Path Planning using RRT* with Repulsive Potential for Probabilistic Node Rejection

Jun Hong Vince Chong and Kamin Palkawong na ayuddhaya

***Abstract*— The optimized rapidly exploring random tree (RRT*) generates and connects random nodes in the free space to create a path to the goal with the mean to achieve the shortest path connecting the start and the goal. However, RRT* tends to sample the nodes that are located too close to the obstacles leading to an overall weak δ-clearance path. In this paper, we proposed the combination RRT* with the repulsive-potential node rejection based on probability to obtain a more robust collision-free path within a limited number of sampling iterations while maintaining relatively good performance metrics in different environments. Our algorithm will randomly sample nodes in the configuration space with RRT* and use the repulsive potential function to determine the probability of whether the planner should reject the particular node based on the repulsive potential-based probability or not. With this behavior, the collision with the obstacles can be avoided and the total number of sampling nodes tends to be reduced. Our algorithm and the base RRT* will be evaluated in four different environments, including the maze, narrow corridor, cluttered, and normal environments with the criterion of memory consumption, convergence rate, and robustness.**

I. INTRODUCTION

The process of finding collision-free paths has been the standard for autonomous navigation ever since the pre-sampling era [14]. Navigating through the collision-free path allows the robot to reach its destination from its initial location while avoiding obstacles in the known environment. It has gained lots of attention due to its potential practical usage in wide-spread applications such as robotics, autonomous vehicles, military, agriculture, and so forth [16], [17], [19]. Among all the motion and path planning algorithms, Artificial Potential Fields is a well-known resolution algorithm for real-time obstacle avoidance. With this algorithm, it mitigated the need for high control complexity that the complete planning algorithm imposed when navigating in a complex environment by fine-tuning its step resolution [9], [12], [17]. Even though the potential field approach has a fast convergence speed with significantly less computational time, it breaks down in higher dimensional spaces or higher complexity environments [5]. Moreover, since the artificial potential field approach is purely reactive in nature, it can also be easily trapped in local minima and does not yield the optimal path [1], [11], [20].

Hence, sampling-based algorithms were introduced as a paradigm shift to solve challenging navigation planning in higher dimensional environments. Its remarkably low computational effort stemmed from its random sampling of nodes from the obstacle-free space, prevented the need for explicit construction of obstacles in the configuration space [1], [5], [17]. Some of the most popular sampling-based planning methods include Probabilistic Road Maps (PRM) [8], Rapidly exploring Random Trees (RRT) [10], and Rapidly exploring Random Trees Star (RRT*) [6]. PRM is a multi-query sampling-based method that constructs the multiple roadmaps of collision-free paths before answering the queries to find the optimal path. However, its implementation does not work efficiently if the environment is not known prior. On the other hand, RRT is a single-query sampling-based method that uses incremental sampling to span across the configuration space to find the feasible path while avoiding obstacles even in an unknown environment [1], [5], [17]. Even though both of these methods have probabilistic completeness, they could not asymptotically converge to an optimal path [5]–[7]. The optimal RRT, namely RRT*, was introduced as the sampling-based algorithm with the capability to converge asymptotically to an optimal path, which is very useful in real-time practical applications [13], [17].

II. RELATED WORKS

However, the drawbacks of RRT* include slow convergence rate, large memory requirement, and long computation time to obtain a feasible, collision-free path [17]. To eliminate these weaknesses of RRT*, several methods have been proposed, such as Bidirectional(Bi-RRT*) and LQR-RRT*. Bi-RRT* is the algorithm that grows two trees simultaneously from the start and the goal towards each other, which improves the search efficiency [21]. However, applying bidirectional trees to RRT* eventually requires rewiring in two trees, which leads to high computational cost [2], [13]. For LQR-RRT*, it uses the linear quadratic regulation to compute the optimal control policies for a linear system with the Gaussian noise [15]. However, its algorithm requires RRT* to re-propagate the tree for each iteration and, consequently, demands high computational costs. [13].

Some of the RRT* variants focused on different sampling strategies or node rejection to reduce the computational time to find the optimal path. RRT*-Smart proposed usage of path optimization, using triangle inequality, after the initial path is found using the RRT*. Then, it employs its intelligent sampling within the chosen radius of deployed beacons at the vertices of the obstacles which resulted in a faster convergence compared to RRT* [4]. However, its complexity increases, resulting in a longer computation time. RRT*FN enforced limits on the number of generated nodes on RRT*. It consists of global and local node removal procedures following the node removal policy, resulting in less memory consumption [1]. However, it resulted in a sub-optimal path since it limits the total number of nodes. EP-RRT* is the algorithm minimizing the sampling area, which reduces computational

time, and it works well in cluttered and narrow environments. However, it is likely to result in a sub-optimal path if there are multiple feasible solutions within the environment [3].

In addition to the node rejection, some modifications of RRT* integrate the artificial potential field. The combination of the bidirectional RRT* with an artificial potential field to help avoid the local minima experienced by the potential-based method alone and make the convergence rate higher while accounting for the safe distance between the robot and the obstacles [19]. However, as mentioned earlier, the bidirectional trees required high computational costs for the attempt to connect both trees. Another interesting approach is the potential-based RRT* that has the same complexity as RRT*, but yields a better convergence to the optimal path, requires less memory, and does not suffer from the local minima [18]. However, it is a one-way search algorithm that finds the optimal path with a weak δ-clearance. This means that the optimal path found is not robust according to the definition discussed in section IV. Additionally, it does not span as much as the RRT* itself.

III. Paper Contribution

In this paper, we propose a modified algorithm of RRT* sampling with rejection probability based on repulsive potential function. The objectives of this modified algorithm are not only to include a novel notion of obstacle avoidance through repulsive rejection probability, but also to maintain a relatively good convergence rate, success rate in different environments, and memory consumption. From a sampling-based planning point of view, it provides asymptotically convergence and probabilistic completeness. From a repulsive potential probability function point of view, it mitigates the need for complex control to incorporate a notion of obstacle avoidance. The effectiveness of our approach will be assessed under the assumptions of the four known different environments, including a normal environment, a maze, a cluttered space, and a narrow corridor, and compared to the base RRT*. The evaluation metrics are the executing times, the number of tree nodes, and the robustness. The success rate is also included to demonstrate the reliability of the algorithm with a fixed limited amount of sampling iterations.

IV. Preliminaries

Our proposed algorithm is based on two widely used algorithms which are the optimal Rapidly exploring Random Trees (RRT*) and Artificial Potential Fields (APF).

A. Optimal Rapidly exploring Random Trees (RRT*) [6]

With the RRT*-based algorithm, we expect our algorithm to utilize its remarkably low computational effort of randomly sampling the obstacle-free space without any need for explicit construction of obstacles in the configuration space while maintaining its optimal properties. The overview of RRT*'s essential functions within the algorithm is listed below:

- **Random Sampling:** A function ***SampleFree*** that randomly samples a node x_{rand} within the given grid space uniformly. Each random sample node needs to be independent and identically distributed.
- **Nearest Neighbor:** A function ***Nearest*** that finds the nearest neighbor $x_{nearest}$ from the random sample node x_{rand} within the tree graph G = (V,E).
- **Steering:** A function ***Steer*** that walk in a straight line to the new node x_{new} from the nearest neighbor $x_{nearest}$ in a direction of the random sample node x_{rand} below a defined maximum number of steps.
- **Collision Check:** There are two functions ***ObstacleFree*** and ***CollisionFree*** that both check for node occlusions and edge collisions between the new node x_{new} and its closest neighbor $x_{nearest}$ or x_{near}.
- **Near Neighbors:** A function ***Near*** that find the neighbors x_{near} that are within the search radius of the new node x_{new}. Within this function, it uses the cardinality card(V) of the total number of generated samples in the grid space that is inversely proportional to the radius of the search radius.
- **Node Cost:** A function ***Cost*** that computes the cost of each nearby node, Euclidean distance accumulating from the starting node to itself, added to its heuristic cost function ***c***, Euclidean Distance between the neighbors and the new node x_{new}.

Based on the components mentioned, the RRT* uses these functions to randomly generate the new node $x_{new} \in X_{free}$ to both find the minimum cost path and rewire its tree graph to obtain the optimal path towards the goal. The probabilistic completeness of the RRT* is based on the notion of robust feasible path planning. As defined, robust path planning has the following properties which are continuous path, collision-free path, and feasible path. These properties depend on the total variation TV(σ), a bounded variation, which is essentially the total length of the traversed path computed with the Euclidean Distance function in $\mathbb{R}^d$.

$$TV(\sigma) = \sup_{n\in\mathbb{N}, 0=\tau_0<\tau_1<\ldots<\tau_n=s} \sum_{i=1}^{n} |\sigma(\tau_i) - \sigma(\tau_{i-1})|$$

where σ is the Euclidean Distance from the starting node to the current node and $\tau \in [0,1]$ is a bounded real number along the traversed path from the starting node to the goal node. First, the path is continuous due to the usage of Euclidean Distance. Second, the found path is collision-free if the generated nodes within the path belong to the free-space X_{free}, given $\sigma(\tau) \in X_{free}$ for all $\tau \in [0,1]$. Finally, the found path is feasible if the path consists of the starting node and the goal node, given $\sigma(0) = x_{init}$ and $\sigma(1) \in cl(X_{goal})$ where $cl(.)$ is a closure set. If the $\sigma : [0,1] \mapsto X_{free}$, the robust free-path has a strong δ-clearance, shown in figure 1, where $\delta > 0$ is the search radius, given $\sigma(\tau) \in int_\delta(X_{free}$ for all $\tau \in [0,1])$.

For the RRT* algorithm to help achieve probabilistic completeness, it essentially manipulates the search radius to be inversely proportional to the number of vertices to the log scale. Using percolation theory for the finite number of vertices in a random geometrical graph, the probability of having the largest distance between a node and the starting

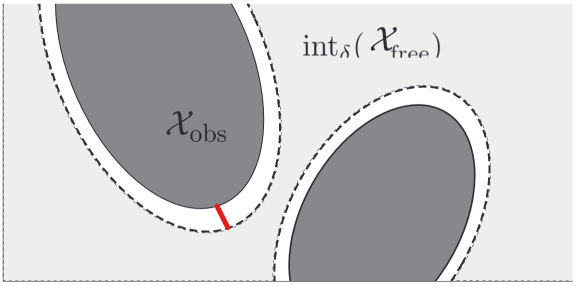


Fig. 1: An illustration of the δ-interior of X_{free}. The distance, the line in red, between the dashed boundary of the gray region to the dark gray region is δ.

node greater than the search radius is a finite sum. This means that the variable search radius function used is based on a scaling function using the number of nodes that is summable. Because the result of this probability is a finite sum, the probability of the event of having the largest distance between the nodes greater than the search radius is zero based on the Borel-Cantelli lemma. Because of such a case, the randomly generated nodes do not span or explore much of the free space X_{free} while the number of nodes goes up to infinity which leads to not having the properties of probability completeness. On the contrary to RRT*, it uses a scaling function based on a logarithm that is not summable.

On the other hand, the RRT* algorithm has an asymptotically optimal property. To obtain such a property, the algorithm needs to have probabilistic completeness to help identify whether there is an optimal solution. With this property, the RRT* can converge to an optimal solution, with its cost Y, almost surely with a probability of either zero or one as the number of generated nodes increases to infinity.

$$\mathbb{P}(\{\limsup_{n\mapsto\infty} {Y_n}^{RRT^*} = c^*\}) = 1 \;\; or \;\; 0$$

In addition, the algorithm also needs to have the monotonicity property. If the graph $G_i^{RRT^*}(\omega) \subseteq G_{i+1}^{RRT^*}(\omega)$, where $\forall\omega \in \Omega$ and $\forall i \in \mathbb{N}$, then its cost $Y_{i+1}^{RRT^*}(\omega) \leq Y_i^{RRT^*}(\omega)$. Since its cost $Y_i^{RRT^*} \geq$ the optimal cost c^*, then the sequence σ_i converges to some limiting value based on the distribution ω used on its configuration space Ω. To relate to the RRT* case, having a larger tree graph means containing a lower minimum cost path as compared to its subset tree graph. If its subset tree graph has the almost surely probability of converging to an optimal minimum cost path, then it can converge to a limit cost value depending on the distribution used for random sampling in the free space X_{free}. Therefore, to achieve the asymptotical optimality property, the RRT* has an approach called ***rewire*** that helps to minimize the path's cost by reconnecting the connected closest neighbors to the new node x_{new}.

B. Artificial Potential Fields (APF) [9]

Besides the RRT*, our algorithm utilizes the repulsive function (eq. 1) referenced from the Artificial Potential Fields. As mentioned in section I, the advantage of the APF is mitigating the need for high control complexity when navigating a complex environment. Because its repulsive function is computed in each iteration, it allows the algorithm to quickly converge to the goal location.

C. Proposed Method: RRT-Probabilistic Node Rejection (RRT*-PNR)*

As reviewed in section II, most RRT* variants are successful in reducing one aspect of the RRT*'s drawbacks, but in turn unfavorably enhance the other drawbacks. Here, our RRT*-PNR algorithm takes a different unique approach by imbuing a sense of obstacle awareness using the repulsive function (eq 1) within the probability function (eq 2).

Since our algorithm is mainly based on RRT*, the node rejection with probability occurs after the function ***Steer***. If the new node x_{new} is generated within the influenced distance of the obstacle, the algorithm will start randomly sampling within the defined search radius of the nearest node $x_{nearest}$ for n times. Along with the temporary new node and the n-generated nodes, the repulsive function (eq 1) will be used to compute the repulsive potential for all the nodes.

$$U_{rep}(x,y) = \begin{cases} \frac{1}{2}(\frac{1}{d(x,y)} - \frac{1}{d_{inf}})^2, \text{if } 0 < \text{d(x,y)} < d_{inf} \\ 0, \text{if d(x,y)} > d_{inf} \\ \text{Undefined, otherwise} \end{cases} \tag{1}$$

Note that this is the repulsive potential for one obstacle. The $d(x,y)$ is the minimum distance between the sampled point and the surface of the obstacle, and d_{inf} is the influenced distance of the obstacle. If there is more than one obstacle, the total repulsive potential will be considered for each node. If the node is located outside of the influence distance of any nearby obstacle, the node would not be accounted for in the probability calculation. After determining the repulsive potential on these points, the algorithm will compute the potential-based probability of each point that has non-zero repulsive potential as shown in eq. 2. Suppose that there are k points rejected due to their zero repulsive potential.

$$P_j = \frac{U_{rep}(x_j)}{\sum_{j=1}^{n-k} U_{rep}(x_j)} \tag{2}$$

where x_j is the points that are not rejected. As seen in the eq.1 and eq.2, the resulting probability is not linear. This non-linear probability tells us about how close the nodes are to each detected obstacle. The higher the probability is, the closer the node to its obstacle is. The algorithm will establish the node as the new node x_{new} that has the lowest probability and add it to the tree graph. It is desirable to have a non-linear probability since it makes the awareness of the node sensitive to its nearby obstacles. This gives the new node x_{new} a chance to re-establish its relative position away from its nearby obstacles without being explicit about it in the configuration space. Additionally, the re-establishing of the new node x_{new} allows it to step backward to the direction of the starting node location. In the same usage of the probability, it also allows the tree to have the opportunity to rewire to

a path with a higher δ-clearance. In hindsight, these will increase both the convergence rate and memory consumption in the short run. However, if the algorithm were to continue sampling the configuration space further, the RRT*-PNR algorithm would find a better optimal path than the RRT* in a safe sense. Thus, the RRT*-PNR algorithm would achieve lower memory consumption while reaching the goal location with a strong δ-clearance and robust path.

In a trivial case, the RRT*-PNR algorithm, built upon the RRT* algorithm, will converge to the asymptotically optimal path with minimal cost if the problem is feasible. The probability node rejection function (algorithm 2) will be active as soon as the new node x_{new} detects the influence region of the nearby obstacles. The probability of the algorithm converging to a strong δ-clearance path depends on both the probability of the newly generated sample hitting a free space X_{free} and choosing the sample that has the lowest repulsive value. The first probability is based on the area of the free space within the sampling radius over the area of the entire sampling radius. The sampling radius is formulated by the following equation:

$$min(\gamma_{RRT^*-PNR}(log(card(V))/card(V))^{1/d}, \eta)$$

where γ_{RRT^*-PNR} is the searching radius, card(v) is the current number of nodes in the tree, d is the dimension of the configuration space, and η is the fixed step size. This equation was chosen, instead of a constant step radius, to maintain the probabilistic completeness of the algorithm as discussed in section IV. This allows the ***rewire*** function to reach each newly generated node. Because the radius slowly decreases according to the number of nodes, the first probability slowly increases in probability since the sampling radius is less likely to be covered by the obstacles. This means that if the rejection algorithm were to sample only one node beside the new node x_{new}, the RRT*-PNR algorithm will always have a higher probability of converging to a stronger δ-clearance path than the RRT* algorithm itself. Additionally, if the rejection algorithm were to sample a sufficiently large number of nodes beside the new node x_{free}, the RRT*-PNR algorithm can converge to a stronger δ-clearance path relatively quickly before reaching the goal node. On the other hand, the second probability always will be 1 if there is at least a sample node beside the new node x_{new} to converge to a stronger clearance path. Therefore, if there is a minimal cost path with a strong δ-clearance, the RRT*-PNR algorithm has a higher probability of requiring lesser memory consumption to converge to the optimal path relatively quickly than the RRT* algorithm itself.

To help with the convergence rate of the algorithm, we used the repulsive function to compute the rejection probability. As discussed in section II, the purpose of the rejection idea is to reduce the memory consumption of the RRT*. However, in return, those RRT* variants increase their computation complexity or result in a sub-optimal collision-free path. Thus, the time to converge to the optimal path increases. On the contrary to our algorithm, it is well-known that the repulsive function, that gives rise to low-level control complexity, can be computed quickly. Hence, our algorithm RRT*-PNR can achieve both memory consumption reduction and a higher convergence rate relatively with a probability compared to those variants. Since we are comparing our algorithm to the RRT*, we hope to achieve relatively the same time complexity due to the significantly low computational time of the repulsive function and reduce its memory consumption to ultimately obtain a more robust path within a limited number of samplings.

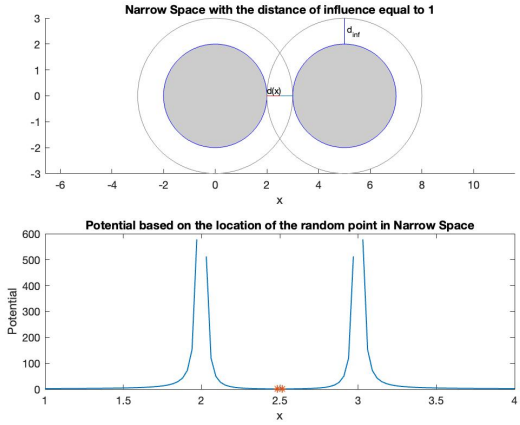


Fig. 2: An illustration of the total potential of the point located between two obstacles with the distance of influence perfectly overlapping each other. Assuming the PNR sampling the space around the point on the line linking between the surface of two obstacles

In the cases of the narrow corridor, cluttered, maze, and normal environments, the RRT*-PNR will mostly behave similarly to RRT*, but with a probability of choosing a node with the lowest repulsive potential, as shown in figure 2. This figure demonstrates that the lowest repulsive potential would be in the middle point between the two obstacles. Therefore, if we are to sample the space for a sufficiently large number of times, the algorithm will move along the path associated with the lowest repulsive potential nodes, implying obstacle avoidance capability with strong δ-clearance. Let d(x,y) be the distance between the node and the first obstacle and d_{inf} - d(x,y) be the distance between the node and the second distance. The repulsive potential is computed separately for each obstacle to the node by using eq. 1. Then, the node will be associated with the obstacle that resulted in the highest potential value if this node detected more than one obstacle.

The RRT*-PNR algorithm and its probability rejection function are displayed in the pseudo-code as shown in algorithm 1 and algorithm 2.

V. Validations

To demonstrate the effectiveness of our RRT*-PNR algorithm, we tested against the RRT* algorithm in four different environments: normal, maze, narrow corridor, and cluttered. To fairly test against the base RRT*, we essentially create our version of the RRT* algorithm so that we can accurately

measure the differences in performance metrics between the RRT* and our proposed algorithm. For both algorithms, they will be tested 100 times for each environment to obtain the average values for each performance criterion: convergence rate, memory consumption, and success rate. In addition, our RRT*-PNR algorithm will randomly sample 10 times within the sampling radius once the new node x_{new} detects the nearby obstacles. Note that we limit the number of iterations at 1,000 sampling iterations, set the maximum step size to 1, and initialize the search radius at 6 for both RRT* and RRT*-PNR.

Algorithm (Python)	Avg Time (s)	Avg Tree Nodes	Success Ratio
RRT*	0.0409586	107	1.0
RRT*-PNR	0.0896492	159	1.0
Algorithm (MatLab)	**Avg Time (s)**	**Avg Tree Nodes**	**Success Ratio**
RRT*	0.019129	107	1.0
RRT*-PNR	0.05043	166	1.0

TABLE I: Performances in the normal environment.

Algorithm (Python)	Avg Time (s)	Avg Tree Nodes	Success Ratio
RRT*	1.5292	174	0.7
RRT*-PNR	2.59764	192	0.56
Algorithm (MatLab)	**Avg Time (s)**	**Avg Tree Nodes**	**Success Ratio**
RRT*	0.23278	204	0.68
RRT*-PNR	0.42962	238	0.62

TABLE II: Performances in the maze environment.

Algorithm (Python)	Avg Time (s)	Avg Tree Nodes	Success Ratio
RRT*	0.496837	196	1.0
RRT*-PNR	1.64737	311	0.94
Algorithm (MatLab)	**Avg Time (s)**	**Avg Tree Nodes**	**Success Ratio**
RRT*	0.051644	184	1.0
RRT*-PNR	0.11087	251	0.97

TABLE III: Performances in the narrow corridor environment.

Algorithm (Python)	Avg Time (s)	Avg Tree Nodes	Success Ratio
RRT*	0.0906482	148	1.0
RRT*-PNR	0.482833	308	0.97
Algorithm (MatLab)	**Avg Time (s)**	**Avg Tree Nodes**	**Success Ratio**
RRT*	0.046382	157	1.0
RRT*-PNR	0.20465	278	0.98

TABLE IV: Performances in the cluttered environment.

A. Convergence Rate (Average Time (s))

The convergence rate is evaluated based on the time difference between the start and the end of the iterative sampling of the algorithm. Note that the time differences include both scenarios in which the planner successfully searches for the path or fails to find the goal within the defined step size and the limited number of iterations. According to the the result tables, our RRT*-PNR converges slower than RRT* by the multiplications of 2.41, 1.78, 2.73, and 4.87 for the normal, maze, narrow corridor, and cluttered environments respectively. Since our RRT*-PNR is built based on the original RRT*, the convergence rate of our algorithm is likely to be slower than RRT* due to the additional functions. In the maze environment, the difference in the convergence rate is the least as compared to other environments in which the zig-zag motion of its spanning tree is not favorable, as shown in figures 5 and 6. On the other hand, the difference is the most when comparing the algorithms in the cluttered environment, where the zig-zag motion of RRT* helps the tree to turn around corners efficiently. Furthermore, our algorithm RRT*-PNR is programmed to allow the new node to step backward as mentioned in the section IV-C, which consequently contributes to the slow convergence rate. However, the structure of the algorithm for the probability rejection node affects the convergence rate the most. As shown in the algorithm 2, we first sample random nodes within the sampling radius. Then, compute the repulsive potential value and its probability for each accepted node. After that, we choose the lowest repulsive potential value based on its probability. In this series of steps, the time complexity for this algorithm is $O(n^3)$ for both Python and MATLAB. Therefore, the difference in the time complexity between the RRT* and our RRT*-PNR algorithms can be approximated at $O(n^3)$.

B. Memory Consumption (Average Tree Nodes)

The memory consumption is evaluated based on the total number of nodes stored within the tree graph. To summarize the tables shown below, our RRT*-PNR has more memory consumption than the RRT* by an approximate factor of 1.519, 1.103, 1.475, and 1.926 for the normal, maze, narrow corridor, and cluttered environments respectively. Based on the average number of nodes from the result tables, our RRT*-PNR algorithm performs the best in the maze environment as compared to other environments while it performs the worst in the cluttered environment. Despite the huge differences in convergence rates, the average memory consumption for our algorithm does not deviate too much from that of the RRT* even though our algorithm allows the node to be generated backward toward the source of the tree graph. This is proof that the RRT* algorithm tends to move in a zig-zag motion, which is unfavorable in the tight environment due to the uniform randomness of the node generation in the configuration space. Thus, the same reasoning for these performances can be found in the subsection V-A. However, it is important to note that the memory consumption is linked to the convergence rate in the way that the algorithm will stop generating random nodes once the goal node is reachable within the defined step size, implying feasibility. The other reason that contributes to memory consumption is the usage of the sampling radius as shown in section IV-C. The base RRT* uses a defined constant step size to move forward, based on the ***steer*** function, to the new node x_{new} unless the randomly

generated node is nearer to the nearest node $x_{nearest}$. On the other hand, our PNR function, as described in algorithm 2, uses a step size that is shrinking as the number of tree nodes increases to choose a random node with a lower repulsive potential value than the new node x_{new}. Therefore, the step size of our algorithm will eventually become lower than the RRT* but in exchange for higher robustness, with a stronger δ-clearance, in the path planning for different environments as it will be discussed in the following subsection V-C.

C. Robustness of RRT*-PNR algorithm

As described in the section IV, an algorithm is robust to different environments when it can produce a feasible, continuous, and collision-free path. The significant noticeable difference between our RRT*-PNR algorithm and the RRT* algorithm is the reproducibility of a stronger δ-clearance path which is shown in the figures 3 to 10. Note that, the differences between the figures are the appearance of the rejected nodes, which is the initial new node before being replaced by the node with lower repulsive potential. Throughout the four different tested environments, our RRT*-PNR algorithm has overall a stronger δ-clearance path as compared to the base RRT* algorithm. The significance is especially shown when the algorithms attempt to plan the path between two obstacles as strongly displayed in the narrow corridor and cluttered environments. Because the RRT* is moving in a zig-zag motion while expanding its tree graph, it tends to get quite close to the obstacles which leads to an overall weak δ-clearance. However, the robustness of our algorithm depends heavily on the probability of the random sampling within the sampling radius. As mentioned, we tested our algorithm in the four different environments with a defined number of random sampling of 10 for each new node x_{new} detected the distance of influence. Figure 8b shows the weak δ-clearance of our algorithm where it failed 10 times to sample a lower repulsion value at the entrance of the narrow corridor. However, our algorithm starts to deviate its tree away from the obstacles and steer it towards the middle between the two obstacles once the random sampling manages to sample a lower repulsive node. If the RRT* were to take a similar route, it would grow its tree close to the obstacles as displayed by the red color rejected node without intervention from our PNR function. Hence, it helps clarify visually the two probabilities mentioned in the section IV.

VI. Conclusions

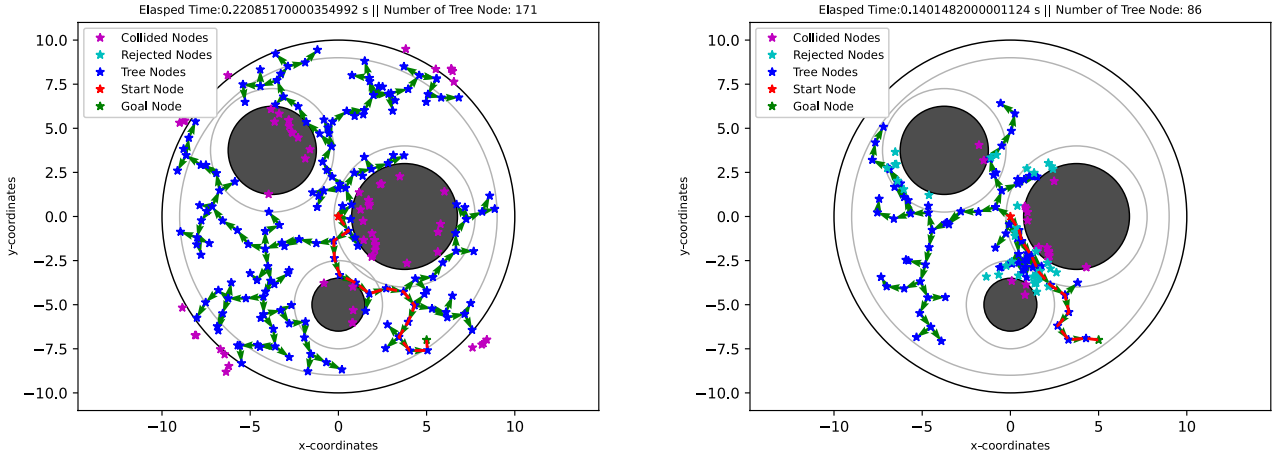
In this paper, we have presented a new variant of the RRT* by including a notion of obstacle avoidance without explicitly representing the obstacles in the configuration space. The novelty of our proposed RRT*-PNR algorithm uses probability to reject nodes with high repulsive potential values and accept the node from random sampling with the lowest repulsive potential value. Because the RRT*-PNR is built upon the RRT* algorithm, the convergence rate of the RRT*-PNR algorithm will always be lower than the RRT*. On the other hand, the memory consumption of our algorithm is likely to be larger than that of the RRT*'s because the PNR function has the probability of generating the new node backward to the direction of the starting node. However, we illustrated the significance of our RRT*-PNR algorithm in obtaining a feasible path with stronger δ-clearance than the RRT* algorithm, demonstrating higher robustness in our algorithm in different environments as compared to the RRT*. To improve the convergence rate of our algorithm, it is essential to reduce the time complexity of our PNR function by either simplifying or optimizing the function's logic. In addition to the time complexity, both the convergence rate and memory consumption can also be improved by implementing a notion of heuristic cost or directional bias to prevent the random sampling from generating the nodes backward and reduce the number of redundant samplings.

Algorithm 1 RRT*- PNR

Define the environment
Initialize the variables ($V \leftarrow x_{init}$, $E \leftarrow \varnothing$,...)
for $Iteration = 1, 2, \ldots$ **do**
    Generate a random node (***SampleFree***)
    Expand the tree from the nearest node to the new node towards the direction of the random node (***Nearest*** and ***Steer***)
    if $dist(new\ \ node, obstacle) < d_{inf}$ **then**
        Run PNR function
    end if
    Check collision (***ObstacleFree*** and ***CollisionFree***)
    if no collision **then**
        Find nearby nodes (***Near***)
        Find the parent node of the new node (***Cost***) // Connect along a minimum-cost path
        Update the tree (***Rewire***) // Rewire the tree
    end if
    if the goal is reached within the step size **then**
        Find the path
        Break out of the loop
    end if
end for
Return the tree graph $G = (V, E)$

Algorithm 2 Probability Node Rejection (PNR) Function

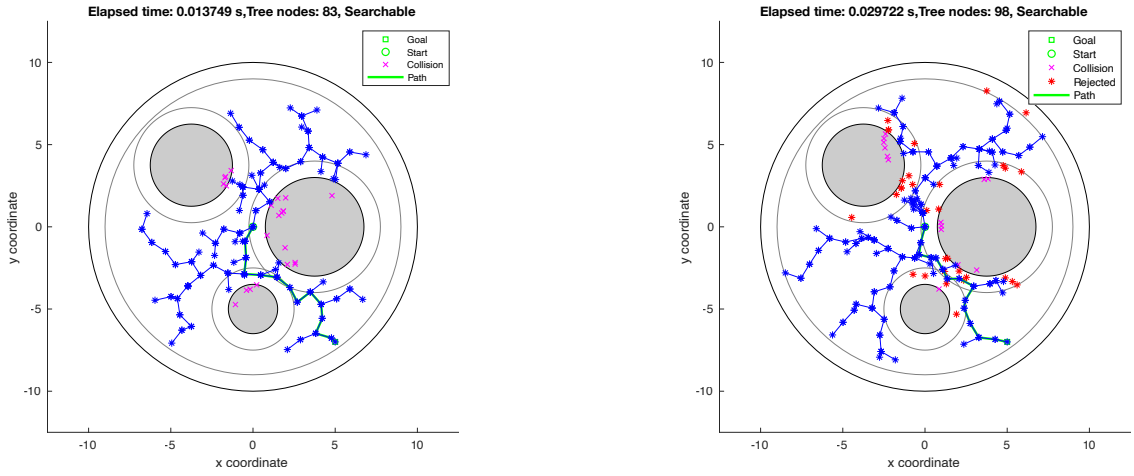
Sample random nodes within the sampling radius
if $dist(sampling\ \ node, obstacle) < d_{inf}$ **then** // node inside the distance of influence
    Calculate the repulsive potential value for the node
    Add both the node and its value into the accepted array
else
    Reject the node // node outside of the distance of influence
end if
Calculate the probability for each accepted node based on its value
Choose the node with the lowest probability of being the new node



(a) RRT* algorithm

(b) RRT*-PNR algorithm

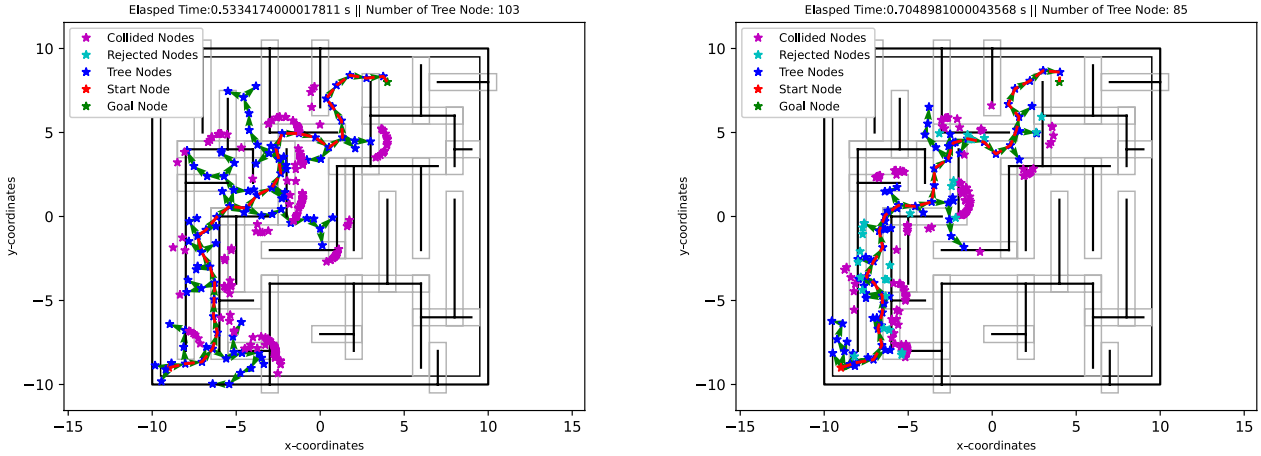
Fig. 3: The performances of the RRT* and RRT*-PNR algorithms in the normal environment (Python).



(a) RRT* algorithm

(b) RRT*-PNR algorithm

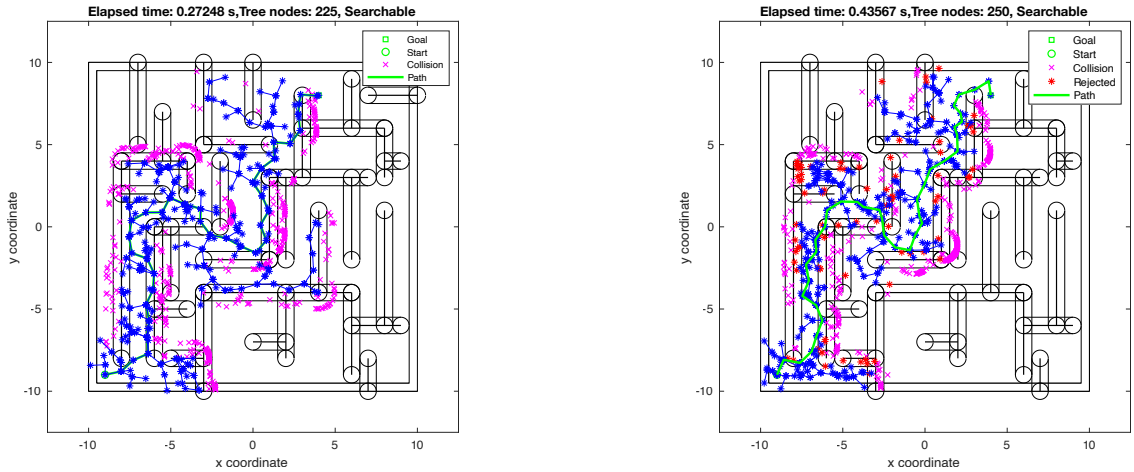
Fig. 4: The performances of the RRT* and RRT*-PNR algorithms in the normal environment (MATLAB).



(a) RRT* algorithm

(b) RRT*-PNR algorithm

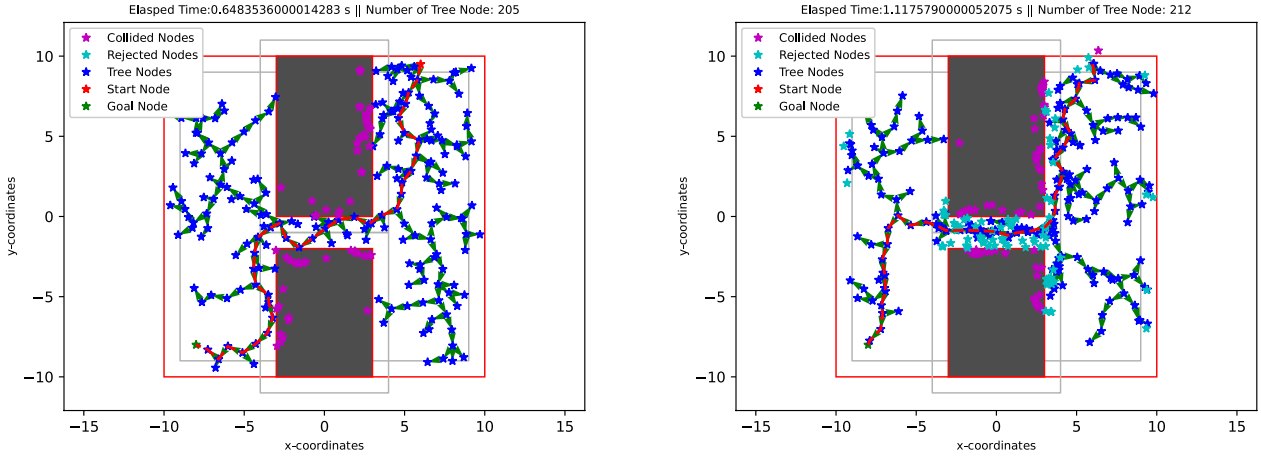
Fig. 5: The performances of the RRT* and RRT*-PNR algorithms in the maze environment (Python).



(a) RRT* algorithm

(b) RRT*-PNR algorithm

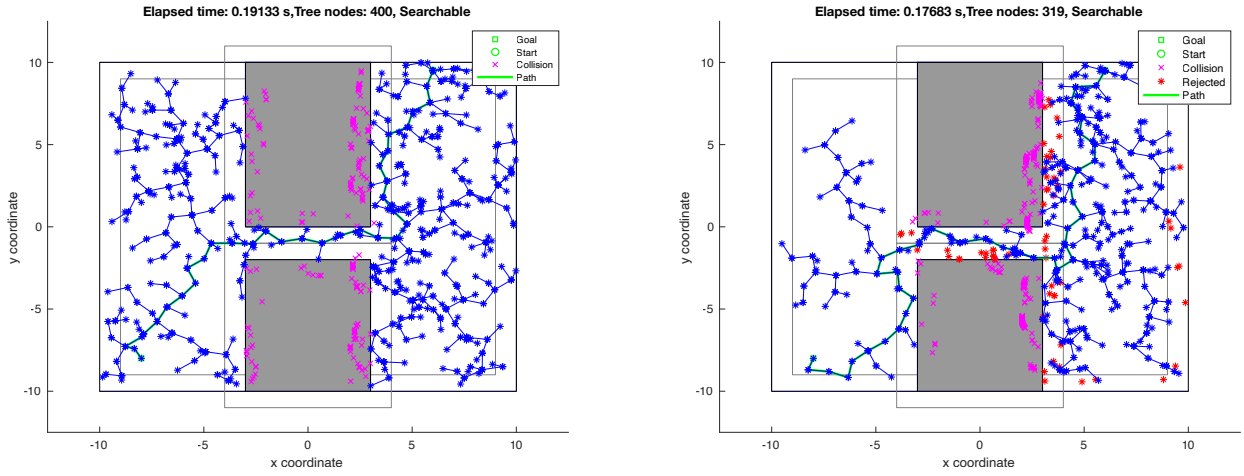
Fig. 6: The performances of the RRT* and RRT*-PNR algorithms in the maze environment (MATLAB).



(a) RRT* algorithm

(b) RRT*-PNR algorithm

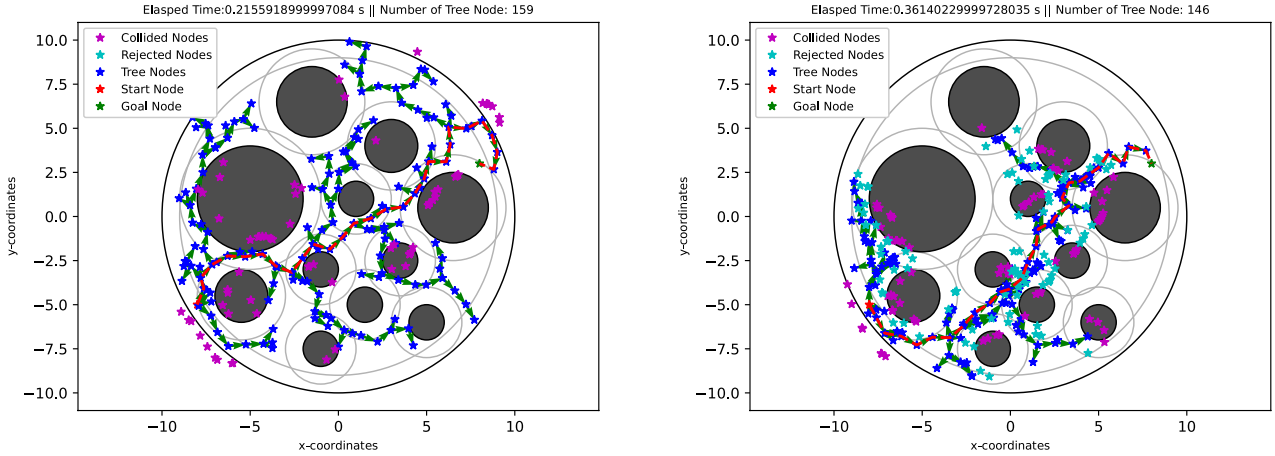
Fig. 7: The performances of the RRT* and RRT*-PNR algorithms in the narrow corridor environment (Python).



(a) RRT* algorithm

(b) RRT*-PNR algorithm

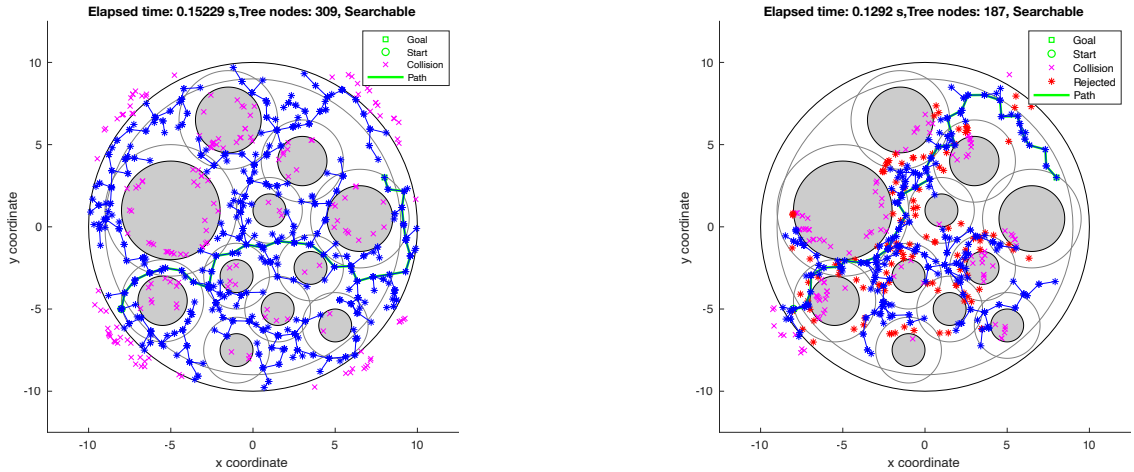
Fig. 8: The performances of the RRT* and RRT*-PNR algorithms in the narrow corridor environment (MATLAB).



(a) RRT* algorithm

(b) RRT*-PNR algorithm

Fig. 9: The performances of the RRT* and RRT*-PNR algorithms in the cluttered environment (Python).



(a) RRT* algorithm

(b) RRT*-PNR algorithm

Fig. 10: The performances of the RRT* and RRT*-PNR algorithms in the cluttered environment (MATLAB).

REFERENCES

[1] O. Adiyatov and H. A. Varol. Rapidly-exploring random tree based memory efficient motion planning. In *2013 IEEE International Conference on Mechatronics and Automation*, pages 354–359, 2013.

[2] B. Akgun and M. Stilman. Sampling heuristics for optimal motion planning in high dimensions. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2640–2645, 2011.

[3] J. Ding, Y. Zhou, X. Huang, K. Song, S. Lu, and L. Wang. An improved rrt* algorithm for robot path planning based on path expansion heuristic sampling. *Journal of Computational Science*, 67:101937, 2023.

[4] F. Islam, J. Nasir, U. Malik, Y. Ayaz, and O. Hasan. Rrt-smart: Rapid convergence implementation of rrt towards optimal solution. In *2012 IEEE International Conference on Mechatronics and Automation*, pages 1651–1656, 2012.

[5] S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning, 2010.

[6] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[7] 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*, pages 1478–1483, 2011.

[8] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[9] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98, 1986.

[10] S. M. LaValle. Rapidly-exploring random trees : a new tool for path planning. *The annual research report*, 1998.

[11] S. X. Lu, E. Li, and R. Guo. An obstacles avoidance algorithm based on improved artificial potential field. In *2020 IEEE International Conference on Mechatronics and Automation (ICMA)*, pages 425–430, 2020.

[12] M. C. L. Min Gyu Park. A new technique to escape local minimum in artificial potential field based path planning. *KSME International Journal*, 2003.

[13] I. Noreen, A. Khan, and Z. Habib. Optimal path planning using rrt* based approaches: A survey and future directions. *International Journal of Advanced Computer Science and Applications*, 7(11), 2016.

[14] A. Orthey, C. Chamzas, and L. E. Kavraki. Sampling-based motion planning: A comparative review, 2023.

[15] A. Perez, R. Jr, G. Konidaris, L. Kaelbling, and T. Lozano-Perez. Lqr-rrt : Optimal sampling-based motion planning with automatically derived extension heuristics. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2537–2542, 05 2012.

[16] J. Qi, H. Yang, and H. Sun. Mod-rrt*: A sampling-based algorithm for robot path planning in dynamic environment. *IEEE Transactions on Industrial Electronics*, 68(8):7244–7251, 2021.

[17] A. H. Qureshi and Y. Ayaz. Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments. *Robotics and Autonomous Systems*, 68:1–11, 2015.

[18] A. H. Qureshi and Y. Ayaz. Potential functions based sampling heuristic for optimal path planning. *Autonomous Robots*, 40(6):1079–1093, 2015.

[19] X. Tang and F. Chen. Robot path planning algorithm based on bi-rrt and potential field. In *2020 IEEE International Conference on Mechatronics and Automation (ICMA)*, pages 1251–1256, 2020.

[20] P. Vadakkepat, K. C. Tan, and W. Ming-Liang. Evolutionary artificial potential fields and their application in real time robot path planning. In *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)*, volume 1, pages 256–263 vol.1, 2000.

[21] P. Xin, X. Wang, X. Liu, Y. Wang, Z. Zhai, and X. Ma. Improved bidirectional rrt* algorithm for robot path planning. *Sensors*, 23(2):1041, Jan 2023.