sensors

MDPI

Article

An Overview and Comparison of Traditional Motion Planning Based on Rapidly Exploring Random Trees

Yang Chu [1], Quanlin Chen [2] and Xuefeng Yan [1,*]

[1] School of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China; chuyang_716@nuaa.edu.cn
[2] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China; quanlinchen@smail.nju.edu.cn
* Correspondence: yxf@nuaa.edu.cn

Abstract: Motion planning is a fundamental problem in robotics that involves determining feasible or optimal paths within finite time. While complete motion planning algorithms are guaranteed to converge to a path solution in finite time, they are proven to be computationally inefficient, making them unsuitable for most practical problems. Resolution-complete algorithms, on the other hand, ensure completeness only if the resolution parameter is sufficiently fine, but they suffer severely from the curse of dimensionality. In contrast, sampling-based algorithms, such as Rapidly Exploring Random Trees (RRT) and its variants, have gained the increasing attention of researchers due to their computational efficiency and effectiveness, particularly in high-dimensional problems. This review paper introduces RRT-based algorithms and provides an overview of their key methodological aspects.

Keywords: kinodynamic planning; motion planning; sampling-based algorithms

check for updates

Academic Editor: Yongmin Zhong

Received: 14 January 2025
Revised: 17 March 2025
Accepted: 17 March 2025
Published: 26 March 2025

Citation: Chu, Y.; Chen, Q.; Yan, X. An Overview and Comparison of Traditional Motion Planning Based on Rapidly Exploring Random Trees. *Sensors* **2025**, *25*, 2067. https://doi.org/10.3390/s25072067

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Motion planning is a well-known and fundamental problem in robotics that involves finding a (optimal) path for a robot from its initial position toward goal regions while avoiding collisions with any obstacles in its environment. While not the only fundamental problem of robotics, motion planning has garnered significant attention from researchers due to widespread applications across various fields, including robotics [1], computer animation [2], medical procedures [3], and manufacturing [4,5].

Many motion planning algorithms have been proposed, including complete, resolution complete, heuristic, potential field, and sampling-based algorithms. Each method offers distinct advantages and limitations in terms of space complexity, time complexity, and path optimization.

Complete algorithms primarily focus on geometric methods [6–9] that are guaranteed to find a path solution in finite time. However, both finding the shortest path and finding a feasible path under velocity constraints are proven to be NP-hard [10].

Resolution-complete algorithms include cell decomposition methods [11] and graph search methods [12–14]. They can guarantee completeness and optimality if fine resolution parameters are provided. However, their computation time and memory space grow exponentially with increasing dimensionality.

Heuristic algorithms consist of neural networks (NNs) and meta-heuristic optimization algorithms. On the one hand, NNs, with their advantages such as nonlinear mapping capabilities and learning ability, are employed in various aspects of navigation, including processing sensory data [15], obstacle avoidance, and path planning [16–19]. Additionally,

end-to-end learning approaches for driving have gained significant attention in recent years, such as Deepdriving [20] and ChauffeurNet [21]. However, their disadvantages include slow training speeds, the need for large datasets, and a lack of traceability. On the other hand, meta-heuristic optimization algorithms offer high success rates and the ability to manage complex constraints. Notable examples include Brain Storm Optimization [22], Genetic Algorithm [23], Particle Swarm Optimization [24], and Ant Colony Optimization (ACO) [25]. However, they require considerable time to reach optimal solutions, especially for large-scale problems.

The Artificial Potential Field (APF) [26] is a well-known method usually used to handle unknown and dynamic environments [27]. However, APF suffers from the problem of local minima. To address this issue, APF is combined with various heuristics to achieve path optimality and safety, such as APF with random sampling [28], APF with ACO [29], and APF with GA [30].

Sampling-based algorithms have emerged as particularly prominent in recent years for two key reasons: (i) They avoid the explicit construction of obstacle configurations (which can be challenging in high-dimensional spaces). (ii) They allow the incorporation of differential constraints to generate dynamically feasible paths. Among sampling-based methods, Rapidly Exploring Random Tree (RRT) [31] and its asymptotically optimal variant (RRT* [32]) are widely recognized as foundational algorithms.

Despite the success of RRT*-based algorithms, there are still some major limitations [33]: (i) Their slow convergence rate to optimal solutions; (ii) Their significantly large memory requirements due to numerous iterations utilized to calculate the optimal path. Various improvements have been proposed to address these challenges. Some approaches focus on enhanced sampling strategies to replace pure uniform sampling, such as Informed-RRT* [34], RRT*-Smart [35], and CE-RRT* [36]. Other methods employ two directional trees with greedy connect heuristics to enhance the convergence rates of RRT and RRT*, including RRT-Connect [37], B-RRT* [38], and IB-RRT* [33].

Recent research has increasingly focused on extending sampling-based algorithms to handle dynamic systems. While RRT and RRT* excel at high-level path planning, they lack a low-level control synthesis. Several approaches attempt to bridge the gap by designing controllers to steer the system between two generated vertices, such as Kinodynamic-RRT* [39] and LQR-RRT* [40]. For environments with dynamical obstacles, CBF-RRT [41] incorporates control barrier functions (CBFs) into RRT to achieve dynamical obstacle avoidance. Similarly, CBF-RRT* [42] combines CBFs for safety with control Lyapunov functions (CLFs) to generate safe motion trajectories between vertices of RRT*.

This paper is structured as follows: Section 2 introduces fundamental components of RRT-based motion planning. Various RRT-based algorithms are introduced in Section 3. Section 4 introduces RRT-based algorithms for dynamic systems. Section 5 presents experimental results and analysis, and the conclusion is in Section 6.

2. Background

In this section, we introduce key local motion planning components commonly integrated into RRT-based algorithms. We focus on three fundamental approaches: linear quadratic regulators (LQRs), which optimize control inputs to minimize cost functions, control barrier functions (CBFs), which ensure safety constraints and obstacle avoidance, and control Lyapunov functions (CLFs), which guarantee stability and convergence properties.

2.1. System Dynamics

Consider robot dynamics modeled as affine control dynamics,

$$\dot{\mathbf{x}} = f_{\text{cl}}(\mathbf{x}, \mathbf{u}) := f(\mathbf{x}) + g(\mathbf{x})\mathbf{u}, \tag{1}$$

with $f : \mathbb{R}^n \to \mathbb{R}^n$ and $g : \mathbb{R}^n \to \mathbb{R}^{n\times m}$ locally Lipschitz, the state $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^n$, and the control input $\mathbf{u} \in \mathcal{U} \subset \mathbb{R}^m$. The initial state is denoted as $\mathbf{x}_{\text{init}}$ and the goal region is denoted as $\mathcal{X}_{\text{goal}} \subset \mathcal{X}$. $\mathcal{O}_i \subset \mathcal{X}$ represents the i-th obstacle region, where $i = 1, \ldots, n$. The obstacle-free set is defined as $\mathcal{X}_{\text{safe}} := \mathcal{X} - \bigcup_{i=1}^{n} \mathcal{O}_i$.

2.2. Linear Quadratic Regulator

An LQR is a widely used optimal control strategy that minimizes a quadratic cost function. Consider a linear time-invariant system

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u},$$

which is a special case of the general affine system (1) where $f(\mathbf{x}) = \mathbf{A}\mathbf{x}$, $g(\mathbf{x}) \equiv \mathbf{B}$. The LQR problem is to find control inputs $\mathbf{u}(t)$ that minimize the following quadratic cost criterion:

$$J_{\text{LQR}} := \int_0^\infty \mathbf{x}(t)^\top \mathbf{Q}\mathbf{x}(t) + \mathbf{u}(t)^\top \mathbf{R}\mathbf{u}(t)dt,$$

where $\mathbf{Q}$ and $\mathbf{R}$ are positive-definite weighting matrices.

According to Theorem 10.1 in [43], if there exists a symmetric solution $\mathbf{P}$ to the following algebraic Riccati equation:

$$\mathbf{A}^\top \mathbf{P} + \mathbf{P}\mathbf{A} + \mathbf{Q} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^\top \mathbf{P} = \mathbf{0}, \tag{2}$$

for which $\mathbf{A} - \mathbf{B}\mathbf{R}^{-1}\mathbf{B}^\top \mathbf{P} = \mathbf{0}$ is a stability matrix, then the optimal feedback control law is given by

$$\mathbf{u} = -\mathbf{K}\mathbf{x}, \text{ where } \mathbf{K} := \mathbf{R}^{-1}\mathbf{B}^\top \mathbf{P},$$

with the optimal cost equal to $\mathbf{x}(0)^\top \mathbf{P}\mathbf{x}(0)$. This feedback law also stabilizes the closed-loop system.

In addition, LQR can be extended to handle nonlinear system dynamics through local linearization. More specifically, a local linearization can be made around an equilibrium point $(\mathbf{x}^{\text{eq}}, \mathbf{u}^{\text{eq}})$ using first-order Taylor expansion [43]:

$$\dot{\delta\mathbf{x}} = \hat{\mathbf{A}}\delta\mathbf{x} + \hat{\mathbf{B}}\delta\mathbf{u},$$

defined by the following Jacobian matrices:

$$\hat{\mathbf{A}} := \frac{\partial f_{\text{cl}}(\mathbf{x}^{\text{eq}}, \mathbf{u}^{\text{eq}})}{\partial \mathbf{x}}, \ \hat{\mathbf{B}} := \frac{\partial f_{\text{cl}}(\mathbf{x}^{\text{eq}}, \mathbf{u}^{\text{eq}})}{\partial \mathbf{u}},$$

where $\delta\mathbf{x} := \mathbf{x} - \mathbf{x}^{\text{eq}}$, $\delta\mathbf{u} := \mathbf{u} - \mathbf{u}^{\text{eq}}$, $\dot{\delta\mathbf{x}} := \dot{\mathbf{x}} - \dot{\mathbf{x}^{\text{eq}}}$. Therefore, this local linearization allows the application of LQR techniques to nonlinear systems in the neighborhood of the equilibrium point.

2.3. Control Barrier Functions

CBFs are employed to generate control inputs that steer the system (1) towards an exploratory point while ensuring obstacle avoidance. Let the *safe set* $\mathcal{C}$ be defined as

$$\mathcal{C} = \{\mathbf{x} \in \mathcal{D} : h(\mathbf{x}) \geq 0\},$$

for a continuously differentiable function $h : \mathcal{D} \subset \mathbb{R}^n \to \mathbb{R}$. The boundary and interior of $\mathcal{C}$ are given by

$$\partial\mathcal{C} = \{\mathbf{x} \in \mathcal{D} : h(\mathbf{x}) = 0\},\ \mathrm{Int}(\mathcal{C}) = \{\mathbf{x} \in \mathcal{D} : h(\mathbf{x}) > 0\}.$$

The set $\mathcal{C}$ is called *forward invariant* for the system (1) if $\mathbf{x}(0) \in \mathcal{C}$ implies $\mathbf{x}(t) \in \mathcal{C},\ \forall t > 0$. Intuitively, set invariance ensures that any trajectory starting within the invariant set will never leave it, thereby guaranteeing safety.

First, consider a model without control inputs, $\dot{\mathbf{x}} = f_{\mathrm{cl}}(\mathbf{x})$. According to Nagumo's Theorem [44], the necessary and sufficient conditions for set invariance are

$$\dot{h}(\mathbf{x}) \geq 0\ \forall \mathbf{x} \in \partial\mathcal{C}.$$

Alternatively, another necessary and sufficient condition [45] is the existence of an extended class $\mathcal{K}$ function α such that

$$\dot{h}(\mathbf{x}) \geq -\alpha(h(\mathbf{x})),\ \forall \mathbf{x} \in \mathcal{D}. \tag{3}$$

The function h, which defines the forward invariant set, is called a *zeroing barrier function*.

Next, consider the system dynamics (1). The derivative of h becomes

$$\dot{h}(\mathbf{x}) = \mathcal{L}_f h(\mathbf{x}) + \mathcal{L}_g h(\mathbf{x})\mathbf{u}.$$

Combining the above equation with Equation (3), for any point $\mathbf{x} \in \mathcal{D}$, the set of all controllers rendering the set $\mathcal{C}$ forward invariant is defined as

$$\mathcal{K}_{\mathrm{zcbf}}(\mathbf{x}) := \left\{\mathbf{u} \in \mathcal{U} : \mathcal{L}_f h(\mathbf{x}) + \mathcal{L}_g h(\mathbf{x})\mathbf{u} \geq -\alpha(h(\mathbf{x}))\right\}.$$

The function h is called a *zeroing control barrier function* (ZCBF) if $\mathcal{K}_{\mathrm{zcbf}}(\mathbf{x}) \neq \emptyset,\ \forall \mathbf{x} \in \mathcal{D}$.

Finally, given a ZCBF h, safety can be ensured by modifying an existing controller in a minimal way. More specifically, given a feedback controller $\mathbf{u} = k(\mathbf{x})$ for the control system (1), it may happen that $k(\mathbf{x}) \notin K_{\mathrm{zcbf}}(\mathbf{x})$ for some $\mathbf{x} \in \mathcal{D}$. To guarantee safety, one can determine the minimum perturbation to $\mathbf{u}$ by solving the following quadratic program (CBF-QP):

$$\begin{aligned} u(\mathbf{x}) = \underset{\mathbf{u} \in \mathbb{R}^m}{\arg\min}\ & \frac{1}{2}\|\mathbf{u} - k(\mathbf{x})\|^2 \\ \text{s.t.}\ & \mathcal{L}_f h(\mathbf{x}) + \mathcal{L}_g h(\mathbf{x})\mathbf{u} \geq -\alpha(h(\mathbf{x})). \end{aligned} \tag{4}$$

2.4. Control Lyapunov Functions

CLFs can be employed to generate control inputs that drive a system to a desired *equilibrium point* $(\bar{\mathbf{x}}, \bar{\mathbf{u}})$ that is, $f_{\mathrm{cl}}(\bar{\mathbf{x}}, \bar{\mathbf{u}}) = \mathbf{0}$. Without loss of generality, the equilibrium point is typically assumed at the origin $(\bar{\mathbf{x}} = \mathbf{0}, \bar{\mathbf{u}} = \mathbf{0})$. Additionally, it is typically assumed that the equilibrium point is stable and there exists an asymptotically stable neighborhood around it. More specifically, the equilibrium point is *stable* [46] if $\forall \epsilon > 0\ \exists \delta > 0$,

$$\|\mathbf{x}(0)\| < \delta \Rightarrow \|\mathbf{x}(t)\| < \epsilon,\ \forall t \geq 0.$$

Intuitively, the stability implies that a trajectory starting in a δ neighborhood of the origin will never leave the ϵ neighborhood. Moreover, a neighborhood $\mathcal{S}$ of the stable equilibrium point is called the region of *asymptotic stability* [46] if

$$\|\mathbf{x}(0)\| \in \mathcal{S} \Rightarrow \lim_{t \to \infty} \mathbf{x}(t) = \mathbf{0}.$$

Intuitively, the asymptotic stability implies that any point starting within $\mathcal{S}$ converges to the stable equilibrium point as t tends to infinity.

First, consider a model without control inputs, $\dot{\mathbf{x}} = f_{\text{cl}}(\mathbf{x})$. According to Theorem 2.26 in [44], a neighborhood $\mathcal{S}$ of origin is the region of asymptotic stability if there exists a locally Lipschitz-positive definite function Ψ such that

$$\mathcal{S} \subset \mathcal{N}[\Psi, v] := \{\mathbf{x} : \Psi(\mathbf{x}) \leq v\} \text{ for some } v > 0,$$

and for all $\mathbf{x} \in \mathcal{N}[\Psi, v]$,

$$D^+\Psi(\mathbf{x}) \leq -\phi(\|\mathbf{x}(t)\|) \text{ for some } \mathcal{K} \text{ function } \phi, \tag{5}$$

where D^+ is a Dini derivative. And this function Ψ is called the *Lyapunov function* inside $\mathcal{S}$.

Next, consider the system dynamics (1). The Dini derivative of Ψ is given by

$$D^+\Psi(\mathbf{x}) = \mathcal{L}_f\Psi(\mathbf{x}) + \mathcal{L}_g\Psi(\mathbf{x})\mathbf{u}.$$

Combining the above equation and Equation (5), the set of all stabilizing controllers for every point $\mathbf{x} \in \mathcal{S}$ is defined by

$$\mathcal{K}_{\text{clf}}(\mathbf{x}) := \left\{\mathbf{u} \in \mathcal{U} : \mathcal{L}_f\Psi(\mathbf{x}) + \mathcal{L}_g\Psi(\mathbf{x})\mathbf{u} \leq -\phi(\|\mathbf{x}(t)\|)\right\}.$$

The function Ψ is called the *control Lyapunov function* (CLF) if $\mathcal{K}_{\text{clf}}(\mathbf{x}) \neq \emptyset$, $\forall \mathbf{x} \in \mathcal{S}$.

Finally, given a CLF Ψ and ZCBF h, one can steer the state $\mathbf{x}$ to another state $\mathbf{x}'$ while guaranteeing safety by solving a sequence of QPs. More specifically, suppose $(\mathbf{x}', \bar{\mathbf{u}})$ is the equilibrium point of the system. Define new variables $\mathbf{z}(t) := \mathbf{x}(t) - \mathbf{x}'$ and $\mathbf{v}(t) := \mathbf{u}(t) - \bar{\mathbf{u}}$ through a translation to establish a new system whose equilibrium point is the origin. The initial state is set to $\mathbf{z}(0) = \mathbf{x} - \mathbf{x}'$. By restricting the control inputs $\mathbf{v}(t) \in \mathcal{K}_{\text{clf}}$ for all $t > 0$, one can guarantee $\mathbf{z}(t)$ converges to the origin. This constraint can then be incorporated into the CBF-QP, resulting in a new problem termed the CLF-CBF-QP:

$$\begin{aligned} v(\mathbf{z}) = \underset{\mathbf{v}\in\mathbb{R}^m}{\arg\min} \quad & \frac{1}{2}\|\mathbf{v} - k(\mathbf{z})\|^2 \\ \text{s. t.} \quad & \mathcal{L}_f h(\mathbf{z}) + \mathcal{L}_g h(\mathbf{z})\mathbf{v} \geq -\alpha(h(\mathbf{z})) \\ & \mathcal{L}_f\Psi(\mathbf{z}) + \mathcal{L}_g\Psi(\mathbf{z})\mathbf{v} \leq -\phi(\|\mathbf{z}(t)\|). \end{aligned} \tag{6}$$

3. The RRT-Based Motion Planning

In this section, several incremental RRT-based motion planning algorithms are introduced.

3.1. The RRT Algorithm and Its Variations

The Rapidly Exploring Random Tree (RRT) is introduced as an efficient data structure and sampling scheme to explore efficiently nonconvex high-dimensional spaces by growing the search tree toward goal areas. The major components of the RRT are as follows:

- Sample: The procedure provides independent identically distributed samples from $\mathcal{X}_{\text{free}}$.
- Steer: Given two states $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$, the Steer procedure returns a state $\mathbf{z} \in \mathcal{X}_{\text{free}}$ that lies between $\mathbf{x}$ and $\mathbf{x}'$ while maintaining $\|\mathbf{z} - \mathbf{x}\| \leq \eta$, for a prespecified constant $\eta > 0$. It is typically defined by

$$\text{Steer}(\mathbf{x}, \mathbf{x}') = \underset{\mathbf{z}\in\mathcal{X}_{\text{free}}, \|\mathbf{z}-\mathbf{x}\|\leq\eta}{\arg\min} \|\mathbf{z} - \mathbf{x}'\|.$$

- Nearest node: Given a tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ with a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E}$, the Nearest$(\mathcal{V}, \mathbf{x})$ procedure provides a node in $\mathcal{V}$ that is closest to $\mathbf{x}$, i.e.,

$$\text{Nearest}(\mathcal{V}, \mathbf{x}) = \arg\min_{\mathbf{x}' \in \mathcal{V}} \|\mathbf{x} - \mathbf{x}'\|.$$

- Collision test: Given two points $\mathbf{x}, \mathbf{x}' \in \mathcal{X}_{\text{free}}$, the procedure ObstacleFree$(\mathbf{x}, \mathbf{x}')$ returns True if and only if the line segment between $\mathbf{x}$ and $\mathbf{x}'$ lies entirely within $\mathcal{X}_{\text{free}}$, i.e.,

$$\text{ObstacleFree}(\mathbf{x}, \mathbf{x}') = \begin{cases} \text{True}, & \text{if } [\mathbf{x}, \mathbf{x}'] \subset \mathcal{X}_{\text{free}}, \\ \text{False}, & \text{otherwise.} \end{cases}$$

The body of RRT algorithms is presented in Algorithm 1. The algorithm proceeds as follows. First, a state is sampled from the configuration space, denoted as $\mathbf{x}_{\text{samp}}$. Next, the Extend procedure grows the tree $\mathcal{T}$ towards $\mathbf{x}_{\text{samp}}$, as outlined in Algorithm 2. More specifically, the Nearest procedure selects the closest neighbor node $\mathbf{x}_{\text{nearest}} \in \mathcal{T}$ to $\mathbf{x}_{\text{samp}}$. The Steer procedure steers the state from $\mathbf{x}_{\text{nearest}}$ to $\mathbf{x}_{\text{samp}}$, resulting in a new state $\mathbf{x}_{\text{new}}$. Finally, the state $\mathbf{x}_{\text{new}}$ and the edge $(\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{new}})$ are added to $\mathcal{T}$ if the ObstacleFree procedure returns True.

Additionally, RRT-Connect, a variation of RRT, employs a two-RRT planner to achieve rapid convergence to a solution. More specifically, it constructs two trees, $\mathcal{T}_1$ and $\mathcal{T}_2$, starting from $\mathbf{x}_{\text{init}}$ and $\mathbf{x}_{\text{goal}}$, respectively. In each iteration, both trees are extended towards a random state $\mathbf{x}_{\text{samp}}$ simultaneously. The algorithm maintains both trees at all times until they connect and a solution is found.

Last but not least, these algorithms ensure *probabilistic completeness*, i.e., as the number of iterations approaches infinity, the probability of finding a solution (if one exists) approaches one (see Theorem 3 in [47]).

Algorithm 1: Body of RRT and RRT* algorithms

Input: N
Output: $\mathcal{T}$
Initialization: $\mathcal{V} \leftarrow \{\mathbf{x}_{\text{init}}\}, \mathcal{E} \leftarrow \emptyset, \mathcal{T} = (\mathcal{V}, \mathcal{E})$;
for $i \leftarrow 1$ **to** N **do**
  $\mathbf{x}_{\text{samp}} \leftarrow \text{Sample}(\mathcal{X})$;
  $\mathcal{T} \leftarrow \text{Extend}(\mathcal{T}, \mathbf{x}_{\text{samp}})$;
return $\mathcal{T}$

Algorithm 2: $\text{Extend}_{\text{RRT}}$

Input: $\mathcal{T}$, $\mathbf{x}_{\text{samp}}$
Output: $\mathcal{T}$
$(\mathcal{V}, \mathcal{E}) \leftarrow \mathcal{T}$;
$\mathbf{x}_{\text{nearest}} \leftarrow \text{Nearest}(\mathcal{V}, \mathbf{x}_{\text{samp}})$;
$\mathbf{x}_{\text{new}} \leftarrow \text{Steer}(\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{samp}})$;
if ObstacleFree$(\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{new}})$ **then**
  $\mathcal{V} \leftarrow \mathcal{V} \cup \{\mathbf{x}_{\text{new}}\}$, $\mathcal{E} \leftarrow \mathcal{E} \cup \{(\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{new}})\}$;
return $\mathcal{T} \leftarrow (\mathcal{V}, \mathcal{E})$

3.2. The RRT Algorithm and Its Variations*

Although RRT ensures probabilistic completeness, it does not guarantee optimality, i.e., it converges to a suboptimal solution with probability one (see Theorem 10 in [32]). To address this limitation, RRT* is introduced as an asymptotically optimal variant that asymptotically optimizes the existing tree to achieve *asymptotic optimality* properties, i.e., almost-sure convergence to an optimal solution. Several key components are added in RRT*:

- Near nodes: Given a tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ with a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E}$, the Near$(\mathcal{V}, \mathbf{x})$ procedure utilizes a pre-defined distance $\|\cdot\|$ to identify a set of near nodes $\mathcal{X}_{\text{near}}$ in $\mathcal{T}$ that are close to $\mathbf{x}$. This is typically defined as the set of all nodes within a closed ball of radius r_n centered at $\mathbf{x}$, i.e.,

$$\mathcal{X}_{\text{near}} := \{\mathbf{x}' \in \mathcal{V} : \|\mathbf{x} - \mathbf{x}'\| \leq r_n\},$$

  where the radius is defined by

$$r_n := \gamma \left(\frac{\log n}{n}\right)^{1/d}, \tag{7}$$

  n is the number of vertices in the tree, d is the dimension of the space, and γ is a constant.
- Choose Parent: The procedure tries to find collision-free paths between a selected node $\mathbf{x}$ and all its near nodes, and selects the near node that yields the lowest cost of $\mathbf{x}$ as the parent of $\mathbf{x}$.
- Rewire: The procedure attempts to connect a selected node $\mathbf{x}$ with each node in its neighborhood $\mathcal{X}_{\text{near}}$. If the trajectory from $\mathbf{x}$ to the near node $\mathbf{x}_{\text{near}}$ results in a lower cost for $\mathbf{x}_{\text{near}}$, then $\mathbf{x}$ becomes the new parent of $\mathbf{x}_{\text{near}}$.

It is worth noting that the Rewire procedure guarantees the asymptotic optimality properties of RRT*.

The body of the RRT* algorithm is outlined in Algorithm 1, while the Extend procedure for the RRT* is detailed in Algorithm 3. The Extend procedure operates as follows. First, the Nearest procedure selects the closest neighbor node $\mathbf{x}_{\text{nearest}} \in \mathcal{T}$ of $\mathbf{x}_{\text{samp}}$. Second, the Steer procedure steers the state from $\mathbf{x}_{\text{nearest}}$ to $\mathbf{x}_{\text{samp}}$, resulting in a new state $\mathbf{x}_{\text{new}}$. Third, the ChooseParent procedure identifies a potential parent node for $\mathbf{x}_{\text{new}}$. If a suitable parent node $\mathbf{x}_{\text{parent}}$ is found, then $\mathbf{x}_{\text{new}}$ and the edge $(\mathbf{x}_{\text{parent}}, \mathbf{x}_{\text{new}})$ are added to the tree $\mathcal{T}$. Fourth, the Rewire procedure attempts to connect $\mathbf{x}_{\text{new}}$ with each node in its neighborhood $\mathcal{X}_{\text{near}}$. If the trajectory from $\mathbf{x}_{\text{new}}$ to the near node $\mathbf{x}_{\text{near}}$ results in a lower cost for $\mathbf{x}_{\text{near}}$, then $\mathbf{x}_{\text{new}}$ becomes the new parent of $\mathbf{x}_{\text{near}}$.

Although RRT* ensures asymptotic optimality, its convergence rate is relatively slow. More specifically, let $f(\mathbf{x})$ be the cost of an optimal path from $\mathbf{x}_{\text{init}}$ to $\mathbf{x}_{\text{goal}}$ constrained to pass through $\mathbf{x}$. The set of states that could improve the current solution is defined by

$$\mathcal{X}_f := \{\mathbf{x} \in \mathcal{X} : f(\mathbf{x}) < c_{\text{best}}\},$$

where c_{best} is the cost of the current solution. In RRT*, the probability of sampling states from $\mathcal{X}_f$ becomes arbitrarily small for large subsets (see Remark 1 in [34]), leading to a slow convergence rate. Rapid convergence can be expected only if the probability of sampling states from $\mathcal{X}_f$ increases.

Algorithm 3: Extend$_{\text{RRT}^*}$

Input: $\mathcal{T}$, $\mathbf{x}_{\text{samp}}$
Output: $\mathcal{T}$

$(\mathcal{V}, \mathcal{E}) \leftarrow \mathcal{T}$;
$\mathbf{x}_{\text{nearest}} \leftarrow \text{Nearest}(\mathcal{V}, \mathbf{x}_{\text{samp}})$;
$\mathbf{x}_{\text{new}} \leftarrow \text{Steer}(\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{samp}})$;
if $\text{ObstacleFree}(\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{new}})$ **then**
  $\mathcal{X}_{\text{near}} \leftarrow \text{NearNode}(\mathcal{V}, \mathbf{x}_{\text{new}})$;
  `// ChooseParent for` $\mathbf{x}_{\text{new}}$`:`
  $\text{minCost} \leftarrow \infty$;
  for $\mathbf{x}_{\text{near}} \in \mathcal{X}_{\text{near}}$ **do**
    if $\text{ObstacleFree}(\mathbf{x}_{\text{near}}, \mathbf{x}_{\text{new}})$ *and* $\mathbf{x}_{\text{near}}.\text{cost} + \text{cost}(\mathbf{x}_{\text{near}}, \mathbf{x}_{\text{new}}) < \text{minCost}$ **then**
      $\text{minCost} \leftarrow \mathbf{x}_{\text{near}}.\text{cost} + \text{cost}(\mathbf{x}_{\text{near}}, \mathbf{x}_{\text{new}})$;
      $\mathbf{x}_{\text{min}} \leftarrow \mathbf{x}_{\text{near}}$;
  $\mathcal{V} \leftarrow \mathcal{V} \cup \{\mathbf{x}_{\text{new}}\}$, $\mathcal{E} \leftarrow \mathcal{E} \cup \{(\mathbf{x}_{\text{min}}, \mathbf{x}_{\text{new}})\}$;
  `// Rewire:`
  for $\mathbf{x}_{\text{near}} \in \mathcal{X}_{\text{near}}$ **do**
    if $\text{ObstacleFree}(\mathbf{x}_{\text{new}}, \mathbf{x}_{\text{near}})$ *and* $\mathbf{x}_{\text{new}}.\text{cost} + \text{cost}(\mathbf{x}_{\text{new}}, \mathbf{x}_{\text{near}}) < \mathbf{x}_{\text{near}}.\text{cost}$ **then**
      $\mathbf{x}_{\text{near}}.\text{cost} \leftarrow \mathbf{x}_{\text{new}}.\text{cost} + \text{cost}(\sigma)$;
      $\mathbf{x}_{\text{near}}.\text{parent} \leftarrow \mathbf{x}_{\text{new}}$;
return $\mathcal{T} \leftarrow (\mathcal{V}, \mathcal{E})$

Several methods aim to enhance the probability of sampling states from $\mathcal{X}_f$ through sample biasing. For instance, Informed-RRT* employs the heuristic $\hat{f}(\mathbf{x}) := \|\mathbf{x}_{\text{init}} - \mathbf{x}\|_2 + \|\mathbf{x} - \mathbf{x}_{\text{goal}}\|_2$ to estimate the unknown $f(\mathbf{x})$. The set $\mathcal{X}_f$ is then estimated by

$$\mathcal{X}_{\hat{f}} := \{\mathbf{x} \in \mathcal{X} : \|\mathbf{x}_{\text{init}} - \mathbf{x}\|_2 + \|\mathbf{x} - \mathbf{x}_{\text{goal}}\|_2 \leq c_{\text{best}}\}.$$

The Sample procedure of Informed-RRT* is modified to generate uniformly distributed samples from the hyperellipsoid $\mathcal{X}_{\hat{f}}$.

RRT*-Smart also implements biased sampling to accelerate the convergence rate. It tends to sample new states as close as possible to obstacles, inspired by visibility graph techniques, which optimize the existing path at turns.

Another approach is adaptive sampling, utilized by CE-RRT*, which applies the cross-entropy method to estimate the probability $\mathbb{P}(\mathcal{X}_f)$. This estimator is then used to sample trajectories, resulting in a faster convergence rate.

Similar to RRT-Connect, there is also a bidirectional version of RRT* known as B-RRT*. B-RRT* employs a slight variation of the greedy RRT-Connect heuristic for connecting two trees. However, using two directional trees with a greedy connection heuristic does not guarantee asymptotic optimality.

4. The RRT-Based Motion Planning with Dynamics

It is challenging to apply RRT* to problems with dynamics due to the need for two domain-specific extension heuristics [40]: a distance metric and a steering procedure.

On the one hand, while the Euclidean distance is widely used in high-level path planning, it may not accurately reflect the dynamics of particular domains, especially in

cases involving complex or underactuated dynamics. This can lead to poor exploration and slow convergence to optimal solutions.

On the other hand, a prevalent steering procedure in high-level path planning involves generating straight paths between vertices. However, steering safe trajectories between vertices in RRT* becomes challenging when considering low-level control synthesis.

In recent years, significant efforts have been made to adapt RRT-based algorithms to dynamic domains. This section will look at these methods that address the said challenges by leveraging either LQRs or CBFs.

4.1. The LQR-RRT Algorithm*

LQR-RRT* [48] is an extension of RRT* that integrates LQR into the Steer, Near, and Nearest procedures for optimal control synthesis. More specifically, given an equilibrium point $\mathbf{x}$, $\text{cost}(\mathbf{v}, \mathbf{x})$ is defined as the LQR cost from any state $\mathbf{v}$ to $\mathbf{x}$ under the LQR policy. The cost function over the neighborhood of $\mathbf{x}$ can be expressed as

$$\text{cost}_{\mathbf{x}}(\mathbf{v}) = (\mathbf{v} - \mathbf{x})^{\top} \mathbf{P} (\mathbf{v} - \mathbf{x}),$$

where $\mathbf{P}$ is a solution to Equation (2). Then, the three components of RRT* are modified as follows:

- LQRNearest: The $\text{LQRNearest}(\mathcal{V}, \mathbf{x})$ procedure identifies a node in $\mathcal{V}$ that is closest to $\mathbf{x}$ according to the cost function, i.e.,

$$\mathbf{x}_{\text{nearest}} = \arg\min_{\mathbf{v} \in \mathcal{V}} \text{cost}_{\mathbf{x}}(\mathbf{v}).$$

- Near nodes: the $\text{LQRNear}(\mathcal{V}, \mathbf{x})$ procedure employs the cost function to find a set of near nodes $\mathcal{X}_{\text{near}}$ in $\mathcal{T}$ that are close to $\mathbf{x}$, i.e.,

$$\mathcal{X}_{\text{near}} := \{\mathbf{v} \in \mathcal{V} : \text{cost}_{\mathbf{x}}(\mathbf{v}) \leq r_n\},$$

  where r_n is consistent with Equation (7).
- Steer: Given two states, $\mathbf{x}, \mathbf{x}'$, the LQR controller generates a sequence of optimal controls $\{\mathbf{u}_i\}$ that steer a state trajectory from $\mathbf{x}$ to $\mathbf{x}'$.

4.2. The CBF-RRT Algorithm and Its Variations

CBF-RRT is an extension of RRT that incorporates CBFs into the steering function to ensure safety, eliminating the need for the Nearest or ObstacleFree procedures of RRT. The Steer procedure is modified as follows:

- Safe steer: Given a state $\mathbf{x}_0$, a time horizon t_h, and a control reference $k(\mathbf{x})$, the $\text{SafeSteer}(\mathbf{x}_0, t_h, k)$ procedure steers the state $\mathbf{x}_0$ to an exploratory new state $\mathbf{x}_{\text{new}}$. Concretely, it generates a sequence of control inputs $\mathbf{u}(t)$ by solving a sequence of CBF-QPs (Equation (4)) that steers a collision-free trajectory to $\mathbf{x}_{\text{new}}$ at time $t_0 + t_h$.

The overall algorithm of CBF-RRT is outlined in Algorithm 4.

Similar to CBF-RRT, CBF-RRT* incorporates CBFs into the RRT* algorithm to ensure safety. In addition to safety, RRT* requires the exact local motion planning because it needs to steer $\mathbf{x}_{\text{near}}$ to $\mathbf{x}_{\text{new}}$ in the ChooseParent procedure and steer $\mathbf{x}_{\text{new}}$ to $\mathbf{x}_{\text{near}}$ in the Rewire procedure. Consequently, CBF-RRT* implements the exact motion planning by solving a sequence of CLF-CBF-QPs (Equation (6)). The state trajectory from $\mathbf{x}_{\text{new}}$ to $\mathbf{x}_{\text{near}}$ can be also utilized to evaluate the cost between the two states, offering greater precision than the LQR cost-to-go function used in LQR-RRT*.

Algorithm 4: CBF-RRT

Input: N
Output: $\mathcal{T}$
Initialization: $\mathcal{T} = (\mathcal{V}, \mathcal{E}), \mathcal{V} \leftarrow \{(\mathbf{x}_{\text{init}}, t_{\text{init}})\}, \mathcal{E} \leftarrow \emptyset$;
for $i \leftarrow 1$ **to** N **do**
  $\mathbf{v}_{\text{samp}} \leftarrow \text{VerticesSample}(\mathcal{V})$;
  Select a control reference $k(\mathbf{x})$;
  $\mathbf{x}_{\text{new}}, t_{\text{new}}, \mathbf{u}_{\text{traj}}, \mathbf{x}_{\text{traj}} \leftarrow \text{SafeSteer}(\mathbf{v}_{\text{samp}}, t_h, k)$;
  $\mathcal{V} \leftarrow \mathcal{V} \cup \{(\mathbf{x}_{\text{new}}, t_{\text{new}})\}, \mathcal{E} \leftarrow \mathcal{E} \cup \mathbf{x}_{\text{traj}}$;
return $\mathcal{T} \leftarrow (\mathcal{V}, \mathcal{E})$

LQR-CBF-RRT* [48], an incremental version of CBF-RRT*, not only incorporates CBFs into the steering function for safety but also employs LQR for optimal control synthesis. Additionally, it proposes truncating the state trajectory that violates CBF constraints to reduce the burden of solving the QPs, leading to a new Steer procedure:

- LQR-CBF-Steer: Given two states $\mathbf{x}, \mathbf{x}'$, the LQR controller generates a sequence of optimal controls $\{\mathbf{u}_i\}$ that steer a state trajectory from $\mathbf{x}$ to $\mathbf{x}'$. The CBF constraints are checked to ensure that the generated trajectory is collision-free. If none of the CBF constraints are violated, then $\mathbf{x}'$ is added to the tree. Otherwise, the trajectory is truncated so that the end state $\mathbf{x}_{\text{new}}$ is safe, and the end state $\mathbf{x}_{\text{new}}$ is then added to the tree.

5. Experimental Results

In this section, we conduct an experimental study to compare the performance of several baseline methods.

5.1. Path Planning

We evaluated RRT, RRT*, RRT-Connect, Informed-RRT*, and RRT*-Smart on two examples for planning the motions of rigid objects in 2D. The implementations of RRT, RRT*, RRT-Connect, RRT*-Smart, and Informed-RRT* were based on the reference code from the package https://github.com/zhm-real/PathPlanning (accessed on 10 December 2024). The hyperparameters for these methods were set as follows: step_len $=$ 1, goal_sample_rate $=$ 0.1, and search_radius $=$ 12 (applied to RRT*-based methods only).

Figure 1 illustrates the evolution of baseline methods across both examples. The comparison between RRT and RRT* demonstrates that RRT* can asymptotically optimize the existing path but its convergence rate is notably slow. The comparison between RRT and RRT-Connect demonstrates that the greedy behavior of RRT-Connect improves both the convergence rate and final solution quality of RRT. With the direct sampling of an ellipsoidal subset, Informed-RRT* improves the convergence rate and final solution quality compared to RRT*. Additionally, with intelligent sampling and path optimization, RRT*-Smart produces significantly shorter paths than RRT*.

Figure 2 presents the convergence pattern of the costs associated with baseline methods on these two examples. RRT*-Smart demonstrates the fastest convergence rate and the highest final solution quality among all baseline methods. This highlights the effectiveness of the intelligent sampling and path optimization techniques introduced by RRT*-Smart. Notably, RRT-Connect also performs exceptionally well, as its greedy connect heuristic significantly outperforms both RRT* and Informed-RRT*. Additionally, Informed-RRT* shows improvements in both convergence rate and final solution quality compared to RRT*.

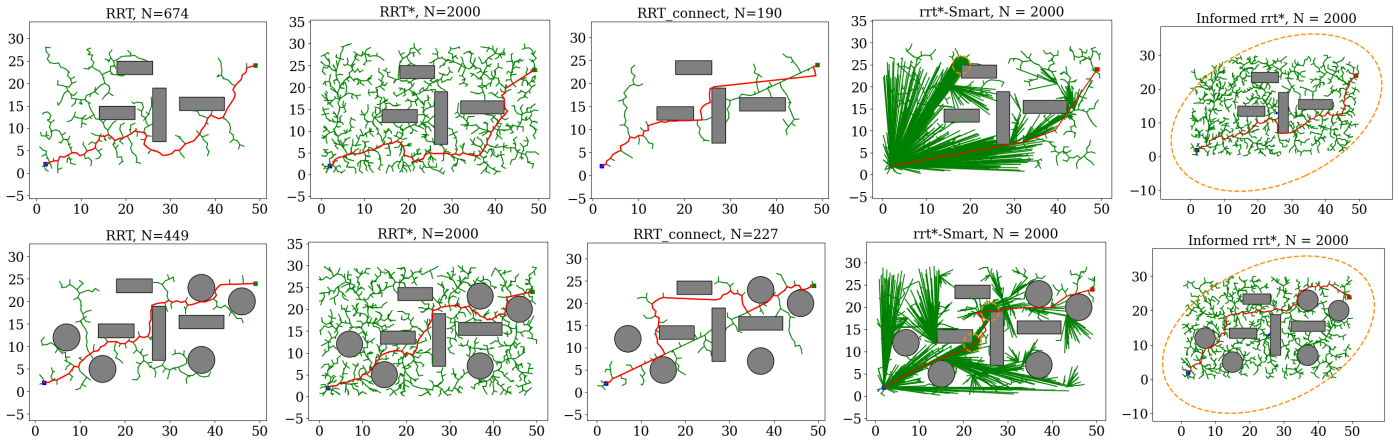


Figure 1. The evolution of baseline methods on two examples. In the visual representation, green edges represent sampled trajectories, the red trajectory highlights the final solution achieved after 1000 iterations, and gray areas represent obstacles.

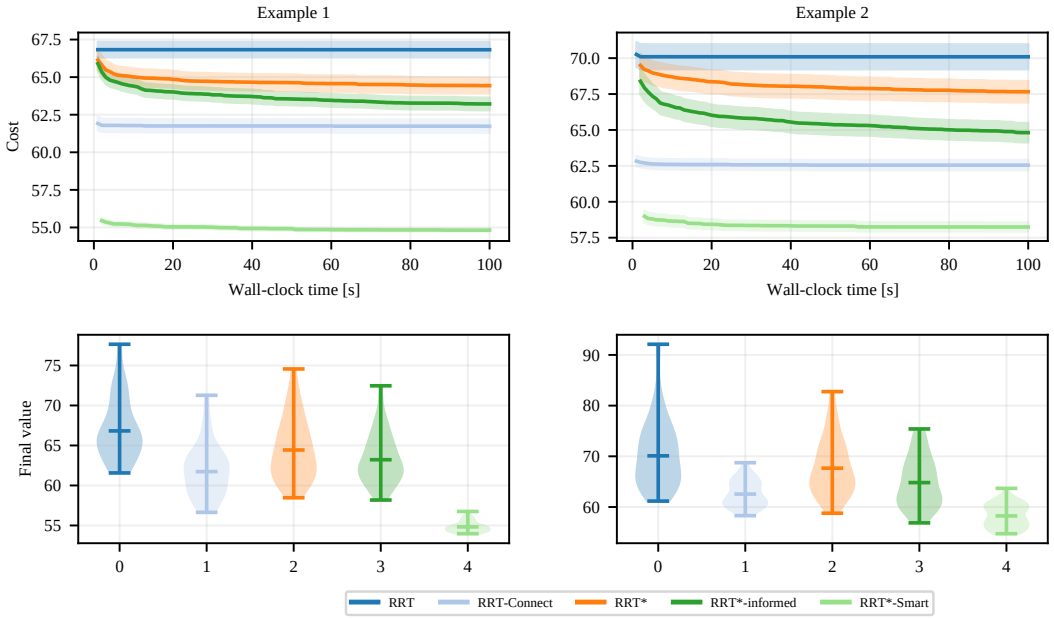


Figure 2. (**Top row**) Costs are plotted against the running time showing the convergence rate of baseline methods, averaged over 50 runs. (**Bottom row**) The distribution over the final costs over 50 runs.

5.2. Double Integrator Model

Following [48], we considered a double integrator model with linear dynamics,

$$\ddot{x_1} = u_1, \ \ddot{x_3} = u_2$$

with the state $\mathbf{x} = [x_1, x_2, x_3, x_4]$ where $[x_1, x_3]$ denotes the position and $[x_2, x_4]$ denotes the velocity. The control input $\mathbf{u} = [u_1, u_2]$ consists of acceleration.

We present an experimental performance comparison between LQR-RRT* and LQR-CBF-RRT*. The LQR-RRT* and LQR-CBF-RRT* used reference implementations from the package https://github.com/mingyucai/LQR_CBF_rrtStar (accessed on 10 December 2024). The hyperparameters for these methods were set as follows: step_len $= 10$, goal_sample_rate $= 0.1$, and search_radius $= 20$.

Figure 3 shows the evolution of these two methods on the double integrator model. It can be observed that, given the same number of iterations, LQR-CBF-RRT* achieves a

higher-quality final solution compared to LQR-RRT*. However, as is shown in Figure 4, LQR-RRT* achieves a faster convergence rate with respect to running time than LQR-CBF-RRT*. This difference may be attributed to the increased time complexity introduced by using CBFs in LQR-CBF-RRT*.

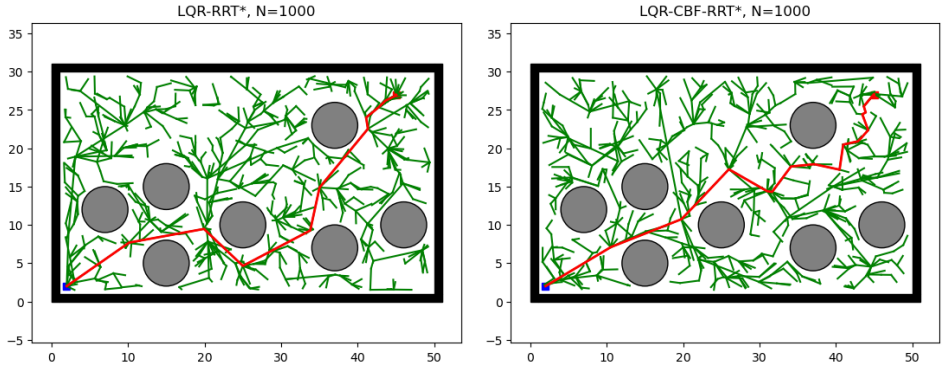


Figure 3. The evolution of LQR-RRT* and LQR-CBF-RRT* on the double integrator model. In the visual representation, green edges represent sampled trajectories, the red trajectory highlights the final solution achieved after 1000 iterations, and gray areas represent obstacles.

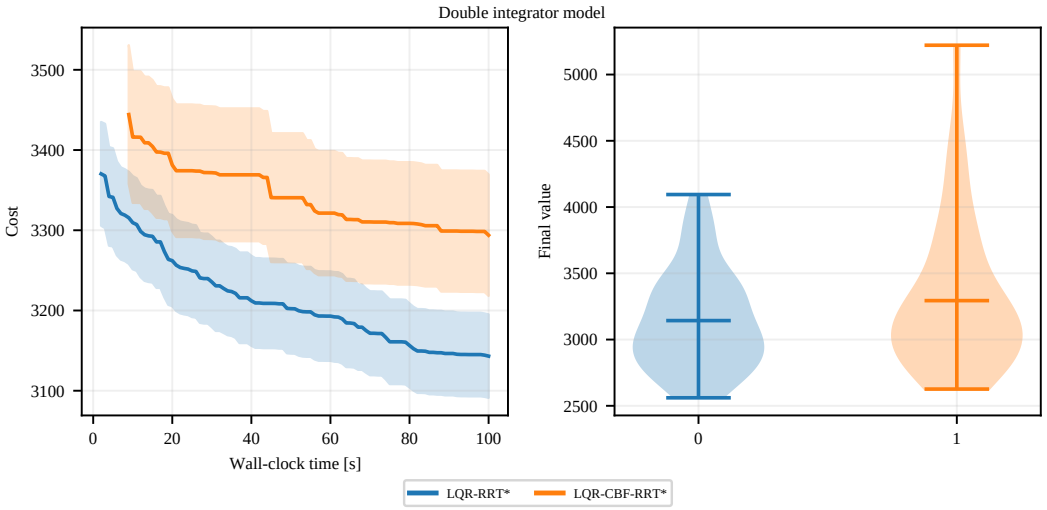


Figure 4. (**Left**) Costs are plotted against the running time showing the convergence rate of LQR-RRT* and LQR-CBF-RRT*, averaged over 50 runs. (**Right**) The distribution over the final costs over 50 runs.

6. Conclusions

In this paper, we have summarized the current state of RRT-based algorithms and conducted an experimental study to compare the performance of several baseline methods. Our results confirm a good performance of the greedy connection heuristic introduced by RRT-Connect, the intelligent sampling and path optimization introduced by RRT*-Smart, and the effectiveness of LQR-RRT*. However, the field still has much to explore. There are undoubtedly many new challenges, discoveries, and insights awaiting further investigation in the future.

Author Contributions: Conceptualization, Y.C. and X.Y.; investigation, Q.C.; writing—original draft preparation, Q.C.; writing—review and editing, Y.C. and X.Y.; visualization, Q.C.; supervision, X.Y.; project administration, Y.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. González, D.; Pérez, J.; Montero, V.M.; Nashashibi, F. A Review of Motion Planning Techniques for Automated Vehicles. *IEEE Trans. Intell. Transp. Syst.* **2016**, *17*, 1135–1145. [CrossRef]
2. Liu, Y.; Badler, N.I. Real-Time Reach Planning for Animated Characters Using Hardware Acceleration. In Proceedings of the 16th International Conference on Computer Animation and Social Agents, CASA 2003, New Brunswick, NJ, USA, 7–9 May 2003; IEEE Computer Society: Washington, DC, USA, 2003; pp. 86–93. [CrossRef]
3. Taylor, R.H.; Menciassi, A.; Fichtinger, G.; Fiorini, P.; Dario, P. Medical Robotics and Computer-Integrated Surgery. In *Springer Handbook of Robotics*; Siciliano, B., Khatib, O., Eds.; Springer Handbooks; Springer: Berlin/Heidelberg, Germany, 2016; pp. 1657–1684. . [CrossRef]
4. Latombe, J. Motion Planning: A Journey of Robots, Molecules, Digital Actors, and Other Artifacts. *Int. J. Robot. Res.* **1999**, *18*, 1119–1128. [CrossRef]
5. Beyer, T.; Jazdi, N.; Göhner, P.; Yousefifar, R. Knowledge-based planning and adaptation of industrial automation systems. In Proceedings of the 20th IEEE Conference on Emerging Technologies & Factory Automation, ETFA 2015, Luxembourg, 8–11 September 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 1–4. [CrossRef]
6. Lozano-Pérez, T.; Wesley, M.A. An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles. *Commun. ACM* **1979**, *22*, 560–570. [CrossRef]
7. Schwartz, J.T.; Sharir, M. On the "piano movers" problem. II. General techniques for computing topological properties of real algebraic manifolds. *Adv. Appl. Math.* **1983**, *4*, 298–351. [CrossRef]
8. Asano, T.; Asano, T.; Guibas, L.J.; Hershberger, J.; Imai, H. Visibility-Polygon Search and Euclidean Shortest Paths. In Proceedings of the 26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21–23 October 1985; IEEE Computer Society: Washington, DC, USA, 1985; pp. 155–164. [CrossRef]
9. Alexopoulos, C.; Griffin, P.M. Path planning for a mobile robot. *IEEE Trans. Syst. Man Cybern.* **1992**, *22*, 318–322. [CrossRef]
10. Canny, J.F. *The Complexity of Robot Motion Planning*; MIT Press: Cambridge, MA, USA, 1988.
11. Brooks, R.A.; Lozano-Pérez, T. A subdivision algorithm in configuration space for findpath with rotation. *IEEE Trans. Syst. Man Cybern.* **1985**, *15*, 224–233. [CrossRef]
12. 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. [CrossRef]
13. Koenig, S.; Likhachev, M.; Furcy, D. Lifelong Planning A. *Artif. Intell.* **2004**, *155*, 93–146. [CrossRef]
14. Koenig, S.; Likhachev, M. Fast replanning for navigation in unknown terrain. *IEEE Trans. Robot.* **2005**, *21*, 354–363. [CrossRef]
15. Gavrilut, I.; Tepelea, L.; Gacsádi, A. CNN processing techniques for image-based path planning of a mobile robot. In Proceedings of the WSEAS 15th International Conference on Systems, Corfu Island, Greece, 14–16 July 2011; pp. 259–263.
16. Al-Sagban, M.; Dhaouadi, R. Neural-based navigation of a differential-drive mobile robot. In Proceedings of the 12th International Conference on Control Automation Robotics & Vision, ICARCV 2012, Guangzhou, China, 5–7 December 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 353–358. [CrossRef]
17. Engedy, I.; Horvath, G. Artificial neural network based mobile robot navigation. In Proceedings of the 2009 IEEE International Symposium on Intelligent Signal Processing, Budapest, Hungary, 26–28 August 2009; pp. 241–246. [CrossRef]
18. Zhong, Y.; Shirinzadeh, B.; Yuan, X. Optimal Robot Path Planning with Cellular Neural Network. In *Advanced Engineering and Computational Methodologies for Intelligent Mechatronics and Robotics*; IGI Global: Hershey, PA, USA, 2013; pp. 19–38. [CrossRef]
19. Hills, J.; Zhong, Y. Cellular neural network-based thermal modelling for real-time robotic path planning. *Int. J. Agil. Syst. Manag.* **2014**, *7*, 261–281. [CrossRef]
20. Chen, C.; Seff, A.; Kornhauser, A.L.; Xiao, J. DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving. In Proceedings of the 2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, 7–13 December 2015; IEEE Computer Society: Washington, DC, USA, 2015; pp. 2722–2730. [CrossRef]
21. Bansal, M.; Krizhevsky, A.; Ogale, A.S. ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst. In Proceedings of the Robotics: Science and Systems XV, Freiburg im Breisgau, Germany, 22–26 June 2019. . [CrossRef]
22. Zhou, Q.; Gao, S.; Qu, B.; Gao, X.; Zhong, Y. Crossover Recombination-Based Global-Best Brain Storm Optimization Algorithm for Uav Path Planning. *Proc. Rom. Acad. Ser. A-Math. Phys. Tech. Sci. Inf. Sci.* **2022**, *23*, 209–218.
23. Roberge, V.; Tarbouchi, M.; Labonté, G. Comparison of Parallel Genetic Algorithm and Particle Swarm Optimization for Real-Time UAV Path Planning. *IEEE Trans. Ind. Inform.* **2013**, *9*, 132–141. [CrossRef]
24. Kim, J.; Lee, J. Trajectory Optimization With Particle Swarm Optimization for Manipulator Motion Planning. *IEEE Trans. Ind. Inform.* **2015**, *11*, 620–631. [CrossRef]
25. Morin, M.; Abi-Zeid, I.; Quimper, C. Ant colony optimization for path planning in search and rescue operations. *Eur. J. Oper. Res.* **2023**, *305*, 53–63. [CrossRef]

26. Khatib, O. Real-time obstacle avoidance for manipulators and mobile robots. In Proceedings of the Proceedings of the 1985 IEEE International Conference on Robotics and Automation, St. Louis, MO, USA, 25–28 March 1985; IEEE: Piscataway, NJ, USA, 1985; pp. 500–505. [CrossRef]
27. Bahwini, T.; Zhong, Y.; Gu, C. Path planning in the presence of soft tissue deformation. *Int. J. Interact. Des. Manuf. (IJIDeM)* **2019**, *13*, 1603–1616. [CrossRef]
28. Chiang, H.; Malone, N.; Lesser, K.; Oishi, M.; Tapia, L. Path-guided artificial potential fields with stochastic reachable sets for motion planning in highly dynamic environments. In Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26–30 May 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 2347–2354. [CrossRef]
29. Zhao, D.; Yi, J. Robot Planning with Artificial Potential Field Guided Ant Colony Optimization Algorithm. In Proceedings of the Advances in Natural Computation, Second International Conference, ICNC 2006, Xi'an, China, 24–28 September 2006. Proceedings, Part II; Lecture Notes in Computer Science; Jiao, L., Wang, L., Gao, X., Liu, J., Wu, F., Eds.; Springer: Berlin/Heidelberg, Germany, 2006; Volume 4222, pp. 222–231. [CrossRef]
30. Hu, Y.; Yang, S.X. A Knowledge based Genetic Algorithm for Path Planning of a Mobile Robot. In Proceedings of the Proceedings of the 2004 IEEE International Conference on Robotics and Automation, ICRA 2004, New Orleans, LA, USA, 26 April–1 May 2004; IEEE: Piscataway, NJ, USA, 2004; pp. 4350–4355. [CrossRef]
31. LaValle, S.M. *Rapidly-Exploring Random Trees : A New Tool for Path Planning*; TR 98-11; Computer Science Department, Iowa State University: Ames, IA, USA, 1998. Available online: https://msl.cs.illinois.edu/~lavalle/papers/Lav98c.pdf (accessed on 10 December 2024).
32. Karaman, S.; Frazzoli, E. Sampling-based algorithms for optimal motion planning. *Int. J. Robot. Res.* **2011**, *30*, 846–894. [CrossRef]
33. Qureshi, A.H.; Ayaz, Y. Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments. *Robot. Auton. Syst.* **2015**, *68*, 1–11. [CrossRef]
34. 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 2014, Chicago, IL, USA, 14–18 September 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 2997–3004. [CrossRef]
35. Islam, F.; Nasir, J.; Malik, U.; Ayaz, Y.; Hasan, O. RRT*-Smart: Rapid convergence implementation of RRT* towards optimal solution. In Proceedings of the 2012 IEEE International Conference on Mechatronics and Automation, Chengdu, China, 5–8 August 2012; pp. 1651–1656. [CrossRef]
36. Kobilarov, M. Cross-Entropy Randomized Motion Planning. In *Proceedings of the Robotics: Science and Systems VII, University of Southern California, Los Angeles, CA, USA, 27–30 June 2011*; Durrant-Whyte, H.F., Roy, N., Abbeel, P., Eds.; MIT Press: Cambridge, MA, USA, 2012. . [CrossRef]
37. Kuffner, J.J., Jr.; LaValle, S.M. RRT-Connect: An Efficient Approach to Single-Query Path Planning. In Proceedings of the 2000 IEEE International Conference on Robotics and Automation, ICRA 2000, San Francisco, CA, USA, 24–28 April 2000; IEEE: Piscataway, NJ, USA, 2000; pp. 995–1001. [CrossRef]
38. Jordan, M.; Perez, A. *Optimal Bidirectional Rapidly-Exploring Random Trees*; Technical Report MIT-CSAIL-TR-2013-021; Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology: Cambridge, MA, USA, 2013.
39. Webb, D.J.; van den Berg, J. Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics. In Proceedings of the 2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, 6–10 May 2013; IEEE: Piscataway, NJ, USA, 2013; pp. 5054–5061. [CrossRef]
40. Perez, A.; Platt, R., Jr.; Konidaris, G.D.; Kaelbling, L.P.; Lozano-Pérez, T. LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics. In Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2012, St. Paul, MN, USA, 14–18 May 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 2537–2542. [CrossRef]
41. Yang, G.; Vang, B.; Serlin, Z.; Belta, C.; Tron, R. Sampling-based Motion Planning via Control Barrier Functions. In Proceedings of the Proceedings of the 2019 3rd International Conference on Automation, Control and Robots, Prague, Czech Republic, 11–13 October 2019; ICACR 2019; pp. 22–29. [CrossRef]
42. Ahmad, A.; Belta, C.; Tron, R. Adaptive Sampling-based Motion Planning with Control Barrier Functions. In Proceedings of the 61st IEEE Conference on Decision and Control, CDC 2022, Cancun, Mexico, 6–9 December 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 4513–4518. [CrossRef]
43. Hespanha, J.P. *Linear Systems Theory*, 2nd ed.; Princeton University Press: Princeton, NJ, USA, 2018.
44. Blanchini, F.; Miani, S. *Set-Theoretic Methods in Control*, 2nd ed.; Springer: Berlin/Heidelberg, Germany, 2008.
45. Xu, X.; Tabuada, P.; Grizzle, J.W.; Ames, A.D. Robustness of Control Barrier Functions for Safety Critical Control. In Proceedings of the 5th IFAC Conference on Analysis and Design of Hybrid Systems, ADHS 2015, Atlanta, GA, USA, 14–16 October 2015; IFAC-PapersOnLine; Egerstedt, M., Wardi, Y., Eds.; Elsevier: Amsterdam, The Netherlands, 2015; Volume 48, pp. 54–61. [CrossRef]
46. Khalil, H. *Nonlinear Systems: Global Edition*; Pearson: London, UK, 2015.

47. LaValle, S.M.; Kuffner, J.J., Jr. Randomized Kinodynamic Planning. *Int. J. Robot. Res.* **2001**, *20*, 378–400. [CrossRef]
48. Yang, G.; Cai, M.; Ahmad, A.; Belta, C.; Tron, R. Efficient LQR-CBF-RRT*: Safe and Optimal Motion Planning. *arXiv* **2023**, arXiv:2304.00790.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.