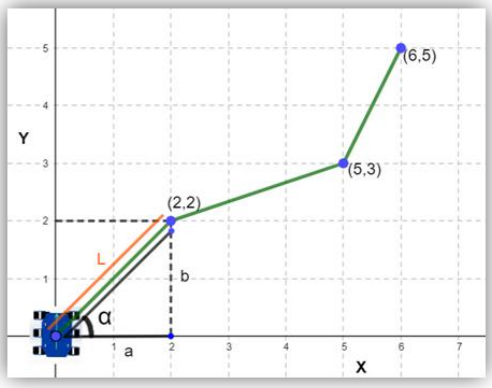

Figure 4. Robot's representation of the XY axis

The green line is assumed to be the result of path planning and the blue dots are the nodes or waypoints. L is the length of the distance between one point to another. α is the angular angle, which is the orientation of the robot's face towards the X axis. The calculation of α and x is described in (1) and (2).

$$L = \sqrt{a^2 + b^2} \quad (1)$$

$$\alpha = \tan^{-1}\left(\frac{b}{a}\right) \quad (2)$$

L is the length of the distance between points, a is the y-axis coordinate point, b is the x-axis coordinate point while α = the angle between the robot's direction based on the x-axis.

Pose-to-Pose motion control consists of rotational motion and translational motion. Rotational motion is controlled using (1), the equation is to determine the angular angle of the robot, so that it can be known how many degrees and in which direction the robot will rotate or rotate. While translational motion is controlled using (2), the equation is to find out how much distance between one point to the next point, so that it can be known how far the robot will move straight. In general, the flowchart sytem of this research is depicted in Figure 5.

RESULTS AND DISCUSSION

This experiment aims to improve the path planning optimality and convergence time. The proposed method is RRT* algorithm with path optimization. There are some problem limitations in this paper, first, the autonomous robot designed and used is a six-legged robot with three DOF in each leg. Second, Implement the triangular inequality method for path optimization of the RRT* algorithm. Third, Path Planning testing is being done with MATLAB simulation. Fourth, Path Planning simulation results is used as Path Tracking Input. Fifth, Implement the concept of Pose-to-Pose approach for Path Tracking. Sixth, the test is carried out on the environmental information that has been given, with a static environment. Seventh, Motion Model is non-dynamic.

In order to make the suggested triangle inequality-based RRT* algorithm more optimum than RRT*, it rewires the path planned by the RRT* algorithm using the notion of triangle inequality between nodes. This study presents a rewiring approach known as the "Triangular-Rewiring" method, which is based on the triangle inequality concept. Figure 6 shows the Triangular-Rewiring process graphically.

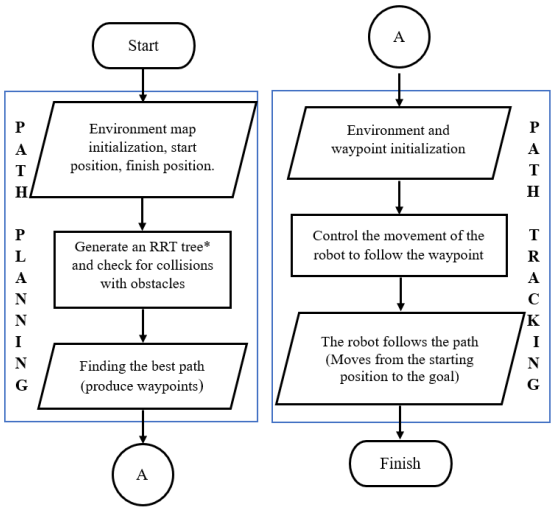

Figure 5. Flowchart System

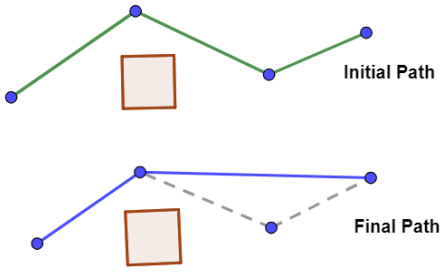

Figure 6. Triangular-Rewiring Process

Pseudocode RRT* Path Optimization

T = (V, E) ← RRT* Path Optimization (*z init)*
1 *T*← InitializeTree();
2 *T*← InsertNode(Θ, z init, *T);*
3 ***for*** *i=0 to i=N* **do**
4 *z rand* ← Sample(*i);*
5 *z nearest* ← Nearest*(T, z rand);*
6 *(z new , U new)* ← Steer *(Z near, Z rand);*
7 *If* ObstacleFree *(z new)* **then**
8 *z near* ← Near*(T, z new, |V|);*
9 *z min* ← Chooseparent *(z near, z nearest, z new);*
10 *T* ← InsertNode *(z min, z new, T);*
11 *T* ← Rewire *(T, z near, z min, z new);*
12 ***if*** InitialPathFound **then**
13 n ← *i*;
14 *(T, directcost)* ← Path Optimization*(T, z init, z goal);*
15 ***if*** *(directcost new < directcostold)* **then**
16 **Z beacons** ← PathOptimization(*T, Z ini, Z goal*);
17 return *T*

To support an ideal comparison, the observations made assume that all knowledge

about the environment is already present in the robot because the robot has already solved the SLAM problem. For testing there are 2 maze-like maps. The difference between the maps is the width of the path, map 1 has a wider path and map 2 has a narrower path.

The difference between maps 1 and 2 is in the level of complexity. Map 1 has wider and easier routes than map 2. But map 1 is larger in dimension at 100x100 while map 2 is 80x80. The green node is the robot's initial point (Start Node) and the red node is the robot's final point (Goal Node). Figure 7 and Figure 8 show the Map 1 and Map 2.

Path Planning testing was carried out by comparing 2 Path Planning algorithms, the RRT* Algorithm with the RRT* Algorithm using Path Optimization in MATLAB software, which was carried out 10 times. The comparison has 4 parameters, which are:

1. Time. Time is the length of time the algorithm is executed in software or a simulation, in this case in MATLAB, if the faster the time then the execution of the algorithm is faster, it is better.
2. Number of iterations. The number of iterations shows the effort in finding a more optimal solution, if the smaller the number of iterations then the effort made to find a solution is low, it is better.

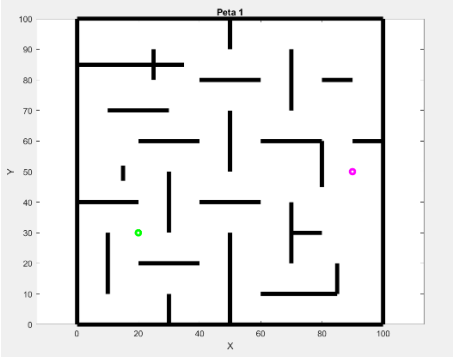

Figure 7. Map 1

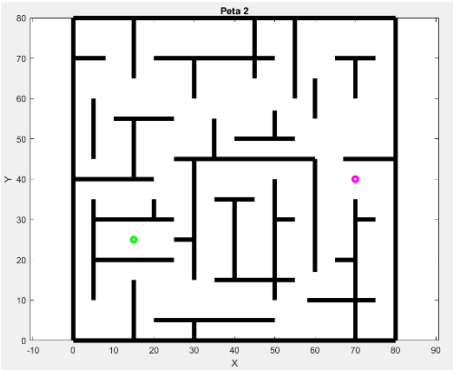

Figure 8. Map 2

3. Distance. Distance is the length of the path resulting from the algorithm's solution and is directly proportional to cost. The cost of the path considered as an estimate of the length of the path traveled in path planning, meaning that if the distance is shorter, it is better.
4. Number of nodes. Many vertices are how many vertices are traveled on the solution path, in its application the number of nodes represents the number of turns to be traveled, the fewer vertices or turns it will make it easier for the robot to reach its destination, it is better.

In map 1, based on the test results in Table 1 and Table 2, it is shown that RRT* with Path Optimization is better than RRT*. Of the 4 parameters, RRT* with Path Optimization is better in 3 parameters, namely the computation time of RRT* Path Optimization is faster, with an average result of 5.72 seconds compared to 10.448 seconds. The number of iterations of RRT* Path Optimization is less with an average result of 130 compared to 222.2. RRT* Path Optimization has fewer vertices with an average result of 11.8 compared to 23.8. While RRT* is only slightly superior in distance, the results of RRT* distance are shorter with an average result of 90.35 compared to 93.33. The data also shows that in general the consistency or stability of the results is better RRT* Path Optimization than RRT*. For the RRT* Path Optimization test data on map 1 can be seen in Table 2.

Table 1. RRT* Data On Map1

No.	Time	Numbers of Iterations	Distance	Number of Nodes
1.	6.73 s	176	93.86	26
2.	39.26 s	592	107	29
3.	8 s	208	84.74	23
4.	5.36 s	142	83.15	22
5.	10 s	181	102.9	27
6.	7.44 s	201	86.45	20
7.	9.9 s	242	88.76	23
8.	7 s	183	79.11	21
9.	7.43 s	195	91.61	24
10.	3.36 s	102	86	23
x̄	10.448	222.2	90.35	23.8

Table 2. RRT* Path Optimization Data On Map 1

No.	Time	Numbers of Iterations	Distance	Number of Nodes
1.	4.22 s	72	94.37	12
2.	3.63 s	99	107,9	13
3.	8.9 s	156	93.99	12
4.	5.92 s	147	88	12
5.	2.79 s	77	74	10
6.	6.97 s	166	94.95	12
7.	2.92 s	79	98.98	11
8.	11.4 s	242	108	15
9.	5.11 s	131	85.2	10
10.	5.39 s	131	88	11
x̄	5.72 s	130	93.33	11.8

The results path of the RRT* and RRT* Path Optimization on map 1can be seen in Figure 9-Figure 12. Figure 9. is the best path of RRT* on Map 1. The path is less smooth (jagged), it is because there are many nodes on the path. This causes the path to be longer. Figure 10 is the best path from RRT* Path Optimization. It can be seen that the path is smoother and there are less nodes. Thus the path will be shorter.

Figure 11 The bad path of RR* in the test, similar to Figure 9. the resulting path is less smooth and jagged, and this path is much longer because it passes through the top unlike the optimal path through the bottom.

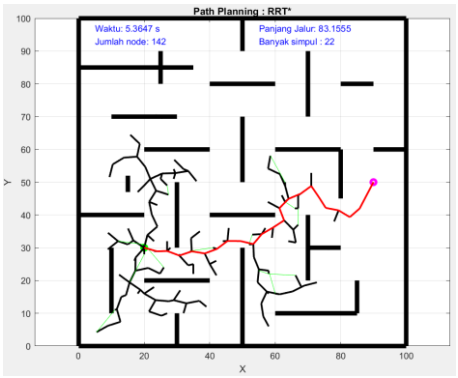

Figure 9. The Best Path of RRT* on Map 1

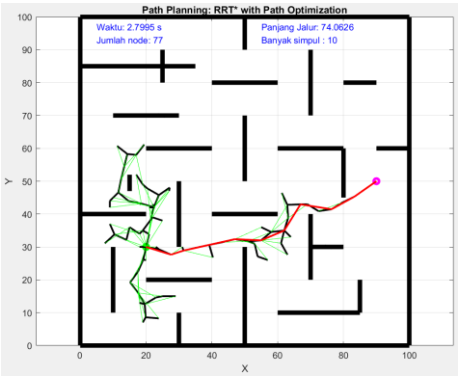

Figure 10. The Best Path of RRT* Path Optimization on Map 1

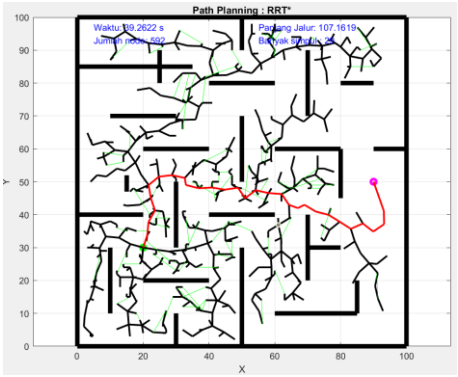

Figure 11. The Bad Path of RRT* on Map 1

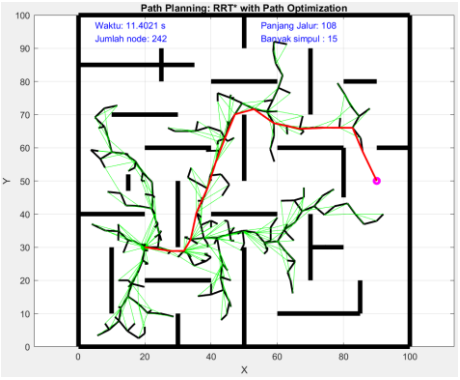

Figure 12. The Bad Path of RRT* Path Optimization on Map 1

Visually, RRT* with Path Optimization looks more smooth, which will make it easier for the robot to follow the path when applied in real or live. And it can be seen that the number of nodes actually represents the smoothness and number of turns on the path. Paths such as RRT* will be more difficult and longer in tracing the path, it is because there are many waypoints that must be computed so that it takes longer.

In map 2, based on the data in Table 3 and Table 4, it is shown that RRT* with Path Optimization is better than RRT*, even in all parameters tested, RRT*, namely RRT* Path Optimization computation time is faster, with an average result of 10 seconds against 12.39 seconds. The number of iterations of RRT* Path Optimization is less with an average result of 186.2 against 227.8. The number of RRT* Path Optimization nodes is less with an average result of 12.2 against 26.3. The path distance result of RRT* Path Optimization is shorter with an average result of 91.96 against 95.76. For the RRT* Path Optimization test data on map 2 can be seen in Table 4. If we just look at the best path, then RRT* is better.

However, when compared with the bad path and the average test results, it can be seen that there is a large gap between the best path of RRT* and the bad path of RRT*.

Table 3. RRT* Data On Map 2

No.	Time	Numbers of Iterations	Distance	Number of Nodes
1.	6.73 s	176	93.86	26
2.	39.26 s	592	107	29
3.	8 s	208	84.74	23
4.	5.36 s	142	83.15	22
5.	10 s	181	102.9	27
6.	7.44 s	201	86.45	20
7.	9.9 s	242	88.76	23
8.	7 s	183	79.11	21
9.	7.43 s	195	91.61	24
10.	3.36 s	102	86	23
$\bar{x}$	10.448	222.2	90.35	23.8

Table 4. RRT* Path Optimization Data On Map 2

No.	Time	Numbers of Iterations	Distance	Number of Nodes
1.	4,22 s	72	94,37	12
2.	3,63 s	99	107,9	13
3.	8,9 s	156	93,99	12
4.	5,92 s	147	88	12
5.	2,79 s	77	74	10
6.	6,97 s	166	94,95	12
7.	2,92 s	79	98,98	11
8.	11,4 s	242	108	15
9.	5,11 s	131	85,2	10
10.	5,39 s	131	88	11
x̄	5,72 s	130	93,33	11,8

When compared to the test results of RRT* Path Optimization, it can be seen that the data generated shows better consistency of results than RRT*.

The results path of the RRT* and RRT* Path Optimization on map 1 can be seen in Figure 13-Figure 16. Figure 13 is the best path of RRT* based on testing. This path is already quite optimal. Figure 14 is the best path of RRT* Path Optimization based on testing. This path is already better than path in Figure 16. Figure 15 is the bad path of RR* based on testing. It can be seen that there is a route selection on the side of Figure 13. In this bad path, the route passes through the top of the map.

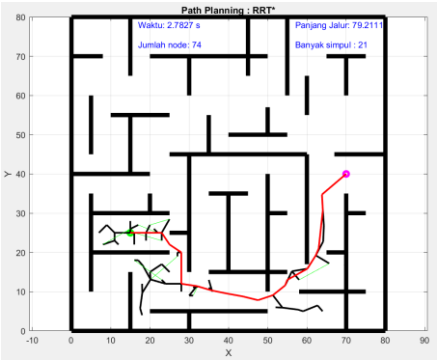

Figure 13. The Best Path of RRT* on Map 2

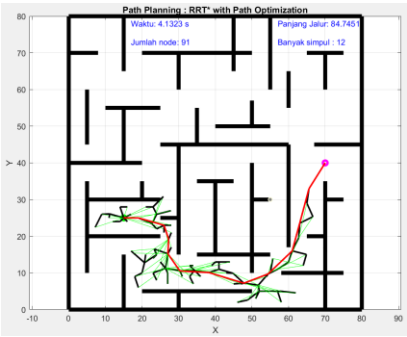

Figure 14. The Best Path of RRT* Path Optimization on Map 2

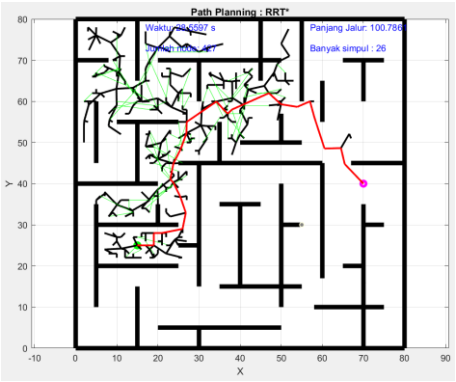

Figure 15. The Bad Path of RRT* on Map 2

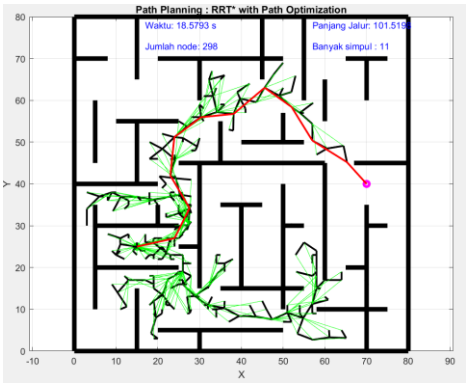

Figure 16. The Bad Path of RRT* Path Optimization on Map 2

Figure 16 is the bad path of RRT* Path Optimization based on testing. It can be seen that there is a route selection on the side of Figure 18. In this bad path, the route passes through the top of the map.

On map 2 with a narrower path, it can be seen that the RRT* path optimization results remain smoother than the RRT* path optimization results.

After the experiment, it can be concluded that there is an increase in the optimality of the path resulting from RRT* path optimization. Then next will be testing the Path Tracking pose-to-pose approach applied to the hexapod robot. The path that will be tracked is the best path resulting from RRT* Path Optimization as shown in Figure 14. Figure 17 shows a pose-to-pose flowchart.

Then the test arena is made based on map 1, but the test arena is made 3x the size of the map for the Path Planning simulation, to 3 meters x 3 meters. Waypoints will be (60, 90), (81, 81), (93, 87), (117, 90), (141, 96), (165, 98), (186, 105), (201, 129), (228, 123), (249, 135), (270, 150). Waypoints in the test are marked with white and green endpoints, as shown in Figure 18.

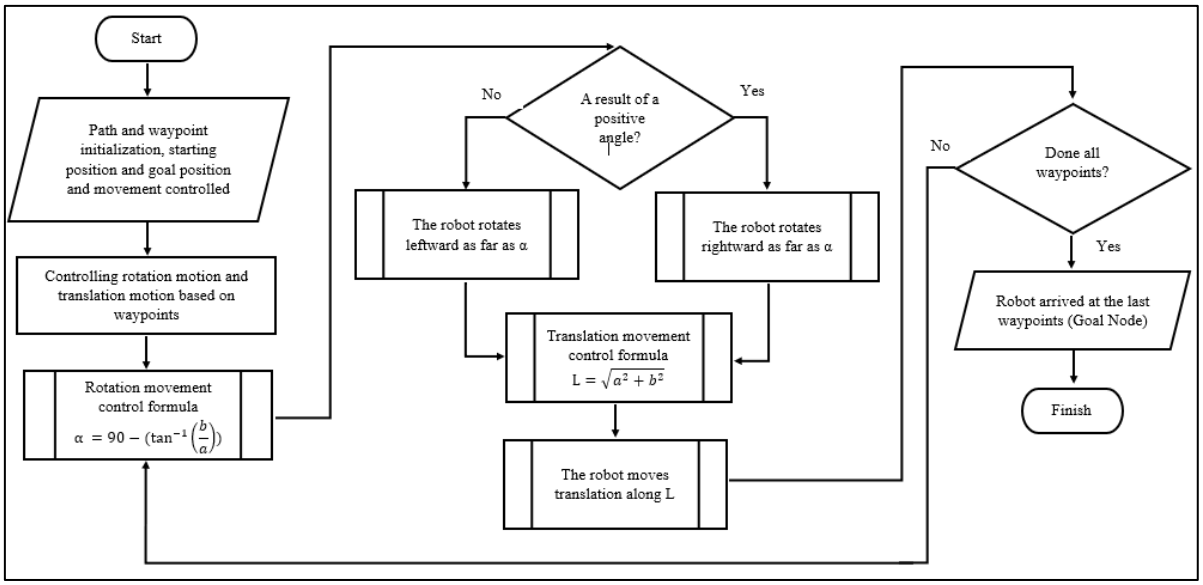

Figure 17. Pose-to-Pose Flowchart

Figure 18. Robot in the arena of testing

The advance speed of the hexapod robot is 23.8 seconds to reach 100 cm or an average of 4.2 cm/s and the rotation speed of the robot is 9 seconds to reach 90° or an average of 10°/s.

Then the result of walking time is obtained as shown in Table 5. Then for the gap between the waypoints and the robot test results are shown in Table 6.

Table 5. Data Robot's Spent Time

Description	Distance	Time
1st to 2nd node	18 cm	19 s
2nd to 3rd node	11 cm	11 s
3rd to 4th node	27.5 cm	10 s
4th to 5th node	25 cm	9 s
5th to 6th node	29 cm	19 s
6th to 7th node	13.9 cm	14 s
7th to 8th node	29.6 cm	11 s
8th to 9th node	28.6 cm	16 s
9th to 10th node	20.2 cm	11 s
10th to 11th node	31.6 cm	10 s
Total	234.4 cm	130 s

Table 6. (X, Y) position data between waypoints and test

Node	Waypoints	Test Result	Gap (X,Y)	Gap Distance
1	(60, 90)	(60, 90)	(0,0)	0 cm
2	(81, 81)	(77, 84)	(4, 3)	5 cm
3	(93, 87)	(88, 83)	(5, 4)	6,4 cm
4	(117, 90)	(114, 92)	(3, 2)	3,6 cm
5	(141, 96)	(139, 93)	(2, 3)	3,6 cm
6	(165, 98)	(168, 94)	(3, 4)	5 cm
7	(186, 105)	(181, 99)	(5, 6)	7,8 cm
8	(201, 129)	(197, 124)	(4, 5)	6,4 cm
9	(228, 123)	(225, 118)	(3, 5)	5,8 cm
10	(249, 135)	(244, 125)	(5, 10)	11, 1 cm
11	(270, 150)	(274, 135)	(4,15)	15, 5 cm

Based on the data in Table 6, the gap between waypoints and test results is the lowest at 3.6 cm and the longest at 15.5 cm. The distance is calculated by finding the Euclidean distance between the waypoints and the test result points using (2). Node one is not considered because it is the starting point of the robot where the robot is placed exactly the same as the waypoints.

As a result, there is a gap between the x,y position of the waypoints and the test results. The gap between the x coordinate nearest is 2, the longest is 5. While the gap between the y coordinate nearest is 2, the longest is 15. This can be caused by several factors, such as a design that is not symmetrical, the condition of the testing ground that is not perfectly flat, and does not apply body kinematic to the robot and does not use sensors. So when there is So when there is a slight deviation, it will gradually become more and more deviant because there is no feedback for the robot.

CONCLUSION

The conclusion of this research is that there is an increase in the optimality of the path resulted by RRT* Path Optimization and it is proven to be better than RRT*. RRT* with Path Optimization excels in all parameters tested, such as time, number of iterations, distance and number of nodes. The application of Pose-to-Pose on the hexapod robot successfully makes the hexapod robot follow the path that has been resulted by Path Planning RRT* Path Optimization nicely without crashing. With the nearest distance gap of 3.6 cm and the longest distance gap of 15.5 cm to the waypoints. Thus, it can be concluded that in general the application of Path Planning and Path Tracking on the 3 DOF hexapod robot is successful and there is an increase in the performance of the hexapod robot in terms of mobility and navigation to be more optimal and faster time to reach the destination point. The robot travel time is 130 seconds for a path length of 234.4 cm.prospects of further studies into the next (based on result and discussion) can also be added.

ACKNOWLEDGMENT

This research is supported by the Center for Research and Community Service of Universitas Mercu Buana, with funding from the international cooperation research scheme.

REFERENCES

[1] F. Ferro, F. Nardi, S. Cooper, and L. Marchionni, “Robot control and navigation: ARI’s autonomous system,” *29th IEEE International Conference on Robot & Human Interactive Communication ROMAN-2020, 2020*, pp. 1-3, doi: 10.36227/techrxiv. 14350571.v1

[2] F. Gul, W. Rahiman, and S. S. Nazli Alhady, “A comprehensive study for robot navigation techniques,” *Cogent Engineering*, vol. 6, no. 1. Cogent OA, Jan. 01, 2019, doi: 10.1080/23311916.2019.1632046.

[3] H. Liu, “Rail transit robots in manufacturing,” in *Robot Systems for Rail Transit Applications*, Elsevier, pp. 1–36, 2020, doi: 10.1016/B978-0-12-822968-2.00001-2.

[4] H. Y. Zhang, W. M. Lin, and A. X. Chen, “Path planning for the mobile robot: A review,” *Symmetry (Basel)*, vol. 10, no. 10, 2018, doi: 10.3390/sym10100450.

[5] N. A. I. Ruslan, N. H. Amer, K. Hudha, Z. A. Kadir, S. A. F. M. Ishak, and S. M. F. S. Dardin, “Modelling and control strategies in path tracking control for autonomous tracked vehicles: A review of state of the art and challenges,” *Journal of Terramechanics*, vol. 105. Elsevier Ltd, pp. 67–79, Feb. 01, 2023, doi: 10.1016/j.jterra.2022.10.003.

[6] J. Gao, Y. Zheng, K. Ni, Q. Mei, B. Hao, and L. Zheng, “Fast path planning for firefighting UAV based on A-star algorithm,” in *Journal of Physics: Conference Series*, IOP Publishing Ltd, Sep. 2021, doi: 10.1088/1742-6596/2029/1/012103.

[7] F. A. Raheem and U. I. Hameed, “Path Planning Algorithm using D* Heuristic Method Based on PSO in Dynamic Environment,” *American Scientific Research Journal for Engineering*, vol. 49, no. 1, pp. 257-271, 2018.

[8] A. Ubaidillah and H. Sukri, “Application of Odometry and Dijkstra Algorithm as Navigation and Shortest Path Determination System of Warehouse Mobile Robot,” *Journal of Robotics and Control (JRC)*, vol. 4, no. 3, 2023, doi: 10.18196/jrc.v4i3.18489.

[9] Q. Zhang, L. Li, L. Zheng, and B. Li, “An Improved Path Planning Algorithm Based on

RRT," in *2022 11th International Conference of Information and Communication Technology (ICTech))*, IEEE, Feb. 2022, pp. 149–152. doi: 10.1109/ICTech55460.2022.00037.

[10] Z. Li, L. Li, W. Zhang, W. Wu, and Z. Zhu, "Research on Unmanned Ship Path Planning based on RRT Algorithm," in *Journal of Physics: Conference Series*, Institute of Physics, 2022. doi: 10.1088/1742-6596/2281/1/012004.

[11] Y. Shi, Q. Li, S. Bu, J. Yang, and L. Zhu, "Research on Intelligent Vehicle Path Planning Based on Rapidly-Exploring Random Tree," *Mathematical Problems in Engineering*, vol. 2020, 2020, doi: 10.1155/2020/5910503.

[12] L. Jin, H. Chaowei, and P. Minqiang, "Path Planning Algorithms for Self-Driving vehicle based on improved RRT-Connect," *Transportation Safety and Environment*, vol. 5, no. 3, Jun. 2022, doi: 10.1093/tse/tdac061.

[13] H. Jin, W. Cui, and H. Fu, "Improved RRT-Connect Algorithm for Urban low-altitude UAV Route Planning," in *Journal of Physics: Conference Series*, IOP Publishing Ltd, Jun. 2021, doi: 10.1088/1742-6596/1948/1/012048.

[14] T. T. Enevoldsen and R. Galeazzi, "Grounding-aware RRT* for path planning and safe navigation of marine crafts in confined waters," in *IFAC-PapersOnLine*, Elsevier B.V., 2021, pp. 195–201. doi: 10.1016/j.ifacol.2021.10.093.

[15] I. Noreen, A. Khan, and P. Zulfiqar Habib, "Optimal Path Planning using RRT* based Approaches: A Survey and Future Directions," *International Journal of Advanced Computer Science and Applications(IJACSA)*, vol. 7, no. 11, pp. 97-107, 2016, doi: 10.14569/IJACSA.2016.071114.

[16] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, San Francisco, CA, USA, 2000, pp. 995-1001 vol. 2, doi: 10.1109/ROBOT.2000.844730.

[17] S. Karaman and E. Frazzoli, "Sampling-based Algorithms for Optimal Motion Planning," *The International Journal of Robotics Research*, vol. 30, no. 7, May 2011, doi: 10.1177/0278364911406761

[18] I. Noreen, A. Khan, and Z. Habib, "A Comparison of RRT, RRT* and RRT*-Smart Path Planning Algorithms," *International Conference on Mechatronics and Automation (ICMA)*, 2016, doi: 10.1109/ICMA.2012.6284384

[19] M. Samuel, M. Maziah, M. Hussien, and N. Y. Godi, "Control of Autonomous Vehicle Using Path Tracking: A Review," *Advanced Science Letters*, vol. 24, no. 6, pp. 3877–3879, Apr. 2018, doi: 10.1166/asl.2018.11502.

[20] H. Wu, H. Zhang, and Y. Feng, "MPC-Based Obstacle Avoidance Path Tracking Control for Distributed Drive Electric Vehicles," *World Electric Vehicle Journal*, vol. 13, no. 12, Dec. 2022, doi: 10.3390/wevj13120221.

[21] S. Tippannavar, Y. S D, H. R, and S. Jain, "Stanley Controller based Autonomous Path planning and Tracking in Self-Driving Cars," *International Journal of Innovative Research in Advanced Engineering*, vol. 10, no. 03, pp. 40–48, Mar. 2023, doi: 10.26562/ijirae.2023.v1003.01.

[22] H. Wang, G. Li, J. Hou, L. Chen, and N. Hu, "A Path Planning Method for Underground Intelligent Vehicles Based on an Improved RRT* Algorithm," *Electronics (Switzerland)*, vol. 11, no. 3, Feb. 2022, doi: 10.3390/electronics11030294.

[23] D. Wang, S. Zheng, Y. Ren, and D. Du, "Path planning based on the improved RRT∗ algorithm for themining truck," *Computers, Materials and Continua*, vol. 71, no. 2, pp. 3571–3587, 2022, doi: 10.32604/cmc.2022.022183.

[24] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, San Francisco, CA, USA, 2000, pp. 995-1001 vol.2, doi: 10.1109/ROBOT.2000.844730.

[25] H. Suwoyo, A. Adriansyah, J. Andika, A. U. Shamsudin, 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, 2023, doi: 10.31436/iiumej. v24i1.2529.

[26] W. K. Yousif and A. A. Ali, "Simulation of Pose to Pose Moving of the Mobile Robot with Specified GPS Points," *Journal of Engineering*, vol. 26, no. 11, pp. 195–208, Nov. 2020, doi: 10.31026/j.eng.2020.11.13.

[27] A. A. Ashari, E. Setiawan, and D. Syauqi, "Sistem Navigasi Waypoint Pada Robot Beroda Berdasarkan Global Positioning System Dan Filter Kalman," *Jurnal*

Pengembangan Teknologi Informasi Dan Ilmu Komputer, vol. 4, no. 7, pp. 2075-2082, 2020.

[28] N. Lilansa, M. N. Rizal, P. Anggraeni, N. J. Ramadhan,"Implementation consensus algorithm and leader-follower of multi-robot system formation," *SINERGI*, vol. 27, no. 1, 2023, pp. 45-56, doi: 10.22441/sinergi.2023.1.006

[29] F. H. Kristanto & Z. Iklima,"Collision avoidance of mobile robot using Alexnet and NVIDIA Jetson Nano B01," *Journal of Integrated and Advanced Engineering (JIAE),* vol. 4, no. 1, pp. 9-20, 2024, doi: 10.51662/jiae.v4i1.118