PATH PLANNING IN 2D AND 3D ENVIRONMENTS

Project Report by “Team NP-Easy”
AER 1516: Robot Motion Planning

APRIL 19, 2023

Geet Patel (Student ID# 1003585293)
Harshil Thakkar (Student ID# 1007619739)
Sari Daniel Tarabay (Student ID# 1009685057)

Table of Contents

1. INTRODUCTION

Planning in robotics addresses the automation and actuation of mechanical systems that have sensing, actuation, and computation capabilities. As autonomous vehicles and autonomy in general is a growingly popular topic in robotics currently, a fundamental function in robotics is to have algorithms that convert high-level specification of tasks from humans into low-level discretized algorithms that set a path and execute the planned trajectory. This field, i.e., motion planning or trajectory planning has been commonly referred to as the Piano Mover's Problem. As early research in the field aimed at finding a theoretical path between two points, recent research focuses on adding on classical approaches and developing new strategies order to decrease the algorithm's space and time complexity while increasing the converge rate towards an optimal path.

Planning in robotics addresses the automation of mechanical systems that have sensing, actuation, and computation capabilities. As autonomous vehicles and autonomy in general is growing topic in robotics currently, a fundamental function of robotics is to have algorithms that convert high-level specification of tasks from humans into low-level discretized algorithms that set a path and execute a planned trajectory. This field, i.e., motion planning or trajectory planning has been commonly characterized by the Piano Mover's Problem. As early research in the field aimed at finding a theoretical path between two points, modern research focuses on adding on classical approaches and developing new strategies to decrease the algorithm's space and time complexity while increasing the converge rate towards the optimal path.

Since actions in the physical world are subject to physical laws, motion planning algorithms should consider uncertainty, differential constraints, and modeling errors. Overall, the algorithm should be complete, optimal and efficient. Completeness is an intrinsic value of the algorithm referring to resolution completeness or probabilistic completeness. The latter theorizes that given enough time, the algorithm should converge to the optimal path, if one exists. Whereas the former states that an algorithm that is resolution complete is an algorithm that would find an optimal path given adequate grid resolution and the existence of a path from start to goal. Optimality criterion represents an essential characteristic of the algorithm in which the path generated should be the best path from start to finish, and finally, efficiency refers to the algorithm finding a path within a reasonable timeframe.

Graphs are common approaches to model a motion planning problem in discretized or continuous space. Graph theory allows the representation of a robot's environment as a graph, where the nodes represent the robot's possible positions, and the edges represent the robot's possible motion between two nodes / poses. Conventional approaches to transform an environment into a graph include probabilistic sampling of nodes in a free space or discretization of the environment into a representative grid space. Luckily, graph theory has been extensively studied and algorithms such as Dijkstra's, A*, RRT, RRT* are able to solve for a path connecting a starting node to a goal node. In this study, we compare the performance of a grid-based algorithm, i.e., A* and two probabilistic based approaches, i.e., RRT* and PRM subject to a cluttered and "slit-based" environment in 2D and 3D.

2. BACKGROUND and RELEVANCE

The discipline of robot motion planning was launched in the mid 60's but did not gain popularity until Lozano-Perez's revolutionary contribution on spatial planning drew the attention of researchers in the field [1]. Lozano-Perez introduced a framework to approach robot motion planning problems by formalizing the configuration space definition, which unified mechanics, geometry, and computation to address motion planning for any shape of robots with any number of degrees of freedom. Lozano-Perez's work ignited the interest in robot motion planning research and was an inspiration to many.

However earlier work on motion planning dates to 1956 when Edsger Dijkstra was concerned of finding the shortest path between two cities in the Netherlands, Rotterdam, and Groningen. This led to the development of the shortest path algorithm, also known as Dijkstra's Algorithm. According to Dijkstra's,

his revolutionary algorithm was nothing but a "twenty-minute invention" conceived over a cup of coffee on a terrace with his fiancée Ria, which he had met at the *Mathematisch Centrum*, a research center in the field of mathematics and computer science… Initially Dijkstra's algorithm was used to demonstrate the potential of a new computer called ARMAC. It was not until 1959 that his work was published in a three-page article titled *"A Note on Two Problems in Connexion with Graphs"*.

Traditional Dijkstra's algorithm relies on a greedy path planning strategy, in which the shortest path to every node in the graph is found. The algorithm works by finding the next closest vertex by maintaining new vertices in a priority min-queue and storing only one intermediate node to find a unique shortest path from starting node to a goal node [2]. Dijkstra's algorithm is only concerned with finding the shortest path on a positive weighted graph and does not pay attention to the pragmatism of the solution. It is therefore memory heavy as it considers all possible outcomes to find the optimal path, yielding a computation complexity of $O(n^2)$, where n is the number of nodes in the graph [3].

Dijkstra's Algorithm was improved to yield a seminal graph search algorithm, A*. It operates like its predecessor except that it guides the search towards the most promising states through a heuristic, saving significant computation time [4]. The heuristic functions for guidance are, but not limited to, Euclidean distance, Manhattan distance, diagonal distance, or any suitable user defined heuristic for a specific application. A* defines three types of nodes (usually referred to as states), i.e., Unvisited, Dead and Alive states. Unvisited states represent all the states that have not been visited yet, initially it represents all the states except the starting state. Dead states are states that have been visited and for which each possible next state has also been visited. Finally, Alive states represent states that have been visited but have unvisited next states. This framework has been at the heart of the A* algorithm making it suitable for path planning in static environments. Computation efficiency of A* and its variants correlates with the accuracy of the heuristic functions which has been discussed in our literature review section.

A* was an undeniable improvement of the Dijkstra's algorithm and an important milestone in the field of motion planning, however it can be directly inferred that the reach of the A* algorithm is limited by its most common application to static environments. Therefore, making it unsuitable for tasks in mobile robotics that require online planning. This pushed the development of probabilistic motion planning algorithms, namely Probabilistic Roadmaps and Rapidly exploring Random Trees.

Probabilistic Roadmaps, also known as RPMs, belong to the family of probabilistic algorithms in which nodes are sampled in the free space, and each node is connected to nodes in its visibility space. This heuristic algorithm is probabilistically complete and therefore guarantees to find a path given the existence of a path and sufficient time horizon [1]. PRM's local sampling proved to be efficient, however it undermines the complexity of connecting two poses. In other words, if each connection is akin to a non-linear control problem, or non-linear model dynamics, then, considering all the possible connections in the configuration space seems very impractical and costly in robotics motion planning problems and related areas.

This was solved by Rapidly exploring Random trees algorithm, known as RRT, developed by Steven M. LaValle in 1998. LaValle introduced a randomized data structure design for a broad range of path planning problems. More specifically, RRT weas designed to handle nonholonomic constraints and high degrees of freedom [5] which was not possible with the aforementioned algorithms. This represented a major step forward in the field of robotics, motion planning and virtual prototyping as this approach provided a general framework applicable to a wide range of planning scenarios. RRT has proven to be probabilistically complete, meaning that it is guaranteed to find a path given the existence of a path and sufficient time. RRT does not guarantee finding an optimal path, however its descendant, i.e., RRT* guarantees asymptotic optimality towards an optimal path by continuous rewiring of the tree when a better path cost is found. This class of algorithms can be considered inefficient due to their bias towards unexplored states disregarding the fundamental goal of finding a path from a start state to a goal state. Therefore, researchers worked on an informed way of sampling along with heuristics to balance exploration and exploitation of the environment. Related approaches are presented in the subsequent section.

3. LITERATURE REVIEW

As part of the literature review, several resources were referred to including but not limited to previously published research and studies. An ample amount of independent and individual studies exists for the path planning algorithms of A*, the RRT family of algorithms and PRM, however, there is a dearth of research that takes a deep dive into comparing these algorithms with each other with a wide number of parameters. From the ocean of knowledge related to these algorithms, the research below was used as basis and to deepen the understanding.

3.1. The A* algorithm:

Very similar to some classical algorithms such as Dijkstra's algorithm, A* search algorithm is a widely used technique in finding the most optimal or shortest path in a graph or a grid. [6] However, it has also been utilized much broadly than its counterparts due to the ease of handling larger networks and yet being faster in finding the shortest path. It is also quite effective in finding the optimum path between two given points because it uses heuristics to guide the search towards the end node. [6]

Along with many pioneering works, in [7] Hart *et al* gave one of the first ideas about the applicability of A* in planning the minimum cost path between two points by introducing the heuristic function concept in its theoretical foundational form. It was shown that A* will always find a solution, if one exists, and it will be the shortest path if the heuristic function used for it is admissible. This seminal work in the field of pathfinding and motion planning discusses using a "weighted sum of Euclidean distances" to allow the algorithm to consider the relative importance of different nodes in the graph. This proposed modification can be useful in applications where some of the nodes in the environment are more critical than others.

The fundamentals laid out decades ago are continually being optimized. They are also actively used in many modern applications such as autonomous navigation of ground robots in warehouses, factories, hospitals and even restaurants. In recent research, many modified versions of the A* path planning algorithms are developed and tested in various environments. The speed with which an extension of this algorithm can be designed is remarkably high and hence gives an advantage to the user. In [4] the authors have discussed many such modifications in the commonly used motion planning algorithms and their applications in robotics. These algorithms include but are not limited to A*, D* and ARA*. For a simple 2D grid-like navigation example, the authors have also tested effectivity of an inflation factor (ϵ) – a parameter used in the process of creating a cost map which assigns a cost value to each point in the robot's environment based on the level of difficulty in navigating through that point. [4] Fig. 1 shows this comparison.

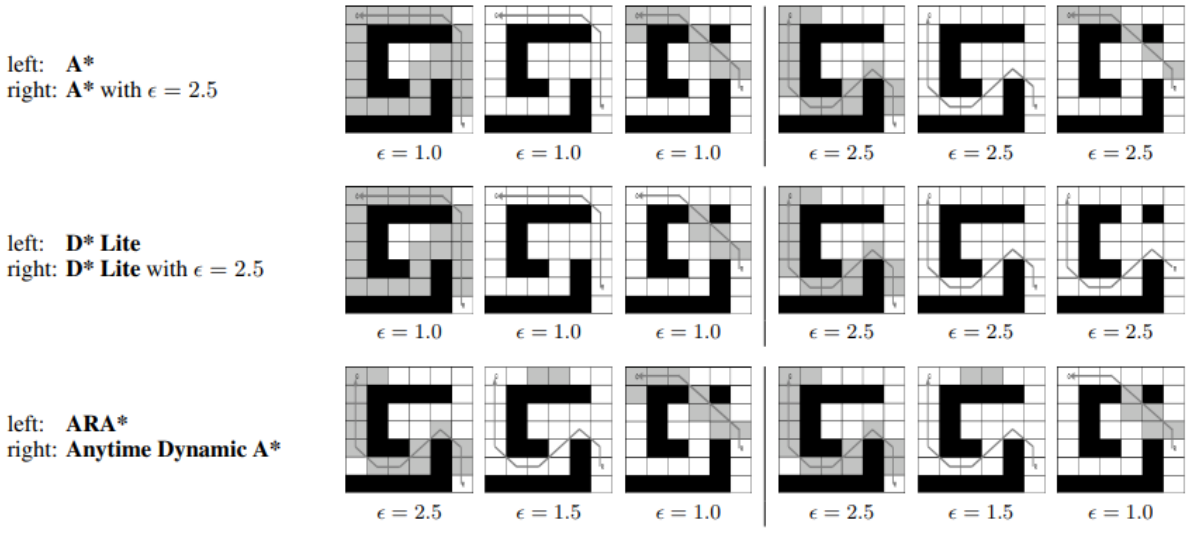


Figure 1: Comparative study of path planning algorithms with variants of A [4]*

Another such variant of A* is discussed in [8] with an aim at addressing some of the limitations of the traditional A* algorithm. Some of these limitations discussed in the article are slow planning speed, closeness to obstacles, etc. EBS-A* has been proposed to have an expansion distance (E), bidirectional search (B) and a smoothing stage (S), hence, its name! The expansion distance refers to the extra safe distance to be maintained from obstacles. While bi-directional search strategy is searching path from the start node and the goal node simultaneously, smoothing is for improving the quality of the path by reducing the number of right-angled turns. [8] showcases the verification of the proposed strategy by utilizing ROS. Fig. 2, Fig. 3, and Fig. 4 below show the three modifications made to a typical A* algorithm by [8].

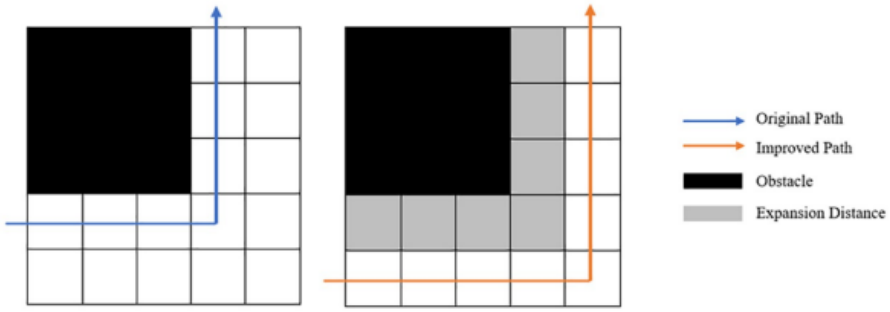


Figure 2: Schematic diagram of expansion distance [8]

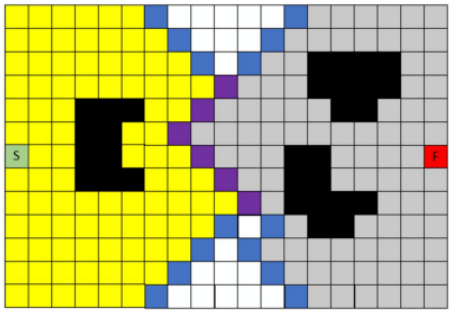


Figure 3: Schematic diagram of bidirectional search [8]

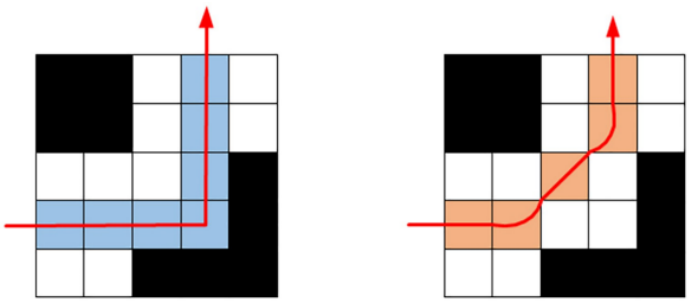


Figure 4: Schematic diagram of smoothing strategy [8]

A detailed comparison is made in this research paper in terms of running time, number of nodes, path length, number of turns, maximum turning angle, etc.

There are plenty of resources available that discuss the application of A* for path planning in various scenarios with most of the resources available in 2D and some also focusing on applicability in 3D environments. [9] is one such sample focused on planning optimal paths considering turning costs for AGVs.

3.2. The RRT* algorithm:

The rapidly exploring random tree, also known as RRT, is a widely used algorithm for motion planning. It belongs to the class of sampling-based algorithms in which a tree data structure is created through random sampling from the workspace of the robot [10]. The sampling process typically follows a uniform spatial distribution, however modern researchers argue that incorporating a non-uniform or informed sampling approach would help speed up the planning and running time of the algorithm. In this section of the literature review, we will survey some variations of RRT* along with their benefits… (the value addition to RRT*)

In [11] Mohammed *et al* explore an informed way of sampling to address random node generation in 2D and 3D, static and dynamic environments. In their work, nodes are not randomly sampled but rather nodes are sampled from a controlled normal distribution. In other words, nodes are generated along a line connecting the start and goal positions and then shifted by a specified gaussian distribution with zero mean and a desired standard deviation. It was shown that within a user specified number of iterations, their proposed approach, called RRT*N proved to always find a valid path in a time efficient manner.

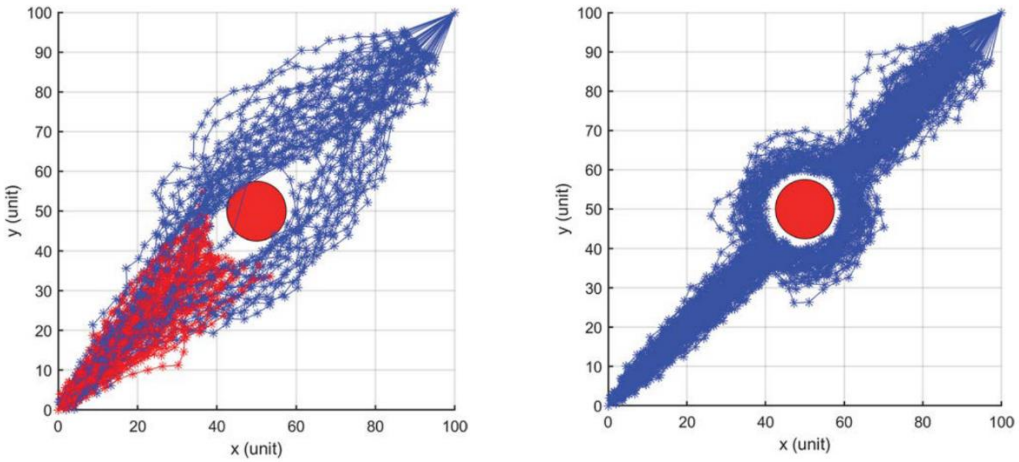


Figure 5: RRT (left) and RRT*N (right) simulation in 2D dynamic Environment. (Courtesy of [11])*

However, this does not solve the sampling problem in a highly cluttered environment or one in which the path to the goal is a narrow path. Therefore, Qureshi *et al* propose a new sampling approach based on potential functions (P-RRT*) in their work [12].The potential function-based sampling heuristic combines both exploration and exploitation in a single objective function. The heuristic is derived by a weighted sum of attractive force towards the goal and repulsive force at the vicinity of obstacles. P-RRT* proved to have an increased convergence rate with a smaller number of samples yet still inherits asymptotic computation complexity and asymptotic optimality form RRT*.

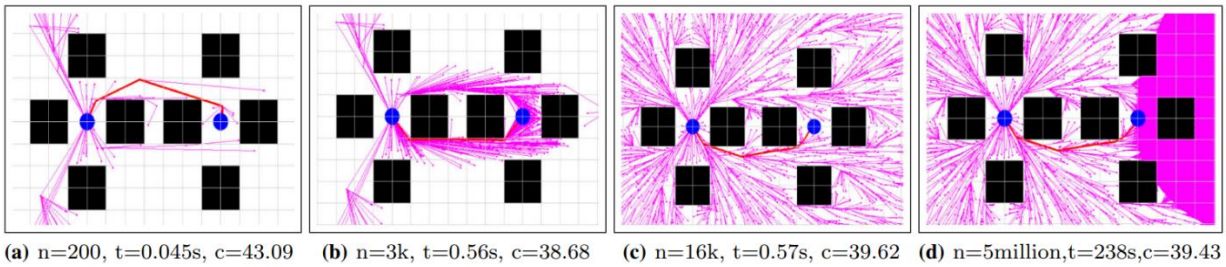
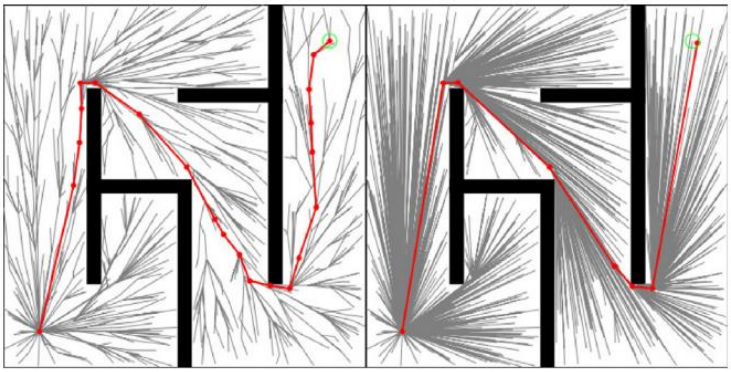


Figure 6: Performance of P-RRT (a, b) and RRT* (c, d) in Cluttered Environment. (Courtesy of [12])*

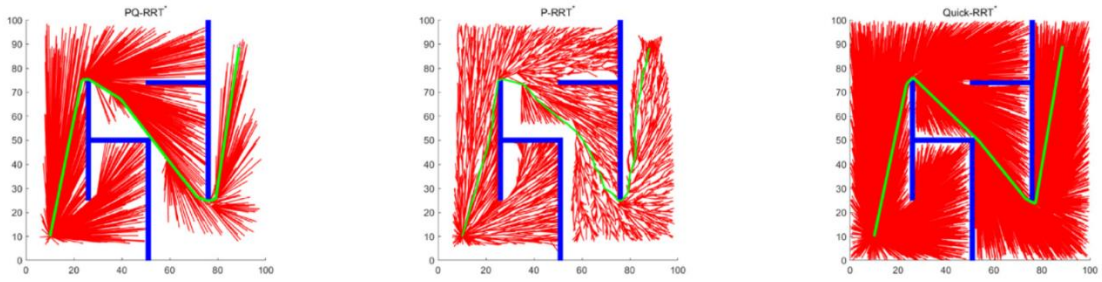
So far, sampling strategy has proven to improve algorithmic efficiency in terms of space complexity and convergence rate, however efficiency can also improve by improving tree growth process and rewiring. Jeong *et al* [13] investigates a new variant of RRT* which they call Quick – RRT* (Q-RRT*). Initial solutions close to the optimal path are generated using triangular inequality followed by a bidirectional search to converge towards the optimal path. In addition, Quick RRT* does not only consider parents of a node defined by a hypersphere but enlarges the set of possible parent vertices, considering their ancestry up to a user defined threshold that they called depth. Therefore the "depth" represents a tunable hyperparameter, or a tradeoff between computation time and convergence speed.



(a) RRT* (b) Q-RRT*

Figure 7: RRT and Q-RRT* performance (Courtesy of [13])*

Finally, Li *et al* [14] proposes an algorithm that merges two RRT* variants, namely the P-RRT* and the Q-RRT* algorithm to develop the PQ-RRT* algorithm combining the advantages of both approaches. The goal is to deal with the asymptotic convergence of the conventional RRT* algorithm which makes it inefficient in certain rapid path planning situations. The authors evaluated the performance of their algorithm on a two-wheeled and a six-wheeled robot. Failure was defined by the inability to find a path within 300 seconds. The authors' approach performed best in terms of repeatability, i.e., the standard deviation on the path length over several trials proved to be robust and consistent.



(a) PQ-RRT* (b) P-RRT* (c) Q-RRT*

Figure 8: Performance of the three algorithms in the environment maze 2 (Courtesy of [14])

3.3. The PRM algorithm:

Another simplistic and widely used path planning approach is Probabilistic Roadmap (PRM). It involves construction of a roadmap by sampling random points in the environment and connecting them. The connected points make a graph from which the collision-free path for the robots can be planned.

One of the pioneering by Kavraki *et al* in [15] indicates the usage of PRM for optimizing the performance of path planning using Gaussian distribution to generate random points in the environment

and plan a roadmap which represents the environment more closely. This work implemented the planner in C and presented results that realize the outputs visually, as shown in Fig. 9 below.

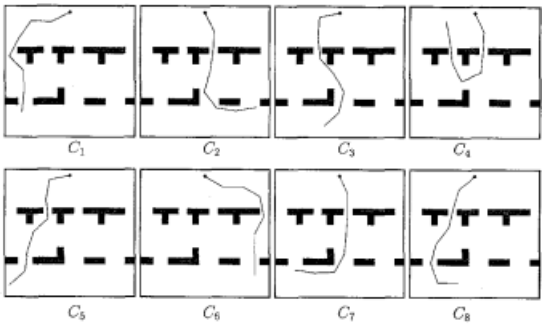


Figure 9: PRM implementation for a 7-joint fixed-base robot [15]

As per [15], the algorithm intended to run in phases known as the Build phase – where random points are generated and connected as a graph and the Query phase finds the path to the goal node. However, since it must construct the roadmap before running the planner, it is well designed for static environments with no internal or external moving elements in it. Due to the nature of the original PRM algorithm, it is inherently slow when attempting to utilize end-to-end for changing environments.

Like A* and RRT*, the PRM also has multiple variants to deal with its drawbacks as above. Lazy PRM introduced in [16] assumes that all the nodes generated in the environment initially are collision-free and instantly continues with the task to look for the shortest path. During the search for the shortest path, it evaluates the path for any collisions and updates the path if required. Lazy PRM contributes to saving planning time required by the planner. [16] is also implemented in a real-life scenario for a robot traversing in a work-cell with 3D obstacles.

Just like Lazy PRM, Toggle PRM is proposed to mitigate the difficulties with planning the path in the narrow passages. [17] Toggle PRM maps not only the free space but also the obstacle space, attempting to improve the overall efficiency of the path planner in the narrow passages as shown in Fig. 10.

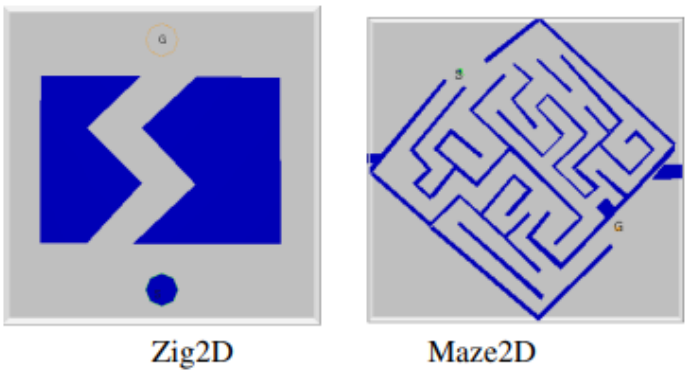


Figure 10: 2D environments with narrow passages used to test Toggle PRM [17]

For dynamically changing environments, T-PRM was introduced [18] which takes into consideration the time availability of the moving obstacles in the environment to avoid the collision during traversal. As the research in the sampling-based path planning approach has progressed, it also has seen many more versions of Lazy PRM and Toggle PRM, such as the combined version of both i.e., Lazy Toggle PRM that reduces the querying time and is more effective for denser environments. [19]

With plenty of resources available for individual research on each of the above algorithms, there is a dearth of comparative studies in terms of various performance parameters. A few of the studies reviewed as part of this project that include a level of comparison between algorithms include [20] where A* and RRT

algorithms are compared for path planning in the environment where UAVs must traverse. The algorithms are conveniently modified to consist of a smoothing step for this 3-dimensional path planning application.

[20] has simulated the algorithm in a 3D environment with planar obstacles as shown in Fig. 11. The comparisons are made only between parameters such as path length, computational time and smoothing computational time. There are no considerations of any qualitative parameters.

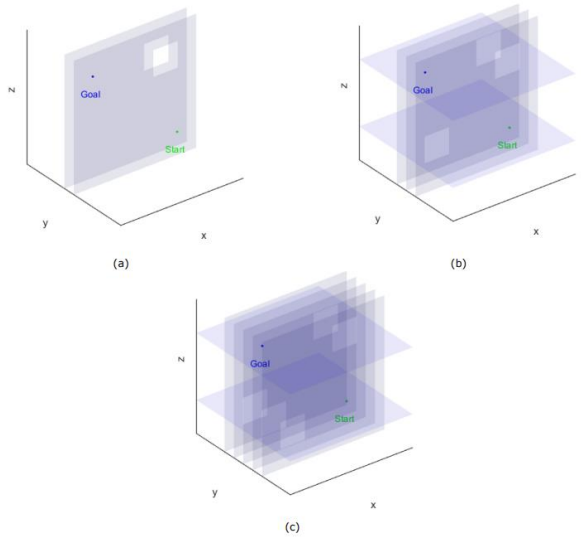


Figure 11: 3D obstacles used for the simulation [20]

[21] shows a comparison between RRT, PRM and a hybrid algorithm of the two focused at 2-dimensional environments in grid-like setups (Fig. 12). The parameters compared for the given algorithms are in terms of the path length, the runtime, number of nodes in the path, number of nodes in the roadmap and the number of iterations.

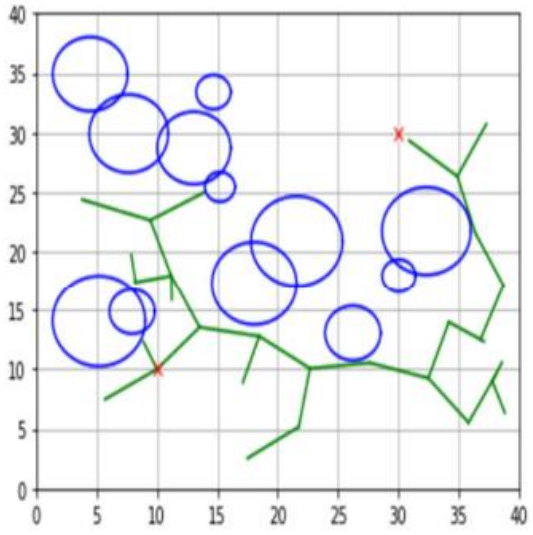


Figure 12: Search tree for RRT [21]

4. IMPLEMENTATION

Only a small number of works have studied path planning algorithms from the qualitative comparison point of view. The angle of the turns made in the path; the number of turns made are such parameters that can be inferred to as factors affecting the quality of the motion of robot. For example, for an autonomous taxi, the sharper the turn it takes the jerkier the motion may be. Similarly, the higher the number of turns a delivery quadrotor makes, the higher the time and power it consumes. This study has been conducted with the primary aim of offering some insights into the effectiveness of commonly used path planning algorithms on these qualitative parameters.

4.1. Experimental Setup

Our goal is to comment on the potential uses of different algorithms. To do so, our experimental setups are designed to highlight the common uses of various robots. We have used two primary environments: 2D and 3D, these will help us identify the best path planners for a ground robot and for a UAV. The 2D environment spans 20m x 20m and the 3D environment spans 20m x 20m x 20m. The choice of the dimensions of our 2D and 3D environments was chosen to have a balance between being sufficiently large to mimic the real-world, not being large enough to cause computational impracticality. Further, we have used two sets of obstacle configuration: one, a cluttered environment with many random obstacles and two, a slit-featured environment, with only a few obstacles but positioned with forethought in a way to create slits. The cluttered environment is a representation of a densely packed space, for example any urban environment like a mall. The slit-featured environment is a representation of tasks/environments that require precision in path planning, for example in UAV racing.

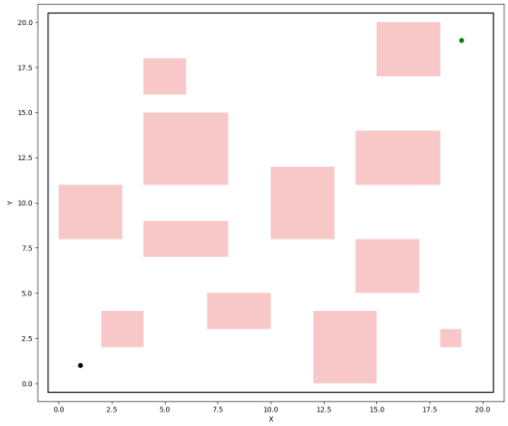


Figure 13: 2D Cluttered Environment.

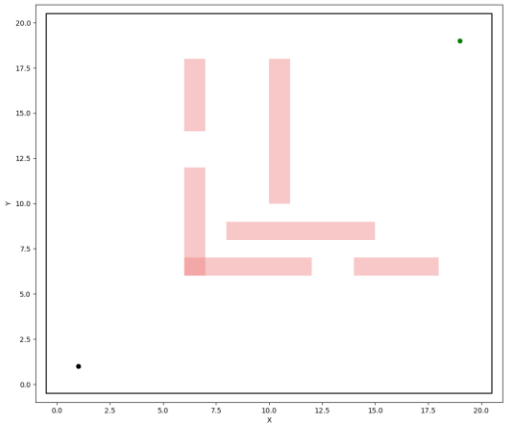


Figure 14: 2D Slit-based Environment.

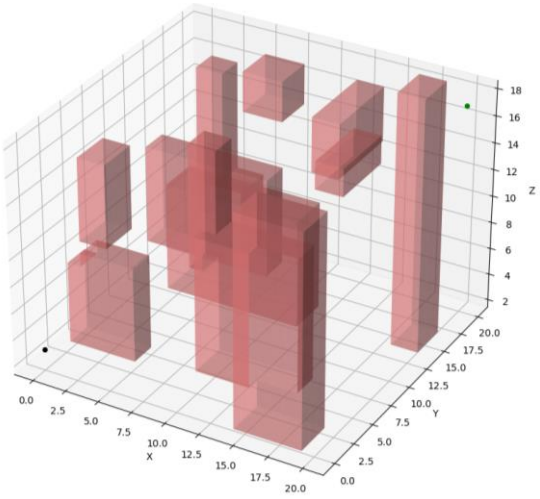


Figure 15: 3D Cluttered Environment.

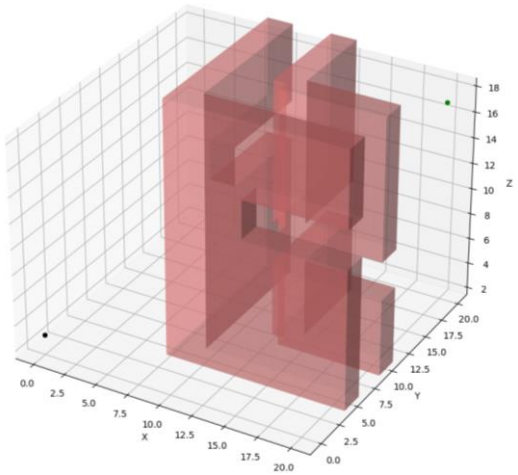


Figure 16: Slit-based Environment.

The environments can be seen in figures 13-16. The start location for 3D was chosen to be (0,0,3) and the goal location as (20, 20, 17), the choice was arbitrary and only for the benefit of better visualization. While the start location for 2D was chosen to be (1,1) and the goal location as (19,19). Note, to add complexity, the slit-based environments in both 2D and 3D, have two misaligned slits. Finally, the 3D and 2D environments allowed the robot to traverse the edge, which is along the border at 20m or 0m from the axis, but no further. Hard bounds on exceeding these borders were implemented.

4.2. Algorithms

We will test and analyze 3 algorithms: A*, Rapidly exploring Random Tree* and Probabilistic Roadmap. The implementation of these three algorithms will now be explained.

The A* path planner was used from an implementation created for another U of T course, by a member of the team. The implementation was based on creating a grid in the environment. The grid size was chosen to be 1m x 1m (x 1m), with bounds as [0, 20] in x, y, and z directions. Each node in the grid has the following attributes: coordinates (x, y, and z), cost to come, cost to go, pointer to parent and a collision flag signifying if that node coincides with an obstacle of not. The algorithm begins by generating and initializing the grid and using the pre-defined obstacles to set the collision flag for all the nodes. Once the grid is built, the search for the optimal path begins. The search is based on iterative exploration of the node with the least total cost. The planner maintains an open list of all the nodes in the grid that are at the frontier of exploration. Initially only the start node is added to this open list. At each iteration, the node with the

least total cost, which is the sum of cost to go and cost to come, is chosen to be explored further. The cost to go is used as the Euclidean distance traversed from the starting node and the cost to go is used as the Euclidean distance to the goal. Further exploration means to identify the neighbor/children of the node being explored. Once the frontier reaches the goal, the search stops and an optimal path using A* is generated.

The RRT* algorithm used for this project was acquired from an open-source GitHub repository [22]. This implementation began by first creating a search space based on the boundaries provided and the obstacles provided. The generation of search space was simply meant to restrict the randomizer to only generate random nodes that lie in the free space. The RRT* search begins by generating a random coordinate within the free space; however, this implementation has a modification that only grows the tree for a tunable length in the direction of the randomly generated coordinate. We tuned this max tree growing length to be 8m. Further, the implementation used the dynamic collision checking approach, in which the path from a node to a new node was discretized into a collection of points using a tunable resolution. Each of these points were then checked against the free space identified in the beginning to check for collision. We tuned this resolution for discretization of a new tree edge as 0.25m, that is break each edge into coordinates 0.25m apart. After, a new random node was added to the tree after collision checking, as per RRT* algorithm its neighbors were identified. This implementation identified 32 of the closest neighbors. In accordance with the RRT* algorithm these 32 neighbors were then used to see it any of them can help optimize the path based on the total cost metric (cost to come + cost to go), which was a Euclidean distance. Finally, this implementation performs the second phase of rewiring to check if changing of the tree can help optimize path to any of the other neighbors. Finally, after adding a new node to the tree, this implementation has one more modification, 1 out of 10 times, it attempts to connect the new node with the goal. When the attempt is successful, the algorithm ends, and a path is generated based on RRT*. However, in case a path cannot be found, simply to prevent the RRT* from running forever we created an upper limit to the number of random nodes that may be generated during this search, which was set to 1000, for both 2D and 3D environments.

PRM algorithm used for this project was acquired from an open-source GitHub repository [23]. We first generate 1000 random coordinates in both 2D and 3D environments. A K-D tree is generated that comprises of these random coordinates. Iteratively edges are constructed, checked for collision, and corresponding a roadmap is developed. The implementation iterates over each random point, identifies another random point in the K-D tree closest to it and the corresponding edge joining the two is checked for collision in a similar dynamic collision checking method, where the edge is discretized and each point on this edge is checked for collision. The edges that are collision free are added to the roadmap. To check for collision another K-D tree is used that is defined by the obstacles in the environment, and the distance to this obstacles' K-D tree is used to decide if an edge is colliding or not. Once the roadmap is formed, with collision free random coordinates connected by collision free edges, the Dijkstra's path planning algorithm is used to search for the path. Dijkstra operates in a similar fashion to A*, with the difference being in the cost used to identify the node that will be explored further. Dijkstra's only uses the cost to go for this purpose.

4.3. Results:

We measured the Path length; the path computation time and the path quality will be discussed in terms of the number of turns and the maximum turning angle for a path. Further, note that the RRT* and PRM are randomization-based algorithms, hence their results are based on an average of 10 runs. The detailed results for each of these 10 runs are provided in the appendix for reference. Sample plots for each algorithm in each environment are shown in figures 17-28.

Table 1: Quantitative Results of Path Planning

	2D					
	Cluttered			Slit-based		
	A*	RRT*	PRM	A*	RRT*	PRM
Path length [m]	28.39	37.83	30.08	29.62	33.97	34.10
Computation Time [s]	0.28	0.10	0.23	0.37	0.048	0.20
Number of Turns	6	4	7	10	2	9
Max Turning angle [deg]	~45	~121	~84	~45	~97	~68
	3D					
	Cluttered			Slit-based		
	A*	RRT*	PRM	A*	RRT*	PRM
Path length [m]	35.51	38.61	49.11	39.67	51.43	53.23
Computation Time [s]	136.35	0.19	0.74	586.20	0.15	0.62
Number of Turns	6	3	11	12	3	12
Max Turning angle [deg]	~45	~83	~133	~45	~113	~107

In 2D, as one might expect, the shortest possible path length is always identified by A*. With the path length of RRT* and PRM being dependent on the environment, in the randomly designed cluttered environment RRT* performs poorly and has significantly higher average path length compared to PRM and A*. While in the slit-based environment both RRT* and PRM have similar path lengths, RRT* performs better. An observation from the detailed results suggests that the longer RRT* works the more optimal path it finds, hence it may help to continue exploring even after the goal is found to attain an optimal path. Implementing such a modification to RRT* will ensure that we always generate shorter with RRT* than with PRM. In terms of computational time, RRT* outperforms the other algorithms for both the obstacle configurations. A* is computationally more demanding than PRM as well. The quality of path is a subjective metric, however, based on how much and how drastically a robot may have to turn, we can observe that A* consistently generates the most moderate path, which is with minimal changes in direction. While RRT* and PRM have a tradeoff between number of turns and the maximum turning angle. RRT* can get to the path in the least number of turns out of all the three algorithms, however, it does so with high turning angles. PRM requires smaller turning angles, but a lot more turns.

In 3D, path length is shortest for A* and followed by RRT* for both the obstacle configurations. The path length from PRM is dependent on the obstacle configuration, with it performing poorly on the slit based configuration. An eye-catching stark difference can be observed in the computational times for the 3D environments. While RRT* and PRM take just a fraction longer than they did in 2D, A* takes significantly longer in 3D. Observing this in A*, we further dived into the computation time of different phases of the algorithm and concluded that 99.88% of the computational time for A* was spent in identifying the list of 26 neighbors of the open node. The implementation for this was based on a for loop that ran 26 times to account for the 26 different possible directions a neighbor might be in, and each time, the algorithm iterated through the entire the grid to identify the node that had the shortest distance from the open node, in that direction. But even in the best-case scenario where all this 99.88% of time is saved, A* was still always computationally the more demanding than RRT* and PRM. The path quality in 3D was again the best for A*, with minimal need for change in direction. However, now, PRM always performs poorly, producing more and sharper turns. RRT* followed the same behavior as in 2D, that is, it got to the goal, with less turn but with sharper turns.

In conclusion, we can say that A* always produces the shortest path and of the best quality in 2D and 3D, however, to achieve this, it takes significantly longer. RRT* on the other hand, can be seen as a close rival to A* with significantly low computational times, and comparable path length. However, choosing RRT* means the quality of the path will be poor.

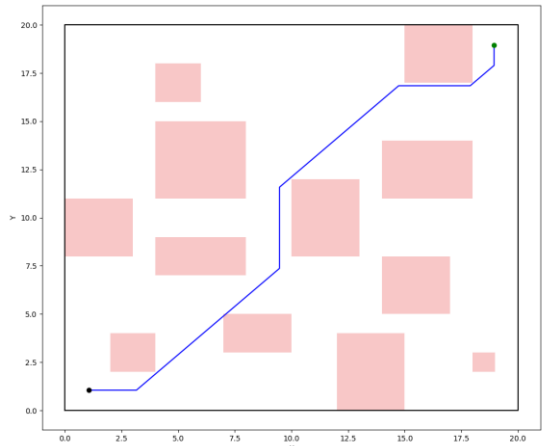


Figure 17: A in 2D Cluttered Environment.*

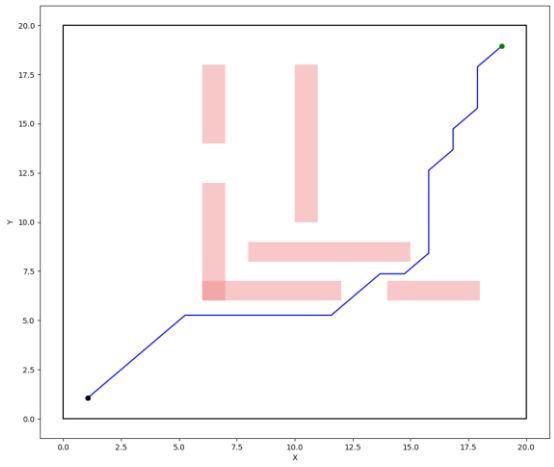


Figure 18: A in 2D Slit-based Environment.*

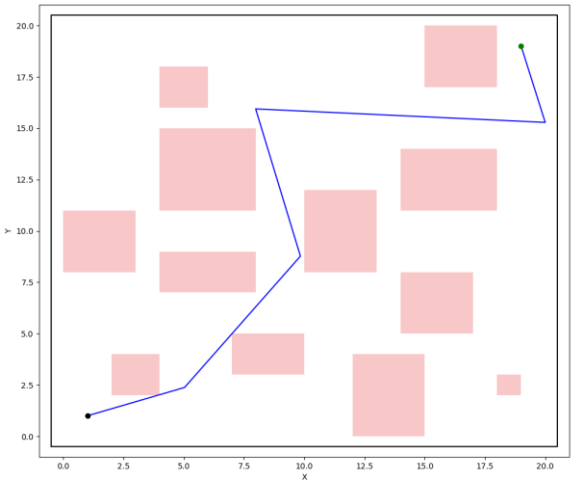


Figure 19: RRT in 2D Cluttered Environment.*

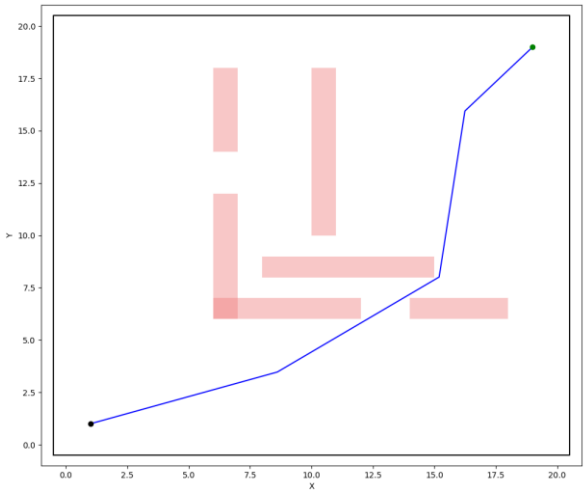


Figure 20: RRT in 2D Slit-based Environment.*

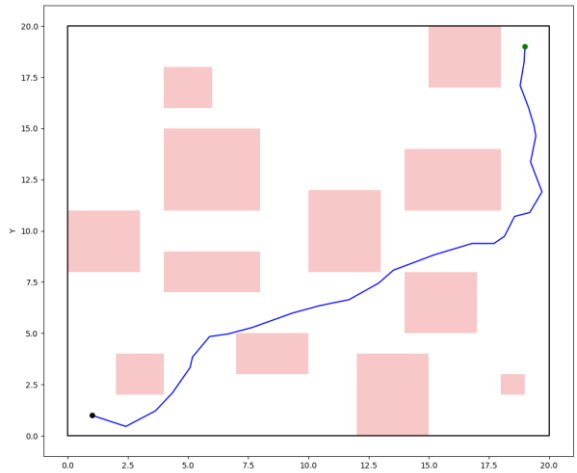


Figure 21: PRM in 2D Cluttered Environment.

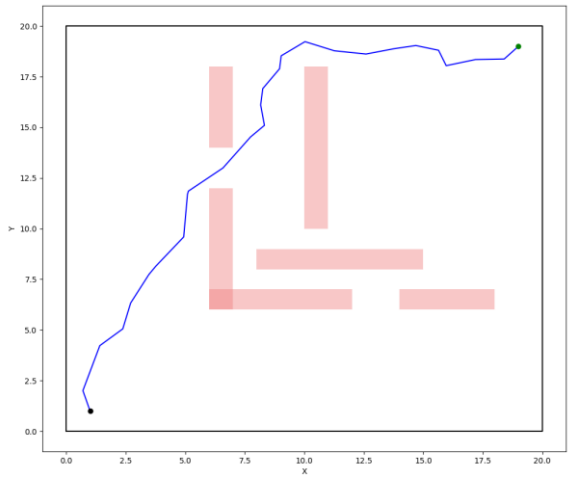


Figure 22: PRM in 2D Slit-based Environment

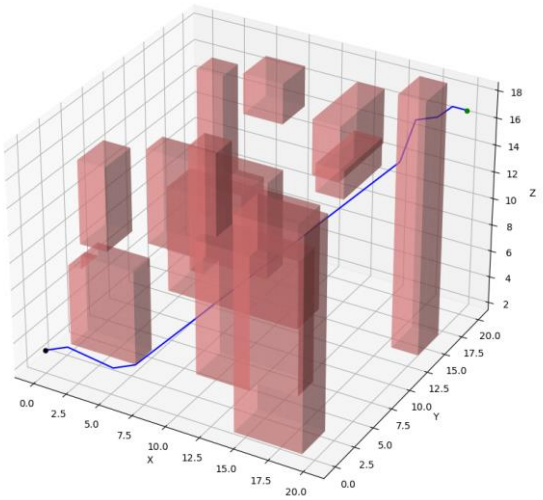


Figure 23: A in 3D Cluttered Environment.*

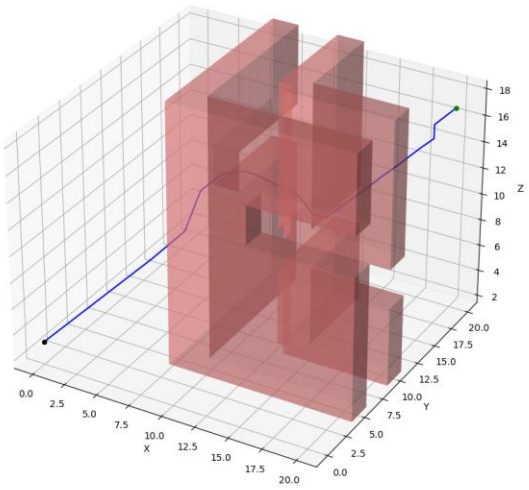


Figure 24: A in 3D Slit-based Environment.*

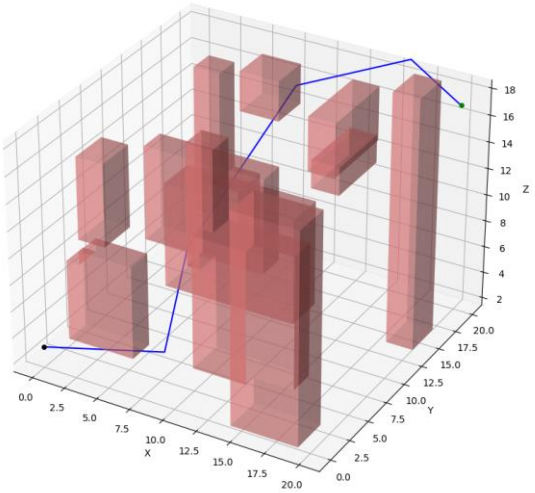


Figure 25: RRT in 3D Cluttered Environment.*

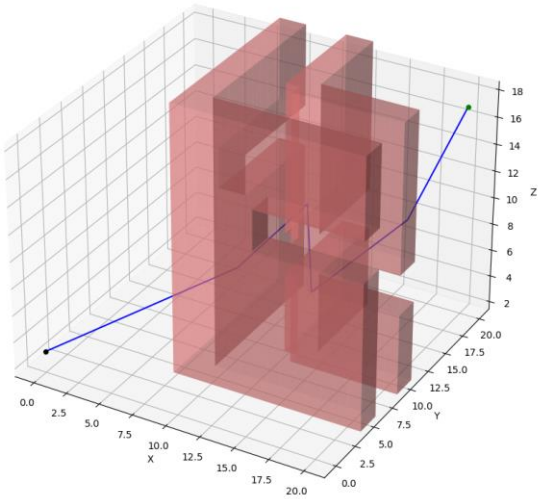


Figure 26: RRT in 3D Slit-based Environment.*

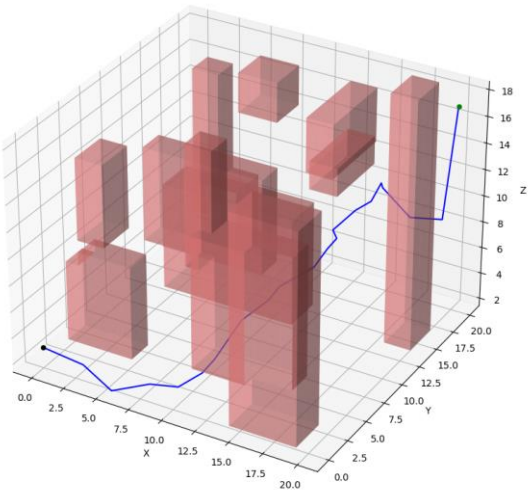


Figure 27: PRM in 3D Cluttered Environment.

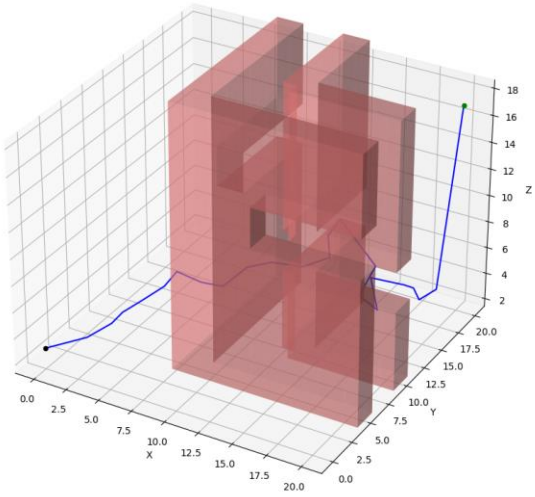


Figure 28: PRM in 3D Slit-based Environment.

5. CONCLUSION

2D path planning is primarily used for ground mobile robots like security guard/patrol at businesses, a customer assistant at shops and malls, a butler at home or an industrial robot at manufacturing plants. These robots have humans in their environments, leading to a critical constraint on the quality of path the robot follows. Hence, RRT* cannot be used in for these robots as the sudden change in direction can alarm or confuse the humans around it. Now, for uses where time is not a constraint, we can use A* to obtain the shortest path of the best quality. However, when time is of the essence, as may be in the case of security guards, it makes more sense to use PRM and trade a good path in exchange for a path that is computed faster. 3D path planning is primarily used for unmanned aerial vehicles or drones, which have applications in delivery service, defense industry, and photography. First, we must eliminate the A* algorithm from being used in these robots, as they are computationally expensive, unless one is able to reduce these times significantly. Out of RRT* and PRM, RRT* generates shorter paths, with less turns and smaller turning angles, hence RRT* should be used for UAVs or drones. Lastly, special uses like UAV racing or fast-reacting security guards, or any case where faster path planning is of importance, we would want to use RRT* when there are no humans in the environment or PRM if there are.

6. FUTURE WORK

The very first extension to our work here would be to explore and compare even more path planning algorithms. As mentioned in our literature review there are many extensions to the traditional A*, RRT* and PRM developed through the years. Further, other algorithms like the Genetic algorithm, the ant colony algorithm or even the Q-learning based algorithms can be studied and evaluated against each other. Our work was limited to static environments, but to truly gauge the applicability of different algorithms on real-world robots, a thorough investigation of the algorithms in dynamic environments must be performed. For future studies of similar kind, it may be important to compare the scalability and adaptability of different algorithms. This comparison will be necessary as we see robots used in different environments like underwater and for intra city drone taxis. Finally, an obvious yet very important future work of this project is implementation and the subsequent analysis and comparison of different algorithm on real-world robots.

7. REFERENCES

[1] Masehian, E., & Sedighizadeh, D, "Classic and heuristic approaches in robot motion planning - a chronological review," *World Academy of Science,* vol. 5, no. 23, pp. 101-106, 2007.

[2] Karur, K., Sharma, N., Dharmatti, C., & Siegel, J. E, "A survey of path palnning algorithms for mobile robots.," *Vehicles,* vol. 3, no. 3, pp. 448-468, 2021.

[3] Fuhao, Z.; Jiping, L. , "An Algorithm of Shortest Path Based on Dijkstra for Huge Data.," in *In Proceedings of the 2009 Sixth International Conference on Fuzzy systems and Knowledge Discovery*, Tianjin, China, 2009.

[4] Ferguson, D.; Likhachev, M.; Stentz, A. , "A Guide to Heuristic-based Path Planning," in *International Workshop on Planning under Uncertainty for Autonomous systems, International Conference on Automated Planning and Scheduling*, Monterey, CA, 2005.

[5] S. LaValle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning.," 1998. [Online]. Available: http://lavalle.pl/papers/Lav98c.pdf. [Accessed 18 April 2023].

[6] Sunjana, "Application of A* Algorithm as a Solution for Finding the Shortest Route," *Turkish Journal of Computer and Mathematics Education (TURCOMAT),* vol. 12 (8), pp. 658-663, 2021.

[7] P. E. Hart, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions of Systems Science and Cybernetics,* Vols. ssc-4, pp. 100 - 107, 1968.

[8] H. Wang, "The EBS-A* algorithm: An improved A* algorithm for path planning," *PLoS ONE,* vol. 17, no. 2, 2022.

[9] K. Fransen, "Efficient path planning for automated guided vehicles using A* algorithm incorporating turning costs in search heuristic," *International Journal of Production Research,* vol. 61, no. 3, pp. 707-725, 2023.

[10] L. G. D. O. Véras, F. L. L. Medeiros and L. N. F. Guimaráes, "Systematic Literature Review of Sampling Process in Rapidly-Exploring Random Trees," *IEEE Access,* vol. 7, pp. 50933-50953, 2019.

[11] Mohammed, H., Romdhane, L., & Jaradat, M. A, "RRT*N: An efficient approach to path planning in 3D for static and Dynamic Environments.," *Advanced Robotics,* vol. 4, no. 35, pp. 168-180, 2021.

[12] Qureshi, A. H., & Ayaz, Y., "Potential functions based sampling heuristic for optimal path planning," *Autonomous Robots,* vol. 40, pp. 1079-1093, 2016.

[13] Jeong, I. B., Lee, S. J., & Kim, J. H., "Quick-RRT*: Triangular inequality-based implementation of RRT* with improved initial solution and convergence rate," *Expert Systems with Applications,* no. 123, pp. 82-90, 2019.

[14] Li, Y., Wei, W., Gao, Y., Wang, D., & Fan, Z., "PQ-RRT*: An improved path planning algorithm for mobile robots.," *Expert systems with applications,* no. 152, p. 113425, 2020.

[15] L. E. Kavraki, "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," *IEEE Transactions on Robotics and Automation,* vol. 12, no. 4, pp. 566-580, 1996.

[16] R. Bohlin, "Path Planning Using Lazy PRM," *Proceedings of the 2000 IEEE International Conference on Robotics & Automation,* 2000.

[17] J. Denny, "Toggle PRM: Simultaneous Mapping of C-free and C-obstacle - A study in 2D," *International Conference on Intelligent Robots and Systems,* 2011.

[18] M. Huppi, "T-PRM: Temporal Probabilistic Roadmap for Path Plannig in Dynamic Environments," in *IEEE/RSJ International Conference on Intelligenet Robots and Systems (IROS)*, 2022.

[19] J. Denny, "Lazy Toggle PRM: A Single-Query Approach to Motion Planning," *International Conference on Robotics and Automation (ICRA),* 2013.

[20] Z. Christian, "Comparison between A* and RRT algorithms for UAV path planning," in *Proceedings of the 2018 AIAA Guidance, Navigation, and Control Conference*, 2018.

[21] J. Jermyn, "A comparison of the effectiveness of the RRT, PRM, and Novel Hybrid RRT-PRM path planners," *International Journal for Research in Applied Science & Engineering Technology (IJRASET),* vol. 9, no. XII, pp. 600-611, 2021.

[22] T. Kose and SZanlongo, "rrt-algorithms," 24 April 2020. [Online]. Available: https://github.com/motion-planning/rrt-algorithms. [Accessed March 2023].

[23] A. Sakai, "PythonRobotics," April 2023. [Online]. Available: https://github.com/AtsushiSakai/PythonRobotics. [Accessed April 2023].

[24] D. Ferguson, "A Guide to Heuristic-based Path Planning," *American Association for Artificial Intelligence,* 2005.

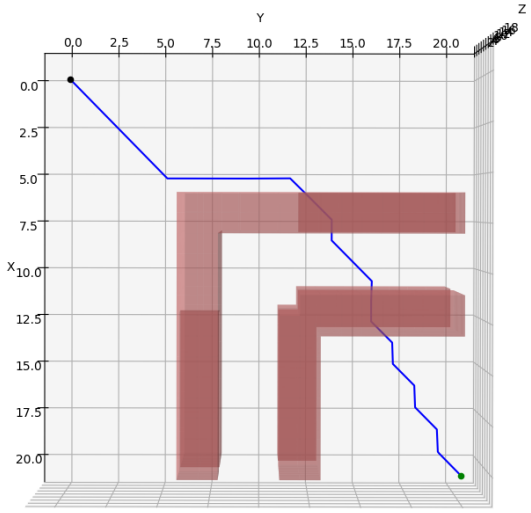
[25] S. Karaman, "Incremental Sampling-based Algorithms for Optimal Motion Planning," *arXiv:1005.0416v1,* 2010.

8. APPENDIX

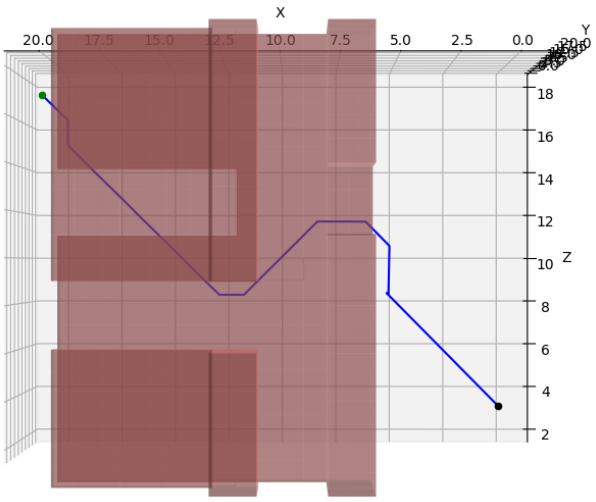
8.1. 3D Visualizations – different angles

To verify that the paths found were collision free, it is important to look at the 3D paths from different angles. Here are the different angles that help us verify a collision free path was found.

A* - Slit-based Environment:

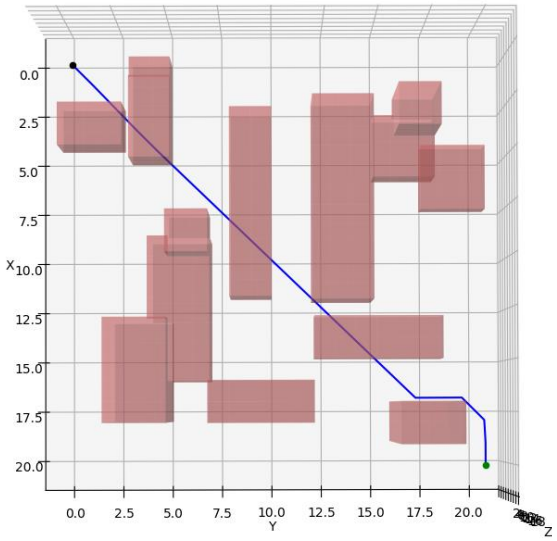


Top View

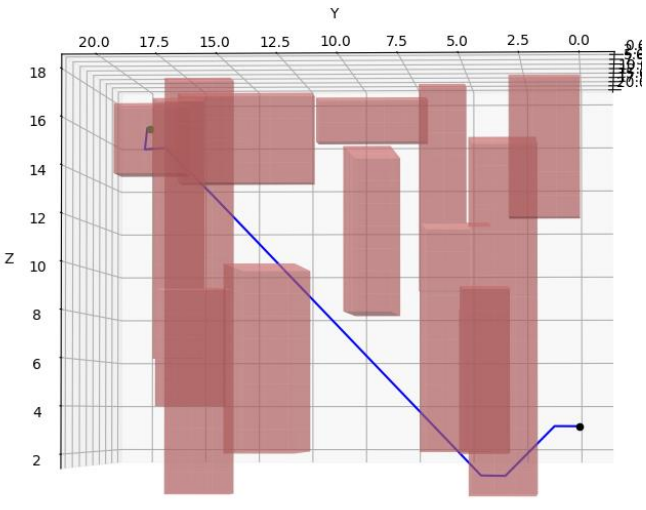


Side View

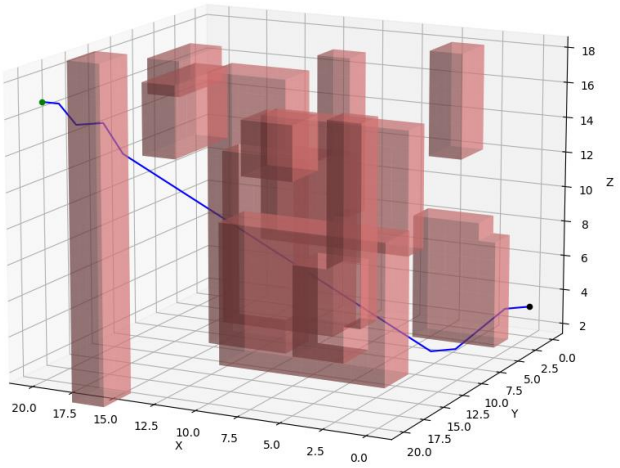
A* - Cluttered Environment:



Top View

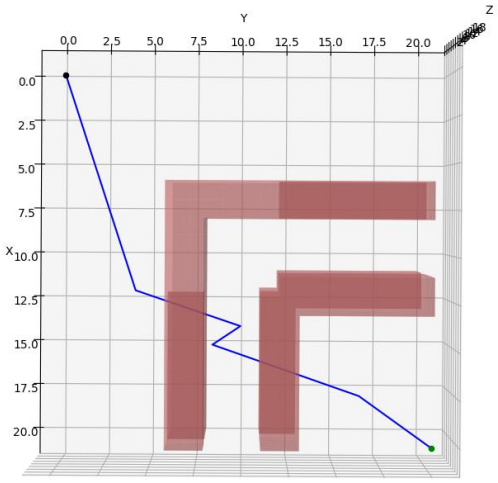


Side View

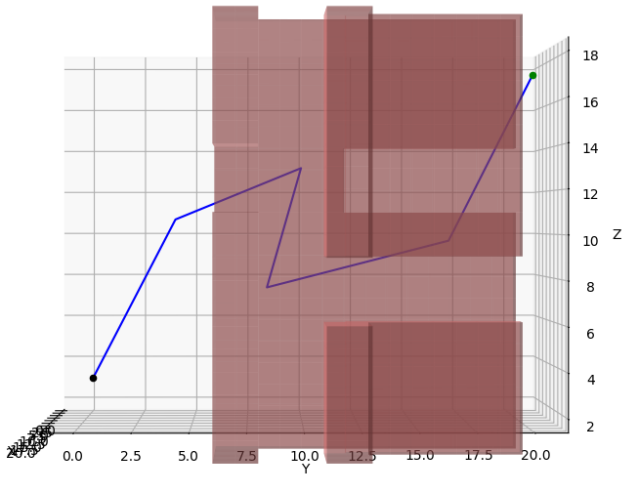


Angled View

RRT* - Slit-based Environment:

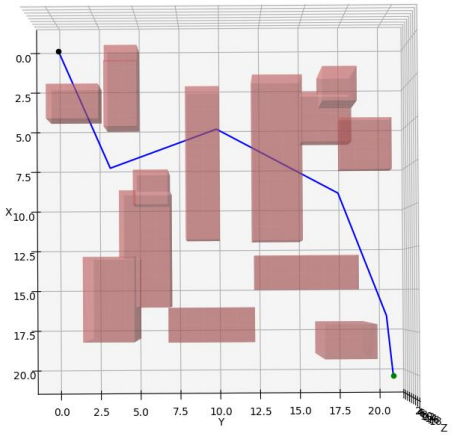


Top View

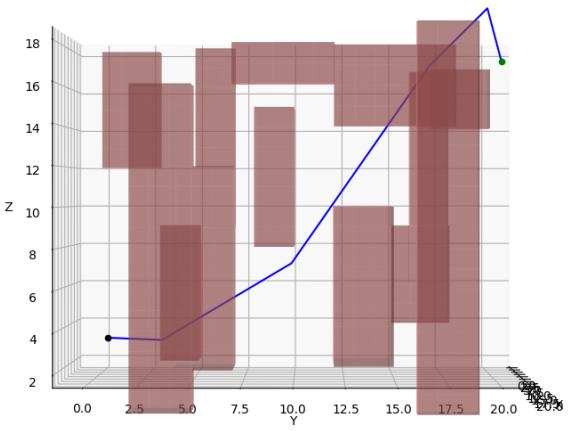


Side View

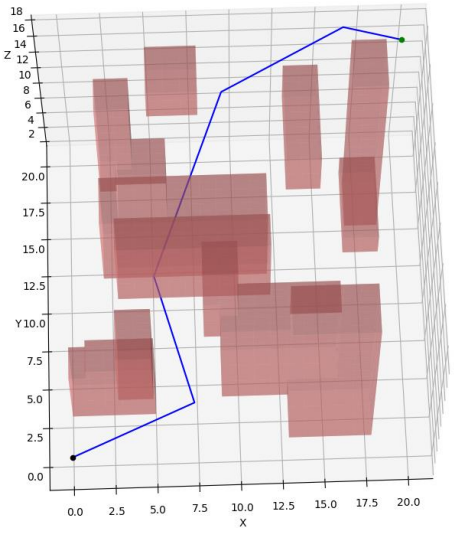
RRT* - Cluttered Environment:



Top View

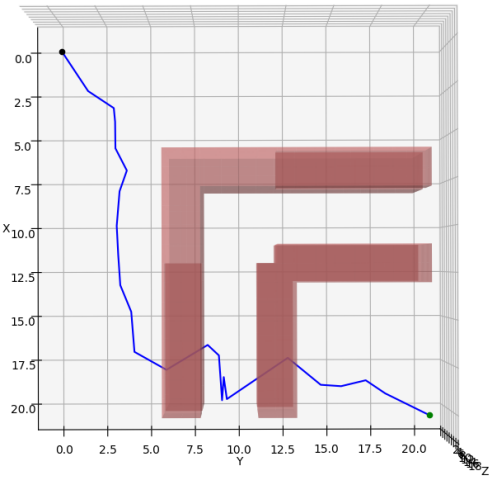


Side View

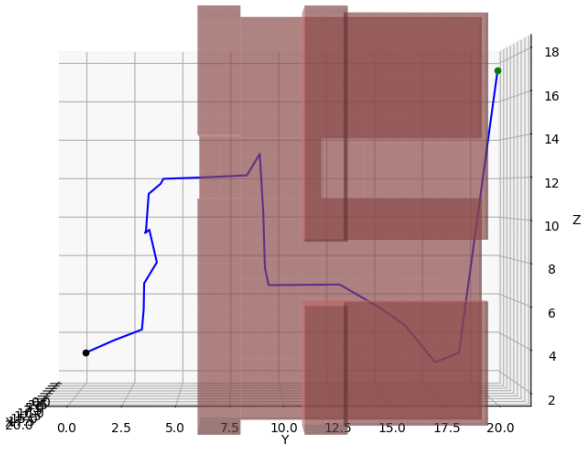


Angled View

PRM – Slit-based Environment:

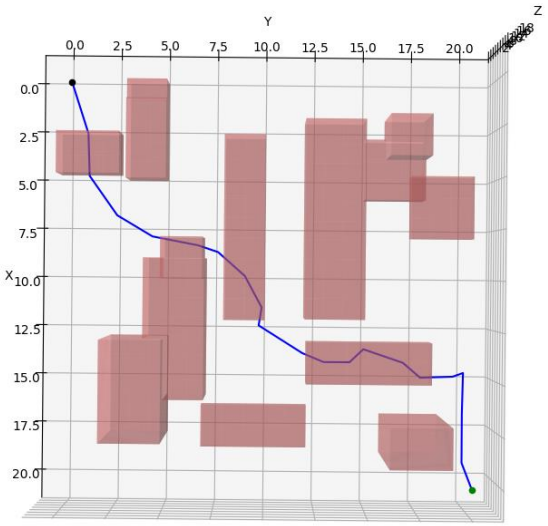


Top View

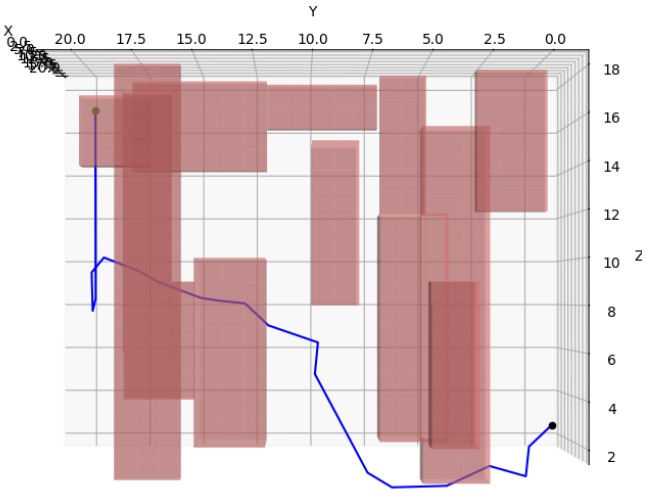


Side View

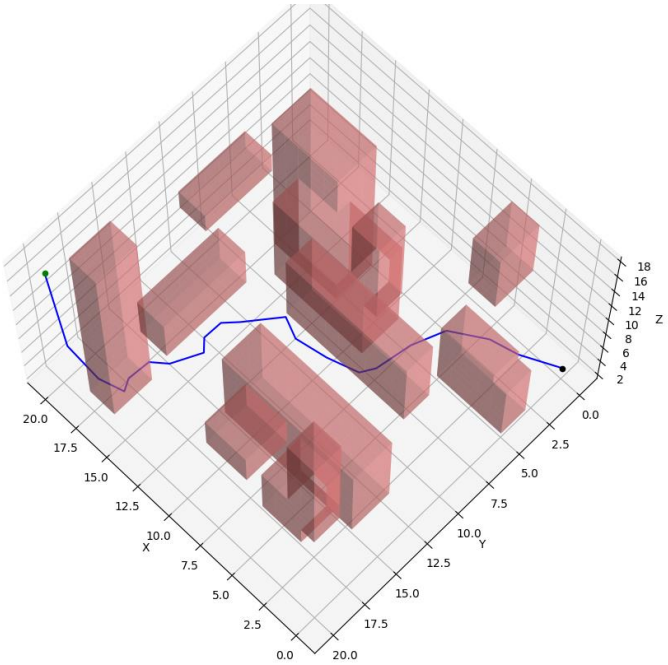
PRM - Cluttered Environment:



Top View



Side View



Angled View

8.2.RRT* and PRM 10-Runs

	2D: Cluttered Environment									
	RRT*									
Run #	1	2	3	4	5	6	7	8	9	10
Path Length	34.15	42.04	35.03	43.96	33.75	42.8	40.07	34.46	40.78	31.3
Computation Time	0.16	0.09	0.16	0.05	0.31	0.043	0.068	0.074	0.027	0.059
Number of turns	4	5	5	5	5	5	5	3	4	4
Max. Turning Angle	110	135	180	90	140	135	135	90	100	100
	PRM									
Run #	1	2	3	4	5	6	7	8	9	10
Path Length	31.11	29.27	28.92	29.01	29.99	29.61	31.6	29.9	30.22	31.26
Computation Time	0.211	0.21	0.24	0.23	0.22	0.23	0.22	0.24	0.25	0.233
Number of turns	10	7	5	7	9	6	6	6	7	7
Max. Turning Angle	80	60	90	70	100	90	90	90	80	90
	2D: Slit-based Environment									
	RRT*									
Run #	1	2	3	4	5	6	7	8	9	10
Path Length	34.75	30.39	35.2	35.26	37.01	34.34	28.13	33.59	38.15	32.91
Computation Time	0.072	0.054	0.026	0.027	0.039	0.036	0.02	0.12	0.046	0.042
Number of turns	2	3	2	3	2	3	2	2	2	2
Max. Turning Angle	90	150	90	110	150	70	45	80	100	90
	PRM									
Run #	1	2	3	4	5	6	7	8	9	10
Path Length	35.02	33.66	33.83	33.81	34.87	33.45	34.01	34	33.78	34.6
Computation Time	0.24	0.19	0.19	0.18	0.21	0.17	0.24	0.19	0.19	0.2
Number of turns	7	8	9	9	10	6	8	9	8	12
Max. Turning Angle	45	45	90	60	90	45	80	90	45	90

	3D: Cluttered Environment									
	RRT*									
Run #	1	2	3	4	5	6	7	8	9	10
Path Length	37.67	39.06	37.22	35.95	37.83	45.19	37.65	38.66	38.55	38.32
Computation Time	0.2	0.09	0.56	0.12	0.31	0.033	0.22	0.16	0.14	0.057
Number of turns	4	3	4	4	2	4	4	3	2	4
Max. Turning Angle	80	60	95	80	60	100	90	90	100	80

	PRM									
Run #	1	2	3	4	5	6	7	8	9	10
Path Length	47.76	54.67	46.29	50.02	44.04	57.65	56.04	43.94	46.33	44.34
Computation Time	0.69	0.72	0.66	0.79	0.76	0.73	0.8	0.71	0.76	0.8
Number of turns	10	15	12	13	11	10	10	9	11	10
Max. Turning Angle	135	160	120	150	90	160	165	120	110	120
	3D: Slit-based Environment									
	RRT*									
Run #	1	2	3	4	5	6	7	8	9	10
Path Length	54.7	50.78	65.4	52.81	41.38	45.54	48.35	47.45	49.96	57.93
Computation Time	0.03	0.15	0.097	0.31	0.23	0.29	0.056	0.041	0.16	0.16
Number of turns	2	4	4	3	3	5	2	3	3	5
Max. Turning Angle	120	120	100	120	90	110	120	100	120	135
	PRM									
Run #	1	2	3	4	5	6	7	8	9	10
Path Length	54.83	52.99	51.87	52.44	52.47	52.29	53.43	54.65	52.58	54.75
Computation Time	0.72	0.63	0.59	0.62	0.59	0.59	0.61	0.62	0.61	0.65
Number of turns	12	10	11	12	13	12	13	13	12	13
Max. Turning Angle	110	100	100	110	120	100	100	135	120	80