Scilight

Journal of Artificial Intelligence for Automation
https://www.sciltp.com/journals/jaia

Review

Optimal Motion Planning for Autonomous Robotic Systems: Foundations, Algorithms, and Challenges

Charles L. Clark and Biyun Xie *

Electrical and Computer Engineering Department, University of Kentucky, Lexington, KY 40506, USA
* Correspondence: Biyun.Xie@uky.edu

How To Cite: Clark, C.L.; Xie,B. Graph Optimal Motion Planning for Autonomous Robotic Systems: Foundations, Algorithms, and Challenges. *Journal of Artificial Intelligence for Automation* **2026**, *1*(1), 5.

Received: 1 January 2026
Revised: 10 February 2026
Accepted: 27 February 2026
Published: 6 March 2026

Abstract: Autonomous robotic systems are increasingly deployed across manufacturing, logistics, and healthcare due to their ability to efficiently complete difficult and dangerous tasks. Motion planning, which generates a safe and feasible motion from an initial state to a goal state, is fundamental to these systems. Many applications also require optimizing performance criteria such as time or energy while satisfying kinematic and dynamic constraints, motivating the study of optimal motion planning. This paper provides a comprehensive review of optimal motion planning algorithms for autonomous robotic systems. The fundamental concepts of optimal motion planning are first introduced. Planners are then categorized into three classes: graph-based, tree-based, and trajectory optimization-based methods, and state-of-the-art algorithms within each category are reviewed. Key properties of each class are compared, and guidance on selecting an appropriate planner for a given application is provided. Finally, challenges and future directions in this research area are discussed.

Keywords: motion and path planning; optimization and optimal control; collision avoidance

1. Introduction

Autonomous robotic systems are now widely employed across a broad range of applications, from manufacturing [1] and healthcare [2] to agriculture [3], transportation [4], and household assistance [5] and they play an increasingly important role in enabling intelligent, efficient, and reliable operation in these domains. To successfully complete a given task, such robotic systems typically follow a pipeline that integrates perception, planning, and control: perception is responsible for sensing and understanding the environment [6], planning determines how the robot should move or act to achieve its objectives [7], and control executes the planned actions safely and accurately on the physical platform [8]. Within this pipeline, motion planning stands out as one of the central problems, guiding the robot from its initial state to the goal state along a safe, collision-free trajectory.

Modern applications demand not only collision-free motion, but also efficiency, smoothness, safety, and robustness. In real-world systems, planners often operate under stringent and competing requirements: industrial manipulators must minimize cycle time while producing smooth motions that reduce mechanical wear [9]; autonomous vehicles must balance efficiency with safety and robustness in dynamic environments [10]; aerial robots must account for energy consumption and dynamic feasibility [11]; and humanoid robots must coordinate high-dimensional motion under balance and contact constraints [12]. These considerations motivate the study of optimal motion planning, in which a planner seeks to optimize a specified objective subject to feasibility constraints rather than simply computing a feasible motion plan.

There are many existing review papers on motion planners which focus on either one specific type of robot, such as mobile robots [13] or humanoid robots [14], or one specific class of algorithm, such as sampling-based [15] or learning-based planners [16]. However, there are limited existing reviews on optimal motion planning, which focus on problem formulation instead of the existing methods [17] or focus on one class of algorithms such as sampling-based [18] or optimization-based [19] approaches. As such, this paper presents a structured and comprehensive review of optimal motion planning for autonomous robotic systems. The foundations of optimal

Copyright: © 2026 by the authors. This is an open access article under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).
Publisher's Note: Scilight stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

motion planning are first introduced, including problem definition, commonly used costs and constraints, and classification of various optimal planners. The current state of the art for each category of optimal planners is then presented, including graph-based optimal planners, tree-based optimal planners, and trajectory optimization-based planners. These three types of optimal planners are compared with an emphasis on their optimality guarantees, completeness properties, constraint-handling capabilities, and computational characteristics. In addition, the selection of appropriate optimal planners based on environment features and task requirements is also discussed. Finally, the challenges and future opportunities in optimal motion planning are briefly addressed.

2. Foundations of Optimal Motion Planning

2.1. Problem Definition

Motion planning is traditionally formulated as a collision-avoidance problem in a robot's configuration space, $\mathcal{C}$, where each configuration uniquely specifies the robot's pose or joint state. Obstacles in the environment induce a forbidden subset of configurations, denoted $\mathcal{C}_{obs} \subset \mathcal{C}$, corresponding to states that result in collision. The collision-free subset of the configuration space is then given by $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$. Given an initial configuration $q_{start} \in \mathcal{C}_{free}$ and a goal region $\mathcal{C}_{goal} \subset \mathcal{C}_{free}$, the motion planning problem consists of finding a continuous path $\pi : [0, 1] \to \mathcal{C}_{free}$ such that $\pi(0) = q_{start}$ and $\pi(1) \in \mathcal{C}_{goal}$. More general formulations extend beyond just the configuration space to include velocities, accelerations, and higher-order derivatives. Despite these generalizations, the fundamental challenge remains unchanged: the planner must search through a high-dimensional space in order to safely and reliably guide the robot from its initial state to a desired goal without violating feasibility constraints.

Optimal motion planning extends the feasibility-based formulation by introducing an explicit objective function and solving for a trajectory that has minimal cost. Rather than solely computing a collision-free path, the goal is to determine an optimal path or trajectory π^* that optimizes a performance criterion of the form

$$J(\pi) = \int_0^T \ell(\pi(t), \dot{\pi}(t), \ddot{\pi}(t), \dots)\, dt,$$

where T is the total time to complete the trajectory and π is subject to feasibility constraints. This formulation captures a wide range of performance considerations relevant to real robotic systems

2.2. Costs and Constraints

The choice of objective function is highly application-dependent and typically includes trade-offs between different performance criteria. Common objectives include minimizing trajectory time [20]; reducing energy or control effort through penalties on torque, power, or fuel consumption [21]; enforcing smoothness by penalizing high velocities, accelerations, or jerk [22]; and maximizing path coverage [23]. Additional objectives may promote safety by encouraging greater clearance from obstacles [24], or robustness by favoring trajectories that remain feasible under disturbances, modeling uncertainty, or actuator failures [25].

In addition to optimizing a cost function, optimal motion planners must satisfy a diverse set of constraints. Collision avoidance, including self-collision, is fundamental [26], [27]. Kinematic constraints such as joint limits, closed-chain constraints, and nonholonomic constraints restrict the allowable motions of the system, while dynamic constraints impose bounds on velocities, accelerations, torques, and actuation [28]. For tasks that require deliberate interaction with the environment, planners must also account for contact and interaction constraints, which are particularly prominent in legged locomotion [29] and manipulation tasks [30].

2.3. Optimal Planner Classes

Different motion planning approaches vary significantly in how naturally and efficiently they can represent different objective functions and incorporate complex constraints. As a result, the structure of the optimization problem and the guarantees that can be provided regarding optimality, feasibility, and computational tractability depends strongly on the underlying planning approach.

A wide range of optimal motion planning algorithms have been developed over the past several decades. These approaches differ significantly in their mathematical foundations, computational properties, scalability, and the strength of the guarantees they provide. Broadly, they can be categorized as either (1) graph-based planners, (2) tree-based planners, or (3) trajectory optimization methods , as shown in Figure 1. While these categories are not mutually exclusive and elements from each class can be combined together, they provide a useful delineation between the fundamental design philosophies and trade-offs of different motion planners.

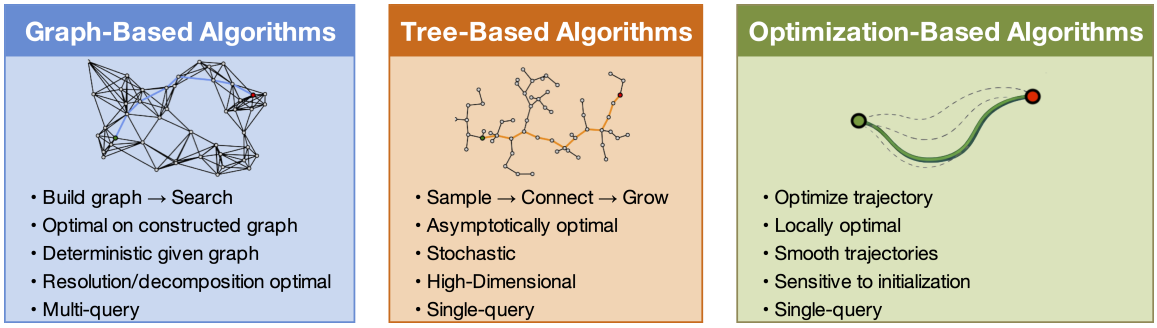

Figure 1. Optimal motion planners can be divided into three main categories: graph-based planners, tree-based planners, and optimization-based planners.

3. Graph-Based Optimal Planners

Graph-based optimal motion planners transform the continuous motion planning problem into a discrete shortest-path problem by explicitly constructing a graph that represents feasible motions, and then searching this graph using classical optimal graph search algorithms. Vertices typically correspond to robot configurations or states, while edges represent feasible motions between states, such as straight-line interpolations, motion primitives, or dynamically feasible trajectories. Once constructed, the planning problem reduces to finding a minimum-cost path in the graph, commonly solved using Dijkstra's algorithm, A*, or their variants. Differences among graph-based planners arise primarily from how the underlying graph is constructed and their optimality guarantees.

3.1. Graphs Constructed by Discretization

Discretization-based planners define a fixed graph structure by discretizing the robot's configuration or state space. Common examples include uniform grids and state lattices for systems with differential constraints. In state lattices, edges correspond to precomputed motion primitives that respect the robot's kinematics or dynamics, making them well suited for nonholonomic systems such as car-like robots or mobile manipulators [31]. Once the discretized graph is constructed and invalid nodes and edges are removed, optimal path planning is performed via graph search. Dijkstra's algorithm guarantees optimality with respect to the given cost function but does not take advantage of heuristics [32]. A* improves upon Dijkstra's by incorporating an admissible heuristic, while preserving optimality guarantees [33, 34]. For dynamic or partially known environments, incremental replanning algorithms such as D* [35] and D* Lite [36] efficiently update paths as the graph changes, maintaining optimality with respect to the updated graph structure. Anytime variants such as AD* [37] further trade off solution quality and computation time, producing increasingly better solutions over time and converging to the optimal discrete solution.

The primary limitation of discretization-based approaches is resolution dependence: optimality is guaranteed only with respect to the chosen grid or lattice resolution. As the dimensionality of the system increases, the number of states grows exponentially, limiting scalability for high-DOF manipulators and complex configuration spaces.

3.2. Graphs Constructed by Sampling

Sampling-based graph planners construct a roadmap graph by randomly sampling configurations in the configuration space and connecting nearby samples via local planners. Unlike discretization-based methods, the graph is not fixed in advance but grows as more samples are added, allowing better scalability to high-dimensional spaces.

The Probabilistic Roadmap (PRM) algorithm is a foundational multi-query planner that constructs a graph representation of the collision-free configuration space through random sampling [38]. In its construction phase, PRM samples collision-free configurations and connects nearby pairs using a local planner, typically straight-line interpolation. Once built, the roadmap can be reused for multiple queries by connecting start and goal configurations to the graph and searching for a path. PRM* is the asymptotically optimal variant of the classical PRM algorithm [39]. It constructs a roadmap by sampling collision-free configurations and connecting pairs of samples that lie within a specified connection radius. By selecting an appropriate connection radius that scales with the number of samples, PRM* guarantees that the cost of the best path in the roadmap converges almost surely to the global optimum as the number of samples increases.

Related variants such as k-PRM* use a fixed number of nearest-neighbor connections, k, while retaining asymptotic optimality provided k is sufficiently large [39]. LazyPRM* reduces collision-checking cost by deferring edge validation until necessary, improving practical performance without sacrificing theoretical guarantees [40].

Sparse roadmap methods such as SPARS and SPARS2 [41] reduce roadmap complexity by retaining only essential samples and connections, while guaranteeing that the resulting paths are close to optimal despite the significantly smaller graph.

Sampling-based graph planners offer strong asymptotic guarantees and good empirical performance in high-dimensional planning problems, though they typically require many samples to approach optimality and provide no finite-time global optimality guarantee.

3.3. Graphs Constructed by Convex Decomposition

A recent innovation in graph-based motion planning formulates planning as a shortest-path problem in a graph of convex sets (GCS), where each vertex corresponds to a convex region of the configuration space and edges represent feasible transitions [42]. By representing the collision-free space as a collection of convex regions, trajectories can be certified as collision-free as long as each segment lies entirely within a single region. Unlike discrete or sampling-based roadmaps, GCS planners jointly optimize both the sequence of regions traversed and the continuous trajectory through them. In the original implementation, the GCS planner solves for optimal trajectories using a mixed-integer convex program. This optimization problem can be relaxed to obtain an extremely efficient convex optimization formulation, where this relaxation introduces a small and bounded optimality gap [43]. To broaden the range of objectives and constraints that can be used, several extensions of the GCS framework have been proposed. These extensions include methods for motion planning in non-Euclidean spaces [44], handling a limited classes of non-convex objectives and constraints [45], optimizing general non-convex cost functions [46], and efficiently computing optimal trajectories with multiple target locations [47].

Graphs of convex sets can also be applied to contact-rich planning problems, where contact modes can be decomposed into a set of convex regions [48]. One downside to this representation is the massive size of the resulting graph. To make planning on large GCS graphs tractable, GCS* uses a heuristic based search to prune high-cost paths while maintaining completeness and optimality [49]. Implicit search variants such as INSATxGCS and IxG* [50] interleave search and partial trajectory optimization to reduce the number of convex programs that must be solved, improving performance on large graphs of convex sets.

While GCS provides a very effective method for formulating general motion planning problems, the construction of the convex sets can be time consuming. For example, computing enough collision-free regions to cover 70% of the free space for a 7R robot can take more than 90 minutes[51]. However, this process is typically performed offline, enabling efficient online planning once the region graph is available. As such, the GCS framework is well-suited for environments with static or mostly-static obstacles.

4. Tree-Based Optimal Planners

Tree-based planning algorithms incrementally construct a search tree rooted at the start configuration by randomly sampling configurations to explore the collision-free space. Unlike graph-based methods that attempt to cover the entire configuration space, tree-based planners grow toward a specific goal, making them more scalable to high-dimensional systems. Many recent variants also incorporate heuristics to focus exploration on promising regions, improving convergence to high-quality solutions.

The Rapidly-Exploring Random Trees (RRT) algorithm grows a tree from the start state by repeatedly sampling random configurations and connecting them to the nearest existing tree node subject to feasibility [52]. Originally designed for efficient exploration in high-dimensional spaces, RRT and some of its variants, such as RRT-Connect [53], excel at finding feasible solutions quickly but do not optimize path cost. It naturally incorporates nonholonomic and kinodynamic constraints through steering functions or forward simulation [54,55]. The RRT* algorithm augments the basic RRT structure to provide asymptotic optimality by incorporating cost-aware parent selection and local rewiring [39]. When a new sample is added, it evaluates candidate neighbors and chooses a parent that minimizes the cumulative cost from the start. It then attempts to rewire nearby nodes through the new sample if this reduces their cost. As the number of samples increases, the cost of the best solution converges almost surely to the global optimum for certain cost functions such as path length.

The RRT# algorithm builds on the RRT* structure by integrating relaxation ideas from incremental search algorithms into the sampling process [56]. The key idea is to maintain cost information such that the search tree contains lowest-cost candidates for vertices that might lie on the optimal path. This has the practical effect of improving convergence behavior in cases where the optimal cost changes significantly as more samples are processed, and it reduces the latency to high-quality solutions without sacrificing asymptotic optimality. RRT$^{\mathrm{X}}$ extends these ideas to dynamic environments where obstacles may appear, disappear, or move unpredictably [57]. When obstacle changes are detected, RRT$^{\mathrm{X}}$ performs a cascading rewiring operation that propagates cost updates

through the affected portion of the tree, repairing the solution without discarding the existing structure. This enables real-time replanning while preserving asymptotic optimality, making RRT^X suitable for applications such as mobile robot navigation in environments with moving obstacles.

Fast Marching Trees (FMT*) takes a different approach by using batch sampling and cost propagation over an implicit random geometric graph [58]. Rather than progressively inserting samples and rewiring, FMT* first draws a batch of samples, then performs a Dijkstra-like dynamic programming recursion to connect nearest neighbors in cost-increasing order. This reduces collision checks and rewiring overhead relative to RRT* while maintaining probabilistic completeness and asymptotic optimality. Its batch nature is advantageous in environments where collision checking is expensive or where high sample density can be exploited efficiently. The BFMT* algorithm extends FMT* by growing trees simultaneously from the start and goal regions and attempting to connect them [59]. This bidirectional strategy improves convergence rates and solution quality in many practical settings while preserving the optimality guarantees of the underlying batch framework.

While RRT* and FMT* guarantee convergence to optimal solutions, they can still require many samples to refine path cost toward the optimum. Informed planners leverage admissible heuristics to prioritize sampling to reduce unnecessary exploration. Informed RRT* improves on RRT* by restricting sampling after a feasible solution is found to a subset of the configuration space that can actually improve the current best solution [60]. For example, when minimizing path length, this region is an ellipse defined by the start, goal, and current best cost bound. This focused sampling leads to faster convergence while retaining asymptotic optimality. Batch Informed Trees (BIT*) unifies sampling-based planning with heuristic graph search by treating a set of samples as an implicit random geometric graph and conducting an informed search over successively denser graphs [61]. Using admissible cost-to-go heuristics to prioritize edge evaluations, BIT* blends the efficiency of A*-like search with the scalability of sampling. It exhibits strong anytime performance, meaning it quickly finds a solution and then refines it as computation continues.

Advanced BIT* variants include Adaptively Informed Trees (AIT*) and Effort Informed Trees (EIT*) [62], which exploit asymmetric bidirectional search. In these approaches, forward and reverse searches continuously inform one another to produce increasingly accurate problem-specific heuristics. This adaptive heuristic refinement enables fast initial solutions even when simple a priori heuristics are weak and accelerates asymptotic convergence across diverse cost objectives such as path length and obstacle clearance.

While the above planners do not directly encode complex dynamic or global constraints, these planners scale to high-dimensional systems and provide a foundation for industrial motion planning tasks. In practice, tree-based optimal planners are often paired with trajectory optimization or time-parameterization stages to produce high quality motions in manufacturing and automation systems.

5. Trajectory Optimization-Based Planners

Trajectory optimization offers an alternative to purely search-based methods by treating path generation as a continuous optimization problem. Rather than exploring a discrete graph of configurations, trajectory optimization methods operate directly in the space of continuous trajectories, iteratively refining an initial path to minimize a cost functional that encodes objectives such as smoothness, efficiency, and collision avoidance.

Covariant Hamiltonian Optimization for Motion Planning (CHOMP) formulates motion planning as the minimization of a cost functional defined over the space of trajectories [63]. The objective typically comprises two components: a smoothness cost that penalizes acceleration or jerk along the trajectory, and an obstacle cost that penalizes proximity to environmental obstacles. CHOMP computes the obstacle cost by integrating a workspace cost field along the body of the robot, where the cost field assigns higher values to points closer to obstacle surfaces. A key contribution of CHOMP is its use of functional gradient descent with a covariant gradient update rule. Standard gradient descent in trajectory space can produce updates that are sensitive to the parameterization of the trajectory, potentially leading to inefficient or oscillatory optimization behavior. By defining the gradient with respect to a metric that accounts for the dynamical structure of the trajectory, CHOMP produces updates that are invariant to reparameterization and tend to yield smoother convergence. The method initializes with a straight-line trajectory in configuration space and iteratively deforms this path away from obstacles while maintaining smoothness. CHOMP demonstrates that gradient-based optimization can efficiently generate collision-free trajectories for high-dimensional manipulators, establishing trajectory optimization as a viable approach for practical motion planning.

While CHOMP is an effective motion planner, its reliance on gradient information introduces limitations when the cost function is non-differentiable or when the cost landscape contains discontinuities. Constraint functions arising from contact conditions, joint limits, or task-specific requirements often lack smooth gradients, limiting the applicability of purely gradient-based methods. Stochastic Trajectory Optimization for Motion Planning (STOMP)

addresses this limitation by employing a sampling-based optimization strategy that does not require explicit gradient computation [64]. At each iteration, STOMP generates a set of noisy trajectory samples by adding correlated noise to the current trajectory estimate. These samples are then evaluated according to the cost functional, and the trajectory is updated by computing a weighted average of the sampled perturbations, where lower-cost samples receive higher weights. This procedure approximates a gradient descent step without requiring the cost function to be differentiable. The noise generation process in STOMP is designed to produce smooth perturbations that respect the correlation structure of the trajectory, preventing the introduction of high-frequency oscillations. By operating without gradients, STOMP can readily incorporate arbitrary cost terms, including those derived from learned models or binary feasibility checks, making it particularly flexible for complex planning scenarios, such as those encountered in manufacturing applications where task constraints may be difficult to express in closed form.

TrajOpt introduces a different optimization strategy based on sequential convex optimization [65]. Rather than performing gradient descent on the original nonconvex problem, TrajOpt iteratively constructs local convex approximations of the cost and constraint functions, solves the resulting convex subproblem, and updates the trajectory within a trust region that limits the step size. TrajOpt also introduces the use of signed distance fields for efficient collision cost computation and, notably, proposes continuous-time collision checking to detect collisions that may occur between discrete waypoints along the trajectory. This continuous-time formulation addresses a fundamental limitation of methods that only evaluate collisions at sampled configurations, which can miss collisions occurring during transitions. The sequential convex programming framework employed by TrajOpt offers favorable convergence properties and handles inequality constraints more naturally than penalty-based approaches, making it well-suited for problems with complex kinematic or geometric constraints typical in industrial manipulation tasks.

The above methods were developed primarily for single-query planning problems and typically execute on CPU architectures, limiting their computational efficiency. The CuRobo algorithm is an attempt to leverage parallel GPU computation for trajectory optimization, enabling real-time motion generation suitable for complex operating environments [66]. CuRobo implements trajectory optimization using GPU-enabled functions that evaluate costs, constraints, and their derivatives across many trajectory waypoints and candidate solutions simultaneously. The algorithm uses a particle-based optimization scheme that maintains a population of trajectory candidates, combining elements of parallel stochastic optimization with gradient-based refinement. By batching computations across the trajectory population and exploiting the parallel processing capabilities of modern GPUs, CuRobo achieves computation times that are orders of magnitude faster than CPU-based implementations. The framework integrates geometric planning components to provide good initializations for the trajectory optimizer, improving robustness in cluttered environments. The emphasis on computational efficiency and real-time capability makes CuRobo particularly relevant for manufacturing and automation applications where robots must respond quickly to changing workpiece locations, dynamic obstacles, or replanning requirements arising from process variations.

Trajectory optimization offers a fundamentally different approach to motion planning by operating directly in continuous trajectory space rather than searching discrete graphs or trees. These methods naturally encode smoothness and efficiency objectives, and the field has evolved to handle increasingly general cost functions, incorporate complex constraints, and exploit parallel computation for faster solve times.

6. Evaluation and Selection of Optimal Motion Planners

6.1. Comparisons between Different Types of Optimal Planners

Each class of motion planning algorithms involves trade-offs between theoretical guarantees, constraint-handling capabilities, solution quality, and computational requirements. This section compares the methods reviewed above along several key dimensions. Additionally, Table 1 provides a summary of the following discussion.

6.1.1. Optimality Guarantees

The strength of optimality guarantees varies considerably across planning approaches. Discretization-based graph planners are resolution optimal, meaning they return the lowest-cost path within the fixed graph structure, but this optimum may deviate from the true continuous optimum depending on the discretization resolution [32]. Sampling-based graph planners and tree-based planners such as PRM*, RRT*, and BIT* provide asymptotic optimality, i.e., the globally optimal path will be computed as the number of samples approaches infinity. However, these methods offer no finite-time optimality bound, and convergence can be slow in practice [39]. GCS planners achieve global optimality with respect to the convex decomposition of the free space when solved via mixed-integer programming; when the convex relaxation is used instead, the resulting solutions may have a small but bounded optimality gap [42]. Trajectory optimization methods are fundamentally local optimizers and converge to locally optimal solutions that are the lowest cost within the local region, but are not the best possible trajectory. Without

a good initial guess, these methods may settle in poor local minima, particularly in environments with many obstacles [46].

Table 1. Comparisons between different optimal motion planners.

	Graph-Based Planners			Tree-Based Planners	Trajectory Optimization
	Discretization-Based Graph	**Sampling-Based Graph**	**Graph of Convex Sets**		
Optimality	Resolution optimal	Asymptotically optimal	Globally optimal (MIP) or bounded gap (relaxation)	Asymptotically optimal	Locally optimal
Completeness	Resolution complete	Probabilistically complete	Complete w.r.t. decomposition	Probabilistically complete	No guarantee
Geometric Constraints	Binary collision check; rejection	Binary collision check; rejection	Implicit via convex regions	Binary collision check; rejection	Cost penalties or inequality constraints
Differential Constraints	Motion primitives	Steering functions or simulation	Limited to convex-compatible forms	Steering functions or simulation	Direct inclusion in cost/constraints
Trajectory Output	Piecewise-linear waypoints	Piecewise-linear waypoints	Bézier curves	Piecewise-linear waypoints	Waypoints with smoothness penalties
Query Type	Multi-query	Multi-query	Multi-query	Single-query	Single-query
Dimensionality	Low (2–3)	Moderate (3–7)	Moderate (3–7)	High ($\geq$7)	High ($\geq$7)

6.1.2. Completeness

Completeness guarantees follow a similar pattern. Discretization-based graph planners are resolution complete: if a path exists that is representable at the chosen resolution, the planner will find it. Sampling-based graph and tree-based planners are probabilistically complete, meaning the probability of finding an existing solution approaches one as the number of samples grows, but no guarantee is provided for any finite sample count [38,52]. GCS planners are complete with respect to their convex decomposition: if a feasible path exists through the decomposed regions, the planner will find it. However, the decomposition itself may fail to capture narrow passages or complex topologies [51]. Trajectory optimization methods provide no completeness guarantees, as they may converge to infeasible local minima or fail to escape poor initializations in cluttered environments [63–65].

6.1.3. Geometric and Kinematic Constraints

Geometric constraints such as obstacle avoidance, self-collision, and joint limits are handled differently across planner classes. Graph-based and tree-based planners address these constraints through explicit collision checking during graph construction or tree extension: invalid configurations and edges are rejected, producing paths that satisfy constraints by construction. This binary accept-reject approach is straightforward but can be computationally expensive in complex environments. GCS planners encode collision avoidance implicitly through the construction of the convex regions; trajectories confined to these regions are guaranteed collision-free without requiring explicit checking during optimization. Trajectory optimization methods typically incorporate geometric constraints as soft penalties or hard constraints within the optimization formulation. CHOMP and STOMP use cost fields that penalize proximity to obstacles, while TrajOpt formulates collision avoidance as inequality constraints using signed distance functions and performs continuous-time collision checking to detect violations between waypoints. Joint limits are naturally enforced as box constraints in trajectory optimization or as bounds during configuration sampling in graph and tree-based methods.

6.1.4. Differential and Dynamic Constraints

Constraints arising from system dynamics, actuator limits, and nonholonomic kinematics can be difficult to handle when generating optimal trajectories. Discretization-based graph planners can incorporate these constraints through motion primitives, as in state lattices, where edges correspond to precomputed dynamically feasible trajectories [31]. This approach is effective for systems with known dynamics, but it requires offline primitive

generation for each system. Sampling-based graph and tree-based planners can accommodate differential constraints through steering functions or forward simulation during edge extension, though designing effective steering functions for complex dynamics is non-trivial [54, 55]. GCS planners can encode certain dynamic constraints, but this capability is limited to constraints expressible in a convex form. Trajectory optimization methods handle differential constraints most naturally, as velocities, accelerations, torques, and other dynamic quantities can be directly included in the cost function or constraint set. Nonholonomic constraints, relevant for mobile robots in warehouse and logistics applications, are similarly incorporated through equality constraints on the trajectory derivatives.

6.1.5. Trajectory Representation and Smoothness

Trajectory representation varies across methods and affects how motions are executed on real-world systems. Graph-based and tree-based planners produce sequences of waypoints connected by straight-line segments in configuration space, yielding piecewise-linear paths that may require post-processing to achieve smooth, executable trajectories. GCS planners can utilize Bézier curve parameterizations, producing trajectories with continuous derivatives by construction [46]. Trajectory optimization methods output waypoint sequences but incorporate smoothness directly in the optimization objective. These methods penalize acceleration or jerk, producing waypoints that are discrete yet approximate smooth motion when interpolated [64].

6.1.6. Computational Profile

Computational characteristics affect the practical applicability of optimal motion planners. Discretization-based and sampling-based graph planners support multi-query planning: the graph is constructed once, potentially offline, and subsequent motion plans are computed using fast graph searches [32, 38]. This approach works well in environments with static geometry, such as fixed manufacturing workcells. GCS planners share this property, as the convex decomposition can be computed offline and reused for multiple queries [42]. Tree-based planners are inherently single-query because a new tree is constructed for each planning problem, though some variants can partially reuse previous computations [52]. Trajectory optimization is also single-query but can be warm-started from previous solutions for efficient replanning. Regarding dimensionality, discretization-based methods suffer from exponential state growth and sampling-based graph planners face similar scaling challenges as they attempt to cover the entire configuration space. Alternatively, tree-based planners and trajectory optimization scale to high dimensions more efficiently. Finally, while classical methods are largely sequential, CuRobo exploits GPU parallelism to achieve order-of-magnitude speedups, enabling real-time performance in dynamic environments [66].

6.2. Choosing an Optimal Motion Planner

The choice of motion planner depends on several factors: the dimensionality of the system, whether the environment is static or dynamic, the frequency of planning queries, and the importance of solution quality versus computation time. For low-dimensional systems in static environments where the same workspace is used repeatedly, discretization-based or sampling-based graph methods are well suited, as the graph can be constructed offline and reused for fast online queries. GCS planners offer similar multi-query benefits with the added advantage of producing smooth, globally optimal trajectories, though they require a convex decomposition of the free space that may be expensive to compute. When planning for high-dimensional manipulators or when each query involves a different environment, tree-based planners such as RRT* and its variants provide a practical balance of scalability and solution quality. These methods require no preprocessing and offer asymptotic optimality, though they produce piecewise-linear paths that may need post-processing for execution.

Trajectory optimization methods are most appropriate when smooth, dynamically feasible trajectories are required and when constraints beyond simple collision avoidance must be satisfied. These methods handle kinematic and dynamic constraints naturally and can optimize for objectives like minimum jerk or energy. However, they are local optimizers and require reasonable initialization to avoid poor local minima; in cluttered environments, they are often paired with a global planner to provide this initialization. For applications demanding real-time performance in dynamic settings, GPU-accelerated implementations such as CuRobo enable trajectory optimization at speeds compatible with online replanning. In practice, many systems combine multiple approaches: using a global planner to find an initial path and trajectory optimization to refine it, which leverages the strengths of each method.

7. Challenges and Future Perspectives

A fundamental challenge in motion planning is handling dynamic environments, where obstacles move unpredictably. Most of the methods reviewed in this paper assume that the environment is static, or that the environment does not change while completing the given task. Graph-based planners that rely on pre-computed

roadmaps or convex decompositions struggle when the environment changes because their graph structure must be updated or recomputed. Tree-based planners such as RRT and its variants can accommodate some degree of environmental change through incremental replanning. Algorithms like RRT^X have been specifically designed for dynamic settings, though when obstacles move continuously, there is no fixed optimum to converge to. Trajectory optimization methods can replan quickly, especially with GPU acceleration, but their local nature means they may fail to find feasible paths when obstacles invalidate the current trajectory. Developing planners that simultaneously provide strong optimality guarantees, real-time performance, and robust handling of dynamic obstacles remains an open problem with significant importance in applications such as warehouse automation and autonomous driving.

A second challenge lies in achieving global optimality for complex, application-specific cost functions. Many real-world tasks involve objectives beyond path length, such as energy consumption, manipulability, or obstacle clearance. Trajectory optimization methods handle diverse objectives naturally but provide only local optimality and are sensitive to initialization. Sampling-based methods offer asymptotic optimality guarantees, but these guarantees typically hold only for additive cost functions and convergence can be slow. GCS-based planners can achieve global optimality through mixed-integer programming, but require costs and constraints to be formulated in convex form. Extending global optimality guarantees to richer cost functions and improving convergence rates remain important directions for future research.

Finally, the field of motion planning would benefit from improved benchmarks and standardization. Comparing motion planning algorithms across publications is difficult due to variations in test environments, robot models, performance metrics, and implementation details. Differences in computing hardware, particularly CPU versus GPU execution, further complicate fair comparison. While benchmark suites such as MotionBenchMaker [67] exist, widespread adoption remains limited. Establishing standardized benchmarks that span diverse robotic systems, environment complexities, and planning objectives would enable more rigorous evaluation of motion planning contributions, facilitate reproducibility, and help practitioners select appropriate methods for their applications.

Funding

This work was supported by the NSF Foundation under Grant #2437812 and Grant #2441654.

Institutional Review Board Statement

Not applicable.

Informed Consent Statement

Not applicable.

Data Availability Statement

Not applicable.

Conflicts of Interest

The authors declare no conflict of interest.

Use of AI and AI-Assisted Technologies

No AI tools were utilized for this paper.

References

1. Keshvarparast, A.; Battini, D.; Battaia, O.; et al. Collaborative Robots in Manufacturing and Assembly Systems: Literature Review and Future Research Agenda. *J. Intell. Manuf.* **2024**, *35*, 2065–2118.
2. Silvera-Tawil, D. Robotics in Healthcare: A Survey. *SN Comput. Sci.* **2024**, *5*, 189.
3. Gil, G.; Casagrande, D.E.; Cortés, L.P.; et al. Why the Low Adoption of Robotics in the Farms? Challenges for the Establishment of Commercial Agricultural Robots. *Smart Agric. Technol.* **2023**, *3*, 100069.
4. Le Mero, L.; Yi, D.; Dianati, M.; et al. A Survey on Imitation Learning Techniques for End-to-End Autonomous Vehicles. *IEEE Trans. Intell. Transp. Syst.* **2022**, *23*, 14128–14147.
5. Szot, A.; Clegg, A.; Undersander, E.; et al. Habitat 2.0: Training Home Assistants to Rearrange Their Habitat. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 251–266.
6. Rahman, Q.M.; Corke, P.; Dayoub, F. Run-Time Monitoring of Machine Learning for Robotic Perception: A Survey of Emerging Trends. *IEEE Access* **2021**, *9*, 20067–20075.

7. Zhou, C.; Huang, B.; Fränti, P. A Review of Motion Planning Algorithms for Intelligent Robots. *J. Intell. Manuf.* **2022**, *33*, 387–424.
8. Abbas, A.K.; Al Mashhadany, Y.; Hameed, M.J.; et al. Review of Intelligent Control Systems with Robotics. *Indones. J. Electr. Eng. Inform.* **2022**, *10*, 734–753.
9. Touzani, H.; Séguy, N.; Hadj-Abdelkader, H.; et al. Efficient Industrial Solution for Robotic Task Sequencing Problem with Mutual Collision Avoidance & Cycle Time Optimization. *IEEE Robot. Autom. Lett.* **2022**, *7*, 2597–2604.
10. Zhao, S.; Zhang, J.; He, X.; et al. A Harmonized Approach: Beyond-the-Limit Control for Autonomous Vehicles Balancing Performance and Safety in Unpredictable Environments. *IEEE Trans. Intell. Transp. Syst.* **2024**, *25*, 15827–15840.
11. Ollero, A.; Tognon, M.; Suarez, A.; et al. Past, Present, and Future of Aerial Robotic Manipulators. *IEEE Trans. Robot.* **2021**, *38*, 626–645.
12. Khatib, O.; Jorda, M.; Park, J.; et al. Constraint-Consistent Task-Oriented Whole-Body Robot Formulation: Task, Posture, Constraints, Multiple Contacts, and Balance. *Int. J. Rob. Res.* **2022**, *41*, 1079–1098.
13. Dong, L.; He, Z.; Song, C.; et al. A Review of Mobile Robot Motion Planning Methods: From Classical Motion Planning Workflows to Reinforcement Learning-Based Architectures. *J. Syst. Eng. Electron.* **2023**, *34*, 439–459.
14. de Lima, C.R.; Khan, S.G.; Tufail, M.; et al. Humanoid Robot Motion Planning Approaches: A Survey. *J. Intell. Robot. Syst.* **2024**, *110*, 86.
15. Orthey, A.; Chamzas, C.; Kavraki, L.E. Sampling-Based Motion Planning: A Comparative Review. *Annu. Rev. Control Robot. Auton. Syst.* **2023**, *7*, 285–310.
16. Wang, J.; Zhang, T.; Ma, N.; et al. A Survey of Learning-Based Robot Motion Planning. *IET Cyber-Syst. Robot.* **2021**, *3*, 302–314.
17. Yang, Y.; Pan, J.; Wan, W. Survey of Optimal Motion Planning. *IET Cyber-Syst. Robot.* **2019**, *1*, 13–19.
18. Gammell, J.D.; Strub, M.P. Asymptotically Optimal Sampling-Based Motion Planning Methods. *Annu. Rev. Control Robot. Auton. Syst.* **2021**, *4*, 295–318.
19. Ioan, D.; Prodan, I.; Olaru, S.; et al. Mixed-Integer Programming in Motion Planning. *Annu. Rev. Control* **2021**, *51*, 65–87.
20. Penicka, R.; Scaramuzza, D. Minimum-Time Quadrotor Waypoint Flight in Cluttered Environments. *IEEE Robot. Autom. Lett.* **2022**, *7*, 5719–5726.
21. Zhu, B.; Bedeer, E.; Nguyen, H.H.; et al. UAV Trajectory Planning in Wireless Sensor Networks for Energy Consumption Minimization by Deep Reinforcement Learning. *IEEE Trans. Veh. Technol.* **2021**, *70*, 9540–9554.
22. Wu, G.; Zhang, S. Real-Time Jerk-Minimization Trajectory Planning of Robotic Arm Based on Polynomial Curve Optimization. *Proc. Inst. Mech. Eng. Part C J. Mech. Eng. Sci.* **2022**, *236*, 10852–10864.
23. Carvalho, J.P.; Aguiar, A.P. Deep Reinforcement Learning for Zero-Shot Coverage Path Planning with Mobile Robots. *IEEE/CAA J. Autom. Sin.* **2025**, *12*, 1594–1609.
24. Sakcak, B.; LaValle, S.M. Complete Path Planning That Simultaneously Optimizes Length and Clearance. In Proceedings of the 2021 IEEE International Conference on Robotics and Automation (ICRA); Xi'an, China, 30 May–5 June 2021; pp. 10100–10106.
25. Wasiela, S.; Cognetti, M.; Giordano, P.R.; et al. Robust Motion Planning with Accuracy Optimization Based on Learned Sensitivity Metrics. *IEEE Robot. Autom. Lett.* **2024**, *9*, 10113–10120.
26. Koptev, M.; Figueroa, N.; Billard, A. Real-Time Self-Collision Avoidance in Joint Space for Humanoid Robots. *IEEE Robot. Autom. Lett.* **2021**, *6*, 1240–1247.
27. Chen, B.; Zhang, H.; Zhang, F.; et al. CIMAP: A High-Performance Motion Planning Algorithm for Robotic Manipulators in Complex Environments Using Clearance Inference Network. *IEEE Trans. Syst. Man Cybern. Syst.* **2025**, *55*, 6383–6396.
28. Wen, Y.; Pagilla, P. Path-Constrained and Collision-Free Optimal Trajectory Planning for Robot Manipulators. *IEEE Trans. Autom. Sci. Eng.* **2022**, *20*, 763–774.
29. Jelavic, E.; Qu, K.; Farshidian, F.; et al. LSTP: Long Short-Term Motion Planning for Legged and Legged-Wheeled Systems. *IEEE Trans. Robot.* **2023**, *39*, 4190–4210.
30. Pang, T.; Suh, H.T.; Yang, L.; et al. Global Planning for Contact-Rich Manipulation via Local Smoothing of Quasi-Dynamic Contact Models. *IEEE Trans. Robot.* **2023**, *39*, 4691–4711.
31. Pivtoraiko, M.; Knepper, R.A.; Kelly, A. Differentially Constrained Mobile Robot Motion Planning in State Lattices. *J. Field Robot.* **2009**, *26*, 308–333.
32. Dijkstra, E.W. A Note on Two Problems in Connexion with Graphs. In *Edsger Wybe Dijkstra: His Life, Work, and Legacy*; Association for Computing Machinery: New York, NY, USA, 2022; pp. 287–290.
33. Hart, P.E.; Nilsson, N.J.; Raphael, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Syst. Sci. Cybern.* **1968**, *4*, 100–107.
34. Zhang, D.; Gai, Y.; Ju, R.; et al. A Cosine Similarity Based Multitarget Path Planning Algorithm for Cable-Driven Manipulators. *IEEE/ASME Trans. Mechatron.* **2024**, *30*, 2379–2388.
35. Stentz, A. *The D* Algorithm for Real-Time Planning of Optimal Traverses*; Technical Report; The Robotics Institute, Carnegie Mellon University: Pittsburgh, PA, USA, 1994.

36. Koenig, S.; Likhachev, M. D* Lite. In *Eighteenth National Conference on Artificial Intelligence*; American Association for Artificial Intelligence: Menlo Park, CA, USA, 2002; pp. 476–483.
37. Likhachev, M.; Ferguson, D.I.; Gordon, G.J.; et al. Anytime Dynamic A*: An Anytime, Replanning Algorithm. In Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling, Monterey, CA, USA, 5–10 June 2005; Volume 5, pp. 262–271.
38. Latombe, J.-C. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Trans. Robot. Autom.* **1996**, *12*, 566–580.
39. Karaman, S.; Frazzoli, E. Sampling-Based Algorithms for Optimal Motion Planning. *Int. J. Rob. Res.* **2011**, *30*, 846–894.
40. Bohlin, R.; Kavraki, L.E. Path Planning Using Lazy PRM. In Proceedings of the 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065), San Francisco, CA, USA, 24–28 April 2000; Volume 1, pp. 521–528.
41. Dobson, A.; Bekris, K.E. Sparse Roadmap Spanners for Asymptotically Near-Optimal Motion Planning. *Int. J. Rob. Res.* **2014**, *33*, 18–47.
42. Marcucci, T.; Petersen, M.; von Wrangel, D.; et al. Motion Planning Around Obstacles with Convex Optimization. *Sci. Robot.* **2023**, *8*, eadf7843.
43. Marcucci, T.; Umenberger, J.; Parrilo, P.; et al. Shortest Paths in Graphs of Convex Sets. *SIAM J. Optim.* **2024**, *34*, 507–532.
44. Cohn, T.; Petersen, M.; Simchowitz, M.; et al. Non-Euclidean Motion Planning with Graphs of Geodesically-Convex Sets. *arXiv* **2023**, arXiv:2305.06341.
45. von Wrangel, D.; Tedrake, R. Using Graphs of Convex Sets to Guide Nonconvex Trajectory Optimization. In Proceedings of the 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Abu Dhabi, United Arab Emirates, 14–18 October 2024; p. 8.
46. Clark, C.L.; Xie, B. Plan Optimal Collision-Free Trajectories with Nonconvex Cost Functions Using Graphs of Convex Sets. *IEEE Trans. Robot.* **2025**, *41*, 5604–5623.
47. Morozov, S.; Marcucci, T.; Amice, A.; et al. Multi-Query Shortest-Path Problem in Graphs of Convex Sets. *arXiv* **2024**, arXiv:2409.19543.
48. Graesdal, B.P.; Chia, S.Y.C.; Marcucci, T.; et al. Towards Tight Convex Relaxations for Contact-Rich Manipulation. *arXiv* **2024**, arXiv:2402.10312.
49. Chia, S.Y.C.; Jiang, R.H.; Graesdal, B.P.; et al. GCS*: Forward Heuristic Search on Implicit Graphs of Convex Sets. *arXiv* **2024**, arXiv:2407.08848.
50. Natarajan, R.; Liu, C.; Choset, H.; et al. Implicit Graph Search for Planning on Graphs of Convex Sets. *arXiv* **2024**, arXiv:2410.08909.
51. Werner, P.; Amice, A.; Marcucci, T.; et al. Approximating Robot Configuration Spaces with Few Convex Sets Using Clique Covers of Visibility Graphs. In Proceedings of the 2024 IEEE International Conference on Robotics and Automation (ICRA), Yokohama, Japan, 13–17 May 2024; pp. 10359–10365.
52. LaValle, S. *Rapidly-Exploring Random Trees: A New Tool for Path Planning*; Research Report 9811; Iowa State University: Ames, IA, USA, 1998.
53. Kuffner, J.J.; LaValle, S.M. RRT-Connect: An Efficient Approach to Single-Query Path Planning. In Proceedings of the 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065), San Francisco, CA, USA, 24–28 April 2000; Volume 2, pp. 995–1001.
54. Hu, B.; Cao, Z.; Zhou, M. An Efficient RRT-Based Framework for Planning Short and Smooth Wheeled Robot Motion Under Kinodynamic Constraints. *IEEE Trans. Ind. Electron.* **2020**, *68*, 3292–3302.
55. Chiang, H.-T.L.; Hsu, J.; Fiser, M.; et al. RL-RRT: Kinodynamic Motion Planning via Learning Reachability Estimators from RL Policies. *IEEE Robot. Autom. Lett.* **2019**, *4*, 4298–4305.
56. Arslan, O.; Tsiotras, P. Use of Relaxation Methods in Sampling-Based Algorithms for Optimal Motion Planning. In Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA), Karlsruhe, Germany, 6–10 May 2013; pp. 2421–2428.
57. Otte, M.; Frazzoli, E. RRT^{X}: Real-Time Motion Planning/Replanning for Environments with Unpredictable Obstacles. In *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*; Springer: Istanbul, Turkey, 2015; pp. 461–478.
58. Janson, L.; Schmerling, E.; Clark, A.; et al. Fast Marching Tree: A Fast Marching Sampling-Based Method for Optimal Motion Planning in Many Dimensions. *Int. J. Rob. Res.* **2015**, *34*, 883–921.
59. Starek, J.; Schmerling, E.; Janson, L.; et al. *Bidirectional Fast Marching Trees: An Optimal Sampling-Based Algorithm for Bidirectional Motion Planning*; Workshop on Algorithmic Foundations of Robotics; Stanford University: Stanford, CA, USA, 2024.
60. Gammell, J.D.; Srinivasa, S.S.; Barfoot, T.D. Informed RRT*: Optimal Sampling-Based Path Planning Focused via Direct Sampling of an Admissible Ellipsoidal Heuristic. In Proceedings of the 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Chicago, IL, USA, 14–18 September 2014; pp. 2997–3004.

61. Gammell, J.D.; Srinivasa, S.S.; Barfoot, T.D. Batch Informed Trees (BIT*): Sampling-Based Optimal Planning via the Heuristically Guided Search of Implicit Random Geometric Graphs. In Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 26–30 May 2015; pp. 3067–3074.
62. Strub, M.P.; Gammell, J.D. Adaptively Informed Trees (AIT*) and Effort Informed Trees (EIT*): Asymmetric Bidirectional Sampling-Based Path Planning. *Int. J. Rob. Res.* **2022**, *41*, 390–417.
63. Ratliff, N.; Zucker, M.; Bagnell, J.A.; et al. CHOMP: Gradient Optimization Techniques for Efficient Motion Planning. In Proceedings of the 2009 IEEE International Conference on Robotics and Automation (ICRA), Kobe, Japan, 12–17 May 2009; pp. 489–494.
64. Kalakrishnan, M.; Chitta, S.; Theodorou, E.; et al. STOMP: Stochastic Trajectory Optimization for Motion Planning. In Proceedings of the 2011 IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China, 9–13 May 2011; pp. 4569–4574.
65. Schulman, J.; Duan, Y.; Ho, J.; et al. Motion Planning with Sequential Convex Optimization and Convex Collision Checking. *Int. J. Rob. Res.* **2014**, *33*, 1251–1270.
66. Sundaralingam, B.; Hari, S.K.S.; Fishman, A.; et al. CuRobo: Parallelized Collision-Free Robot Motion Generation. In Proceedings of the 2023 IEEE International Conference on Robotics and Automation (ICRA), London, UK, 29 May–2 June 2023; pp. 8112–8119.
67. Chamzas, C.; Quintero-Peña, C.; Kingston, Z.; et al. MotionBenchMaker: A Tool to Generate and Benchmark Motion Planning Datasets. *IEEE Robot. Autom. Lett.* **2022**, *7*, 882–889.