Search-based versus Sampling-based Robot Motion Planning:
A Comparative Study

Georgios Sotirchos[1], Zlatan Ajanović [2]

[1]Delft University of Technology
[2]RWTH Aachen University
g.sotirchos@student.tudelft.nl, zlatan.ajanovic@ml.rwth-aachen.de

Abstract

Robot motion planning is a challenging domain as it involves dealing with high-dimensional and continuous search space. In past decades, a wide variety of planning algorithms have been developed to tackle this problem, sometimes in isolation without comparing to each other. In this study, we benchmark two such prominent types of algorithms: OMPL's sampling-based RRT-Connect and SMPL's search-based ARA* with motion primitives. To compare these two fundamentally different approaches fairly, we adapt them to ensure the same planning conditions and benchmark them on the same set of planning scenarios. Our findings suggest that sampling-based planners like RRT-Connect show more consistent performance across the board in high-dimensional spaces, whereas search-based planners like ARA* have the capacity to perform significantly better when used with a suitable action-space sampling scheme. Through this study, we hope to showcase the effort required to properly benchmark motion planners from different paradigms thereby contributing to a more nuanced understanding of their capabilities and limitations. The code is available at https://github.com/gsotirchos/benchmarking_planners

Motion planning for robotic manipulators involves determining a feasible path from a starting configuration to a goal configuration while avoiding collisions with obstacles and obeying kinematic and dynamic constraints. This problem is particularly challenging due to the high-dimensional and continuous configuration space of manipulators, complex geometrical environments, and the real-time requirements on computation time. The complexity of the problem is further amplified when considering the need for discretization of the configuration space, and the cases with dynamic environments where obstacles and goals may change over time, requiring real-time re-planning capabilities.

Modern motion planning algorithms have been often categorized into two main families: search-based and sampling-based methods (LaValle 2006). This categorization might be misleading because both of them search and sample the continuous configuration space. Search-based methods, discretize the space and search for a feasible path in a graph, using algorithms such as Dijkstra's (Dijkstra 1959), A* (Hart, Nilsson, and Raphael 1968), and their variants. These methods are complete and optimal in the discretized space but suffer from the "curse of dimensionality" as the size of the discretized space grows exponentially with the number of dimensions. However, different variants of search-based planning were successfully used for planning footsteps for humanoid robots (Ranganeni et al. 2020), robot manipulation (Mandalika, Salzman, and Srinivasa 2018), underwater vehicles (Youakim et al. 2020), the aggressive flight of UAVs (Liu et al. 2018), as well as for automated driving in unstructured environment (Dolgov et al. 2008; Likhachev and Ferguson 2009; Adabala and Ajanović 2023), urban environments (Ajanović et al. 2018) and performance driving including drifting maneuvers (Ajanović et al. 2019; Ajanović et al. 2023).

Sampling-based methods, on the other hand, randomly sample possible configurations and build a graph that approximates the connectivity of the space. Popular algorithms in this family include the Probabilistic Roadmap Method (PRM) (Kavraki et al. 1996) and Rapidly-exploring Random Trees (RRT) (LaValle 1998). These methods are particularly suited for high-dimensional spaces as they avoid explicit discretization. While they are not complete, they are probabilistically complete, meaning that they can find a solution with probability approaching one as the number of samples increases. Moreover, variants like RRT* and PRM* (Karaman and Frazzoli 2010) can provide asymptotically optimal paths. Informed variants have been proposed, which improve RRT*'s convergence rate and final solution quality by focusing the search through directed sampling in a subset, such as an ellipsoidal defined based previous solutions (Gammell, Srinivasa, and Barfoot 2014). Particularly useful in robotics are real-time variants of RRT*, in which tree expansion and action-taking are interleaved (Naderi, Rajamäki, and Hämäläinen 2015).

Both families of algorithms have their strengths and applications. However, there is fairly limited work towards experimentally comparing the two families in a systematic manner. These algorithms are developed and evaluated using different software frameworks by different research groups and the experimental results of each method are often not directly comparable to those of the other.

The main contributions of our work are threefold:

- We present a generic categorization of existing planning methods based on their functional architecture and relation of search and sampling steps.

- We demonstrate the process and the challenges for objectively comparing the performance of two planning algorithms developed in fundamentally different frameworks.

- We provide a fair experimental comparison between the search-based Anytime Repairing A* (ARA*) and the sampling-based RRT-Connect.

We organize the paper as follows. In the first and second sections, we outline the main components of the robot motion planning problem. In the second second we describe the basic components present in every robot motion planning algorithm along with a generic representation. Based on this, we categorize existing planning algorithms into four main types and explain their distinguishing characteristics. In the third section, we present our benchmarking method with the implementation details and the structure of our experiments are recorded. Next, the results of the experiments are shown and discussed in the fifth section, and in the sixth section related work to ours is analyzed. Lastly, closing remarks and possible directions of future work are presented in the seventh section.

1 Robot Motion Planning Problem

A robot comprises links and joints that can occupy various configurations, or states, depending on the position or velocity of their joints. The collection of all possible states of a system is known as the state space, denoted as X, with individual states represented as x. Not all states within the state space are physically feasible, as some may violate spacial or otherwise specified constraints. These constraints divide the state space X into the constraint-free region X_{free} and its complement $X \setminus X_{\text{free}}$.

A motion planning problem is defined as a tuple $(X_{\text{free}}, x_{\text{I}}, X_{\text{G}})$, which represents the task of finding a path, a continuous function $p : [0, 1] \rightarrow X_{\text{free}}$, from a start state $x_{\text{I}} \in X_{\text{free}}$ to a goal region $X_{\text{G}} \subseteq X_{\text{free}}$ (LaValle 2006). The set of all feasible paths P is referred to as the path space $P(X_{\text{free}}, x_{\text{I}}, X_{\text{G}})$.

There are several variations of motion planning problems, including: *i)* Path planning: This variation addresses the geometric problem, ignoring the system's velocity, time, or dynamics, and is often called the piano mover's problem. *ii)* Kinodynamic planning: This involves planning for a dynamical system with potential constraints on velocity, acceleration, or torque. A kinodynamic planning problem is defined as a tuple $(X_{\text{free}}, x_{\text{I}}, X_{\text{G}}, f)$, where f represents the dynamical equations. *iii* Optimal planning: This focuses on finding a globally optimal path. An optimal path is one that minimizes a given cost functional $c : P \rightarrow R \geq 0$. An optimal motion planning problem is represented as a tuple $(X_{\text{free}}, x_{\text{I}}, X_{\text{G}}, c)$, where the objective is to find a feasible path p^* such that $c(p^*) = c^*$, with c^* being the minimum cost across the path space P.

Robot Models

A robot model is an abstraction, capturing both the physical form and the functional attributes of a robot, including its geometric configuration and kinematic chain. It is required for accurately simulating how the robot can move and interact with the surrounding environment, thereby enabling the planning of paths while respecting the robot's and environment constraints. Central to robot modeling is the notion of configuration space (C-space) as defined by Lozano-Pérez (1990), which represents every possible configuration q the robot might assume across a multidimensional space, each dimension of which corresponds to a specific degree of freedom (DOF) of the robot, enabling a holistic representation of its potential movements. In essence, the robot model is instrumental in the collision detection process, assisting in the clear distinction between navigable areas and those restricted either by obstacles or by the robot's joint limits. This delineation is essential for obtaining paths conforming to the planning scenario's requirements.

Environment models

Environment models in robot motion planning are representations of the world in which the robot operates, including the obstacles, terrain, and other elements that might affect the robot's movement. These models are essential for a robot to determine a path from its starting location to its destination without colliding with any obstacles. In practice, they are used by motion planners in queries for collision checking of new state configurations.

In motion planning, the world, denoted as $\mathcal{W}$, includes both the robot and the obstacles within it. The space can be two-dimensional for simpler, planar motion planning, or three-dimensional for more complex environments that require navigation in 3D space, such as in the case of manipulators. The obstacle region, represented as $\mathcal{O} \subset \mathcal{W}$, is the area within the world that the robot must avoid. This region typically has a boundary defined by simple geometric shapes (e.g., polygons in 2D or polyhedra in 3D).

Objects contained in the environment can be represented in several ways, including analytical geometric descriptions, polygon meshes, or annotated voxel spaces.

Collision checking

Collision checking in motion planning is the process that involves determining whether for a given path or motion any link of the robot intersections with with another link or obstacles within its environment. Collision checking typically involves evaluating a series of positions or configurations that the robot will occupy along its proposed path. Each of these configurations is then tested to see if the robot would intersect with any known obstacles. This can be computationally intensive, as it may require checking numerous points along the path, especially in complex or dynamically changing environments with a high density of obstacles, or in scenarios requiring high precision. Various methods for collision checking exist, ranging from geometric checks in simulation (Pan, Chitta, and Manocha 2012; Hudson et al. 1997), to distance routines like Gilbert, Johnson, and Keerthi (1988) for convex representations, to more complex ones that involve constructing and querying the C-space using learned methods, such as in (Benka et al. 2023; Danielczuk et al. 2021). The effectiveness and efficiency of collision checking can greatly impact the overall performance of the

motion planning system, making it a critical component in the design of autonomous robots.

2 Motion Planning Algorithms

A foundational concept in almost all types of planning algorithms are the tree data structures due to their utility in solving complex problems by breaking down the searched space into manageable segments for exploration and decision-making according to (LaValle 2006). Trees are hierarchical structures composed of nodes and edges, without cycles (as opposed to graphs in general), where each node corresponds to a state or a configuration in the space the planner is exploring, and the edges represent transitions between them. These trees are constructed and used during the planning process, with their root typically representing the starting state, and their branches leading to different possible future states based on the actions taken. In addition, they allow for backtracking, pruning of infeasible paths, and incorporating various optimization criteria, as explained by (Orthey, Chamzas, and Kavraki 2023).

The state tree can be expanded by obtaining new state configurations. The continuous space or domain can be probed by one of several **sampling** methods on the spectrum between orderly sampling in classical grid search and random sampling as in RRT and PRM, as explained by (LaValle, Branicky, and Lindemann 2004). This process is particularly relevant in motion planning for manipulators, where it is impractical to evaluate analytically every possible robot configuration. As discussed by (Kavraki and LaValle 2016), by intelligently sampling the C-space, algorithms can search for feasible paths to navigate, while significantly reducing computational load or balancing exploration and exploitation in unknown environments. To construct a viable route, the set of obtained states (whether by sampling or discretization) must eventually be subjected to some form of **search**. This search process involves systematically exploring the set of states, evaluating the feasibility of transitions, and then incrementally building a sequence that satisfies the motion planning criteria. It can be guided by various heuristics or strategies to improve efficiency, reduce computation time, and increase the likelihood of finding a valid path if one exists.

When solving a planning problem, there are several universal elements that are utilized by every type of planning algorithm. These are the following:

- *Query*: the initial and goal configurations of the robot, q_I and q_G[1] respectively, for which the algorithm is requested to plan a path connecting them.
- *Path*: the path p^* of successive configurations or actions generated by the planning algorithm.
- *Graph*: a graph G with robot configurations (states) as its nodes V, and the transitions connecting them as its edges E. It is used and updated by the algorithm to during the planning process.
- *Geometric models*: representations of the actual objects present in the environment as well as of the robot itself.

---

[1]The goal can also be represented by a set of states $q_G \in G$

- *Collision detection*: a method with which the planner can evaluate whether the robot's configuration in a node is colliding with the environment or with itself, based on their representations.
- *Sampling*: the process with which the algorithm can obtain new sets of nodes that can be subsequently evaluated, added to the graph, and included in possible paths. The sampling domain is specified by the algorithm and can be the state or the action space.
- *Heuristic function*: a function that can provide an estimation of the value of a node that represents some measure of proximity to the goal node.
- *Discrete search*: a way to traverse the graph and obtain a sequence of nodes or edges connecting the initial and goal nodes, while optimizing specific metrics (distance, duration, etc.).
- *Algorithm*: the logic performed by the planning algorithm itself, arbitrating each of the other elements to produce a path given a planning query.
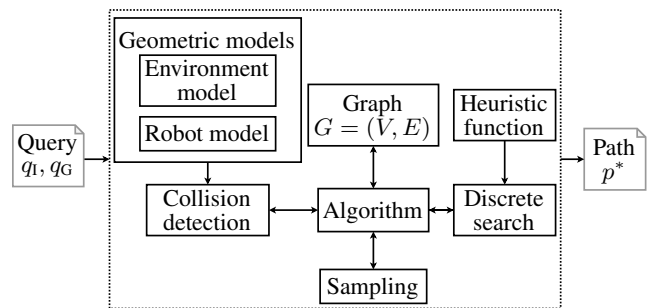

Figure 1: Generic elements of robot motion planning algorithms.

Depending on the sequence and the method in which the above processes are performed the different existing robot motion planning algorithms can be classified in five main categories.

Sample/Decompose-then-Search Motion Planning

The basic principle of these methods is the separation of the graph construction phase and the graph search. The two most prominent methods are Voronoi diagram planning and Probabilistic Roadmap Method (PRM).

In Voronoi based motion planning a Voronoi diagram is constructed based on the complete C-space. As formulated by (Leven and Sharir 1987), the configuration space is partitioned into cells by the Voronoi diagram, where each cell contains points that are closer to a particular obstacle (or set of obstacles) than to any other. The Voronoi diagram essentially divides the space into regions based on the nearest neighbor relation with respect to the obstacles. This generates a graph where the edges represent paths equidistant from the nearest obstacles, ensuring that the path maximizes the clearance from obstacles, which is often preferable for safety and maneuverability (Bhattacharya and Gavrilova 2008).

Subsequently, the shortest or most efficient path along these edges, connecting the start to the goal position, can be searched for using graph search algorithms such as Dijkstra's algorithm or the A* algorithm. This approach is particularly useful in environments where navigating close to obstacles poses a high risk or is undesirable. However, constructing a Voronoi diagram, especially in high-dimensional spaces or complex environments, can be computationally intensive. Furthermore, the paths generated by Voronoi decomposition may not always be the most direct paths between two points, as the primary objective is to maximize clearance from obstacles rather than minimize path length.

The Probabilistic Roadmap Method (PRM) is specifically designed to handle multiple-query situations efficiently by performing substantial preprocessing to build a roadmap. Formulated by (Kavraki et al. 1996), this roadmap is an undirected graph constructed during a preprocessing phase, where vertices represent random configurations in the free workspace, and edges represent collision-free paths between these configurations. Once built, this graph can be used to solve various subsequent path planning queries within the same environment by employing a search algorithm after connecting start and goal configurations to the roadmap.

The planning components utilized by Sample/Decompose-then-Search algorithms are shown in Figure 2. During Voronoi decomposition, the workspace is divided in a structured way, while collision detection is implicit and the geometric models are used directly for obtaining the nodes. It is worth noting that the decomposition process is analogous to sampling as the nodes of the graph are obtained in this phase, albeit in a more systematic manner. By contrast, during PRM's pre-processing phase the whole of C-space is sampled randomly and the nodes that are not in collision are then added to the graph. Subsequently, each query's start and goal nodes are added to the produced graph which is subjected to the search.
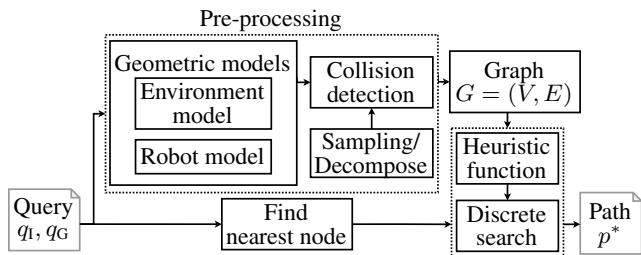

Figure 2: Planning elements of Sample/Decompose-then-Search algorithms.

Sampling-guided Motion Planning

The basic principle of these methods is that the sampling dominates the process and practically guides the search. The most prominent algorithm is Rapidly-exploring Random Tree (RRT). Rapidly-exploring Random Tree (RRT) solves the problem of navigating a space by incrementally building a tree that explores the available space, starting from an initial point and expanding towards unexplored areas. This approach, formulated by (LaValle 1998), is particularly well-suited for high-dimensional spaces or environments with complex obstacles. We classified RRT (and the algorithms based on it) as purely sampling-based because it constructs a path through the space by randomly sampling points and incrementally building a path or tree towards these points, without any explicit searching steps. This method contrasts with discrete search-based algorithms, which might systematically explore the space or with deterministic methods that might solve a mathematical formulation of the problem directly. By sampling, RRT can efficiently explore large spaces and find paths that avoid obstacles without exhaustive search.

One of the key features of RRT is its Voronoi bias, which refers to the tendency of the algorithm to preferentially expand into the largest unsearched areas of the space. New nodes are added by randomly sampling the obstacle-free space and then connected, to the closest node in the tree provided that the path between them is collision-free effectively guiding the tree expansion towards unexplored areas, as discussed by (Lindemann and LaValle 2004). The nature of this process means that regions of the space that are less densely covered by the tree (i.e., parts of the Voronoi diagram with larger cells) are more likely to be selected for expansion, ensuring more efficient coverage of the space. Thus, it encourages rapid exploration of unvisited areas, helping the algorithm to find a path to the goal more quickly than it might otherwise and to navigate around obstacles or explore potential paths that other algorithms might miss due to a more systematic or less exploratory approach.

The planning components used in the RRT algorithm are shown in Figure 3. The tree expansion process entails sampling the C-space to get a new node, performing collision checking to ensure it is a valid state, and then finding the nearest node in the tree (graph) to expand. RRT can be considered as a special case of Monte Carlo Tree Search (MCTS) (Browne et al. 2012), since in the way MCTS uses a *selection* step for choosing the nodes to explore based on some selection policy, RRT also is selecting nodes to *explore* based on its proximity to the newly sampled state. Both algorithms then expand the selected node by adding a new node based on certain rules. Thus, the *heuristic function* and *discrete search* generic components in this context were replaced by finding the nearest node based on the randomly sampled state. This algorithm updates the tree iteratively until a node close enough to the goal is added to the tree.

Search-and-Sample Motion Planning

Search-based motion planning algorithms focus on a discrete representation of the state space. They systematically search through a graph or a grid that represents possible robot states and paths. Modern search-based motion planning algorithms iteratively construct a graph by expanding nodes (or states) trough regular sampling of the continuous configuration space, e.g. with motion primitives (Frazzoli, Dahleh, and Feron 2002). The search is guided according to a specific strategy, such as depth-first, breadth-first, or best-first search strategies, until a path from the start state to the goal state is found. In essence, in every iteration a dis-
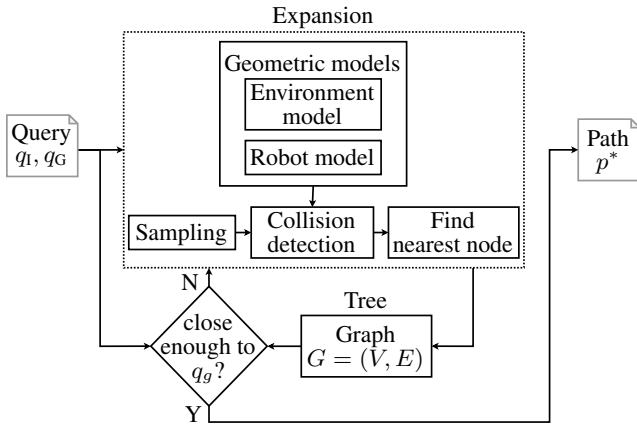
Figure 3: Generic elements of RRT algorithm.

crete space or graph representation of possible next states is obtained by a decomposition method, the most promising next node is then selected based on some heuristic, and the next iteration can then start from that node, as discussed by (LaValle 2006).

In this sense, the search and sampling (graph construction) steps are interleaved during the planning process, with the validity checks for each node performed during sampling. These methods are well-suited for environments where a discrete representation can effectively capture the necessary details for planning. (Cohen, Chitta, and Likhachev 2010) and (Cohen et al. 2011) have proved that these methods guarantee optimality with respect to the discretization of the actions and the state space, and they are resolution complete, meaning that if a solution exists, it will be found given that the resolution of the discretization is sufficiently refined.

The planning components used in search-and-sample methods are shown in Figure 4. In this method, the action space is sampled instead, and a lattice of connected states is obtained. After checking for collisions, the valid states are added to the graph which is subsequently subjected to a discrete search. This process is repeated iteratively until a state close enough to the goal is sampled and connected to the graph.
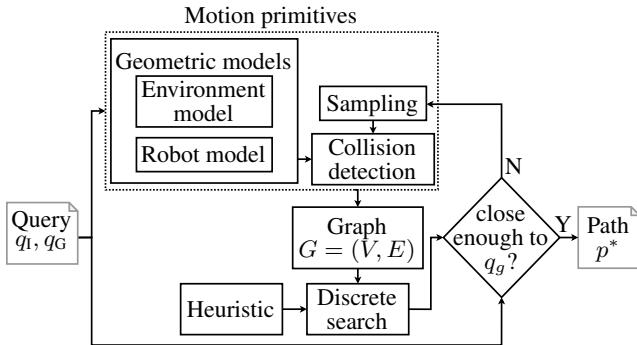

Figure 4: Generic elements of search-and-sample algorithms.

Trajectory Optimization-based Motion Planning

Apart from global planning, trajectory optimization is a class of methods that focus on local refining a given path or trajectory for a robot to minimize or optimize a certain objective, such as the path length, energy consumption, or the smoothness of the path. It starts with an initial guess of a path and iteratively refines this path to find a locally optimal solution that meets the constraints and objectives set. This approach is particularly useful in manipulator planning, where precise control over the motion is required, the dynamics of the system are complex, and the goal is to achieve high-quality motions in terms of efficiency, safety, or adherence to specific constraints as demonstrated by (Kalakrishnan et al. 2011) and (Ratliff et al. 2009).

While other planning methods, such as sampling-based planners, focus on exploring the space to find a feasible path from the start to the goal without necessarily optimizing the path, trajectory optimization directly aims at improving an existing path to optimize a given objective. While sampling-based methods are generally better suited for quickly finding a path in high-dimensional spaces or under complex constraints, they do not inherently optimize the path. In contrast, trajectory optimization takes an existing path and improves it according to the specific optimization criteria, making it more suitable for applications requiring high-quality motions (Mir et al. 2022). Compared to search-based methods, trajectory optimization does not operate on a discretized search space and it is not constrained by the granularity of the discretization. Instead, it is constrained by the computational intensity of solving the optimization problem iteratively, which can be more scalable to high-dimensional systems.

The planning components used in trajectory optimization are shown in Figure 5. Although this method is fundamentally different from the rest, a few parallels can be drawn between its functional architecture and the generic case. In particular, instead of a graph there is a trajectory, and in the place of sampling there is an initial generation of an suboptimal trajectory. Accordingly, instead of having search process, this method iteratively optimizes the initial trajectory until all constraints are satisfied within tolerance.
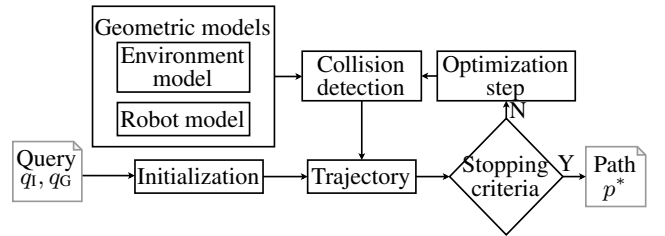

Figure 5: Generic elements of Trajectory Optimization.

3   Benchmarking Method

Properly comparing the two motion planning algorithms' performance imposes a few requirements. First, we defined a comprehensive set of benchmarks selected to represent a variety of problems the algorithms may encounter in practice.

These problems should include different scenarios and task complexities to ensure a broad evaluation spectrum. Second, we decided upon performance metrics that would allow for the quantitative assessment of each algorithm's efficiency, effectiveness, and applicability to different scenarios.

Scenarios

The planning problem instances, or scenarios, we used for benchmarking were identical for both planning frameworks. The following elements in these scenarios are the most critical for our comparison:

- **Robot model**: The specific robot model to be used for planning with a description format compatible with both OMPL and SMPL. In addition, the joints to be used for planning are specified, thus defining the configuration space.

- **Start state**: The start state defines the initial configuration of the robot and is the point to be connected through a viable path to the goal state by the motion planning algorithm.

- **Goal**: The goal state specifies the desired final configuration of the robot and can be a specific point in space or a region specified by allowed ranges for each state variable.

- **Environment**: The environment is a description of the workspace using primitive geometric shapes, encompassing all obstacles and boundaries that the robot must avoid while moving from the start to the goal state.

- **Allowed planning time**: The execution time allowed for the planner to generate a viable path once given the query.

Evaluation Metrics

For assessing the performance and effectiveness of the two algorithms, as evaluation metrics most useful for this comparison we selected the success rate and planning time. Each metric addresses different aspects of a planning algorithm's performance and together they offer a comprehensive view:

- **Success Rate**: This metric evaluates the reliability and robustness of a motion planning algorithm. It is defined as the proportion of planning attempts that successfully result in a feasible path from the start to the goal state without collisions. The success rate helps to gauge an algorithm's effectiveness in navigating complex environments and dealing with variations in it.

- **Planning Time**: This metric measures the efficiency of a motion planning algorithm in terms of computation time. It represents the duration from when the planning request is made until a viable path is found or the algorithm concludes that no such path exists and is crucial in real-time applications, where delays in decision-making could result in operational delays or safety risks. It is desired to minimize planning time while maintaining a high success rate.

4   Implementation and Experiments

We endeavored to align as best as possible each of the modules in the two frameworks aside from the planning algorithms. The intention being to enable an objective view of how effectively each planner utilizes the same planning modules. This entailed using the same experiment specifications to plan with the same robot model description for environments provided in the same format.

Therefore, after selecting the benchmarking problem set, we augmented each framework according to our needs.

Motion Planning frameworks

The two planning frameworks we compared were the Anytime Replanning A* algorithm (ARA*) with motion primitives from the Search-Based Motion Planning Library (SMPL), developed by (Likhachev et al. 2005), and RRT-Connect algorithm with uniform sampling as originally formulated by (Kuffner and LaValle 2000), implemented in the Open Motion Planning Library (OMPL) and set up using Robowflex by (Kingston and Kavraki 2022). We selected these two algorithms as they are both outstanding examples of the search-based and sampling-based families respectively.

ARA* (Anytime Repairing A*) is a variant of the A* search algorithm that is designed to provide solutions quickly and then improve these solutions over time, making it suitable for real-time applications. Unlike traditional A*, which aims to find the least-cost path before producing any solution, ARA* starts by finding a sub-optimal solution quickly and then iteratively improves this solution as time permits. It achieves this by initially using a relaxed version of the heuristic to speed up the search process and then gradually tightening this relaxation to improve the solution's optimality. It provides bounds on sub-optimality and improves the solution's quality by continuously decreasing the heuristic's inflation factor, thereby trading off between search speed and solution optimality. The searched graph of states is constructed using motion primitives, which represent available actions for each robot orientation, as sets of pre-defined and adaptive angles or offsets (Cohen, Chitta, and Likhachev 2010; Cohen et al. 2011).

RRT-Connect is a variant of the RRT algorithm enhancing it by simultaneously growing two trees: one from the start state and another from the goal state, and then attempting to connect these trees at each iteration of the algorithm. This bidirectional approach often finds a path more quickly than the unidirectional RRT, especially in high-dimensional spaces or environments with complex obstacles. RRT-Connect is efficient in finding feasible paths in challenging scenarios, although the paths may not be optimal. It's particularly favored for its simplicity and effectiveness in various planning domains.

The primary goal of this work was to align all the elements that differ between these two implementations in order to obtain a fair comparison. More specifically, this entailed using them in the same planning scenarios; planning for the same robot; performing collision checking in the same manner; and, for ARA*, searching both forward and

backward, in the same manner that RRT-Connect leverages bidirectional expansion.

Scenarios

The benchmarking planning scenarios used were the same used as test sets for the Spark, Flame, and RRT-Connect planners by (Chamzas et al. 2021), generated with Motion-BenchMaker (Chamzas et al. 2022). These are three groups of 100 request variations over three environments. All environments contain shelves with objects inside them which the robot is requested to either approach or take out. The objects are represented by basic cylinder and cube geometric shapes.

Each group has a specific kind of variation, among its 100 scenarios, on the position and orientation of the shelf and the objects it contains. The `shelf_zero_test` group has the shelf in a fixed position while varying the locations of the objects in it; the `shelf_height_test` group includes additional variation on the shelf height; and in the `shelf_height_rot_test` group the shelf's rotation relative to the robot's base reference is additionally varied.

The objects in the environments were primitive geometric descriptions of standard ROS planning scene messages, in the same format that was used to benchmark RRT-Connect with uniform sampling and biased sampling-based planners by (Chamzas et al. 2021). An example of an environment as used for search-based planning is shown in Figure 6b. The start and goal were also randomly varied among all scenarios in each group. The Fetch robot was used (Wise et al. 2016), shown in figure 6a, with the same 8 planning joints for all scenarios, resulting in an 8-dimensional planning space with 1 prismatic and 7 rotational degrees of freedom. The allowed planning time was 60" and was the same for every planning request.
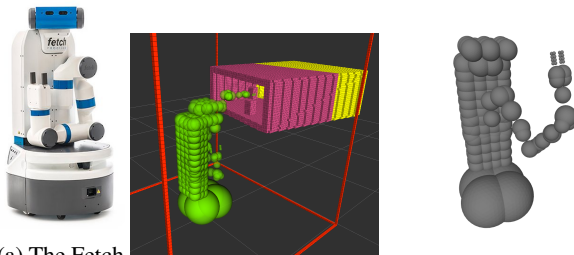
(a) The Fetch robot (Wise et al. 2016). (b) An example of a planning environment. (c) The Fetch robot's spheres-collision model.

Figure 6: The robot and the environment used in benchmarking.

SMPL

The first main extension of SMPL involved enabling the use of the experiment specification files generated by MotionBenchMaker that are used in OMPL. These are two types of YAML formatted files both including entries corresponding to standard ROS *MoveIt!* planning message types: files containing entries for `moveit_msgs/MotionPlanRequest` messages, and files containing entries for `moveit_msgs/PlanningScene` messages, as defined by (Coleman et al. 2014). Originally only SMPL-specific definitions for planning environments and requests were supported, thus not allowing for automating the benchmarking of ARA* with motion primitives on sets of planning problems from other libraries.

The second necessary modification was performing both a forward and a backward search with ARA* for each problem. The reasoning behind this was to offer search-based planning the same advantage of avoiding becoming trapped in convex obstacles in the workspace similar to how RRT-Connect bypasses this problem with bidirectional expansion. Backward search has been proven by (Garrett, Lozano-Pérez, and Kaelbling 2015) to benefit the search when planning for manipulators in high-dimensional state spaces. In a case where the robot is required to reach inside an opening, search-based planning can produce a path for the reverse problem considerably faster while that path is at the same time a valid solution to the forward problem. In our experiments, each problem was tried using both forward and backward search in sequence, and the first solution was kept.

The last major addition was the construction of a Fetch robot description for SMPL. While the basic structural description is obtained by the corresponding ROS description package, several other configuration files were required. The most important are those of the motion primitives definitions, the discretization for each joint state, the distance metric function that had to include an additional prismatic DoF compared to other manipulators, and the collision model of the robot. In this library the robots' collision representations are sets of spheres arranged to resemble the actual links' geometries. This collision model is shown in Figure 6c. This method introduces two main challenges for comparing this framework's planners to others: first, the collision checking with spheres is considerably faster compared to using triangular mesh geometries; and second, although the approximation is tailor-made to be as close as possible to the actual geometry, it is still prone to inaccuracies compared to the original. To alleviate this inconsistency, it is possible to further adjust it.

OMPL with Robowflex

The only modification required on the side of the Robowflex framework was performing collision checking in the same way as in SMPL. This was achieved by creating a custom Fetch robot URDF description file with sphere primitive geometries instead of meshes for each link. In practice, every link's mesh geometry was automatically replaced by sphere definitions with the same centers and radii as those specified in SMPL's model. In this way, *MoveIt!*'s collision checking using primitive geometric descriptions was leveraged accelerating the collision checking duration phase during RRT-Connect's execution.

5   Results and Discussion

The planning experiments were run on a compute node with an Intel Xeon E5-6248R 24C 3.0GHz and 8 GB of RAM

memory per CPU; each experiment executed as a single threaded process. The distribution of planning times for each planning scenario group is shown in Figure 7. The success rates are similarly shown in Figure 8. The vertical axis for the planning time is in logarithmic scale. The specific numbers of successes, failures, and unsolvable cases are presented in Tables 1 and 2.

It is worth noting that in total, ARA* failed in 15 out of the 300 problems. 9 of these were unsolvable due to collisions in the initial or goal state, while for the other 6 it failed to find a path. Out of the first 9 unsolvable by ARA*, RRT-Connect succeeded in 6; and out of the rest 6 where ARA* failed, RRT-Connect succeeded in 4. In total, RRT-Connect failed in 57 out of 300 problems.

Table 1: Benchmarking results for ARA*.

Experiment	Success forward	Success backward	failure	unsolvable
shelf_zero_test	47	53	0	0
shelf_height_test	47	52	0	1
shelf_height_rot_test	43	43	6	8

Table 2: Benchmarking results for RRT-Connect.

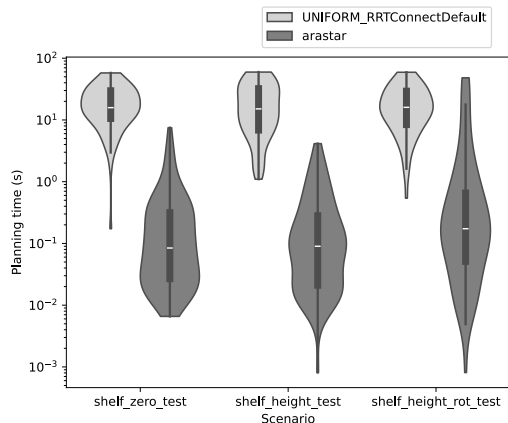| Experiment	Success forward	Success backward	failure	unsolvable
shelf_zero_test	85	-	15	0
shelf_height_test	82	-	18	0
shelf_height_rot_test	76	-	24	0

Figure 7: Average planning time for RRT-Connect and ARA* across the three different planning scenario groups.

Based on the performance of the two frameworks in our experiments, we were able to make several observations. Compared to ARA*, RRT-Connect exhibits more consistent performance in all three groups of problems, both in terms of planning time and of success rate. However, despite the more pronounced variation, ARA* was approximately 100 times faster in planning and had an overall slightly higher success rate.
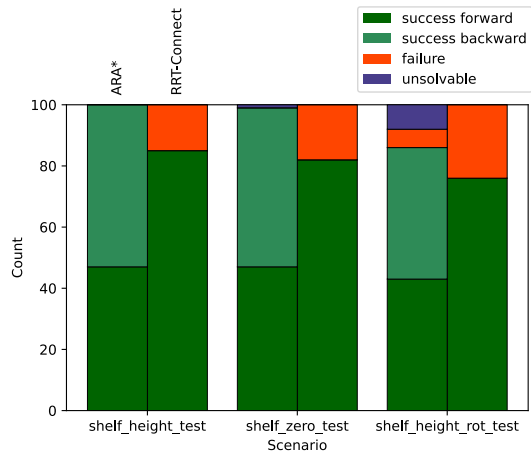

Figure 8: Success rate for RRT-Connect and ARA* across the three different planning scenario groups.

It is important to note that the motion primitives used for the search-based planning of SMPL in this case were appropriately chosen for manipulators and were not arbitrary. To achieve similar performance when planning for a different system in a different environment it would be necessary to configure the motion primitives accordingly. Therefore, the superior performance of search-based planning is tied to this type of planning problem and is not guaranteed when planning for other robots.

On the other hand, the sampling-based planning required no prior adjustments for these particular problems. In principle, it would perform equally well when planning for any other system with the same number of degrees of freedom. Across the board, sampling-based planning is more a more consistent option, while search-based methods can be adjusted to perform better in individual cases.

One caveat of this effort is that RRT-Connect's performance was negatively impacted by the sphere-collision model. Specifically mean and variance of the planning time were increased, and the success rate slightly reduced. At the same time, owing to the voxelization method in SMPL, some of the planning problems were unsolvable by ARA* due to collisions in the initial and goal configurations. Although we strove to prepare as accurate a robot description and collision model as possible, issues like these are bound to occur when trying to bridge the gap between two inherently different implementations.

6   Related work

The relationship between search-based and sampling-based motion planning methods, especially in high-dimensional spaces has been explored by (LaValle, Branicky, and Lindemann 2004). This work pointed the advantages of sampling methods in such contexts and outlined importance of discrepancy and dispersion in understanding motion planning algorithms' efficacy. However, there has been no significant follow-up work to this in the research community. (Cohen, Sucan, and Chitta 2012) introduced a more generic infras-

tructure for benchmarking motion planners, comparing notable algorithms like ARA* and RRT. However, their benchmarks did not account for adjustments in collision models, which could significantly influence the algorithms' performance metrics.

Until recently, there was a notable lack of attempts to create a comprehensive benchmarking software that extended beyond OMPL by (Sucan, Moll, and Kavraki 2012), as pointed out by (Elbanhawi and Simic 2014). The introduction of benchmarking capabilities in OMPL highlighted the need to establish benchmarking standards that facilitate a thorough comparison of sampling-based motion planners. Their work represents a significant effort towards creating a common ground for benchmarking.

Besides global planning for robot manipulation, (Wahab, Nefti-Meziani, and Atyabi 2020) compared classical algorithms, like Dijkstra's, to meta-heuristic algorithms for mobile robot path planning, albeit within the same framework. This approach to benchmarking, focused on planning modules implemented under one tailor-made framework. Similarly, (Spahn, Salmi, and Alonso-Mora 2022) developed an extensible benchmarking suite for local planners which nevertheless requires manual integration of motion planners.

(Chamzas et al. 2022) and (Orthey, Chamzas, and Kavraki 2023) both contributed to the expanding landscape of motion planning benchmarks, with the former providing an extensible tool for dataset generation and comparison of algorithms within OMPL, and the latter comparing several sampling-based motion planning methods using OMPL, among other tools. However, search-based motion planning is not treated. Lastly, for the extended problem of Task and Motion Planning, (Lagriffoul et al. 2018) highlight the challenges in benchmarking, pointing out the absence of a common set of metrics, formats, and problems accepted by the community. Their proposal for a set of benchmark problems and a planner-independent specification format for TAMP challenges seeks to unify the community around standard benchmarks.

7 Conclusion

In this study, we compared two fundamentally distinct approaches: the sampling-based RRT-Connect and the search-based ARA* utilizing motion primitives. In principle, these are two approaches that solve the motion planning problem by approaching the discretization of the continuous planning domain in distinct ways. Through an experimental setup that ensured equal planning conditions, we illustrated the strengths and limitations of each planning paradigm.

RRT-Connect showed a more uniform performance across various high-dimensional scenarios, underscoring its adaptability and broad applicability. Conversely, the search-based ARA* demonstrated a potential for superior performance, especially when paired with an appropriately tailored action-space sampling scheme, highlighting the significance of customization in optimizing algorithmic efficiency.

ARA* was found capable plan almost 100 times faster than RRT-Connect on average. Nonetheless, owing to the custom robot collision model there was a small set of problems exclusively unsolvable by ARA*. At the same time,

the incorporation of the spheres collision model to RRT-Connect's collision checking resulted in a comparable worsening of its performance.

The implications of this work extend beyond the comparative performance metrics, offering an understanding of the complexities inherent in unifying the benchmarking conditions of two independently developed planning frameworks. Moreover, we hope to provide a new perspective on the basic components of motion planning algorithms and the implications their functional structure has for their performance. We hope that this study will also contribute to bridging the gap between these communities.

However, there remain several direction for further exploration and refinement of our research. Future investigations could focus on aligning SMPL's collision checking with OMPL and Robowflex's mesh collision checking. Furthermore, code profiling would provide valuable insight as to which phases of the planning process in each method take significant computation time. Lastly, metrics for judging the solutions' optimality, such as trajectory length or duration, would be particularly informative for determining the preferable use cases of each method.

For future work, search-based algorithms provide a solid foundation for several advancements. The search process can potentially be accelerated by employing burs (Lacevic, Osmankovic, and Ademovic 2016) as more principled motion primitives. Additionally, prior search experience can be utilized to learn heuristics for guiding motion planning (Ajanovic, Lacevic, and Kober 2023), or to learn the distance from the optimal path to prune non-promising branches from the search (Bokan, Ajanovic, and Lacevic 2024).

Acknowledgments

References

Adabala, B.; and Ajanović, Z. 2023. A Multi-Heuristic Search-based Motion Planning for Automated Parking. In *2023 XXIX International Conference on Information, Communication and Automation Technologies (ICAT)*, 1–8.

Ajanovic, Z.; Lacevic, B.; and Kober, J. 2023. Value Function Learning via Prolonged Backward Heuristic Search. In *PRL Workshop Series {\textendash} Bridging the Gap Between AI Planning and Reinforcement Learning*.

Ajanović, Z.; Lacevic, B.; Shyrokau, B.; Stolz, M.; and Horn, M. 2018. Search-Based Optimal Motion Planning for Automated Driving. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 4523–4530. IEEE. ISBN 978-1-5386-8094-0.

Ajanović, Z.; Regolin, E.; Shyrokau, B.; Ćatić, H.; Horn, M.; and Ferrara, A. 2023. Search-Based Task and Motion Planning for Hybrid Systems: Agile Autonomous Ve-

hicles. *Engineering Applications of Artificial Intelligence*, 121: 105893.

Ajanović, Z.; Regolin, E.; Stettinger, G.; Horn, M.; and Ferrara, A. 2019. Search-Based Motion Planning for Performance Autonomous Driving. In *IAVSD Vehicles on Road and Tracks 2019*.

Benka, C.; Gross, C.; Gupta, R.; and Lipson, H. 2023. Direct Robot Configuration Space Construction using Convolutional Encoder-Decoders. ArXiv:2303.05653 [cs].

Bhattacharya, P.; and Gavrilova, M. L. 2008. Roadmap-Based Path Planning - Using the Voronoi Diagram for a Clearance-Based Shortest Path. *IEEE Robotics & Automation Magazine*, 15(2): 58–66. Conference Name: IEEE Robotics & Automation Magazine.

Bokan, D.; Ajanovic, Z.; and Lacevic, B. 2024. SLOPE: Search with Learned Optimal Pruning-based Expansion. In *PRL Workshop Series - Bridging the Gap Between AI Planning and Reinforcement Learning*.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1): 1–43.

Chamzas, C.; Kingston, Z.; Quintero-Peña, C.; Shrivastava, A.; and Kavraki, L. E. 2021. Learning Sampling Distributions Using Local 3D Workspace Decompositions for Motion Planning in High Dimensions. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 1283–1289.

Chamzas, C.; Quintero-Peña, C.; Kingston, Z.; Orthey, A.; Rakita, D.; Gleicher, M.; Toussaint, M.; and Kavraki, L. E. 2022. MotionBenchMaker: A Tool to Generate and Benchmark Motion Planning Datasets. *IEEE Robotics and Automation Letters*, 7(2): 882–889. ArXiv:2112.06402 [cs].

Cohen, B.; Sucan, I. A.; and Chitta, S. 2012. A generic infrastructure for benchmarking motion planners. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 589–595. Vilamoura-Algarve, Portugal: IEEE. ISBN 978-1-4673-1736-8 978-1-4673-1737-5 978-1-4673-1735-1.

Cohen, B. J.; Chitta, S.; and Likhachev, M. 2010. Search-based planning for manipulation with motion primitives. In *2010 IEEE International Conference on Robotics and Automation*, 2902–2908. ISSN: 1050-4729.

Cohen, B. J.; Subramania, G.; Chitta, S.; and Likhachev, M. 2011. Planning for Manipulation with Adaptive Motion Primitives. In *2011 IEEE International Conference on Robotics and Automation*, 5478–5485. Shanghai, China: IEEE. ISBN 978-1-61284-386-5.

Coleman, D.; Sucan, I.; Chitta, S.; and Correll, N. 2014. Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study. ArXiv:1404.3785 [cs].

Danielczuk, M.; Mousavian, A.; Eppner, C.; and Fox, D. 2021. Object rearrangement using learned implicit collision functions. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 6010–6017. IEEE.

Dijkstra, E. 1959. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1: 269–271.

Dolgov, D.; Thrun, S.; Montemerlo, M.; and Diebel, J. 2008. Practical search techniques in path planning for autonomous driving. *Ann Arbor*, 1001(48105): 18–80.

Elbanhawi, M.; and Simic, M. 2014. Sampling-Based Robot Motion Planning: A Review. *IEEE Access*, 2: 56–77.

Frazzoli, E.; Dahleh, M. A.; and Feron, E. 2002. Real-Time Motion Planning for Agile Autonomous Vehicles. *Journal of Guidance, Control, and Dynamics*, 25(1): 116–129. Publisher: American Institute of Aeronautics and Astronautics.

Gammell, J. D.; Srinivasa, S. S.; and Barfoot, T. D. 2014. Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ international conference on intelligent robots and systems*, 2997–3004. IEEE.

Garrett, C. R.; Lozano-Pérez, T.; and Kaelbling, L. P. 2015. Backward-forward search for manipulation planning. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 6366–6373.

Gilbert, E.; Johnson, D.; and Keerthi, S. 1988. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2): 193–203. Conference Name: IEEE Journal on Robotics and Automation.

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2): 100–107.

Hudson, T. C.; Lin, M. C.; Cohen, J.; Gottschalk, S.; and Manocha, D. 1997. V-COLLIDE: accelerated collision detection for VRML. In *Proceedings of the second symposium on Virtual reality modeling language*, VRML '97, 117–ff. New York, NY, USA: Association for Computing Machinery. ISBN 978-0-89791-886-2.

Kalakrishnan, M.; Chitta, S.; Theodorou, E.; Pastor, P.; and Schaal, S. 2011. STOMP: Stochastic trajectory optimization for motion planning. In *2011 IEEE International Conference on Robotics and Automation*, 4569–4574. ISSN: 1050-4729.

Karaman, S.; and Frazzoli, E. 2010. Incremental sampling-based algorithms for optimal motion planning. *Robotics Science and Systems VI*, 104(2).

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

Kavraki, L. E.; and LaValle, S. M. 2016. Motion Planning. In Siciliano, B.; and Khatib, O., eds., *Springer Handbook of Robotics*, 139–162. Cham: Springer International Publishing. ISBN 978-3-319-32552-1.

Kingston, Z.; and Kavraki, L. E. 2022. Robowflex: Robot Motion Planning with MoveIt Made Easy. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3108–3114. ISSN: 2153-0866.

Kuffner, J.; and LaValle, S. 2000. RRT-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 2, 995–1001 vol.2. ISSN: 1050-4729.

Lacevic, B.; Osmankovic, D.; and Ademovic, A. 2016. Burs of free c-space: a novel structure for path planning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 70–76. IEEE.

Lagriffoul, F.; Dantam, N. T.; Garrett, C.; Akbari, A.; Srivastava, S.; and Kavraki, L. E. 2018. Platform-Independent Benchmarks for Task and Motion Planning. *IEEE Robotics and Automation Letters*, 3(4): 3765–3772.

LaValle, S. 1998. Rapidly-exploring random trees: A new tool for path planning. *Research Report 9811*. Publisher: Department of Computer Science, Iowa State University.

LaValle, S. M. 2006. *Planning Algorithms*. Cambridge University Press, 1 edition. ISBN 978-0-521-86205-9 978-0-511-54687-7.

LaValle, S. M.; Branicky, M. S.; and Lindemann, S. R. 2004. On the Relationship between Classical Grid Search and Probabilistic Roadmaps. *The International Journal of Robotics Research*, 23(7-8): 673–692.

Leven, D.; and Sharir, M. 1987. Planning a purely translational motion for a convex object in two-dimensional space using generalized Voronoi diagrams. *Discrete & Computational Geometry*, 2(1): 9–31.

Likhachev, M.; and Ferguson, D. 2009. Planning Long Dynamically Feasible Maneuvers for Autonomous Vehicles. *The International Journal of Robotics Research*, 28(8): 933–945.

Likhachev, M.; Ferguson, D. I.; Gordon, G. J.; Stentz, A.; and Thrun, S. 2005. Anytime dynamic A*: An anytime, replanning algorithm. In *ICAPS*, volume 5, 262–271.

Lindemann, S.; and LaValle, S. 2004. Incrementally reducing dispersion by increasing Voronoi bias in RRTs. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, 3251–3257 Vol.4. New Orleans, LA, USA: IEEE. ISBN 978-0-7803-8232-9.

Liu, S.; Mohta, K.; Atanasov, N.; and Kumar, V. 2018. Search-based motion planning for aggressive flight in se (3). *IEEE Robotics and Automation Letters*, 3(3): 2439–2446.

Lozano-Pérez, T. 1990. Spatial Planning: A Configuration Space Approach. In Cox, I. J.; and Wilfong, G. T., eds., *Autonomous Robot Vehicles*, 259–271. New York, NY: Springer New York. ISBN 978-1-4613-8999-6 978-1-4613-8997-2.

Mandalika, A.; Salzman, O.; and Srinivasa, S. 2018. Lazy receding horizon A* for efficient path planning in graphs with expensive-to-evaluate edges. In *Twenty-Eighth International Conference on Automated Planning and Scheduling*.

Mir, I.; Gul, F.; Mir, S.; Khan, M. A.; Saeed, N.; Abualigah, L.; Abuhaija, B.; and Gandomi, A. H. 2022. A Survey of Trajectory Planning Techniques for Autonomous Systems. *Electronics*, 11(18): 2801.

Naderi, K.; Rajamäki, J.; and Hämäläinen, P. 2015. RT-RRT*: a real-time path planning algorithm based on RRT*. In *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games*, MIG '15, 113–118. New York, NY, USA: Association for Computing Machinery. ISBN 978-1-4503-3991-9.

Orthey, A.; Chamzas, C.; and Kavraki, L. E. 2023. Sampling-Based Motion Planning: A Comparative Review. *Annual Review of Control, Robotics, and Autonomous Systems*. Publisher: Annual Reviews.

Pan, J.; Chitta, S.; and Manocha, D. 2012. FCL: A general purpose library for collision and proximity queries. In *2012 IEEE International Conference on Robotics and Automation*, 3859–3866. ISSN: 1050-4729.

Ranganeni, V.; Chintalapudi, S.; Salzman, O.; and Likhachev, M. 2020. Effective footstep planning using homotopy-class guidance. *Artificial Intelligence*, 103346.

Ratliff, N.; Zucker, M.; Bagnell, J. A.; and Srinivasa, S. 2009. CHOMP: Gradient optimization techniques for efficient motion planning. In *2009 IEEE International Conference on Robotics and Automation*, 489–494. ISSN: 1050-4729.

Spahn, M.; Salmi, C.; and Alonso-Mora, J. 2022. Local Planner Bench: Benchmarking for Local Motion Planning. ArXiv:2210.06033 [cs].

Sucan, I. A.; Moll, M.; and Kavraki, L. E. 2012. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4): 72–82.

Wahab, M. N. A.; Nefti-Meziani, S.; and Atyabi, A. 2020. A comparative review on mobile robot path planning: Classical or meta-heuristic methods? *Annual Reviews in Control*, 50: 233–252.

Wise, M.; Ferguson, M.; King, D.; Diehr, E.; and Dymesich, D. 2016. Fetch & Freight: Standard Platforms for Service Robot Applications. In *Workshop on autonomous mobile service robots*, 1–6. NYC.

Youakim, D.; Cieslak, P.; Dornbush, A.; Palomer, A.; Ridao, P.; and Likhachev, M. 2020. Multirepresentation, Multiheuristic A* search-based motion planning for a free-floating underwater vehicle-manipulator system in unknown environment. *Journal of field Robotics*.