2nd International Conference on Engineering, Natural Sciences, and Technological Developments (ICENSTED 2025)

June 20-23, 2025

Obstacle Avoidance Path Planning for Industrial Robotic Manipulators Using Improved RRT with Bezier and Linear Interpolations

Osama Altalmas[1], Beste Bahceci[1]

[1]Mechatronics Engineering, Bahcesehir University, Istanbul, Türkiye
Corresponding author: Osama Altalmas (e-mail: osamath.altalmas@bahcesehir.edu.tr)

Abstract

Path planning is a fundamental challenge in robotic motion planning, especially in dynamic situations where obstacles must be effectively avoided. This paper rigorously examines and compares three path-planning algorithms—standard rapidly exploring random tree (RRT), rapidly exploring random tree* (RRT*), and an improved RRT algorithm—with a particular emphasis on computational efficiency, path smoothness, and adaptability. The Rapidly Exploring Random Tree algorithm is executed and its performance is evaluated using MATLAB software by calculating the computation time to reach the goal, path length, and visualization of the exploration process illustrating the avoidance of obstacles. Through the use of interpolation, path shortening, and goal biasing, the improved RRT method is executed and simulated. The results indicate that the improved RRT algorithm generates smooth, collision-free paths while minimizing the path length efficiently and reducing the computation time.

Keywords: Path planning, RRT algorithm, Motion planning, Computation time, Trajectory optimization

1. INTRODUCTION

Industrial robotic manipulators have achieved fundamental significance during modern times for intelligent factories because they enable safe human-machine teamwork when performing complex tasks [1]. Industrial robots operate at high speeds by handling tasks jointly with human operators and machines as they adjust to encountered obstacles and operational requirements [1–5]. The fundamental obstacle in robotic human cooperation requires a system that enables manipulators to automatically avoid fixed or moving objects during their movement [3]. Path planning creates collision-free routes which use distance minimization, smoothness optimization, energy-saving functions, and risk reduction depending on application requirements [4]. A robotic manipulator requires this ability for successful deployment within industrial environments [2–6].

The rapidly exploring random tree (RRT) algorithm creates its position as one of the best path-planning algorithms since it can achieve obstacle avoidance with probabilistic completeness and universal usage capabilities [3]. RRT demonstrates strong capability in fast exploration of large search spaces while managing high-dimensional problems and it generates collision-free paths across complex and uncertain environments [4]. RRT constructs its tree through random space sampling which leads toward the goal point while steering clear of obstacles [5]. During each iteration RRT chooses a random point which it connects to its nearest accessible node before further expansion until path discovery or time/iteration limit is reached [6]. Continuous worldwide use of the Standard RRT algorithm faces multiple implementation challenges due to its insufficient convergence performance and the inability to reach tight spaces effectively in addition to irregular path generation [5, 6]. In contexts where impediments evolve at a rapid pace, the implementation of these constraints assumes absolute importance for real-time manipulator adjustments [6–8]. While standard RRT works well in static settings, it has issues in unstructured dynamic settings since random sampling leads to inefficiencies, particularly in high-dimensional spaces with narrow entry points, the convergence rate is a significant difficulty [9, 10]. When trajectories involve abrupt shifts that hinder real-time operations, uneven paths become apparent [9–11]. Since it lacks an effective mechanism to recalculate pathways when new barriers shift or arise unexpectedly, the algorithm performs poorly when faced with dynamic obstacles [11]. Extensive sampling and collision testing significantly increases the computational load, which makes real-time operations impracticable, particularly in robotic systems with multiple degrees of freedom [7, 8].

Rapidly exploring random tree* (RRT*) represents an advanced version of RRT which conducts optimized tree expansion through selecting the least expensive parent node to establish an improved smooth path structure for navigation applications [7]. It proves optimal for robotic manipulators operating in dynamic environments because

it improves their ability to handle obstacles during navigation toward the target of contemporary smart production systems [9]. Moreover, RRT* suffers from disadvantages which include high computational complexity from numerous tree recalculations together with reduced path optimization speed and parameter-based sensitivity causing inefficiencies [7]. Real-time operations could prove difficult for RRT* since it uses substantial memory resources which leads to resource limitations in certain systems [7–9].

The research aims to develop an improved RRT algorithm to enhance path planning for robotic manipulators in dynamic unstructured spaces. The improved RRT algorithm brings optimized parent node selection abilities to RRT* while cutting down search path detours and overall costs. The method uses goal biasing to accelerate convergence while maintaining efficient exploration of the environment. Trajectory smoothness as well as memory efficiency improve through the integration of linear and Bezier interpolation features in the algorithm that deliver straight path interpolation through lines and curved path interpolation through Bezier curves. Better high-dimensional exploration occurs since novel sampling and adaptive step size control improve convergence speed. Real-time operation requires the algorithm to create seamless smooth paths that resolve the sudden path interruptions of the standard RRT. The obstacle-handling system adjusts automatically to both moving obstacles and human interferences. The improved method is simulated through MATLAB and results indicate that the improved RRT algorithm has enhanced performance capabilities.

2. ROBOT KINEMATICS AND PATH PLANNING APPROACH

This study concentrates on an efficient robotic manipulator path planning algorithm through the combination of sampling-based motion planning methods with suitable kinematic models. The OpenManipulator-X robot is employed in this work, because of its research-supporting motion planning capabilities and its open-source design along with its lightweight composition. A path-planning approach is studied which uses goal direction optimization through an improved RRT algorithm. The following section consists of an explanation of OpenManipulator-X's kinematic model alongside details regarding better RRT implementation for developing collision-free robot motion paths.

2.1. OpenManipulator-X Robot

A widely recognized research and teaching robotic manipulator known as OpenManipulator-X appears in Figure 1 is used in this research. OpenManipulator-X continues to be employed in robotic applications because of its accurate adaptable features along with modular design and suitability for trajectory planning and control algorithm development with manipulation applications. The robotic arm has four degrees of freedom (4-DOF) which enables it to complete numerous robotic tasks accurately [12, 13]. Each of the four rotating joints functions to enable the end effector to position itself at points across three dimensions. The design allows the robot to reach operational levels that match practical and educational requirements [12]. A PD controller operates the manipulator to produce steady smooth motion [13].

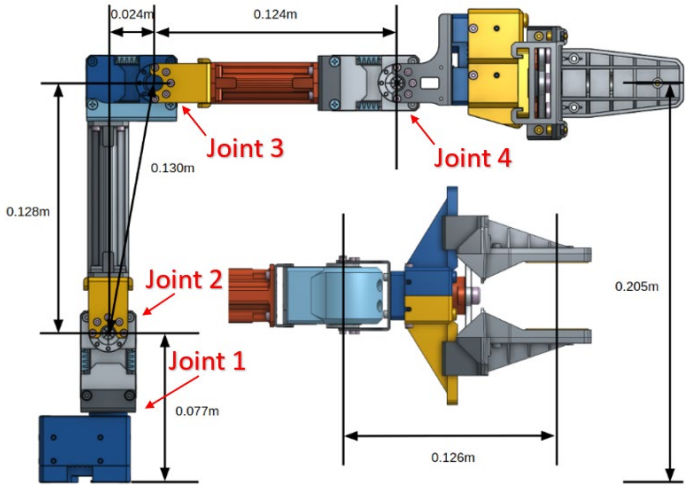


Figure 1. OpenManipulator-X model [13]

Through a standard method that describes link-joint behavioral correlations, the OpenManipulator-X defines its kinematics using the modified Denavit-Hartenberg (DH) convention [12–14]. The model reduces calculation complexity for both forward and inverse kinematics by grouping the robotic system into separate matrix sections.

Each robot joint requires four parameters to create a model under the DH convention using link length and link twist with joint offset and joint angle. The systematic application of these parameters leads to end-effector position and orientation calculation resulting in complete spatial descriptions of link interconnections. Table 1 provides the DH parameter presentation for the robot manipulator.

Table 1. Modified DH [14]

Joint	α_i (degree)	α_i (m)	d_i (m)	θ_i (degree)
1	90	0	0.077	θ_1
2	0	0.130	0	$\theta_2 - \theta_0$
3	0	0.124	0	$\theta_3 + \theta_0$
4	0	0.126	0	θ_4

Where α_i is the link twist, α_i is the link length, d_i is the link offset, and θ_i is the joint angle. Forward kinematics involves determining the position and orientation of the manipulator's end effector based on the given joint parameters. By sequentially applying the Denavit-Hartenberg (DH) parameters for each joint, transformation matrices are derived that describe the relationship between consecutive links. These matrices are then multiplied to compute the overall transformation from the base frame to the end-effector frame. The general transformation matrix is shown below:

$$ {}^{i-1}_{i}T = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \cos\alpha_i & \sin\theta_i \sin\alpha_i & \alpha_i \cos\theta_i \\ \sin\theta_{i0} & \cos\theta_i \cos\alpha_i & -\sin\alpha_i \cos\theta_i & \alpha_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1} $$

Combining variables and parameters with the same subscripts in Table 1, the homogeneous transformation matrix among the four-link coordinate systems ${}^{0}_{1}T$ is the transformation matrix of frame 1 relative to frame 0, ${}^{1}_{2}T$ is the transformation matrix of frame 2 relative to frame 1, ${}^{2}_{3}T$ is the transformation matrix of frame 3 relative to frame 2, and ${}^{3}_{4}T$ is the transformation matrix of frame 4 relative to frame 3 can be obtained.

$$ {}^{0}_{4}T = {}^{0}_{1}T \times {}^{1}_{2}T \times {}^{2}_{3}T \times {}^{3}_{4}T = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2} $$

where, (n_x, n_y, n_z), (o_x, o_y, o_z) and (a_x, a_y, a_z) are the initial frame orientation vector and the value p_x, p_y, p_z are the coordinates of the end-effector. The representative method for robot arm inverse kinematics features a closed-form solution that integrates both geometric and algebraic approaches. The solution is divided into two distinct parts. Robot arm movement on the *x-y* plane begins with determining the angle of joint 1. A calculation for θ_1 angle depends on the robot arm motion projection onto the *x-y* plane which can be calculated as follows:

$$ \theta_1 = \tan^{-1}\left(\frac{p_y}{p_x}\right). \tag{3} $$

In order to make calculating joint angles simpler, the end effector positions are frequently projected onto the *r-z* plane. Determination of the radial distance *r* represents the basis of the calculation since it compiles the end effector's *x* and *y* coordinates as shown in Figure 2. The offset d_1 of the first joint exists in the *z*-axis range with z_3 representing the vertical displacement are both illustrated in Figure 3. The projections simplify the process of finding joint angles 2, 3, and 4 because they reveal the two-dimensional spatial relationships between links and desired locations.

$$ r = \sqrt{p_x^2 + p_y^2}, \tag{4} $$

$$ z_3 = p_z - d_1. \tag{5} $$

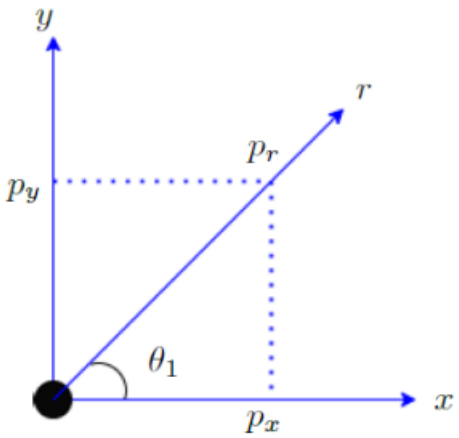


Figure 2. Combination of *x* and *y*-axis as *r*-axis [14]

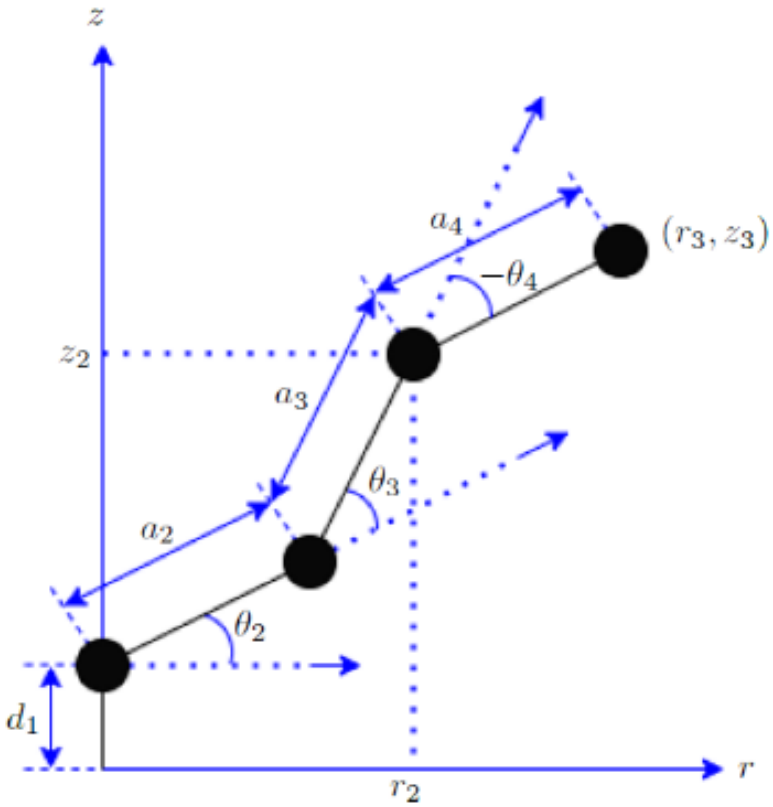


Figure 3. Projection of the r-z surface for the movement of the robot arm [14]

The angles for joints 2, 3, and 4 require geometric and trigonometric relationships to calculate their positions which align the links at the desired end-effector position. The cosine rule enables the calculation of θ_3 (Joint 3) angle by defining the link distance relationship according to the robot.

$$\cos\theta_3 = \frac{r^2 + z_2^2 - \left(a_2^2 + a_3^2\right)}{2a_2a_3}. \tag{6}$$

The position of θ_2 (Joint 2) depends on a combination of radial and vertical end effector measurements which incorporates the influence of joint 3.

$$\cos\theta_2 = \frac{\left(a_2 + a_3\cos\theta_3\right)r_2 + \left(a_3\sin\theta_3\right)z_2}{r_2^2 + z_2^2} \tag{7}$$

The proposed system requires joint 4 as the final angle to connect the end effector properly with its goal orientation, where $\varnothing$ represents the total angle required for the end-effector to reach its target orientation.

$$\theta_4 = \varnothing - (\theta_2 + \theta_3) \tag{8}$$

2.2. Improved RRT Algorithm

The improved RRT Algorithm implemented for this study operates through multiple performance optimization features which enhance efficiency alongside cost reduction and smoother path operations. The refined parenting process develops advanced relationship analysis by allowing new nodes to review multiple potential parents from a designated search area instead of picking one through the nearest proximity. Moreover, it provides the algorithm with the ability to construct effective path structures and eliminate unnecessary detours by selecting the parent node with minimal cost-to-come. The execution of the RRT algorithm for controlling tree growth and creating a safe path to the objective involves multiple vital mathematical procedures. First, the following formula is used to create a random sample x_{rand} (random node) inside the designated workspace bounds:

$$x_{\text{rand}} = \begin{bmatrix} x_{\min} + (x_{\max} - x_{\min}) \cdot \text{rand} \\ y_{\min} + (y_{\max} - y_{\min}) \cdot \text{rand} \\ z_{\min} + (z_{\max} - z_{\min}) \cdot \text{rand} \end{bmatrix}. \tag{9}$$

Where min and max refer to the minimum and maximum values along x, y, z axes. A random number ranging between 0 to 1 with uniform distribution emerges through the "rand" function. The MATLAB program adds additional constraints for sampling to avoid sampling in restricted areas close to obstacles. The distance measure in the approach relies on Euclidean distance for locating the nearest existing node x_{near} (nearest node) in the tree following random sampling:

$$x_{\text{near}} = \arg^{\min}_{q_i \in T} \left\| x_{\text{rand}} - x_i \right\| \cdot \tag{10}$$

Where T denotes the set of all nodes that are currently part of the tree and x_i represents the position of a node in the tree. While maintaining the specified development pattern, the closest node selection ensures that the tree expands successfully towards undiscovered areas. Using the following equation, the tree grows in incremental steps rather than simply connecting x_{rand} (random node) to x_{near} (nearest node):

$$x_{\text{new}} = x_{\text{near}} + \lambda \cdot \frac{(x_{\text{rand}} - x_{\text{near}})}{\left\| x_{\text{rand}} - x_{\text{near}} \right\|}. \tag{11}$$

Where the step size is represented by λ (EPS in MATLAB). This method avoids big leaps that might avoid obstructions or make the path unstable. The algorithm discretizes the straight-line segment between x_{near} and x_{new} into several intermediary points in order to check for collisions before adding x_{new} to the tree. These points are produced using:

$$\text{EdgePoints} = \begin{bmatrix} \text{linspace}(x_{\text{near},1}, x_{\text{new},1}, N) \\ \text{linspace}(x_{\text{near},2}, x_{\text{new},2}, N) \\ \text{linspace}(x_{\text{near},3}, x_{\text{new},3}, N) \end{bmatrix}. \tag{12}$$

Where "linspace" is the function that generates a sequence of evenly spaced points between two given values. A collision detection function compares each of these points against obstructions. The new node is discarded if any of them fall inside the safety margin of an obstacle. The algorithm uses the following equation to determine whether a newly added node is within a predetermined threshold ε epsilone from the goal:

$$\left\| x_{\text{new}} - x_{\text{goal}} \right\| < \varepsilon \cdot \tag{13}$$

Where x_{new} is the new node and x_{goal} is the goal position. Then a refined parenting mechanism is applied, where new nodes evaluate multiple potential parent nodes within a defined parenting search radius. By selecting the parent that minimizes the cost-to-come, the algorithm ensures a more efficient path structure and reduces unnecessary detours. The cost function is calculated as:

$$\text{Cost}(x_{\text{new}}) = \text{Cost}(x_{\text{parent}}) + \left\| x_{\text{new}} - x_{\text{parent}} \right\| \cdot \tag{14}$$

The algorithm determines x_{parent}, parent node is the node in the tree from which a new node is generated among various options by selecting the one offering the minimum overall cost. The cost function gets implemented in the MATLAB code during the parent selection process to ensure nodes establish connectivity through the path with minimal cost. A predefined measurement called λ determines the radius search value within the developed code, λ is set to be 0.15 because a fixed radius eliminates the need to dynamically compute the optimal radius at each iteration, reducing the overall computational burden and improving real-time performance. Also, by maintaining a uniform radius, it ensures that new nodes are consistently connected to their nearest parent nodes within a predefined distance, avoiding unnecessary complexity in the tree structure, and allowing for controlled growth of the tree, ensuring efficient exploration while still promoting directed expansion towards the goal. The improved RRT process produces more affordable solution paths than the default RRT* method without compromising the effective exploration guided by the algorithm. The goal biasing mechanism samples point near the goal position with a 5% chance to accelerate convergence because it generates occasional goal-proximity samples. The goal-biasing method uses the following approach to achieve its purpose:

$$x_{\text{rand}} = x_{\text{goal}} + 0.05 \cdot N(0,1) \cdot \tag{15}$$

Where $N(0,1)$ represents a standard normal distribution. The usage of this strategy improves the chances of obtaining samples that direct the tree toward the target point which shortens processing time. A step size of 5 cm operates as a control mechanism for tree expansion to keep jumps from becoming excessive and damaging precision. The algorithm advances by efficient tree growth but also ensures tree rewiring where existing nodes perform constant assessments of new nodes that might reduce their path costs. The parent node updates to new connections through new nodes when these connections present lower costs to improve trajectory results. After establishing a safe collision-free path the algorithm conducts forceful path reduction operations by studying direct connections of non-contiguous nodes to eliminate superfluous waypoints. These direct connections become feasible through an evaluation of the direct Euclidean distance between the points against the cumulative distance of the original segment:

$$d_{\text{direct}} = \left\| x_i - x_j \right\|, \tag{16}$$

$$d_{\text{cumulative}} = \sum_{k-i}^{j-1} \left\| x_k - x_{k+1} \right\| \cdot \tag{17}$$

Where, x_i and x_j represents any two nodes in the path, and x_k represents an intermediate node between x_i and x_j. The removal of intermediate waypoints becomes possible when a direct connection shows no collisions and leads to minimized distance which consequently reduces the overall route length. The algorithm enhances the trajectory by using linear interpolation combined with Bezier curve interpolation to guarantee smooth transitions and to maintain low memory consumption. The system applies linear interpolation to generate flowing movements between waypoints when direct lines are possible. Through this formula, the algorithm establishes the interpolated point x_{interp} that lies between waypoints x_{start} (initial position) to x_{end} (goal position):

$$x_{\text{interp}} = x_{\text{start}} + \alpha \left(x_{\text{end}} - x_{\text{start}} \right) \cdot \tag{18}$$

Moving along the segment depends on the interpolation fraction α where this value is limited between 0 and 1. Smooth motion occurs through this approach while also preventing irregular direction changes. Bezier interpolation is used to produce seamless, continuous transitions between waypoints for curved portions of the trajectory [15]. The weighted Bernstein polynomial function is used to create the Bezier curve:

$$t = \sum_{j=0}^{n} \omega_j p_j B(n, j, t) \cdot \tag{19}$$

The equation consists of control points p_j together with weights ω_j and Bernstein basis polynomials $B(n,j,t)$:

$$B(n, j, t) = \frac{n!}{j!(n-j)!} t^j (1-t)^{(n-j)} \cdot \tag{20}$$

Where n is the degree of the Bezier curve, j is the index of the control point, and t is the parametric variable (ranges from 0 to 1). The size of the curved area is set to 5% of the segment prior to waypoints in the middle of the path and 5% superior to the same nodes, ensuring smoother rotations through transition points. Each Bezier

interpolation generates 10 nodes, reducing abrupt changes in motion and making the final trajectory more feasible for real-world robotic execution.

3. RESULTS AND DISCUSSION

A MATLAB-designed three-dimensional occupancy map incorporated four spherical obstacles together with one box-shaped obstacle as shown in Figure 4. A 7.5×7.5×7.5 cm^3 box-shaped obstacle exists among five spherical obstacles with each having a 5cm radius. All obstacles maintain a position around the robot while the robot rests at coordinates (0, 0, 0). The map sets a specific green-colored starting location which signifies the robot's end-effector beginning position. The red-colored goal position establishes the destination point that the robot end-effector must reach at the end of its route. The testing and evaluation of path-planning algorithms become possible through this experimental setup which provides complete operational environment representation of robotic systems.

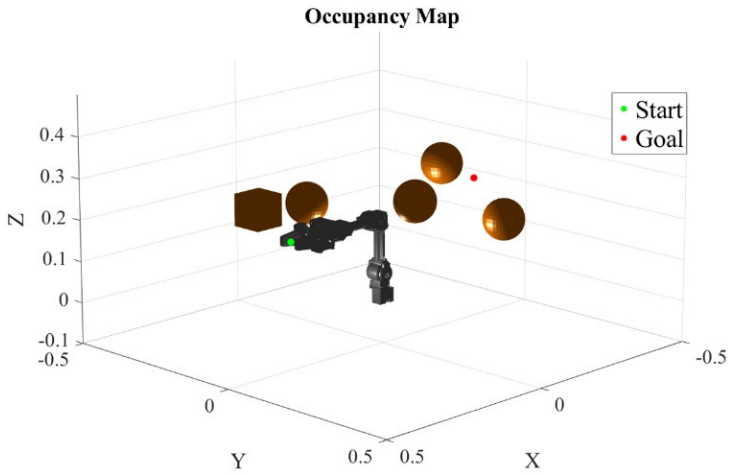


Figure 4. Occupancy map. Where the green dot represents the starting position, while the red dot indicates the goal position

An improved RRT algorithm simulation (Figure 5) investigated its ability to produce optimized smooth trajectories while Figure 6 shows the resulting smooth path. The simulation tracks the behavior of the algorithm by measuring time duration, total path distance, and minimum obstacle distance along with the number of iterations to find the goal and the number of nodes in the path. Starting from a preset start and goal position and using 5 cm steps controlled the systematic exploration of the tree expansion, after path smoothing the original four waypoints expanded into twenty-three waypoints. The improved RRT algorithm achieved the goal within 3.5958 seconds after executing 31 iterations as it proved highly efficient in exploration. The final optimized trajectory from path-shortening and path-smoothing measures reaches a total length of 0.48545 meters.

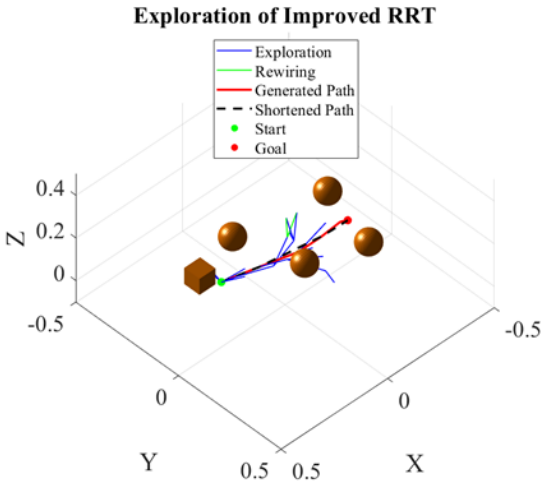


Figure 5. The exploration process of improved RRT algorithm. Where the green dot represents the start position of the robot, while the red dot indicates the goal position. Various paths are shown: exploration (blue), rewiring (green), the generated path (red), and the shortened path (black dashed line)

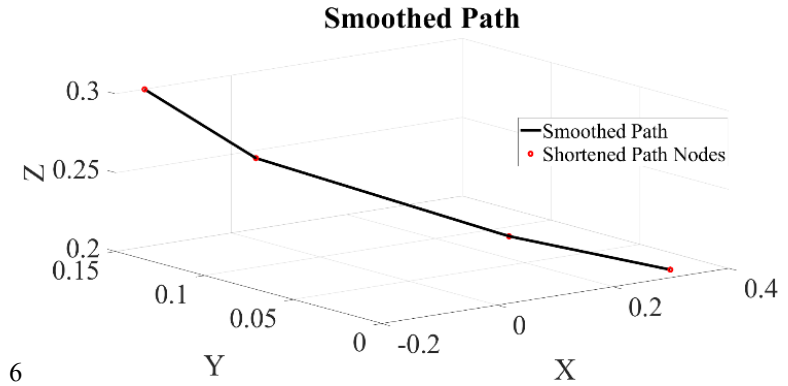


Figure 6. The smoothed path of improved RRT. Where red dots indicate the shortened path nodes and the black line indicates the smoothed path

A 3D simulation was conducted to analyse path tracking of the robot and verify its clearance from obstacles, ensuring a safe and collision-free trajectory. The speed of the manipulator end effector was set at 0.2 m/s, influencing the total time required to complete the path based on its length. Figure 7 depicts the robot navigating along the generated trajectory.

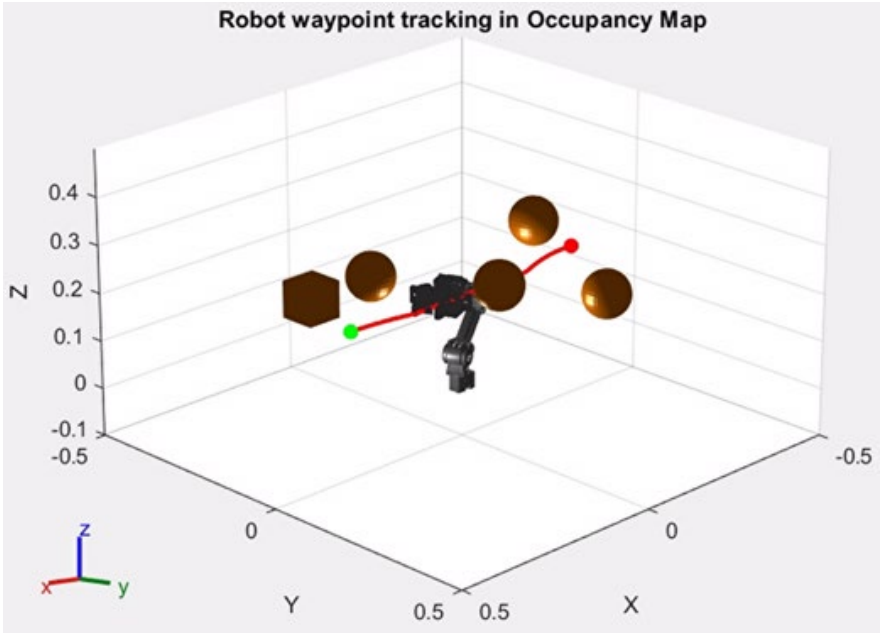


Figure 7. Robot simulation of improved RRT path. Where red line indicates the generated path

Figure 8 presents the joint angle variations over time as the robot follows the improved RRT path. The plot displays the motion of four joints, each represented by a distinct trajectory, measured in radians over a time span of approximately 2.5 seconds. Compared to previous simulations, orientations of the joints have higher levels of smoothness in this version of the algorithm due to the implementation of path shortening and interpolation techniques. The uniform upward trajectory during Joint 1 indicates its governance position for steering the robot toward its overall movement. The orientation pattern of Joint 2 shows a well-balanced behavior since its decrease occurs gradually followed by a directional change. The natural curves executed by Joint 3 and Joint 4 serve to demonstrate Bezier interpolation which provides smooth transitions through the reduction of get-abrupt rotations. Through the refined parental influence, the algorithm achieves economically efficient paths that minimize joint angular fluctuations. The enhanced trajectory design method has eliminated redundant motions which produces a more stable and efficient motion path. The improved RRT algorithm presents smoother and more optimal rotation execution paths that qualify it for real-world robotic systems requiring accurate fluid joint orientations.

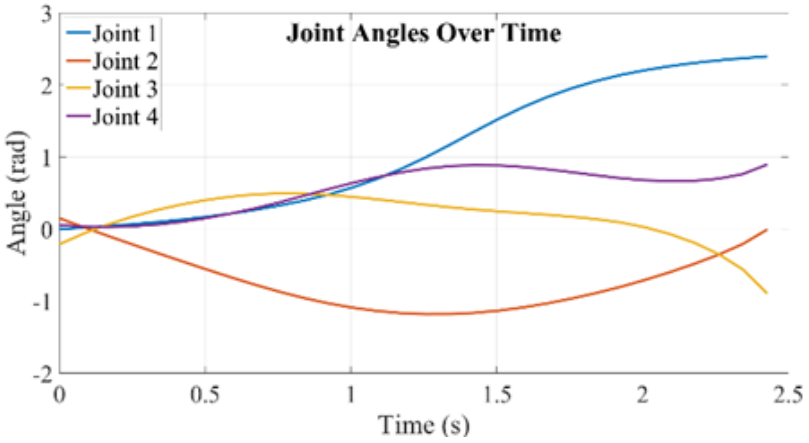


Figure 8. Joints' orientation plot of the robot following improved RRT path. Where the lines indicate as follows, Joint 1 (blue), Joint 2 (orange), Joint 3 (yellow), and Joint 4 (purple)

The 3D occupancy map in Figure 9 depicts path replanning during which the original black path encounters a collision obstacle that makes it unsafe to navigate. A fresh path computing process began following the detection of this collision to devise an obstacle-avoiding route toward the end destination. A smooth version of the new trajectory was developed to improve rotation efficiency and minimize sharp changes in the planned path. By implementing this method, the robot gains the ability to adjust its course in dynamic settings while executing optimized rotations and avoiding obstacles at all times.

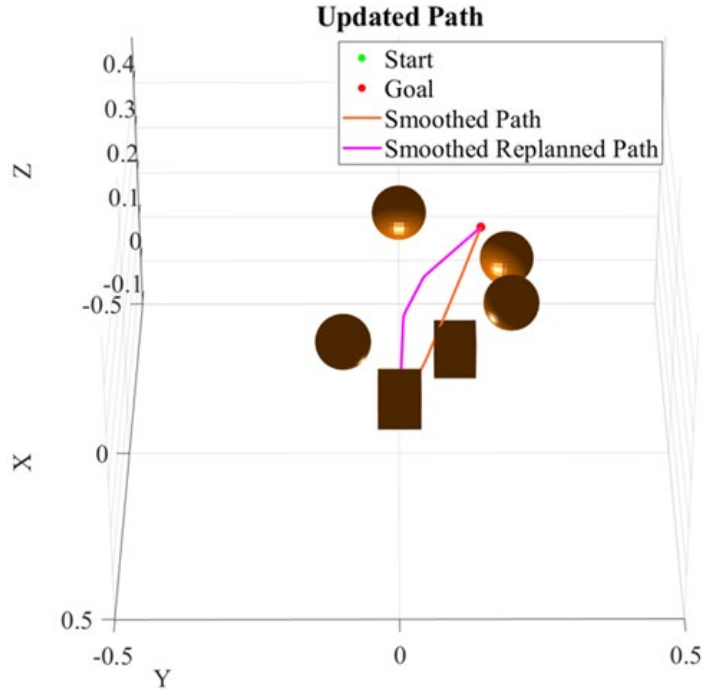


Figure 9. The replanned path. Where the green dot represents the start position of the robot, while the red dot indicates the goal position. Two paths are shown, the orange line is the smoothed path, and the pink line is the smoothed replanned path

A 3D simulation was conducted to evaluate the robot's path tracking in the updated occupancy map, considering the newly introduced obstacle. The robot, moving at a speed of 0.2 m/s, successfully followed the replanned trajectory while maintaining a collision-free path. Figure 10 illustrates the robot navigating the adjusted route, effectively avoiding the obstacle.

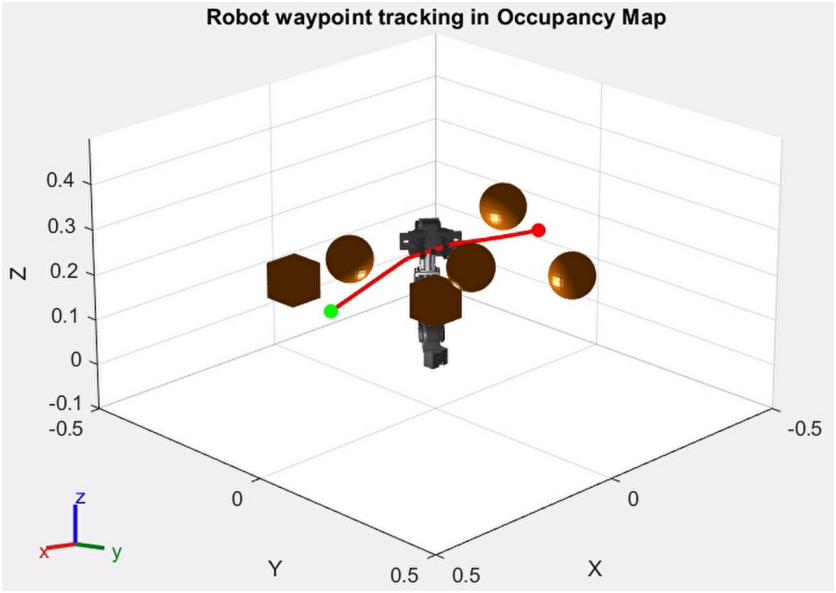


Figure 10. Robot simulation of improved RRT path with a new obstacle. Where the red line is the generated path

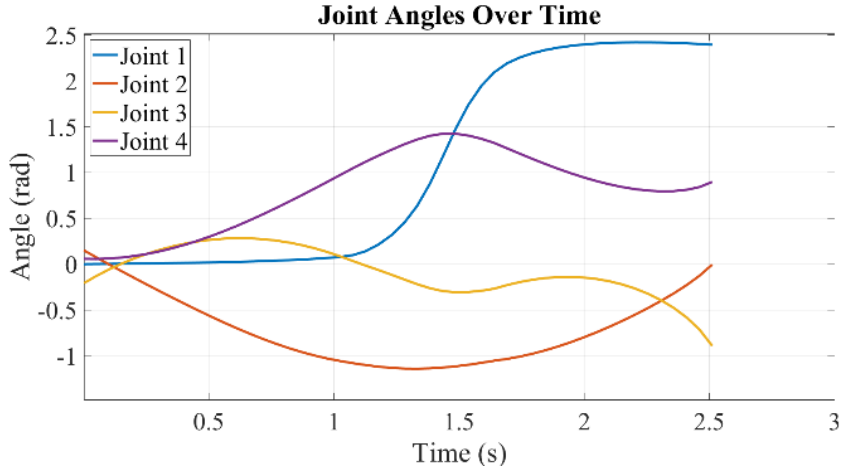
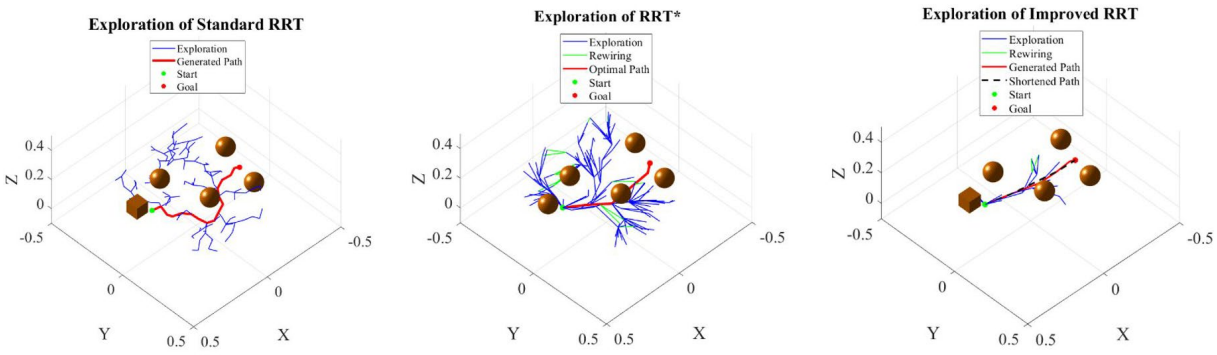


Figure 11. Joints' orientation plot of the robot following the replanned path. Where the lines indicate as follows, Joint 1 (Blue), Joint 2 (Orange), Joint 3 (Yellow), and Joint 4 (Purple)

The joint angle trajectories shown in Figure 11 maintain the same smoothness as the previous simulation before introducing the new obstacle, indicating that the replanned path does not introduce abrupt changes in motion. The synchronization between joints is preserved, and the trajectory smoothly converges to the final position, confirming the effectiveness of the replanned path in adapting to environmental changes while ensuring a feasible and stable execution.

In order to verify the feasibility and effectiveness of the proposed path planning algorithm, a comparison between standard RRT, RRT*, and improved RRT algorithms was conducted. During RRT algorithm's exploration process, it selects many redundant nodes. The random points appear in a disorganized distribution due to the random sampling strategy combined with sampling from the entire space. Moreover, during the RRT* algorithm's exploration process, it refines the tree structure by rewiring connections between nodes. This helps in reducing path cost compared to standard RRT. The sampling process is still random, but the algorithm ensures that the shortest possible path is identified by continuously optimizing the tree structure. Furthermore, The improved RRT algorithm arranges its tree expansion in a structured manner with reduced unnecessary branches when it performs exploration. By utilizing rewiring techniques, the path becomes optimized while additional path-shortening methods enhance efficiency resulting in a direct obstacle-avoiding link between start and goal points. Figure 12 illustrates the exploration process of the three algorithms, comparing standard RRT, RRT*, and improved RRT. Figure 13 represents the evolution of Joint 4's angular displacement over time for different RRT-based path-planning algorithms, comparing the standard RRT, RRT*, and the Improved Algorithm. The trajectory differences highlight the variations in smoothness and efficiency between the methods, with the improved algorithm demonstrating a more optimized and stable motion.



(a) **(b)** **(c)**

Figure 12. Comparison of the generated paths: (a) Path planned by standard RRT, (b) Path planned by RRT*, and (c) Path planned by improved RRT. Where the blue line is exploration and the red line is the generated path

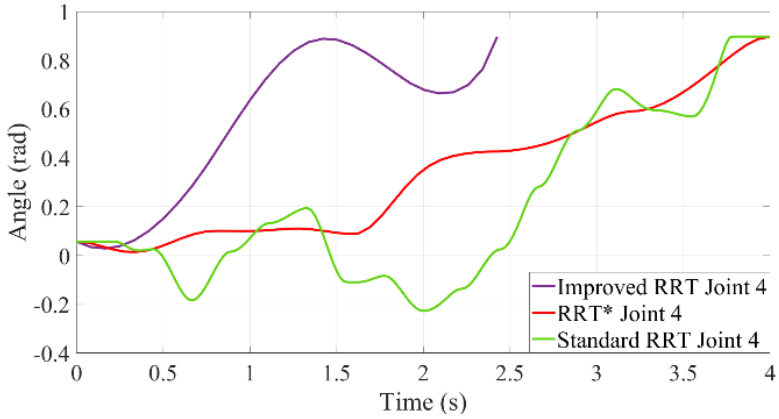


Figure 13. Comparison of Joint 4 orientation across different algorithms. Where the Purple line represents the improved RRT algorithm's Joint 4 orientation, the red line represents RRT*'s Joint 4 orientation, and the green line represents standard RRT's Joint 4 orientation

The statistics in Table 2 demonstrate the efficiency improvements achieved with the improved RRT algorithm compared to the standard RRT and RRT*. The improved RRT significantly reduces computation time, achieving a 71.35% decrease compared to standard RRT and a 73.51% decrease compared to RRT*. Additionally, the path length is shortened by 32.48% relative to standard RRT and 4.52% compared to RRT*, indicating a more optimized trajectory. The number of iterations required is also significantly lower, showing a 66.15% reduction compared to standard RRT, demonstrating improved efficiency in exploration. Moreover, the number of nodes generated is minimized, ensuring a more compact tree structure while maintaining a safe minimum obstacle clearance. These results highlight the effectiveness of the improved RRT in generating shorter, more efficient, and computationally faster paths.

Table 2. 50 Simulation results with 3 different algorithms (without smoothening)

Algorithm	Time (s)	Path Length (mm)	Number of Iterations	Number of Nodes	Min Obstacle Distance (mm)
Standard RRT	14.14687	724.979	271.44	21	69.333
RRT*	15.18044	512.544	285.98	6.3	71.102
Improved RRT	4.021534	489.369	91.88	4.34	71.159

4. CONCLUSION

The improved RRT algorithm conducted in the research is an improved version of RRT*, with additional computations to fasten the exploration process by adding a bias towards the goal, an improved technique to calculate the distance between multiple parents of the existing node, and by smoothening the generated path using linear and bezier interpolations. The improved algorithm proved that the exploration process time was reduced by 73.51%, and the numbers of iterations were reduced by 67.89% compared to RRT* algorithm. The research findings validate that the improved RRT algorithm creates efficient compute-friendly trajectories which establish

itself as a viable option for dynamic environments. The algorithm demonstrates real-time adaptability and path updating abilities because of which it exhibits potential for both industrial robotics applications and collaborative robotic deployments. The improved joint motion smoothness supports longer component lifetime and decreased component wear which makes the implementation highly practical.

References

[1] I. Noreen, A. Khan, and Z. Habib, "Optimal path planning using RRT* based Approaches: A survey and future directions," *International Jou*rnal of Advanced Computer Science and Applications, vol. 7, no. 11, 2016. doi: 10.14569/ijacsa.2016.071114

[2] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the RRT*," in *2011 IEEE International Conference on Robotics and Automation*, Shanghai, China, 2011, pp. 1478–1483.

[3] H. Zhang, Y. Wang, J. Zheng, and J. Yu, "Path Planning of Industrial Robot Based on Improved RRT Algorithm in Complex Environments," *IEEE Access*, vol. 6, pp. 53296–53306, 2018.

[4] J. Nasir, F. Islam, U. Malik, Y. Ayaz, O. Hasan, M. Khan, and M. S. Muhammad, "RRT*-SMART: A rapid convergence implementation of RRT*," *International Journal of Advanced Robotic Systems*, vol. 10, art. no. 299, 2013. doi: 10.5772/56718

[5] G. Huang and Q. Ma, "Research on path planning algorithm of autonomous vehicles based on improved rrt algorithm," *International Journal of Intelligent Transportation Systems Research*, vol. 20, pp. 170–180, 2022. doi: 10.1007/s13177-021-00281-2

[6] C. Zong, X. Han, D. Zhang, Y. Liu, W. Zhao, and M. Sun, "Research on local path planning based on improved RRT algorithm," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, vol. 235, no. 8, pp. 2086–2100, Mar. 2021. doi: 10.1177/0954407021993623

[7] O. Adiyatov and H. A. Varol, "A novel RRT-based algorithm for motion planning in dynamic environments," in *2017 IEEE International Conference on Mechatronics and Automation (ICMA)*, Takamatsu, Japan, 2017, pp. 1416–1421.

[8] A. Lonklang and J. Botzheim, "Improved rapidly exploring random tree with bacterial mutation and node deletion for offline path planning of mobile robot," *Electronics*, vol. 11, no. 9, art. no. 1459, May 2022. doi: 10.3390/electronics11091459

[9] L. Jiang, S. Liu, Y. Cui, and H. Jiang, "Path planning for robotic manipulator in complex multi-obstacle environment based on improved RRT," *IEEE-ASME Transactions on Mechatronics*, vol. 27, no. 6, pp. 4774–4785, Dec. 2022. doi: 10.1109/tmech.2022.3165845

[10] 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, art. no. 113425, Aug. 2020. doi: 10.1016/j.eswa.2020.113425

[11] M. Cao, H. Mao, X. Tang, Y. Sun, and T. Chen, "A novel RRT*-Connect algorithm for path planning on robotic arm collision avoidance," *Scientific Reports*, vol. 15, no. 1, Jan. 2025. doi: 10.1038/s41598-025-87113-5

[12] A. Dadbin, A. Kalhor, and M. T. Masouleh, "A comparison study on the dynamic control of OpenManipulator-X by PD with gravity compensation tuned by oscillation damping based on the phase-trajectory-length concept," in *2022 8th International Conference on Control, Instrumentation and Automation (ICCIA)*, 2022, pp. 1–7. doi: 10.1109/iccia54998.2022.9737207

[13] Z. Zhou, "A mobile robot with a manipulator to alleviate the shortage of health workers in hospitals in covid 19," *Highlights in Science Engineering and Technology*, vol. 39, pp. 872–880, Apr. 2023. doi: 10.54097/hset.v39i.6658

[14] H. Z. Ting, M. H. M. Zaman, M. F. Ibrahim, and A. M. Moubark, "Kinematic analysis for trajectory planning of open-source 4-DoF robot arm," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 6, 2021. doi: 10.14569/ijacsa.2021.0120690

[15] Y. Zhang, W. Zhang, B. Xu, and C. Liu, "An algorithm to realize the real-time interpolation of bezier curves based on DSP," *Advances in Computer Science Research*, Jan. 2015. doi: 10.2991/csic-15.2015.107