A Survey of Trajectory Planning Methods for Autonomous Driving — Part I: Unstructured Scenarios

Yuqing Guo, Zelin Guo, Yazhou Wang, Danya Yao, *Member*, *IEEE*, Bai Li, *Member*, *IEEE*, and Li Li, *Fellow*, *IEEE*

Abstract—**Trajectory planning is a critical function in an autonomous vehicle, which is about generating a local spatio-temporal curve with safety, traverse efficiency, and comfort factors considered. Many survey papers have been published about trajectory planning in structured scenarios while a survey regarding the planners in unstructured scenarios is still absent. Driving in unstructured scenarios involves massive irregularly placed obstacles and mixed forward/backward maneuvers, which easily make the existing structured-scenario planners inapplicable. This paper aims to leverage the challenges in unstructured scenarios, review the existing planners about their strengths, limitations, and computational complexities, develop an open-source library containing prevalent trajectory planners suitable for unstructured scenarios, and provide insights into future developments. Particularly, we advocate the usage of a two-stage computational architecture, where stage one finds a coarse trajectory/path and stage two refines via a gradient-based optimizer. The two-stage architecture has been well adopted in the past score and deserves to be further developed for complex real-world trajectory planning cases in unstructured driving scenarios.**

***Index Terms*—Autonomous driving, trajectory planning, motion planning, unstructured scenario, two-stage planning**

I. Introduction

TRAJECTORY planning methods have been extensively investigated in the field of autonomous driving [1]. The growing significance of intelligent vehicles in enhancing transportation safety, efficiency, and passenger comfort has drawn much interest from both academic and industrial circles, thus establishing trajectory planning as a prominent research topic.

A trajectory, in this context, is a time-stamped curve illustrating the spatial-temporal motion of an agent, and trajectory planning concerns the determination of such a curve. In the research field of autonomous driving, trajectory planning focuses on identifying a spatial-temporal curve for an ego vehicle that connects the initial and goal poses while satisfying several key conditions: 1) ensuring kinematic feasibility to facilitate the tracking control process in a low-level controller, 2) ensuring that the ego vehicle is free from collisions with environmental obstacles, and 3) optimizing local driving behaviors according to a predefined cost function [2]. It is noteworthy that trajectory planning is sometimes referred to as motion planning in some previous studies.

Fig. 1 presents the standard architecture of a pipeline-based intelligent vehicle system, with trajectory planning serving as a pivotal component within the decision and planning module [3]. Within this module, route planning outlines a high-level path on the road network [4], behavioral planning governs localized decisions at critical waypoints along this high-level path, and trajectory planning meticulously crafts precise trajectories between adjacent waypoints [5]. As depicted in this figure, a planned trajectory from the decision and planning module is executed by the subsequent tracking control module through a feedback mechanism. While closed-loop control has a direct impact on vehicle movements, it operates reactively, addressing only immediate tracking deviations or emergent safety failures [6]. Conversely, trajectory planning proactively forecasts future actions, integrating current perception and location data to anticipate potential risks. In essence, a trajectory planner can be perceived as a feedforward controller. Given that autonomous driving involves critical safety considerations, relying solely on feedback control for safety measures is passive and somewhat sluggish, neglecting the inherent intelligence of autonomous driving. In this context, trajectory planning assumes the primary role in ensuring both the safety and intelligence of autonomous driving. There are essentially two empirical approaches to trajectory planning: one involves directly deriving a trajectory, while the other entails first planning a path (path planning) and subsequently defining a time profile (velocity planning) along the derived path. Table I provides a succinct comparison among these mentioned concepts.

Unstructured scenarios constitute a distinctive category within the realm of autonomous driving scenarios. Typical unstructured scenarios are encountered in ports, open-pit mines, parking garages, manufacturing workshops, and logistic warehouses. In contrast to structured scenarios [7], the defining characteristic of unstructured scenarios is the absence of a reference line to guide local driving behaviors. While a

Manuscript received Nov. 17, 2023; accepted Nov. 25, 2023. This work was supported in part by the National Key R&D Program of China under Grant 2022YFB2502905, the National Natural Science Foundation of China under Grant 62103139, and Hejian Youth Talent Program of Hunan Province under Grant 2023RC3115. *(Corresponding authors: Bai Li and Li Li)*

Yuqing Guo and Zelin Guo are with the Department of Automation, Tsinghua University, Beijing 100084, China (e-mails: gyq18@tsinghua.org.cn, gzl23@tsinghua.edu.cn).

Yazhou Wang is with the College of Mechanical and Vehicle Engineering, Hunan University, Changsha 410082, China (e-mail: albert@hnu.edu.cn).

Danya Yao is with the Department of Automation, BNRist, Tsinghua University, Beijing 100084, China, and also with Jiangsu Province Collaborative Innovation Center of Modern Urban Traffic Technologies, Nanjing 210096, China (e-mail: yaody@tsinghua.edu.cn).

Bai Li is with the State Key Laboratory of Advanced Design and Manufacturing for Vehicle Body, Hunan University, Changsha 410082, China (e-mail: libai@zju.edu.cn).

Li Li is with the Department of Automation, BNRist, Tsinghua University, Beijing 100084, China (e-mail: li-li@tsinghua.edu.cn).

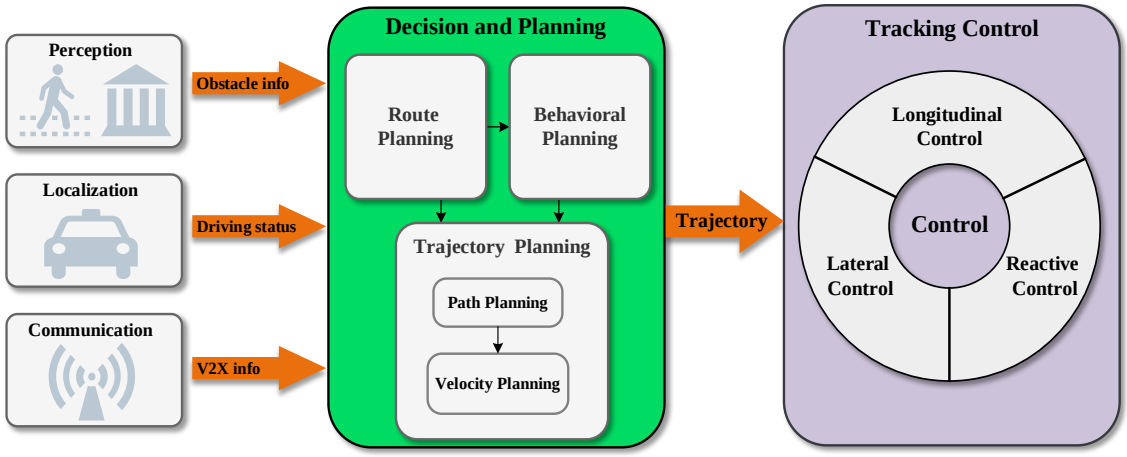


Fig. 1. Fundamenal architecture of a pipeline-based autonomous driving system.

TABLE I. DIFFERENCES AMONG CONCEPTS RELATED TO TRAJECTORY PLANNING

Module	Function	Duty	Constraint	Spatial Horizon	Temporal Horizon
Decision making and planning	Route planning	Schedule a road network-level path that connects the initial and goal spots	Vehicle kinematics × Collision avoidance ×	>100m	>1min <1hour
	Behavioral planning	Make local decisions as per the planned route and environmental restrictions	Vehicle kinematics × Collision avoidance √	>10m <100m	>1s <5min
	Path planning	Generate a spatial path that connects the local initial and goal poses	Vehicle kinematics √ Collision avoidance √	>10m <100m	>1s <1min
	Velocity planning	Generate a velocity profile along the aforementioned path	Vehicle kinematics √ Collision avoidance √	>10m <100m	>1s <1min
	Trajectory planning	Generate a spatial-temporal curve directly instead of doing path-velocity decomposition	Vehicle kinematics √ Collision avoidance √	>10m <100m	>1s <1min
Control	Tracking control	Motivate the ego vehicle to track a planned trajectory	Vehicle kinematics √ Collision avoidance ×	>0.5m <10m	>10ms <1s

reference line could be computed offline for an unstructured scenario, the introduction of new obstacles can significantly deviate the evasive trajectory from this reference line [8], underscoring its limited utility in such contexts. In structured scenarios such as urban roads, the reference line typically corresponds to the road's centerline. When new obstacles appear, the ego vehicle navigates by nudging or changing lanes, during which the ego vehicle's heading angle remains closely aligned with the reference line's direction. Conversely, in unstructured scenarios, evasive trajectories may deviate from the nominal reference line in heading angle due to the cluttered and narrow features of these scenarios. The absence of a reference line implies that many reference-line-based planning algorithms designed for structured scenarios are inapplicable in unstructured scenarios. Furthermore, if backward driving is permitted in unstructured scenarios, the planned trajectories may contain cusps, which are beyond the capability of planners intended for structured scenarios.

Trajectory planning in unstructured scenarios presents several challenges. First, the absence of a reliable reference line renders sampling along the reference line impossible. Second, accurate modeling of vehicle kinematics is imperative when operating in unstructured scenarios at low speeds, indicating that trajectory planning may not be decomposed into two sequential steps (i.e., path planning + velocity planning). Third, obstacles in unstructured scenarios clutter and narrow the environment, resulting in multiple homotopy classes, but only one homotopy class leads to the optimal trajectory.

Trajectory planners for unstructured scenarios have undergone extensive investigation and development over the past decade. However, no comprehensive survey systematically summarizes the performances of these planners and their interconnections. Most published surveys have predominantly focused on trajectory planning methods for structured scenarios. For instance, Dixit et al. [6] concentrated on trajectory planners for overtaking scenarios, while Claussmann et al. [5] emphasized highway trajectory planning. These surveys commonly share certain limitations. First, surveys like [2] and [5] did not differentiate between robots and intelligent vehicles concerning geometric and vehicle kinematic/dynamic considerations. Second, some surveys, such as [2] and [3], placed excessive emphasis on presenting the planners while overlooking analyses of how these planners adapt to various scenarios.

This paper endeavors to present a comprehensive survey of prevalent trajectory planning techniques specifically tailored for unstructured scenarios. As a foundational step, we employ a unified optimal control problem (OCP) framework to delineate generic trajectory planning schemes. Broadly, an OCP consists of a cost function and multiple constraints, all of which are functions of state and control variables. We distinguish various trajectory planning schemes in various types of

TABLE II. FEATURES OF TRAJECTORY PLANNERS FOR VARIOUS UNMANNED SYSTEM TYPES

Category	Highlights	Constraint Features	Applicable Planner Type
Holonomic wheeled robot	High requirements in runtime Footprints are simple Moving speed is constant	High-accuracy boundary conditions High-accuracy collision avoidance Holonomic kinematics	Search-based methods Geometric methods
Rocket/spacecraft	Low requirements in runtime Footprints can be ignored Online receding-horizon replanning is not necessary	Low-accuracy boundary conditions Low-accuracy collision avoidance Nonholonomic kinematics	Search-based methods Optimization-based methods Geometric methods
Surface ship	High requirements in runtime Footprints are simple	High-accuracy boundary conditions High-accuracy collision avoidance Nonholonomic kinematics	Search-based methods Optimization-based methods Geometric methods
Quadcopter	High requirements in runtime Footprints are simple	High-accuracy boundary conditions Low-accuracy collision avoidance Holonomic kinematics	Search-based methods Optimization-based methods Geometric methods
Intelligent vehicle	High requirements in runtime Footprints are complex	High-accuracy boundary conditions High-accuracy collision avoidance Nonholonomic kinematics	Search-based methods Optimization-based methods

unstructured scenarios by elucidating the fundamental disparities among their cost functions and constraints. This approach also elucidates how different categories of planners conceptualize a trajectory planning scheme, thereby establishing a unified link among diverse planners. Given that prevalent planners often amalgamate distinct basic algorithms, we classify them into two categories: one-stage planners and two-stage planners. Specifically, one-stage planners directly determine trajectories in a single step, while two-stage planners first construct a coarse trajectory before executing a post-processing refinement step. This novel classification perspective provides explicit insight into how challenges in a particular planning scheme can be surmounted. This study additionally provides theoretical analyses of the algorithmic complexity of each typical planner. Comparative simulations are conducted to directly illustrate the performance of prevalent trajectory planners. Furthermore, we offer the source code for typical categories of trajectory planners, along with benchmark simulation cases.

The contributions of this survey paper are fourfold. First, the paper provides a systematic review of extant trajectory planners tailored for unstructured environments. A unified OCP framework delineates the trajectory planning task, thereby facilitating the categorization of disparate driving scenarios based on the definitions of cost functions and constraints within the OCP. This unified formulation elucidates the challenges specific to unstructured settings and establishes a basis for further analysis. Second, the paper categorizes existing planners according to the deployment of a two-stage architecture and the functional dynamics of each stage. This empirically motivated classification scheme aids in elucidating current and future research trajectories. Third, the paper offers both analytical and experimental comparisons among widely used planners, providing clear guidance for researchers in the selection of an appropriate planning algorithm for a given task. Fourth, a comprehensive library featuring open-source code of prevalent planners, written in multiple programming languages, is made available to validate our experimental results.

The rest of this paper is organized as follows. Section II formulates the trajectory planning scheme as a unified OCP. Section III is focused on the features of unstructured scenarios in a trajectory planning problem formulation and reviews the prevalent planners. Section IV delves into the encountered challenges, providing current solutions together with complexity analyses and experimental tests. Section V introduces the open-source library and the benchmark set. Section VI concludes the study with future research directions.

II. TRAJECTORY PLANNING PROBLEM FORMULATION

This section defines the trajectory planning task in the form of an OCP. We begin with a brief explanation of the differences between trajectory planning for an intelligent vehicle and that for other types of unmanned systems. Thereafter, a unified OCP is formulated for trajectory planning in an unstructured scenario. The differences between structured and unstructured are discussed at the end of this section.

A. Trajectory Planning for an Intelligent Vehicle versus Other Types of Unmanned Systems

This subsection analyzes the difference between trajectory planning for an intelligent vehicle and other types of unmanned systems. The concerned unmanned systems typically include holonomic wheeled robots, unmanned surface ships, spacecraft, and quadrotors.

Holonomic wheeled robots, commonly known as Automated Guided Vehicles (AGVs), are characterized by their capability to move in every direction, thus enabling the omission of kinematic constraints in trajectory planning. This means that the primary challenge in holonomic wheeled robot trajectory planning is how to avoid collisions. If the workspace does not contain moving obstacles, trajectory planning is degraded as path planning, which can be simply solved by the 2-dim A* search algorithm [9] or similar simple planners. If one expects the derived paths to be smooth, a post-processing step is implemented to polish a coarse path found by the 2-dim A* search [10].

Surface ships, spacecraft, and quadrotors [11]–[13] share similarities with intelligent vehicles in their trajectory planning schemes wherein boundary conditions and vehicle kinematics should be considered. Compared with those in spacecraft and quadrotors, collision-avoidance constraints w.r.t. intelligent vehicles are more complicated in an unstructured environment narrowed by close-range obstacles. Surface ships, spacecraft, and quadrotors usually keep a considerably large buffering distance from the surrounding obstacles for safety (e.g. the downwash effect related to a quadrotor) unless operating in

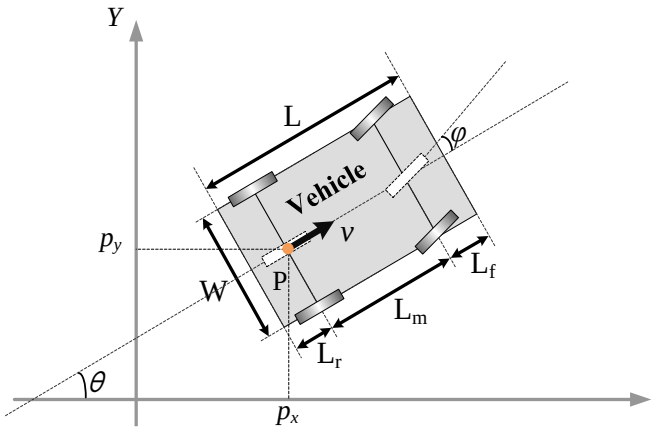


Fig. 2. Schematics on intelligent vehicle geometrics and kinematics.

extreme scenarios such as challenging competitions. Besides that, intelligent vehicle trajectory planning strictly holds an accurate constraint on the goal pose when driving in an unstructured scenario, which is rarely seen in trajectory planning problems of a surface ship, spacecraft, or quadrotor. Table II summarizes the discussions in this subsection.

B. Trajectory Planning Problem Formulation in Unstructured Scenarios

The problem formulation strategy has a substantial influence over the solution quality of a planned trajectory. Two prevalent strategies are continuous and discrete modeling methods.

In a problem formulated via a continuous modeling strategy, the decision variables are continuous functions of time. This renders a time-continuous OCP, which is about minimizing a specified cost function subject to a series of constraints. Such an OCP is straightforward in describing the real-world trajectory planning scheme. Analytical solutions to a generic OCP typically include Calculus of Variations, Hamilton-Jacobi-Bellman equations, and Pontryagin's Maximum Principle. However, these analytical methods cannot find closed-form solutions to all types of OCPs especially the ones we concern in this study, which contain non-convex constraints in collision avoidance and kinematics. Thus, we resort to solve the OCPs numerically, the first step of which is to convert the time-continuous OCP into its discretized form.

Discrete modeling strategies use a link of discrete waypoints characterized by equidistant or non-equidistant intervals to present the trajectory through interpolation, thereby converting the nominal time-continuous OCP into a mathematical programming problem. In this sense, the discrete modeling strategy eases the nominal trajectory planning scheme. However, the conversion from continuous states to discretized states renders a limitation, that is, a constraint violation risk might occur between adjacent waypoints. Thus, a successfully planned trajectory is actually dangerous or infeasible due to the constraint violations between adjacent waypoints.

In general, a time-continuous OCP is formulated as follows to describe a trajectory planning scheme:

$$\begin{aligned}&\min_{t_F,\boldsymbol{x}(t),\boldsymbol{u}(t)} J\big(t_F,\boldsymbol{x}(0),\boldsymbol{x}(t_F)\big)+\int_{\tau=0}^{t_F}\big|\omega\big(\tau,\boldsymbol{x}(\tau),\boldsymbol{u}(\tau)\big)\big|\mathrm{d}\tau\\&\textit{subject to}:\\&\quad \boldsymbol{x}(0)=\boldsymbol{x}_0,\boldsymbol{x}(t_F)=\boldsymbol{x}_F\\&\quad \boldsymbol{x}_{\min}\le\boldsymbol{x}(t)\le\boldsymbol{x}_{\max},\ \boldsymbol{u}_{\min}\le\boldsymbol{u}(t)\le\boldsymbol{u}_{\max}\\&\quad \dot{\boldsymbol{x}}(t)=\boldsymbol{f}(t,\boldsymbol{x}(t),\boldsymbol{u}(t))\\&\quad \mathbb{C}(\boldsymbol{x}(t))\cap \mathrm{O}^m(t)=\emptyset,\forall t\in[0,t_F]\end{aligned},\tag{1}$$

where $\boldsymbol{x}(t)$ is the state variable at time t; $\boldsymbol{u}(t)$ is the control variable at t; $\boldsymbol{x}_0$ and $\boldsymbol{x}_F$ are the vehicle states at the start and end points, respectively; $\boldsymbol{x}_{\min}$, $\boldsymbol{x}_{\max}$, $\boldsymbol{u}_{\min}$, and $\boldsymbol{u}_{\max}$ denote the boundary constraints of the state and control variables, respectively. $f(\,)$ represents the vehicle kinematic constraints, $\mathbb{C}(\boldsymbol{x}(t))$ denotes the space occupied by the footprint of the ego vehicle at t, $\mathrm{O}^m(t)$ denotes the space occupied by all of the m obstacles at t, $\mathbb{C}(\boldsymbol{x}(t))\cap\mathrm{O}^m(t)=\emptyset$ represents the collision-avoidance constraints at time t, $J(\,)$ defines the to-be-penalized polynomial at the terminal moment t_F, and $|\omega(\,)|$ is a non-negative norm-1 cumulative performance metric. As opposed to a norm-2 metric $\omega^2(\,)$, $|\omega(\,)|$ is advantageous to promote the convergence rate in numerically solving the OCP [64].

In addition to the aforementioned constraints such as collision-avoidance constraints and kinematic constraints, special demands should also be considered as constraints [14], such as communication-range constraints in a V2X scenario [15] and engine power constraints in an uneven terrain [7].

It is essential to underscore that (1) is not limited to trajectory planning tasks in unstructured scenarios; its application extends to structured scenarios as well. While the specific meanings of variables $\boldsymbol{x}$ and $\boldsymbol{u}$ vary, the core idea remains the same. A structured scenario is typically incorporated with a reference line, which helps to construct an easier frame than the Cartesian one. Details at this point are discussed in the next subsection.

In using the discrete modeling strategy, a trajectory is characterized by a sequence of equidistant discrete waypoints. Suppose that the number of waypoints is $(n+1)$ and the gross travel time is t_n, thus the equidistant time sub-interval is $\Delta t = t_n/n$. A general discrete formulation of trajectory planning is expressed as

$$\begin{aligned}&\min_{t_n,\boldsymbol{x}(k),\boldsymbol{u}(k)} J(t_n)+\textstyle\sum_{k=0}^{n}\big|\omega\big(\tau,\boldsymbol{x}(\tau),\boldsymbol{u}(\tau)\big)\big|\\&\textit{subject to}:\\&\boldsymbol{x}(0)=\boldsymbol{x}_0,\ \boldsymbol{x}(n)=\boldsymbol{x}_F\\&\boldsymbol{x}_{\min}\le\boldsymbol{x}(k)\le\boldsymbol{x}_{\max},\ \boldsymbol{u}_{\min}\le\boldsymbol{u}(k)\le\boldsymbol{u}_{\max},\ k=0,1,\dots,n,\\&\boldsymbol{x}(k+1)=f_k(\boldsymbol{x}(k),\boldsymbol{u}(k)),\ k=0,1,\dots,n-1,\\&\mathbb{C}(\boldsymbol{x}(k))\cap\mathrm{O}^m(k)=\emptyset,\ k=0,1,\dots,n.\end{aligned}\tag{2}$$

An intelligent vehicle generally drives at a low speed in unstructured scenarios, thus the vehicle lateral slip issue can be safely omitted. In such a driving condition, a single-track bicycle model (see Fig. 2) is commonly used to describe the ego vehicle's kinematic principle [16]. In a trajectory planning problem formulation with single-track kinematic constraints, decision variables include state collocation points $\boldsymbol{x}(k)=[p_x(k),p_y(k),v(k),\theta(k)]^{\mathrm{T}}$, control collocation points $\boldsymbol{u}(k)=[a(k),\varphi(k)]^{\mathrm{T}}$, and the driving duration variable t_n. Herein, $p_x(k),p_y(k),v(k)$, and $\theta(k)$ denote the location, velocity, and yaw angle at the kth time step; $a(k)$ and $\varphi(k)$ represent the acceleration and steering angle at the kth time step, respectively. The discretized kinematic constraints, denoted as $f(\,)$, is written as

TABLE III. COMPARISONS BETWEEN CONTINUOUS AND DISCRETE METHODS FOR DESCRIBING A TRAJECTORY PLANNING TASK

Model Type	Equation Type	Representative Methods	Solution Space Construction Principle
Continuous	Differential / algebraic	Function approximation methods such as polynomial and trigonometric functions	Define approximation function types/basis types, and then determine coefficients/parameters in the defined approximation functions
Discrete	Algebraic	Collocation methods such as orthogonal collocation direct transcription, Euler method, Simpson method	Define collocation types, and then determine collocation points by solving a mathematical programming problem

TABLE IV. COMPARATIVE INFLUENCES OF STRUCTURED AND UNSTRUCTURED SCENARIOS ON TRAJECTORY PLANNING TASKS

Scenario Type	Features Of Road	Shape and Placement Of Obstacles	Vehicle Dynamics	Planning Scope	Coordinate Frame	Kinematic Model	Decision Variables	Featured Constraints	Decomposition Method	Applicable Planner Types
Structured	Strong road geometry, with roads consisting of straight lines and arcs, sometimes with fixed paths	Regular	Limited dynamic behavior (straight ahead and lane change), no reversing behavior	Small planning scope, high replanning frequency, high real-time requirements	Frenet	Lateral-longitudinal decomposition	Lateral offset and longitudinal offset in Frenet frame	Linear collision-avoidance constraints Linear kinematic constraints Time-window constraints Spatially constrained conditions such as speed limits [7]	Path-velocity decomposition Lateral-longitudinal decomposition	Search-based methods Optimization-based methods Geometric methods Learning-based methods
Unstructured	No obvious road geometry, no fixed path.	Irregular	Variety of dynamic performance with reversing behaviors	Large planning scope, low replanning frequency, relatively low real-time requirements	Cartesian	Single-track bicycle model	Position, orientation, speed, wheel deflection angle, etc.	Nonconvex collision-avoidance constraints Nonlinear kinematic constraints Uphill and Downhill Engine Limiting Constraints [16], Up and Down Shoulders Tire Orientation Restraint [19], etc.	Path-velocity decomposition or non-decomposition	Search-based methods Optimization-based methods Learning-based methods

$$\begin{bmatrix} p_x(k+1) - p_x(k) = \Delta t \cdot v(k) \cdot cos\,\theta\,(k) \\ p_y(k+1) - p_y(k) = \Delta t \cdot v(k) \cdot sin\,\theta\,(k) \\ \theta(k+1) - \theta(k) = \Delta t \cdot v(k) \cdot \tan\varphi\,(k)/\mathrm{L_W} \end{bmatrix}, \quad (3)$$

where $\mathrm{L_W}$ denotes the ego vehicle's wheelbase.

Bounding constraints are imposed on the aforementioned state and control profiles at the discretized moments. Abstract bounding constraints $\boldsymbol{x}_{\min} \leq \boldsymbol{x}(k) \leq \boldsymbol{x}_{\max}$ and $\boldsymbol{u}_{\min} \leq \boldsymbol{u}(k) \leq \boldsymbol{u}_{\max}$ are presented as

$$\begin{bmatrix} p_{x\min} \\ p_{y\min} \\ v_{\min} \\ a_{\min} \\ \theta_{\min} \\ \varphi_{\min} \end{bmatrix} \leq \begin{bmatrix} p_x(k) \\ p_y(k) \\ v(k) \\ a(k) \\ \theta(k) \\ \varphi(k) \end{bmatrix} \leq \begin{bmatrix} p_{x\max} \\ p_{y\max} \\ v_{\max} \\ a_{\max} \\ \theta_{\max} \\ \varphi_{\max} \end{bmatrix}, k = 0,1,\dots,n. \quad (4)$$

Together with formulating the trajectory planning task in a discretized way, one needs to define a principle to approximate the time-continuous state/control profiles $\boldsymbol{x}(t)$ and $\boldsymbol{u}(t)$ based on the discretized collocations points $\boldsymbol{x}(k)$ and $\boldsymbol{u}(k)$. Typical approximation methods include orthogonal polynomials and geometric curves. Through this, the original OCP is converted into a mathematical programming problem, which aims to determine the finite collocation points. Variable values between adjacent collocation points can then be analytically derived via interpolation. It deserves to be noted that the usage of an approximation method renders underlying limitations to cover all of the possible trajectory candidates. A common practice to resolve this issue is to increase the density of collocation points, but that adds to the computational burden of the solution process. Constraint violations between two adjacent collocation

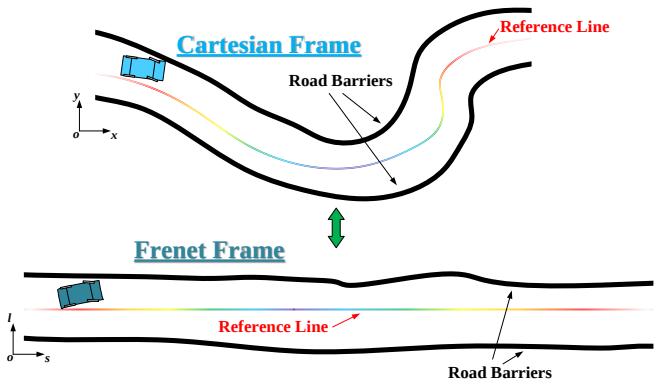


Fig. 3. Schematics on conversion between Cartesian and Frenet frames in autonomous driving trajectory/path planning.

points also deserve to be considered especially when the violations may cause serious troubles [93].

As opposed to the aforementioned discrete modeling strategy, the continuous modeling strategy is commonly used in trajectory planners of structured roads, where the planned trajectories are aligned to the road centerline. In such structured scenarios, quintic spline and Bézier curves are efficient in modeling vehicle kinematics. In unstructured scenarios, nonetheless, complex maneuvers such as combined forward/backward motions make the selection of approximation functions difficult. An improper selection of approximation functions would lead to a solution failure. Compared to the continuous modeling strategy, the discrete modeling strategy is straightforward and unified, thus we recommend applying the the discrete modeling strategy for describing trajectory planning tasks in unstructured scenarios, as summarized in Table III.

C. Differences between Trajectory Planning in Unstructured and Structured Scenarios

This subsection discusses the differences between unstructured- and structured-based trajectory planning problems. At this point, we realize that the choice of the coordinate system plays an important role in shaping the decision variables and constraints, thus we elaborate on the coordinate system selection problem first.

There are two coordinate systems widely employed, namely the Frenet frame and Cartesian frame [17]. The Frenet frame transforms a road into a straight tunnel with left and right boundaries. This transformation converts nonlinear collision-avoidance constraints into linear ones, thus making a planning problem simple in its formulation (Fig. 3). Additionally, the Frenet frame enables the division of coupled kinematic constraints into longitudinal and lateral dimensions independently. This indicates that the nominal trajectory planning problem is decoupled into two sequential subproblems with reduced complexity. However, the Frenet frame may inaccurately model the actual kinematics of the ego vehicle and distort the ego vehicle's shape. These problems are serious when the road curvature is overly high [18].

While the Cartesian frame may not simplify the expression of collision-avoidance constraints and kinematic constraints as the Frenet frame does, the Cartesian frame has the advantage of ensuring trajectory continuity and kinematic feasibility. In unstructured scenarios where trajectory curvature is usually high, kinematic feasibility is important, without which the closed-loop tracking controller easily fails. In this sense, the Cartesian frame is our preferred choice when planning trajectories in an unstructured environment.

The selection of a coordinate system has a significant impact on the representation of decision variables and constraints in trajectory planning. As shown in Table IV, decision variables are typically set as lateral offset and longitudinal offset in the Frenet frame. Conversely in unstructured scenarios described in the Cartesian frame, decision variables often consist of the vehicle's state profiles such as position, orientation, speed, and steering angle at each time step.

Compared to structured scenarios, unstructured scenarios exhibit stronger coupling effects between lateral and longitudinal dynamics, thus decoupling these two dimensions is difficult. In conclusion, considering the benefits of trajectory continuity and kinematic feasibility in unstructured environments, we recommend using the Cartesian coordinate for trajectory planning. Its ability to represent complex trajectories and handle the coupling effects between lateral and longitudinal dynamics makes it a suitable choice for tackling the challenges in unstructured scenarios.

III. Discrete Trajectory Planners for Unstructured Scenarios

This section presents the fundamental concepts of trajectory planning in unstructured scenarios. Inherently, trajectory planning is a search and optimization process, thus we utilize a generic optimization perspective to review all trajectory planning methods. We provide a detailed overview of their differences in decision variables, constraints, and solvers to establish the intrinsic connections among existing methods. The criteria for categorizing different methods are constraints and solvers. In a broad classification, we categorize existing planners into search-based, optimization-based, and learning-based methods.

The core idea of search-based methods is to start from the initial state and iteratively determine the state variables of the ego vehicle at each time step or search iteration until a predefined goal state is reached [20]. During each step of the search process, it is ensured that the transition from the previous state to the subsequent state is guaranteed to approximately satisfy specific constraint conditions [29].

An optimization-based approach is oriented toward directing the ego vehicle from its initial state to a goal state within a specified number of intervals or steps [25], [26], [27]. In this optimization problem, decision variables correspond to the state variables of the vehicle at these intervals or steps [26], [27]. Optimal values for these decision variables are determined using numerical optimization techniques, which ensures that the transition from a previous state to a subsequent state approximately satisfies the constraints [26], [27].

Learning-based approaches involve collecting a substantial dataset of empirically feasible solutions to diversified trajectory planning problems [36], [37]. This dataset is used to train a neural network for learning the inherent characteristics of these solutions [79]. The ultimate goal of the neural network is to comprehend the mapping relationship from specific inputs (e.g.,

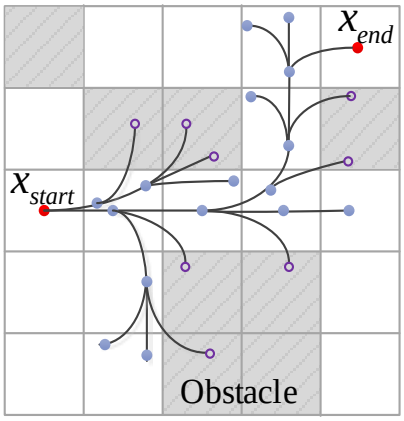


Fig. 4. Schematics on random sampling and search operations in CL-RRT.

initial state, environmental attributes, and target state) to the corresponding output (i.e., an optimal trajectory) [79]. Throughout this training process, the neural network becomes proficient at directly recalling a feasible or optimal trajectory for a given input, which is composed of vehicle states at n intervals or steps [92].

Each of these methods has its advantages and limitations, which are presented in Section III.A. Differences among the planners are introduced in Section III.B. In practical applications, multiple methods are usually combined to form what are known as two-stage methods. These two-stage approaches usually achieve better solutions for feasible and optimal trajectories by leveraging the strengths of different methods. The integration of different types of planners is introduced in Section III.C.

A. Basic Planner Categories

1) Search-based Methods with Primitives

Search-based methods with primitives rely on motion primitives to represent the state transfer process, which makes the solution space a discretized graph of grids and edges. This discretization reduces the runtime when search-based planners are used online. In simpler terms, the method actually requires the presence of $p_x(k+1) - p_x(k) = 0$ or $p_y(k+1) - p_y(k) = 0$ for each search process. Search-based methods reduce the complexity of searching the full state space by exploring trajectories within a finite set of motion primitives. The primary computational complexity lies in graph search in the graph generated by motion primitives. Commonly used techniques such as breath first search (BFS) and heuristic search enable search-based methods to find trajectories with optimal resolution in the grid space. In unstructured scenarios, search-based methods commonly utilize Reeds–Shepp curves or motion primitives as basic elements to build complex paths or trajectories. Two prevalent search-based methods are the hybrid A* search algorithm [29] and the state lattice planner [20].

Compared with the conventional 2-dim A* search algorithm [91], the hybrid A* search algorithm enhances the capability to satisfy vehicle kinematics, but the derived paths are still discontinuous in their curvatures. By contrast, the state lattice algorithm can generate paths with higher-order continuity, which results in better kinematic principles for the vehicle. Furthermore, state lattice-based planners typically run faster than the hybrid A* search algorithm, especially when an offline Hult table is deployed to accelerate the heuristic search process.

The hybrid A* search algorithm and the state lattice planner share a common limitation regarding the usage of motion primitives. Specifically, setting a satisfactory resolution for sampling the primitives *a priori* is difficult. If the primitives are sampled densely, then the search process runs slowly due to the curse of dimensionality. Conversely, if the primitives are sparse, then a search-based planner with such primitives will produce jerky paths or fail to find any feasible path.

2) Search-based Methods with Random Sampling

As opposed to the primitive-based searchers that explore an offline graph consisting of nodes and edges, a searcher based on random sampling has flexible capabilities in expressing path candidates. Typical planners of this type include probabilistic roadmap (PRM) and rapidly exploring random trees (RRT). These approaches progressively build an online space-filling graph by expanding nodes randomly. Searchers based on primitives are resolutionally complete, whereas those based on random sampling are probabilistically optimal.

Random sampling-based searchers can be divided into two types: state-space sampling, such as kinematics RRT* [21], [22], and control-space sampling, including closed-loop RRT (CL-RRT) [23]. Specifically, Fig. 4 shows the performance of CL-RRT in generating kinematically feasible paths by steering a virtual chassis based on control commands. Random sampling-based searchers have a common drawback: the paths they produce may be jerky. This randomness not only leads to unpredictable and unreliable outcomes but also complicates algorithm development and debugging.

Notably, some trajectory planners do not involve search operations; instead, they rely on sampling or a combination of sampling and sorting. In this paper, we refer to these planners as sampling-based methods. For instance, the Dynamic Window Approach (DWA) algorithm [88] determines a spatiotemporal and local space satisfying kinematic constraints based on the current position and velocity states. It evaluates multiple trajectories within the dynamic window and selects the trajectory with the highest score. This process iterates continuously before the ego vehicle reaches the destination. An adaptive pure pursuit path planner [89] samples path primitives by simulating the pure-pursuit tracking control process of a virtual chassis, which renders kinematically feasible paths.

3) Optimization-based Methods

Optimization-based trajectory planners adopt optimization theories to find high-quality trajectories, which are categorized into gradient-based and metaheuristic optimizers. Gradient-based optimizers utilize gradient information through methods such as Newton's method, the gradient descent method, and sufficient/necessary optimality conditions to determine iterative search directions and step lengths. Metaheuristic optimizers treat optimization problems as black boxes, which indicates that they do not use the gradient information inherent in the concerned optimization problems.

Gradient-based optimizers can handle optimization problems with complex constraints and run quickly to achieve local

TABLE V. TRAJECTORY PLANNING METHOD INTRODUCTION

Value Space		Method Type	Representative Methods	Optimization Framework			Advantages	Disadvantages
				Decision Variables	Constraints	Solution Strategy		
Function fitting space		Curve fitting	Quintic polynomials Bessel curves [9], [28]	Continuous coefficients of fitting functions	Collision-avoidance constraints Two-point boundary-value constraints Constraint handling approach: direct solution	Solve QP problem by interior-point method	Small computational scale Short runtime High smoothness in derived trajectories	Limitations on the vehicle's full maneuvering capabilities
Discrete value space of motion primitives	Polynomial curve	Search-based methods with primitives	2-dim A* search Hybrid A* search [29] State lattice [20]	Finite values for location and orientation angle	Collision-avoidance constraints (A*, D* algorithm); Collision-avoidance and curvature constraints (Hybrid A* search, state lattice); Constraint handling approach: substitution test	Utilize depth-first search, breadth-first search, and best-first search to find a near-optimal coarse path/trajectory Determine adjacent waypoints using fixed step-size gradient descent	Quick to find smooth paths in open space	Inflexible setting of kinematic primitives Challenging to pre-determine the resolution of motion primitives Involve continuous trial and error
	Dubins curve, Reeds-Shepp curve							
	State lattice							
Randomly explore the value space		Search-based methods with random sampling	PRM RRT Informed RRT* CL-RRT* [31]	Waypoints	Collision-avoidance constraints (RRT, PRM, etc.); Collision-avoidance and kinematic constraints (CL-RRT*); Constraint handling approach: substitution test or control space sampling	Determine adjacent waypoints using adaptive step-size gradient descent method	Short runtime Suitable for local or low-accuracy planning	Inflexible setting about the number of sampling iteration
Values of states and control variables	Numerical optimization	Single-point solution	APF [32], [33]	Waypoints	Collision-avoidance and road barrier constraints, with little consideration of kinematic constraints Constraint handling approach: penalty function method	Determine adjacent waypoints using fixed step-size gradient descent method	Short runtime Suitable for local planning	Susceptible to getting stuck in local minima Challenging to adjusting the coefficients of potential field methods
	Numerical optimization	Soft-constrained methods	TEB CHOMP STOMP	Continuous coefficients at finite collocation points that present state and control variables	Collision-avoidance and kinematic constraints Constraint handling approach: penalize collision-avoidance constraints to the cost function	Satisfy constraints as much as possible via penalty functions	Straightforward	Difficult to set the penalty function with numerous constraints

	Numerical optimization	Hard-constrained methods	Iterative LQR SLiFS H-OBCA	Continuous variables: state and control variables at uniform time intervals	Collision-avoidance and kinematic constraints; Constraint handling approach: represent collision-avoidance constraints to convex corridor or dual variables	Satisfy constraints strictly Use methods such as the interior-point method to solve the formed Quadratic Constraint (QC) or Quadratic Constraint Quadratic Programming (QCQP) problems (gradient descent)	Straightforward Precise satisfaction of constraints	Highly dependent on the initial value: Improper initial value leads to getting trapped in local minima and failing to find a feasible solution
	Intelligent optimization [34], [35]	Metaheuristic algorithms	SA [91] PSO [90] SFS [24]	Continuous/discrete variables: state and control variables at uniform time intervals	Collision-avoidance constraints; Constraint handling approach: substitution test	Iterative solution during the optimization process by introducing random perturbations	Less commonly applied due to the scarcity of complex logical constraints in trajectory planning	Challenging to control evolution speed Difficult to meet real-time requirements Adjust too many empirical parameters
Learn value space of other functions	Learning-based methods [36], [37]	Machine learning	Supervised learning	Continuous/discrete variables: state and control variables at uniform time intervals	Collision-avoidance and kinematic constraints; Constraint handling approach: associative memory or substitution test	Stochastic gradient descent Batch gradient descent	Able to find human-like trajectories	Expensive and time-consuming data collection Human-like trajectories may not be the best choice
			Reinforcement learning [38]	Continuous/discrete variables: state and control variables at uniform time intervals	Constraint handling approach: trial and error	Deep reinforcement learning that involves gradient descent	No reliance on traditional data labels	Low learning efficiency Challenging exploration of feasible solutions Non-inexplicability
			Parallel learning [39]	Continuous/discrete variables: state and control variables at uniform time intervals	Collision-avoidance and kinematic constraints; Constraint handling approach: associative memory or substitution test	Mutual coupling of data processing and action learning: 1. Fit the function space in data processing 2. Evaluate by gradient descent in action learning	Overcome the drawbacks of relying on labeled data and extensive interaction with the environment	Details that still require comprehensive theoretical proof

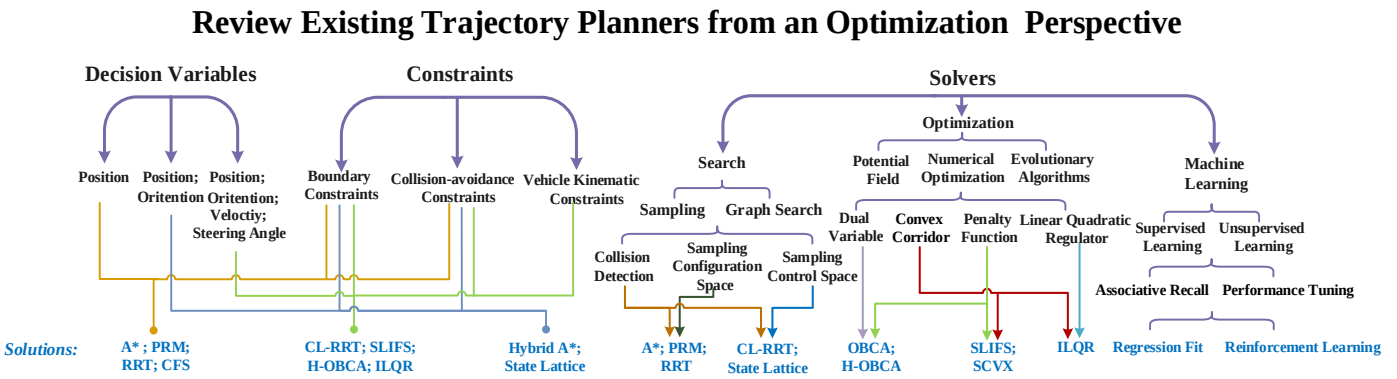


Fig. 5. Classification of prevalent trajectory planners by differences in decision variables, constraints, and solvers.

TABLE VI. PREVALENT TWO-STAGE TRAJECTORY PLANNERS FOR UNSTRUCTURED SCENARIOS

Method Full-name	Abbreviation	Stage 1 Method Type	Stage 2 Method Type
Hybrid A* search algorithm [40]	Hybrid A*	Search-based with primitives	Optimization-based
Improved path planning by tightly combining lattice-based path planner and optimal control [20]	Lattice OP	Search-based with primitives	Optimization-based
Successive linearization in feasible set [38]	SLiFS	Search-based with random sampling	Optimization-based
Hierarchical optimization-based collision avoidance [41]	H-OBCA	Search-based with primitives	Optimization-based
Elastic band-based rapidly exploring random tree [42]	EB-RRT	Search-based with random sampling	Optimization-based
Extended constrained iterative linear quadratic regulator [43]	EC-ILQR	Sampling-based	Optimization-based
Lightweight iterative optimization method [44]	LIOM	Search-based with primitives	Optimization-based
Timed elastic band [45]	TEB	Search-based with primitives	Optimization-based
Neural optimization [46]	Neural OP	Learning-based	Optimization-based

optimality. However, if the constraints are too complex (e.g., being highly non-convex), then gradient-based optimizers run slowly unless the concerned optimization problems have special structures in their formulations that can be utilized to define distinct solution structures. For example, the trajectory planning problem in unstructured scenarios is a non-convex optimization problem, but it can be reformulated into a sequence of convex optimization problems, which significantly enhances computational efficiency. By contrast, metaheuristic optimizers are not as proficient in handling complex constraints. Nevertheless, they can explore global optimality.

Metaheuristic algorithms have two main categories: trajectory-based methods like simulated annealing (SA) and swarm intelligence (SI)-based methods such as particle swarm optimization (PSO) [90] and stochastic fractal search (SFS) algorithm [24]. SA proceeds along a gradient descent trajectory, while an SI algorithm starts from an initial population and iteratively updates solutions. Trajectory-based methods are inefficient in handling large-scale problems. An SI algorithm is well suited for large-scale optimization problems with complex constraints. However, an SI algorithm requires manual settings of parameters and runs slowly to find a feasible solution when the feasible region in the concerned solution space is narrow. In conclusion, metaheuristics are rarely used in autonomous driving trajectory planning problems.

Gradient-based optimizers are classified into two types based on the way constraints are handled: 1) soft constraint methods that penalize constraint violations, and 2) hard constraint methods that request rigorous constraint satisfactions [25]. The artificial potential field (APF) method [26] represents a prototypical soft-constrained gradient-based optimizer. APF tackles trajectory planning by reformulating it into an unconstrained optimization problem through potential fields. These fields encompass repulsion fields from obstacles, repulsion fields from road boundaries, and gravitational fields toward the target destination. By harmonizing these potential fields, APF derives an integrated trajectory that optimizes the balance between reaching the destination and evading obstacles and boundaries [26]. However, soft constraint methods, such as APF, cannot uniformly represent intricate environment-related constraints and ensure their strict satisfaction. By contrast, hard-constrained gradient-based optimizers excel in handling diverse types of constraints uniformly while guaranteeing strict adherence to them [27].

4) Learning-based Methods

The aforementioned trajectory planners are commonly used online, but they are not truly real-time planners from a strict perspective. Learning-based methods offer a real-time solution to this issue because most computational burdens are addressed by their learning operations offline. In brief, a learning-based planner builds a mapping between the input and output of the trajectory planning scheme. The input typically includes integrated images of obstacles and the planning environment, whereas the output represents the state at various moments along the trajectory. Although the offline training process is time-consuming, the learned strategies would be extracted in real-time when a learning-based planner is used online [92].

Learning-based methods can be categorized into two: supervised learning and unsupervised learning. However, these methods suffer from the following limitations. First, human driving data may not always be optimal, which may affect learning performance. Second, preprocessing steps such as data collection and labeling can be time-consuming.

Reinforcement learning (RL) is another approach to automatically generate labeled data for training and explore the

TABLE VII. COMPARISON AMONG COMMONLY USED INITIAL GUESS GENERATION METHODS

Category	Method	Method Introduction	Advantages	Disadvantages	Computational Complexity
Search-based planners with primitives	A* search [29]	Attach a time-optimal velocity profile to a path derived by A* search algorithm	Short runtime	Low feasibility in vehicle kinematics, especially when backward maneuvers are enabled	$O(N_E + N_N \log(N_N))$
	Hybrid A* search [47]	Attach a time-optimal velocity profile to a path derived by hybrid A* search algorithm	High feasibility	Low optimality in vehicle kinematics (i.e., the derived paths are jerky) Long runtime when the environment is fulfilled by cluttered obstacles	$O(N_E + N_N \log(N_N))$
	Lattice OP [20]	Attach a time-optimal velocity profile to a path derived by state lattice planner	High feasibility	Low optimality in vehicle kinematics	$O(N_E + N_N \log(N_N))$
Learning-based planners	Supervised learning	Generate a spatio-temporal curve as an initial guess directly after offline training operations	Short runtime	Deep reliance on offline training performance, which means that an initial guess is poor if poorly trained due to overfitting and/or lack of training samples	$O(1)$
Optimization-based planners	Iterative optimization [18] [44]	Solve a series of optimization subproblems with increasing difficulties	Solution feasibility gradually improves Anytime solution	Long runtime if excessive iterations are needed Failures to solve intermediate subproblems block the whole solution process	$O({N_w}^3)$
	Operation only on critical variables [48]	Optimize critical variables along a coarsely planning 2-dim A* path and regard the derived trajectory as an initial guess	Short runtime	Low success rate to find an initial guess, especially when backward maneuvers are enabled	$O(N_w)$

optimal trajectory for different driving environments. However, RL requires excessive interactions with the environment for feedback training. The complexity of unstructured scenarios also adds to the challenges for RL. Extra strategies are often used alongside RL as remedies for safety risks to mitigate the aforementioned issues.

B. Differences among Various Basic Planner Categories

The aforementioned search-based, optimization-based, and learning-based planners differ from one another (Fig. 5). Despite this, they share a common action, that is, they all aim to solve an optimization problem while the differences lie in the way they describe the decision variables, constraints, and cost function in the optimization problem, as well as the way to design a solver.

The selection of decision variables is closely combined with the constraints in the concerned trajectory planning scheme. For example, in planning the trajectory for a holonomic AGV, setting the decision variables as the 2-dim locations is sufficient because a holonomic AGV is not subject to complicated vehicle kinematic constraints. Methods such as the conventional 2-dim A* search algorithm and basic RRT are planners suitable for a holonomic AGV. Conversely, state information in higher dimensions is needed when dealing with nonholonomic vehicles. This information includes variables such as yaw angle and steering angle, and constraints in this context extend to encompass nonholonomic kinematics. Approaches such as the hybrid A* search algorithm and CL-RRT fall into this category.

The value spaces of decision variables can be divided into two categories: the space of functions constructed using orthogonal functions, and the space of interpolated values formed by configuration points. This categorization is based on whether the planning time is treated discretely or continuously.

Optimization-based methods operate in a continuous decision variable space, whereas search-based methods work in discrete spaces. Learning-based methods fit the function space through data or action learning. The success of machine learning stems from its adeptness at pattern recognition, particularly on extensive, independent, and uniformly distributed datasets.

Two ways exist to deal with constraints. One is to check if a specific solution candidate causes constraint violations; the other is to solve an optimization embodied with the concerned constraints. Search-based methods, optimization-based methods with metaheuristics, and some learning-based methods [7] fall into the first category. Optimization-based methods with gradients fall into the second category.

Different methods relatively differ from each other in the way they present the decision variables and constraints. Thus, this work innovatively regards all the existing methods as ways to solve optimization problems via searching for descent directions. Accordingly, they are reviewed in a unified manner.

Optimization-based methods with gradients naturally satisfy the concept of descent search. Learning-based methods utilize techniques like batch gradient descent, stochastic gradient descent, and mini-batch gradient descent to iteratively optimize the weighting parameters in neural networks. In optimization theory, descent methods involve two primary steps: identifying the descent direction, and selecting the appropriate step size for the descent. The trajectory planning approaches with respect to search and potential fields satisfy these steps, given that each iterative progression entails the determination of direction and step length for the subsequent movement of trajectory points. APF is a special branch in the gradient-based optimization methods. In APF, the direction of each step is determined by simulating the synthetic effect of several types of forces.

Search-based methods decide each search step by guidance from the heuristic function under a best-first-search policy. They use motion primitives to make the solution space a discretized graph of grids and edges. These motion primitives

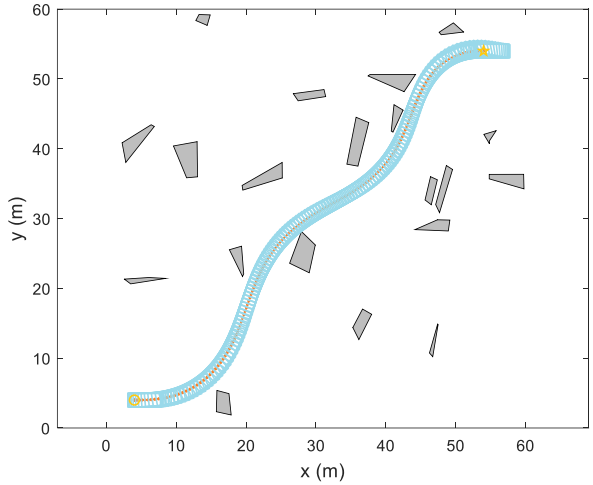


Fig. 6. Example of an unstructured simulation experiment scenario.

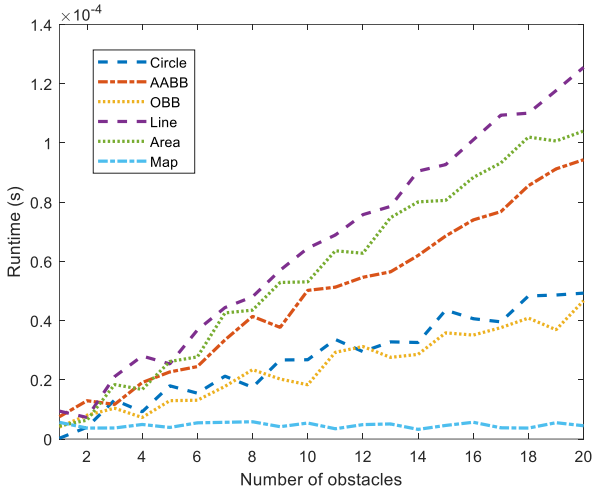


Fig. 7. Variation of time spent with number of obstacles for different collision checking methods (result statistics in C++ release mode).

represent the direction and step size for each respective search and can be viewed as an alternative form of descent direction and step size.

Table V summarizes the properties of the aforementioned planner types.

C. *Two-stage Trajectory Planning*

Empirically, different types of planners have to be integrated to mitigate their limitations and leverage their strengths. This work innovatively classifies the aforementioned methods into one-stage or two-stage methods to create a unified framework for them. One-stage methods refer to directly determining the trajectory in a single step, whereas two-stage methods first identify a rough path/trajectory and then refine it to enhance feasibility and optimality. Table VI lists the prevalent two-stage planners. The table indicates that optimization-based methods are often used in the second stage for path/trajectory refinement. In such methods, the first stage establishes an initial trajectory to serve as the starting point for the optimization in the second stage. Collision-avoidance constraints are the most complex part of a planning problem, which is highly non-convex. A high-quality initial guess facilitates the handling of collision-avoidance constraints. If a planner has clear steps to create a coarse initial guess before further refinement, then the method is regarded as a two-stage one. Conversely, some methods such as Convex Feasible Sets (CFS), Covariant Hamiltonian Optimization for Motion Planning (CHOMP), and Stochastic Trajectory Optimization for Motion Planning (STOMP) do not emphasize the importance of initial guess generation. Therefore, they are regarded as one-stage methods. One-stage methods rely on good initial guesses by default, without which the solution process would be difficult and slow.

Generating an initial guess typically includes two steps. The first step is to identify a feasible homotopy class, and the second step is to find a path/trajectory in the identified homotopy class. These steps are combined as one in some cases. The search-based planners with primitives and learning-based planners are commonly used to generate an initial guess. Other incremental strategies divide the overall challenge of finding an initial guess into multiple subproblems before conquering them in a sequence [18].

Table VII summarizes the prevalent methods to generate initial guesses. We define a trajectory planning case in Fig. 6 and report the comparative results in Table VIII to demonstrate the performances of the initial guess generators. Parametric settings and scenario setup are listed in Appendix B.

In the first stage, supervised learning boasts a nearly negligible time requirement, followed closely by the conventional 2-dim A* search algorithm. However, supervised learning may output an initial guess that intersects with obstacles, which makes the second stage difficult. The conventional 2-dim A* search algorithm generates a collision-free path as the initial guess but makes the second stage time-consuming in some cases because the initial guess generated in the first stage is kinematically infeasible.

Compared with the conventional 2-dim A* search algorithm, the hybrid A* search method [47] and the state lattice algorithm [20] can present vehicle kinematic principles more concretely. Thus, they can generate paths/trajectories with backward maneuvers. Such paths/trajectories are bounded in their curvatures. As a result, they are close to being kinematically feasible, although the curvatures may be discontinuous. A near-feasible initial guess can largely facilitate the optimization-based refinement operation in the second stage, but the first stage may spend more runtime if the hybrid A* search algorithm or state lattice algorithm is used instead of the conventional 2-dim A* search algorithm. A recent study [49] shows that the search operations in the first stage easily involve the curse of dimensionality in dealing with vehicles with higher dimensions of kinematic systems. The optimization-based refinement method in the second stage, if well developed, is not affected by the growth in the dimension of vehicle kinematics [48]. We also notice from simulations that, as opposed to iteratively solving a series of suboptimal problems to find an initial guess, only partial variables need to be optimized if they are regarded as critical in the first stage, which saves much runtime. The method that only optimizes critical variables can enhance the quality of an initial guess, reduce the runtime in the first stage, and further decrease the runtime needed for the second stage. In conclusion, we recommend adopting the method that only tackles critical variables for better performance in the first stage [48] when dealing with complicated unstructured scenarios.

TABLE VIII. COMPARISONS AMONG DIFFERENT STAGE COMBINATIONS (TESTED UNDER C++ RELEASE MODE)

Category	Planner	Runtime at Stage 1 (s)	Runtime at Stage 2 (s)	Gross runtime (s)
Search-based with primitives	A* search algorithm	0.008	0.789	0.797
	Hybrid A* search algorithm	0.871	0.176	1.047
	State lattice algorithm	0.924	0.162	1.086
Learning-based	Supervised learning tools	0.0004	0.218	0.218
Optimization-based	Iterative trajectory optimization	/	0.438	0.438
	Optimization of critical variables only	0.012	0.232	0.244

TABLE IX. COMPARISONS AMONG COMMON COLLISION CHECKING METHODS

Methods	Basic Principle	Complexity Analysis
Circle	Compare the sum of the radius of the circumscribed circles with the distance between their centers	$O(N_o)$
AABB	Check whether adjacent obstacles collide, build axis-aligned bounding boxes, iteratively sort the obstacle bounding boxes, and apply the separation axis theorem	$O(N_o)$
OBB	Check whether adjacent obstacles collide, build oriented bounding boxes, sort the obstacle bounding boxes, and apply the separation axis theorem	$O(N_o)$
Line	Check four edges of the rectangular vehicle and obstacle edges for intersections	$O(N_o)$
Area	Check if the ego vehicle's vertexes are in a convex obstacle via a triangle-area criterion and vice versa	$O(N_o)$
Map	Check if any of the grids covered by the vehicle is 1 based on a binary occupancy gridmap	$O(N_x \cdot N_y \cdot N_o)$

IV. CHALLENGES OF TRAJECTORY PLANNING IN UNSTRUCTURED SCENARIOS AND EXISTING SOLUTIONS

The two most complex elements in a trajectory planning problem are the collision-avoidance constraints and vehicle kinematic constraints. Trajectory planning inherently requires simultaneously satisfying both types of constraints while seeking among the feasible trajectory candidates for one that minimizes a predefined cost function. This section reviews the prevalent planners by how they handle the two types of constraints.

A. Collision-Avoidance Constraints

Trajectory planners are broadly categorized into two groups as per how they treat collision-avoidance constraints. The first category checks collisions along a given candidate trajectory. Typical examples of this category are search-based planners. Methods in the second category formulate an optimization problem with nominal collision-avoidance constraints incorporated. The rest of this subsection elaborates on the advantages and disadvantages of the two categories.

1) Deploying a Collision Checker

Search-based and learning-based methods commonly check collisions along candidate trajectory segments and discard those invalid ones. Given that unstructured scenarios are usually tiny, the shape of vehicle footprint should be appropriately modeled, otherwise collision risks may occur. Common checking models include external circle, axis-aligned bounding box (AABB), oriented bounding box (OBB), line detection, triangle area, and occupancy gridmap [50]. Table IX summarizes the features of these methods.

The AABB method [30] improves collision check accuracy by enclosing objects with axis-aligned boxes. However, the accuracy improvement is limited as the creation of the surrounding box needs to align with the coordinate axis. The OBB method [51] is an improved collision check approach that uses a rectangle aligned with the object's orientation, providing enhanced accuracy compared to the AABB method. In theory, the calculation amount of the OBB method will be slightly larger than the ABB method, due to factors like the consideration of vehicle orientation. While in algorithm implementation, the orientation can be efficiently incorporated using vector multiplication rather than computationally expensive trigonometric functions, resulting in a relatively small increase in the amount of computation. Line detection [52] determines collisions by checking if the four vertices of the vehicle rectangle are inside the obstacle and if the two diagonal lines intersect with the obstacle. Triangle area detection [18] calculates the triangle area to determine if the vehicle vertex is inside the obstacle. However, this method is unable to check cases where the ego vehicle crosses over an obstacle.

Suppose that there are N_O obstacles, the computational complexity for collision check using the aforementioned methods is constant for each obstacle. Therefore, the total computational complexity for all the above methods is $O(N_O)$. However, by dividing the planning space into slices during collision check, it is possible to eliminate the detection of distant obstacles and improve detection efficiency. It's important to note that although the computational complexity of these methods is $O(N_O)$, the actual running time may vary significantly due to some specific computational operations involved, such as area calculations.

Unlike the above detection method of $O(N_O)$, the occupancy gridmap checker [50] deploys a 0-1 occupancy gridmap, where 0 represents a collision-free grid and 1 represents a grid occupied by obstacles. In our concerned trajectory planning problem, the occupancy gridmap checker examines whether all the grids covered by the ego vehicle have any non-zero value. In this gridwise checker, the primary computational complexity arises from the initialization of the occupancy gridmap, which is related to the map scale, i.e., $O^o_{\text{build}} = O(N_x \cdot N_y \cdot N_o)$.

To visually compare the computation time differences among the collision check methods, simulation tests were conducted. As depicted in Fig. 6, the 60m×60m simulation scene is discretized into grids with a 0.5m×0.5m resolution. The number of obstacles in the scene ranges from 0 to 20. Each obstacle's position satisfies a uniform distribution within the

interval [0,60], and the area of each obstacle follows a uniform distribution within the interval [1,8]. The experiment was repeated 100 times to derive the results depicted in Fig. 7. From Fig. 7, it is evident that even for collision check methods with a complexity of $O(N_O)$, the line detection method takes the longest time, while the OBB method takes the shortest time. In contrast, a map-based collision checker with a resolution of 0.5m×0.5m gridmap does not cause a drastic increase in runtime with the obstacle number.

According to the complexity analysis O^o_{build}, we can observe that if the gridmap's resolution is 0.25m×0.25m, the map-based collision check time will be four times the existing time when there are 20 obstacles. After improving the resolution, the runtime consumed by the map-based collision checker will be approximately the same as the OBB-based collision check time. In unstructured scenarios with irregularly shaped obstacles or narrow passages, a higher-resolution gridmap is required to represent the free space. Under such conditions, the map-based checker may not have an advantage in terms of runtime. For more accurate results in these specific situations, we recommend using the OBB inspection method. However, in most general scenarios where a resolution of 0.5m×0.5m is sufficient to accurately represent the driving area, the map-based collision check method shows a small computation time at this resolution. Therefore, for our future experiments, we will utilize the map-based method for collision check with a resolution of 0.5m×0.5m.

2) Imposing Collision-avoidance Constraints

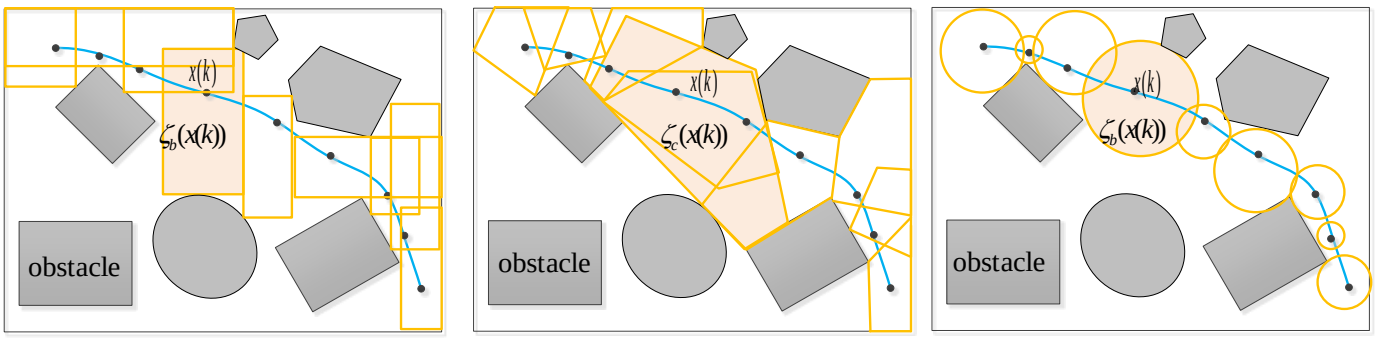
The optimization-based approach for handling collision-avoidance constraints involves incorporating these constraints into an optimization problem and solving it to obtain a safe trajectory that satisfies the safety-related demands. This approach can be further categorized into soft constraint processing, where the constraint is penalized in the cost function, and hard constraint processing, where the constraint is formulated as quadratic, mixed-integer, or linear.

The APF method and the Timed Elastic Band (TEB) method [45] can be considered soft constraint processing, as they ensure vehicle safety by penalizing the distance between the vehicle and obstacles. However, in unstructured scenarios complicated by irregularly placed obstacles, determining appropriate penalty function coefficients for soft constraint processing can be challenging. Improper choices may cause the solver to converge to infeasible local solutions.

The hard constraint processing methods can be further classified into direct analytical model processing, pairwise variable processing, and convex corridor processing. The direct analytical model processing method is commonly applied in 2-dim scenarios, where the vehicle and obstacles are represented by non-overlapping convex sets. Analytical formulas are derived to ensure that the convex sets representing the vehicle and obstacles do not intersect, thereby enforcing collision-free constraints between the vehicle and obstacles. For example, [53] represents vehicles and obstacles using multiple triangles and directly checks for intersections between these triangles. Similarly, [54] characterizes vehicles and obstacles as multiple circles, and directly calculates the distance between the centers of the circles to determine if there is any intersection. However, the direct analytical processing model approach introduces three major drawbacks in the final optimization problem: non-convex and non-differentiable nature, large problem size, and heavy reliance on the initial feasible trajectory solution.

The pairwise variables treatment involves treating both the vehicle and the obstacle as convex polygons. The method then applies the convex set separation theorem to formulate the original convex optimization problem, ensuring no overlap between the vehicle and obstacles. Subsequently, the problem is transformed into a pairwise problem based on the principles of convex optimization, enabling further computation. This method has been in existence for a considerable time and is discussed in detail in a book entitled "Convex Optimization", which was published in 2004 [55]. In 2018, this idea was revisited with a new name Optimization-Based Collision Avoidance (OBCA) algorithm [41], [47], [56] but the core techniques in OBCA were actually well proposed in 2004. The OBCA method indeed provides an accurate representation of collision-avoidance constraints. However, the introduction of a large number of dyadic variables, which grows linearly with the number of obstacles, significantly increases the dimensionality of the optimization problem. As a consequence, the solution process is slowed down, especially in scenarios with a higher number of obstacles.

Convex corridor methods depict the travelable area between obstacles to avoid collisions and ensure the vehicle trajectory stays within the travelable area using hard constraints or soft penalty functions in optimization. To simplify computations, this method characterizes the travelable area around each vehicle state as a specific convex set. However, this approximation may often lead to a smaller travelable area than the actual limit. Fig. 8 shows that, compared to OBCA methods, convex corridor methods do not introduce additional pairwise variables, thus can reduce the number of decision variables for the optimization problem.



(a) Box/Corridor (b) CFS (c) Bubble

Fig. 8. Schematics on layout of convex corridors.

TABLE X. COMPARISONS OF COMMON METHODS FOR HANDLING COLLISION-AVOIDANCE CONSTRAINTS

Method	Corridor Shape	Advantages	Disadvantages	Complexity	Type of Collision-Avoidance Constraints
Box/Corridor	Rectangle	Linear constraints simplify the optimization problem Computational complexity is not affected by obstacle number	Rely on a reference trajectory	$O^o_{\text{build}}+O(\lceil r/r_\varepsilon \rceil)$ or $O^o_{\text{build}}+O(\log(\lceil r/r_\varepsilon \rceil))$	Linear
CFS	Polygon	Linear constraints simplify the optimization problem Independent of the initial guess	Computational complexity is related to obstacle scale	$O(N_o)$	Linear
Circle	Round	Intuitive and convenient formation of convex feasible solution	Quadratic constraints are more difficult than linear ones	$O(N_o \log(\lceil r/r_\varepsilon \rceil))$	Quadratic

TABLE XI. COMPARISON OF SOLUTION RUNTIMES OF DIFFERENT METHODS IN HANDLING COLLISION-AVOIDANCE CONSTRAINTS

Method	Runtime to Form a Convex Corridor (s)	Solution Runtime per Iteration (s)	Total Number of Iterations	Total Runtime (s)
Box	0.0014	0.026	8	0.24
CFS	0.0016	0.025	7	0.17
Bubble	0.006	0.101	9	1.18

Typical convex corridor methods are Box [44], Corridor [57]–[59], CFS [40], and Bubble [60]. Box, Corridor, and CFS form linear convex corridor collision-avoidance constraints, while Bubble forms quadratic constraints.

The Box and Corridor methods construct the convex corridor by continuously expanding the length and width of the rectangle until they encounter obstacles within occupied grids in the occupancy gridmap. The building complexity of an occupancy gridmap complexity is O^o_{build}. In the extended region, Box uses an iterative search with the complexity of $O^o_{\text{build}} + O(\lceil r/r_\varepsilon \rceil)$ and Corridor uses a binary search with the complexity of O^o_{build}+O($\log(\lceil r/r_\varepsilon \rceil)$), where r denotes the expected rectangle length and r_ε denotes the grid resolution. Both methods form a well-arranged and orderly rectangle convex corridor that is easy to compute. This results in a linear constraint with left and right boundaries, simplifying the subsequent optimization solution. CFS [40] analyzes the geometrical relationship between the initial trajectory points and the points and edges of obstacles, forming a convex corridor that covers a large area. CFS has low computational complexity, which is related to the number of obstacles and can be expressed as $O(N_w N_o\, log\, N_o) \cdot O_{Tri}$. Bubble [60] extends the circle radii dichotomously until it intersects with an obstacle. Its computational complexity is related to the number of obstacles and is expressed as $O(N_o \log(\lceil r/r_\varepsilon \rceil))$. The constraints formed by bubbles are quadratic, which complicate the subsequent optimization solution process. The convex safe feasible set methods mentioned earlier primarily focus on ensuring collision avoidance at discrete trajectory points. To ensure safety between collocation points, two main approaches are commonly used: the geometric expansion method [61] and the trajectory point density increase method [52].

Geometric expansion methods [61] ensure safety between discrete points by expanding the vehicle body based on geometric numerical relationships. However, this method only guarantees collision avoidance with static obstacles, and further study is needed to strictly guarantee collision avoidance with dynamic obstacles. The trajectory point density increase method ensures safety between discrete points by making the discrete trajectory as close as possible to the continuous trajectory. However, too many discrete points can increase the size of the optimization problem, making it challenging to meet real-time requirements. The appropriate number of trajectory points can be determined based on the accessibility analysis of the vehicle model, but heuristic selection is often used to minimize computation time. Typically, the interval between trajectory points is approximately equal to 1/4 of the vehicle length [60].

Table X summarizes the advantages, disadvantages, and complexity of common methods that can deal with collision-avoidance constraints. We performed simulation experiments to visually compare the formation times of different convex safe feasible sets and studied their influence on the overall optimization problem solution. The parameter settings and details of the simulation experiments can be found in Appendix B.

Table XI shows that the optimization process is faster when the formed convex corridor is linearly constrained. For the Box and CFS methods that yield linear collision-avoidance constraints, the average runtime spent on corridor generation and constraint handling in each iteration is quite close. However, CFS yields a larger coverage area, resulting in fewer total solution iterations and less total solution time. Therefore, we recommend using CFS to handle collision-avoidance constraints.

However, if an initial guess derived at the first stage is not close to being feasible, applying the convex corridor constraint based on such a low-quality initial guess easily leads to underlying violations of vehicle kinematic constraints, thus leading to a solution failure in the second stage. One empirical remedy for this issue is to enlarge the convex corridor area to cover more free spaces.

B. Vehicle Kinematic Constraints

When dealing with kinematic constraints, different planning methods can be broadly categorized into two approaches: methods sample kinematically feasible primitives and methods incorporate kinematic constraints into the optimization problem. In the former, represented by search methods, search nodes are

TABLE XII. COMPARISONS AMONG COMMON MODELING METHODS FOR NONLINEAR KINEMATIC CONSTRAINTS

Method	Brief principle	Advantages	Disadvantages	Time complexity
Sequential Programming (SP)	Penalize the nonlinear kinematic constraints to the cost function	Few parameters to be adjusted	Non-convex optimization with low computation speed	$O({N_w}^3)$
L1 Sequential Convex Programming (SCP)	Linearize the nonlinear kinematic constraint Penalize cost function with the L1-norm	High computation speed	Many parameters to be adjusted	$O(N_w)$
L2 SCP	Linearize the nonlinear kinematic constraint Penalize cost function with the L2-norm	High computation speed	Many parameters to be adjusted	$O({N_w}^3)$

TABLE XIII. COMPARISONS OF COMMON METHODS FOR LOW-SPEED UNSTRUCTURED SCENARIOS

Planner Type	Representative Methods	Scene 1: An open-space cluttered by obstacles			Scene 2: Parallel parking scenario		
		Time (s)	Track Costs	Tracking Error	Time (s)	Track Costs	Tracking Error
Search-based planners with primitives	Hybrid A* search	0.899	59.18	9.14	0.014	32.15	4.32
	State lattice	0.982	60.38	8.28	0.012	28.14	2.89
Search-based planners with random sampling	Kinematic RRT*	0.778	89.42	11.14	0.008	36.12	7.93
	CL-RRT*	0.892	83.41	14.15	0.011	38.13	6.89
Optimization-based	H-OBCA	0.648	51.19	2.56	0.476	24.16	1.32
	SLiFS	0.278	52.18	2.42	0.124	23.12	1.48
Spline-based	Bezier	0.124	56.78	2.78	/	/	/

extended by kinematic checking or pre-generating trajectory segments that satisfy the kinematics. In the latter, represented by optimization methods, trajectories that satisfy kinematic constraints are generated directly. Sections IV.B.1 and IV.B.2 will introduce commonly used methods in these two categories and their respective advantages and disadvantages.

1) Sampling Kinematically Feasible Trajectory Primitives

Search-based and learning-based methods are capable of directly generating trajectory primitives that satisfy kinematic constraints. These methods can be further divided into two categories: one is to back-propagate the configuration space detection while the other is to simulate in the control space. The former, represented by the hybrid A* search algorithm and the state lattice algorithm, detects whether adjacent trajectory segments, generated based on kinematic primitives, satisfy kinematic constraints during the search process. The latter, represented by CL-RRT, generates trajectory segments that directly fulfill kinematic constraints while expanding nodes in the control space. Learning-based methods relying on data labels can be seen as generating trajectories that satisfy the kinematic constraints through direct association.

2) Imposing Kinematic Constraints

When dealing with kinematic constraints, optimization-based methods fall into two main categories: linear quadratic regulator methods (LQR) [62] and penalty function methods [63]. The linear-quadratic regulator method transforms vehicle kinematic constraints into a linear-quadratic OCP by linearizing them. While the LQR method makes it easy to form a closed-loop optimal control and is computationally simple. However, in unstructured environments, the highly nonlinear vehicle kinematics easily lead to linearization errors, further yielding kinematically infeasible paths.

The penalty function method penalizes the nonlinear kinematic equation constraints by incorporating them into the cost function. This method includes the parametric penalty method and the augmented Lagrangian function method. The parametric penalties include the L1 parametric penalty and the L2 parametric penalty (also known as the quadratic penalty function), i.e., $\min_x J(\boldsymbol{x}) + \|f(\boldsymbol{x})\|_p$, $p = 1$ or 2 [64]. Some methods may not explicitly handle nonlinear kinematic equality constraints separately and explicitly. Instead, they leverage optimization solvers that internally handle these constraints using function penalty techniques.

To further accelerate optimization efficiency, linearizing the nonlinear equation within a trust region followed by Parametric or Augmented Lagrangian Penalty has gained more attention in recent years. By doing so, specialized optimizers designed for these problem types, like cplexqp [65], can be utilized, greatly accelerating the optimization process. The cost function after each linearization of the vehicle kinematic constraints can be written as $\min_x J(\boldsymbol{x}) + \left\|f(\boldsymbol{x}^{(h)}) + \nabla f(x^{(h)})(\boldsymbol{x} - \boldsymbol{x}^{(h)})\right\|_p$, where $\boldsymbol{x}^{(h)}$ denotes the solution obtained in the previous iteration.

Table XII summarizes the advantages, disadvantages, and computational complexity of commonly used methods for processing nonlinear kinematic constraints. Compared to the L2 norm penalty function, the L1 norm penalty function maintains the block-like separation characteristics in forming the cost function, which results in a Hessian matrix that remains strip sparse. This sparsity property allows for more efficient computations and reduces the computational complexity from $O({N_w}^3)$ to $O(N_w)$.

Simulation experiments in [64] demonstrate the superiority of using the L1 paradigm. Based on this, we recommend using the L1 parametrization for nonlinear kinematic constraints and adopting the L1 norm penalty function in the two-stage canonical approach in subsequent simulation experiments.

C. Summary

To compare the calculation times of the various methods, we also compared all the methods in the parallel parking scenario and the reverse parking scenario, in addition to testing in an obstacle-cluttered scenario shown in Fig. 6. Since trajectories

planned by methods such as the hybrid A* search algorithm do not satisfy G^2 continuity, we use cumulative tracking control error of the trajectories as an indicator to quantitatively portray the impact of G^2 continuity. Fig. 9 builds a simulation scene in CarSim, which generates testing data as listed in Table XIII. Readers may consult Appendix B for the simulation setup information.

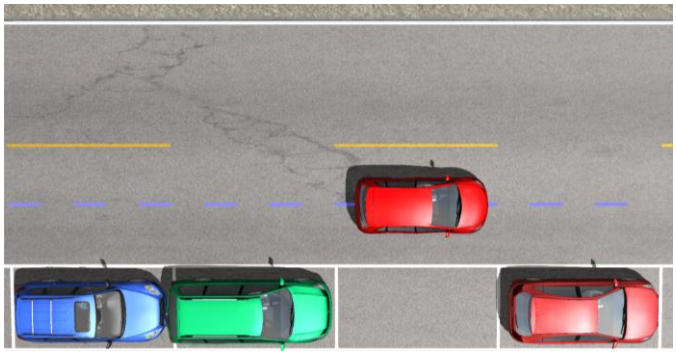


Fig. 9. Typical parallel parking scenes in CarSim.

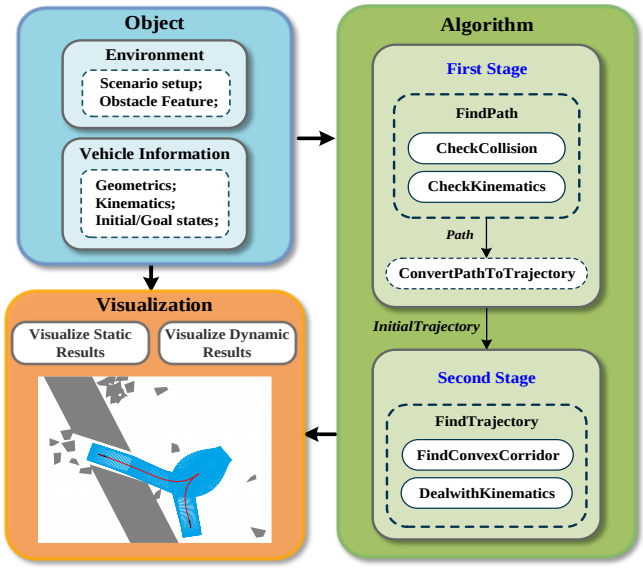


Fig. 10. Three modules of the trajectory planning method library: object, algorithm and visualization.

The simulation results demonstrate that the Successive Linearization in Feasible Set (SLiFS) method performs well in both scenarios in terms of solution time, trajectory quality, and traceability. The running time of other methods such as the hybrid A* search algorithm, state lattice algorithm, kinematic RRT*, and CL-RRT* is largely influenced by the complexity of the environment, particularly the obstacle scale. As the number of obstacles increases, these methods may encounter challenges in the runtime spent to find a solution.

Although the hybrid A* search algorithm and kinematic RRT* can find trajectories in a very short time in Scenario 2, the trajectories they find do not fully satisfy the vehicle kinematics and are rather jerky, which indicates that the planned trajectories are low-quality. If one planner does not satisfy the G^2 continuity, then the paths/trajectories it planned are hard to be tracked by a low-level controller. While the state lattice planner and CL-RRT* can find trajectories that satisfy vehicle kinematics, their performance can be affected by the complexity of the environment, particularly when there are numerous obstacles. In such scenarios, these methods may take a long time to search for a feasible trajectory. Additionally, there are still many detours in the trajectory, leading to an increase in trajectory length. The Bezier method can quickly find feasible smooth trajectories in Scenario 1, and because it only needs to optimize the coefficients of the Bezier curve, the number of decision variables and the scale of the problem are much smaller than SLiFS. However, in scenario 2, the Bezier method cannot portray trajectories with discontinuous curvature, making it unable to find parking trajectories with cusps.

As a summary of this section, we recommend employing a two-stage algorithm in trajectory planning for an unstructured scenario. In the first stage, the initial guess should be obtained by optimizing critical variables [48]. In the second stage, the CFS method can be used to address collision-avoidance constraints, and the L1 norm penalty function can be utilized to handle nonlinear kinematic constraints [64].

V. Library and Benchmarks

To advance autonomous driving trajectory planning, we have established the open-source trajectory planner library and scenario library: Libraries and Benchmarks for Autonomous Driving Trajectory Planning (LBADTP). The library includes a collection of methods and test scenarios, which are available at https://github.com/gyq18/AutonomousDrivingTrajectoryPlanning.

A. Trajectory Planner Library

1) Introduction to Prevalent Planner Libraries

Since the 1940s, libraries of trajectory planners have emerged with the development of trajectory planning techniques. Table XIV lists the current open-source libraries of mainstream planning methods.

Among them, the Search-based Planning Library (SBPL) [66], developed in collaboration between Maxim Likhachev from the University of Pennsylvania and Willow Garage, implements a set of general-purpose motion planners using search-based planning techniques. The Open Motion Planning Library (OMPL) [67], developed by the Kavraki lab at Rice University, focuses on search-based planners with primitives and search-based planners with random sampling. CHOMP [63], jointly developed by Google, Carnegie Mellon University, and the University of Pennsylvania, is a gradient-based trajectory optimization program. CHOMP improves the smoothness of generated trajectories through an optimization process. In practical usage, CHOMP often serves as a post-processing optimization technique for OMPL to further refine the results of motion planning. STOMP [68] is a trajectory optimization method based on Policy Improvement with Path integral. Unlike CHOMP, STOMP does not require gradient information, allowing it to optimize arbitrary cost functions.

The four libraries, namely SBPL, OMPL, CHOMP, and STOMP, are primarily designed for planning in the context of robotic arms, rather than specifically tailored for autonomous

TABLE XVI. SCENE PARAMETRIC SETTINGS

Parameter	Description	Settings
S_x, S_y	Scale of trajectory planning sceme (m)	60, 60
R_x, R_y	Resolution of occupancy gridmap (m)	0.5, 0.5
N_x, N_y	Number of discretized grids in occupancy gridmap	120, 120

TABLE XVII. EGO-VEHICLE GEOMETRIC PARAMETRIC SETTINGS

Parameter	Description	Settings (m)
L_L	Vehicle length	4.689
L_B	Vehicle width	1.942
L_W	Vehicle wheelbase	2.800
L_F	Vehicle front hang	0.960
L_R	Vehicle rear hang	0.929

TABLE XVIII. EGO-VEHICLE KINEMATIC PARAMETRIC SETTINGS

Parameter	Description	Settings
$v_{\max}, v_{\min}$	Upper and lower bounds of $v(t)$ (m/s)	2.5, -2.5
$a_{\max}, a_{\min}$	Upper and lower bounds of $a(t)$ (m/s^2)	1, -1
$\phi_{\max}, \phi_{\min}$	Upper and lower bounds of $\phi(t)$ (rad)	0.75, -0.75
$\omega_{\max}, \omega_{\min}$	Upper and lower bounds of $\omega(t)$ (rad/s)	0.5, -0.5
$r_{\min}$	Minimum turning radius of the ego vehicle, i.e., $L_W/\tan(\phi_{\max})$ (m)	3.0056

driving trajectory planning.

Furthermore, these four libraries do not come with built-in modules for collision detection, kinematic checks, or result visualization. These libraries require users to instantiate and utilize these modules themselves to examine the generated trajectory segments.

In practice, these libraries are often used in conjunction with MoveIt [69]. MoveIt is a software designed for manipulation tasks, providing external interfaces for the planners SBPL, OMPL, CHOMP, and STOMP to perform collision detection and kinematic checks.

PythonRobotics, developed by Atsushi Sakai, includes various methods such as search-based, sampling-based, spline-based, and potential field-based methods, making it more comprehensive than the aforementioned SBPL, OMPL, CHOMP, and STOMP. Unlike the aforementioned libraries, which primarily target robotic arms, PythonRobotics has a broader scope and can handle holonomic AGVs, vehicles, and other objects. However, PythonRobotics is more oriented towards educational purposes. It lacks a well-designed algorithm interface, making it less adaptable to various scenarios. For instance, significant modifications to the entire algorithm are required when the shape of obstacles in the environment changes.

Apollo developed by Baidu Inc. and CommonRoad developed by the Technical University of Munich have effectively addressed the modularity and integration of planning libraries. However, they include a relatively limited number of algorithms. For instance, Apollo primarily employs lattice-based search, EM algorithm, and Hierarchical OBCA (H-OBCA) algorithm as its planning algorithms, while

TABLE XIV. COMPARISONS AMONG TRAJECTORY PLANNING LIBRARIES

Library	Coding Language	Content	Embedded Functions	Applications
SBPL	MATLAB C++	Search-based planners with primitives	Collision check × Kinematic feasibility check × Visualization ×	Robotic arm Intelligent vehicles
OMPL	Python C++	Search-based planners with primitives, search-based planners with random sampling	Collision check √ Kinematic feasibility check × Visualization ×	Robotic arm
CHOMP	C++	Covariant Hamiltonian optimization method for motion planning	Collision check × Kinematic feasibility check × Visualization ×	Robotic arm
STOMP	C++	Stochastic trajectory optimization method for motion planning	Collision check × Kinematic feasibility check × Visualization ×	Robotic arm
Trajopt	Python C++	Sequential convex optimization method	Collision check √ Kinematic feasibility check √ Visualization √	Robotic arm
PythonRobotics	Python MATLAB	Search-based planners with primitives, search-based planners with random sampling, sampling-based planners, potential field-based planners, spline-based planners	Collision check √ Kinematic feasibility check × Visualization √	Robotic arms Holonomic wheeled robots Intelligent vehicles
Autoware	C++	Search-based planners with primitives	Collision check √ Kinematic feasibility check √ Visualization √	Intelligent vehicles driving in structured (major) and unstructured (minor) environments
Apollo	C++	Search-based planners with primitives, search-based planners with random sampling, sampling-based planners, optimization-based planners, learning-based planners	Collision check √ Kinematic feasibility check √ Visualization √	Intelligent vehicles driving in structured (major) and unstructured (minor) environments
CommonRoad	Python	Search-based planners with primitives, Interfaces to learning-based planners	Collision check √ Kinematic feasibility check √ Visualization √	On-road intelligent vehicles
This work	MATLAB Python C++	Search-based planners with primitives, search-based planners with random sampling, sampling-based planners, optimization-based planners, learning-based planners	Collision check √ Kinematic feasibility check √ Visualization √	Intelligent vehicles driving in unstructured environments

TABLE XV. SYMBOL DEFINITIONS AND PARAMETRIC SETTINGS

Symbol	Description	Settings	
S_x, S_y	Scale for the planning range (m)	60, 60	
R_x, R_y	Resolution of an occupancy gridmap (m)	0.5, 0.5	
N_x, N_y	Number of x-y grids in an occupancy gridmap. $N_x = S_x/R_x, N_y = S_y/R_y$	120, 120	
N_d	Number of expansion trials in node expansion procedure	2-dim A* search	8
		Hybrid A* search	14
		State lattice	8
		Complex lattice	8
N_a	Number of actions for each direction in the searching graph	2-dim A* search	1
		Hybrid A* search	2
		State lattice	8
		Complex lattice	10
N_N	Number of nodes in the searching graph. $N_N = N_x \cdot N_y \cdot N_d$	e.g., $N_x = 120, N_y = 120, N_d = 8$	
N_E	Number of edges in the searching graph. $N_E = N_x \cdot N_y \cdot N_d \cdot N_a$	e.g., $N_x = 120, N_y = 120, N_d = 8, N_a = 8$	
N_o	Number of obstacles	-	
N_w	Number of all waypoints. $N_w \propto N_x$	-	
N_{iq}	Number of inequalities. $N_{iq} \propto N_o \cdot N_w$	-	
N_s	Number of samples/iterations for search-based planners with random sampling	-	
O_{coll}	Complexity for collision checking	-	
O_{Trigo}	Complexity for trigonometric computation	-	
O^o_{build}	Complexity for building an occupancy gridmap	$O^o_{\text{build}} = O(N_x \cdot N_y \cdot N_o)$	
O^g_{build}	Complexity for establishing a graph	$O^g_{\text{build}} = O(N_N) = O(N_x \cdot N_y \cdot N_d)$	
O_{search}	Complexity for searching in an established graph	$O_{\text{search}} = O(N_E + N_N \log(N_N))$	

CommonRoad mainly focuses on primitives and interfaces to learning-based planners. CommonRoad functions more like an integration platform, providing algorithm interfaces for other planning algorithms, including optimization-based and learning-based planning methods, but it has fewer implemented algorithms of its own. Additionally, CommonRoad is primarily designed for structured road environments and is inefficient for unstructured scenarios.

In conclusion, the current method libraries for trajectory planning suffer from the following issues:

1. Limited comprehensiveness: The implemented trajectory planning methods in the current libraries lack inclusiveness, as they do not include popular numerical optimization-based methods and fail to provide adequate modules and interfaces for two-stage planning methods.
2. Orientation towards robotic arms and holonomic AGVs: The existing libraries mainly target robotic arms, holonomic AGVs, etc., without fully considering the characteristics of intelligent vehicles.
3. Limited modularity and integration: The existing libraries lack comprehensive modularity and integration.
4. Poor reproducibility and improvement: Some libraries have undisclosed or incomplete source codes, hindering reproducibility and improvement, thereby limiting their potential to drive technological advancements in the field of trajectory planning.

To address the limitations of the existing libraries, we propose a trajectory planning method library specifically designed for autonomous driving. The library is implemented in three programming languages: MATLAB, Python, and C++. The core design principle of this library framework is modularity. Modularity facilitates the decoupling of different functionalities, enhancing the library's scalability and extensibility. New models and algorithms can be inherited from existing modules, allowing for easy deployment without significant modifications to the core architecture of the library.

2) Introduction of the Proposed Method Library Module

To increase the scalability of the method library, we have developed a modular trajectory planning method library. Based on the overall process of trajectory planning, the library is divided into object modules, algorithm modules, and visualization modules, as shown in Fig. 10.

The object module is responsible for providing inputs to the planning algorithm and contains two sub-modules: the environment module and the ego vehicle module.

The environment module includes sub-modules for scene and obstacle settings. The scene setup module contains information such as scene size and resolution, while the obstacle setting module contains information such as the number, position, and shape of obstacles.

The ego vehicle module includes the physical and kinematic information of the ego vehicle, such as its shape and kinematic constraints. Additionally, it defines the status of the ego vehicle at the starting and ending points. These pieces of information together form the task of trajectory planning and serve as inputs to subsequent trajectory planning algorithms.

In the object module, we support the direct modification of parameters from the submodule algorithm to design the scenario. Within the obstacle setting module, a random obstacle generation procedure is provided. Additionally, we offer an external data import interface in the object module, allowing the export of trajectory planning tasks containing scene settings, obstacle settings, vehicle start/stop status, and other relevant information by reading ".csv" files. Appendix C provides details on the specific ".csv" file format.

The algorithm module takes the trajectory planning tasks generated by the object module as input and produces trajectories as output. The algorithm module is divided into a first-stage module and a second-stage module.

```
Matlab/
    ├ Main.m
    ├ Planners/
        ├ Graphbased/
            ├ PlanAStarPath.m
            ├ PlanHybridAStarPath.m
            ├ PlanHybridAStarPath.m
            ├ PlanSimpleStateLatticePath.m
            ├ PlanControlLatticeTrajectory.m
            └ ...
        ├ Samplingbased/
            ├ PlanInformedRRTPath.m
            ├ PlanKinodynamicRRTTrajectory.m
            ├ PlanCLRRTTrajectory.m
            └ ...
        ├ Optimizationbased/
            ├ PlanOBCATrajectory.m
            ├ PlanSCPTrajectory.m
            ├ PlanL1SCPTrajectory.m
            ├ PlanL2SCPTrajectory.m
                ├convex corridors for collision constraints/
                    ├FindCFS.m
                    ├FindBox.m
                    ├FindBubble.m
        └ ...
    ├ CheckCollision/
        ├ CheckByCircle.m
        ├ CheckByAABB.m
        ├ CheckByOBB.m
        └ CheckByArea.m
    ├ ConvertPathtToTrajectory/
        ├ PlanSpeedForStaicScenarios.m
        ├ PlanSpeedForDynamicScenarios.m
    └ ...
```

Fig. 10. Source codes of planning methods ans relavent functions available in our library.

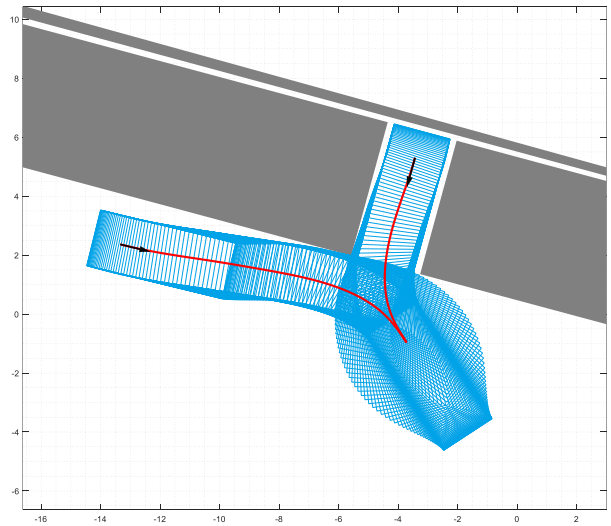


Fig. 11. A typical parking trajectory planning scenario from TPCAP 2022.

The first-stage module primarily consists of search-based methods with primitives, search-based methods with random sampling, optimization-based methods, and learning-based methods, which are employed to generate initial trajectories. The second-stage module focuses on optimization-based methods, utilizing the initial trajectory obtained from the first stage as an initial guess for the optimization process to further refine and optimize the generated trajectory.

Since the primary challenges in autonomous driving trajectory planning lie in handling collision avoidance constraints and kinematic constraints, and there are various methods for processing them, we have introduced the CheckCollision submodule and CheckKinematics submodule in the first stage, as well as the FindConvexCorridor submodule and DealwithKinematics submodule in the second stage. This allows for a convenient combination of different collision avoidance and kinematic constraint handling methods.

Since the planning algorithm in the first stage needs to verify whether generated trajectory segments satisfy collision avoidance and kinematic constraints, we use the generated trajectory segments as input to the CheckCollision and CheckKinematics sub-modules to verify their feasibility.

The CheckCollision sub-module implements various collision detection methods, including the AABB detection method, the OBB detection method, the collision detection method based on circle approximation, the collision detection method based on triangle area, and the collision detection method based on an occupancy gridmap. The CheckKinematics sub-module implements vehicle kinematic detection based on bicycle models. Other vehicle kinematic models will be added to the module in the future.

In the second stage, the FindConvexCorridor submodule generates convex feasible sets of different shapes based on the previous iteration trajectory solution, scene information, and obstacle distribution, according to the selected options. The submodule mainly includes four options: Box, Corridor, Bubble, and CFS, for finding convex feasible sets. When Box and Corridor are chosen, the shape of convex feasible sets is a box, and the FindConvexCorridor submodule outputs box constraints associated with each trajectory point position. When CFS is chosen, the shape of convex feasible sets is a polygon and the FindConvexCorridor submodule outputs linear inequality constraints related to each trajectory point position. Lastly, when bubbles are determined, the shape of convex feasible sets is a bubble and the FindConvexCorridor submodule outputs quadratic constraints related to each trajectory point position.

In the DealwithKinematics submodule, various methods such as SCP, L1-norm, and L2-norm are employed to handle nonlinear kinematic equality constraints. This submodule involves linearizing the violations of kinematic constraints around the previous iteration solution and penalizing them in the cost function. The inputs to this submodule include the previous iteration solution, the original cost function, and the penalty coefficient for kinematic violation. The output is the linearized cost function that incorporates penalty terms.

Using the convex feasible set constraints generated by the FindConvexCorridor sub-module and the cost function with a kinematic penalty from the DealwithKinematics sub-module, along with other constraints from the original planning task, optimization algorithms are employed in the second stage to solve the optimization problem and obtain feasible trajectories.

After introducing the submodules for handling collision avoidance constraints and kinematic constraint processing in the first and second stages, we will now provide a detailed description of the overall logic for the first stage and second stage.

The first-stage module takes the trajectory planning task from the object module as input and generates an initial trajectory as output. The FindPath module in the first stage

includes implementations of search-based methods with primitives such as 2-dim A* search, hybrid A* search, SimpleLattice, and ControlLattice; search-based methods with random sampling such as InformedRRT, KinodynamicRRT, and CL-RRT; as well as optimization-based methods such as OBCA and CFS. Fig. 10 shows the algorithms we provide.

It is important to note that searchers like A*, hybrid A*, etc. are only used for path planning (methods with "Path" suffix in the Planers folder). Subsequently, depending on the presence of dynamic obstacles in the scene, the appropriate path conversion method should be selected from the ConvertPathToTrajectory folder to add temporal attributes to the path to transform it into a trajectory.

The initial trajectory obtained in the first stage serves as the initial guess for the second stage, guiding the solution based on numerical optimization methods.

It is worth noting that in the first stage, we also provide a trajectory planning algorithm based on SL. This algorithm utilizes a deep neural network regression approach to learn the mapping between trajectory planning tasks and output trajectories from SLiFS. Through offline learning, the algorithm can generate trajectories in real time by recalling the learned mapping when faced with new trajectory planning tasks. However, the trajectories generated by the supervised learning-based approach sometimes collide with obstacles due to the limited sample size and constraints of the offline generation process. Therefore, in the second stage, further adjustments to the trajectories are made through optimization methods.

In the second stage, the FindTrajectory module takes the trajectory planning tasks from the object module and the initial trajectories obtained in the first stage as inputs, and outputs feasible trajectories. The module primarily implements optimization-based algorithms such as SLiFS and H-OBCA. Once the algorithm finds a trajectory, the visualization module's functions can be invoked to examine the static and dynamic results of the trajectory planning.

The VisualizeStaticResults module takes the trajectory planning tasks from the object module and the feasible trajectories obtained from the second stage FindTrajectory module as inputs and generates static images that depict the obstacles and vehicle motion trajectories. On the other hand, the VisualizeDynamicResults module has the same inputs as the VisualizeStaticResults module but produces an animation that illustrates the ego vehicle's motion.

Appendix C provides usage examples of the entire method library. For detailed instructions, readers are suggested visit https://github.com/gyq18/AutonomousDrivingTrajectoryPlanning.

B. Test Scenario Library

Since the DARPA Urban Challenge Driverless Challenge was held in 2004 [70], various types of autonomous driving competitions have flourished. In recent years, competitions focusing on autonomous driving trajectory planning have also emerged. Examples include the CommonRoad competition organized by the Technical University of Munich [17], and the On-site competition organized by Tongji University [71]. However, existing trajectory planning competitions mainly focus on structured environments, with few competitions specifically designed for complex unstructured environments.

In 2022, we held the Trajectory Planning Competition of Automated Parking (TPCAP) for unstructured parking environments [72], which was part of the 2022 IEEE Intelligent Transportation Systems Conference. This competition featured multiple realistic engineering scenarios, offering high-quality and targeted test environments. The performance of trajectory planning algorithms from different participating teams was evaluated using standardized trajectory quality assessment functions. Fig. 11 illustrates a benchmark case in TPCAP.

In addition to the testing scenarios in TPCAP, this paper generated a database of 10,000 test scenarios with universal testing significance. Each test scenario consists of the following elements: planning area, initial and final states of the ego vehicle, obstacle settings, cost function, vehicle kinematic model, and kinematic and dynamic constraints. By adjusting parameters such as the size of the planning area and the number of obstacles, the difficulty of the generated test scenarios can be varied.

Among the 10,000 test scenarios randomly generated, each scenario has a size of $60\text{m} \times 60\text{m}$. These scenarios consist of static polygonal obstacles, with their positions randomly generated following a uniform distribution within the range of $[0\text{m} \times 0\text{m}] - [60\text{m} \times 60\text{m}]$. The area of each obstacle is uniformly distributed within the range of $1\text{m}^2 - 8\text{m}^2$. The 10,000 randomly generated test scenarios aim to simulate complex and unstructured environments, reflecting challenges encountered in real-world scenarios.

We tested the methods in our library on 10,000 test scenarios, and the experimental results were consistent with those presented in Section IV.C. In these scenarios, we employed a two-stage optimization approach for trajectory planning. In the first stage, we utilized a key variable optimization search to determine the initial guess [48]. In the second stage, we applied the CFS method to handle collision avoidance constraints and employed the L1 norm penalty function to address nonlinear kinematic constraints during trajectory optimization [64]. This two-stage approach effectively leverages the speed of obtaining approximately feasible trajectories in the first stage and achieves high-quality trajectory results through optimization-based methods in the second stage. Under real-time conditions, we can rapidly obtain a feasible and relatively optimal trajectory. Compared to other methods, our approach demonstrates a higher success rate in finding trajectories and exhibits smaller tracking errors. Therefore, we recommend adopting this two-stage optimization method for trajectory planning.

VI. Remaining Challenges and Future Perspectives

A. Computational Burden Brought by Complex Kinematic Constraints

In autonomous driving trajectory planning, simplified vehicle kinematic models such as single-track bicycle models are commonly used to reduce computational complexity and meet real-time planning requirements. However, in extreme operating conditions such as excessively high vehicle speed, these simple models cannot depict the true motion state of the vehicle, resulting in the planned trajectory being unable to be tracked by the vehicle, which causes huge tracking errors [73].

Although complex vehicle kinematic models can portray the

real motion state of the vehicle, the complexity and strong nonlinearity of the vehicle models greatly increase the computational burden of trajectory planning, making it hard to plan feasible trajectories in real time. Therefore, finding a feasible trajectory in real-time under extreme conditions is a pressing research problem [74].

In addition, for the planning of tractor-trailer vehicles [75]–[77], the trailer complicates the kinematic principle by greatly increasing the dimension of the state space. Therefore, it is worth exploring how to reduce the trajectory planning time for this particular type of vehicle.

To mitigate the computational complexity associated with complex vehicle models, recent advancements in autonomous driving trajectory planning have incorporated the generation of trajectories based on actual vehicle dynamic models into planning and control [78]. These approaches combine empirical sampling and vehicle kinematic and dynamic simulation, enabling the pre-sampling of various potential control actions and establishing a forward mapping from the start to the destination. Moreover, deep neural networks are utilized to establish the inverse mapping from the start-to-destination point to the corresponding control behavior, enabling the generation of control actions and trajectories from the target position [79]. The results presented in [79] indicate that by focusing on trajectory planning, it is possible to achieve effective path correction during tracking control using a simple proportional feedback controller. However, how to consider the collision avoidance of generated trajectories when sampling trajectories through vehicle kinematic simulation still needs further research.

B. Computational Burden Brought by Complex Constraints in Special Scenes

In specialized scenarios such as mining scenarios, additional constraints are imposed on autonomous driving trajectory planning. These constraints often exhibit strong non-convexity, nonlinearity, and even non-differentiability. For example, in roadside parking scenarios, to prevent tire blowouts, vehicles must adhere to specific constraints on wheel angles when climbing onto the roadside [14]. In scenarios involving slopes without fences [7], vehicles must satisfy orientation constraints to prevent them from sliding off slopes. Moreover, in mining areas, unmanned vehicles face numerous complex constraints when performing transportation tasks in unstructured environments [80].

Currently, a basic way to handle these special constraints is to approximate them and transform them into constraints that are beneficial for the trajectory planners to handle. Especially in optimization-based planners, these constraints are usually convexized or linearized. However, how to ensure the accuracy of the approximation and the ease of calculation, as well as how to effectively define these new spatial constraints and establish a unified processing framework still need further research and exploration.

C. Uncertainty Handling

In autonomous driving systems, uncertainties in sensing, localization, prediction, and other modules pose significant challenges to trajectory planning. On one hand, these uncertainties place high demands on the real-time performance of trajectory planning, which requires trajectory planning to handle sudden obstacles promptly [81], [82]. On the other hand, trajectory planning also needs to exhibit a certain level of fault tolerance and robustness [83], [84].

Currently, trajectory planners for addressing uncertainties can be categorized into two main paradigms based on uncertainty modeling: probabilistic uncertainty representations and deterministic uncertainty representations [27], [85].

Probabilistic uncertainty representation methods refer to requiring collision avoidance probabilities to meet certain thresholds when planning a trajectory. By adjusting the collision threshold, the trajectory smoothness can be flexibly reduced. However, a drawback of these methods is that they cannot guarantee vehicle safety with 100% confidence [82].

Deterministic uncertainty representation methods employ certain sets to model uncertainty and ensure trajectory safety through extra redundancy. For example, when the precise position of obstacles cannot be accurately determined, a robust obstacle avoidance strategy can be achieved by expanding the safety distance between feasible trajectories and obstacles during the planning phase. However, excessive redundancy may lead to a significant reduction in traffic efficiency. Finding a suitable trade-off between traffic safety and efficiency is a problem worthy of research.

When addressing prediction uncertainty, accurately modeling the behavior of dynamic obstacles is challenging, and predictions about the behavior and trajectories of these dynamic obstacles are often influenced by the future behavior of the ego vehicle. Currently, a common solution is to employ probabilistic search tree methods to predict and plan trajectories for various potential dynamic obstacle behaviors [86]. However, further in-depth research is needed due to the curse of dimensionality associated with this approach.

D. Enhancing Optimizer Algorithm Performance

Optimizer algorithms have gained increasing attention in trajectory planning due to their high-quality solutions. However, there is still relatively limited research on the fundamental nature of optimization problems in trajectory planning. The highly non-convex and nonlinear nature of optimization problems in autonomous driving trajectory planning often leads existing solvers to local optima or even infeasible solutions in certain cases. It is imperative to explore the characteristics of optimization problems in autonomous driving trajectory planning and ensure that existing solvers can find feasible trajectories in real time with a high success rate.

E. Deep Integration of Learning-based and Optimization-based Planners

The integration of learning and optimization methods has shown great potential in vehicle trajectory planning for unstructured environments. Quickly determining a good enough initial feasible solution through learning methods, and then quickly strengthening it through optimization methods, may be a powerful solution for vehicle trajectory planning in non-structural environments in the future. Additionally, it is worthwhile to explore how optimization-based methods can be leveraged to rapidly generate labeled trajectory data for neural network learning.

Parallel learning, which combines data processing and action

learning, offers a promising approach for integrating these methods. Parallel learning couples data processing with action learning, utilizing predictive learning to address how to explore data over time; using ensemble learning to address how to explore data in spatial distribution; and employing instructive learning to address the direction of data generation exploration. By combining virtual simulation with real-world testing data, it can effectively solve challenges such as low learning efficiency and the need for extensive interaction with the environment [37], [84], [87].

Parallel learning facilitates the advancement of learning-based methods and optimization-based methods. In the near future, it is expected that a more comprehensive application framework based on parallel learning will be developed [39].

VII. Conclusions

This review focuses on trajectory planning methods for single-vehicle autonomous systems in unstructured scenarios. It presents a comprehensive overview of the existing methods in this field and summarizes a collection of open-source implementations. Through theoretical analysis and experiments, the following conclusions are drawn:

1. Traditional robotic trajectory planning methods are not suitable for autonomous driving trajectory planning, and the computational complexity of trajectory planners specifically designed for autonomous driving is much higher compared to those designed for holonomic AGVs.
2. The size of the intelligent vehicle cannot be ignored in trajectory planning. When performing collision checks, it is recommended to use collision check methods based on the occupancy gridmap, which requires minimal computation time and exhibits slow runtime variation with the number of obstacles compared to other methods such as AABB and OBB collision checkers.
3. In unstructured environments, we recommend the two-stage planning algorithm based on numerical optimization, in which the first stage should use key variable optimization search to determine the initial guess [48] and the second stage should use CFS to deal with collision-avoidance constraints and use L1 norm penalty function to deal with nonlinear kinematic constraints [64]. The two-stage planning algorithm can make full use of the rapidity of finding the rough trajectory in the first stage and the high quality of the trajectory optimization in the second stage, which can quickly find a feasible trajectory with high quality under real-time conditions.

Appendix

A. Symbol Definition

The definitions of the symbols used in this paper are shown in Table XV.

B. Experimental Parameter Configuration

1) Planning Scope Configuration

Configurations of the planning scope are presented in Table XVI.

2) Obstacle Configuration

Scenario 1: Static obstacles with polygonal shapes are randomly generated. The position of each obstacle follows a uniform distribution within the intervals $[0, S_x]$ and $[0, S_y]$ respectively, while the area of each obstacle follows a uniform distribution within the interval [1,8].

3) Vehicle Physical Parameters Configuration

The configuration of vehicle physical parameters is provided in Table XVII.

4) Vehicle Motion Parameters Configuration

The configuration of vehicle motion parameters is presented in Table XVIII.

C. Usage Example of the Proposed Method Library

When using the proposed method library, the focus is mainly on configuring the parameters within the "Object" module, as shown in Fig. 10. The parameter settings in the "Object" module, which primarily pertain to environment, obstacles, and ego vehicle configuration for trajectory planning tasks, are as follows. These parameters serve as global variables and are utilized as built-in inputs for the algorithm module. Subsequently, the trajectory planning algorithms within the algorithm module can be invoked to search for trajectories as required.

Within the "Object" module, we provide two methods for configuring trajectory planning task parameters. One way is to directly read the parameter settings in the SetTrajectoryTask() program, as shown in Fig. 12 a-e.

The other one, by calling the program ReadTrajectoryTask(), reads the data from the ".csv" file, which includes information about the vehicle's start/stop status and obstacles. The data format in the ".csv" file follows the same structure as the test scenario "CaseX.csv" used in the TPCAP competition. Each ".csv" file contains a vector of n rows and 1 column, where the first 6 rows record the initial and target attitudes of the vehicle.

Suppose the vector is V and the element of its ith row is $V[i]$ (note that the indicator i starts from 1, not 0). Then we have $p_x(0) = V[1]$, $p_y(0) = V[2]$, $\theta(0) = V[3]$, $p_x(n) = V[4]$, $p_y(n) = V[5]$, $\theta(n) = V[6]$, where n represents the last trajectory waypoint moment. The $7th$ row of the vector, i.e. $V[7]$, records the total number of obstacles in the scene. The number of vertices of the jth obstacle is recorded in $V[7 + j]$, where the index *j* ranges from 1 to $V[7]$. After that, the vertices of each obstacle are represented by 2-dim coordinate values on the x and y axes, corresponding to the data recorded after $V[7 + j]$. Detailed benchmark cases are available at https://www.tpcap.net/#/benchmarks.

Once the trajectory planning task is configured, the trajectory planning method in the algorithm module can be called to carry out the trajectory planning process. For example, in the first stage, the A* search algorithm can be called using the command $[p_x, p_y, \theta$, path_length, completeness_flag] = PlanAStarPath(). The command outputs the position $[p_x(k), p_y(k)]^{\mathrm{T}}$ and vehicle steering angle $\theta(k)$ for each trajectory point, the total path length, and a flag indicating whether the path is found. After the path is found, the PlanSpeedForStaticScenario program in the ConvertPathToTrajectory folder is called to plan the speed of the path, converting the path into an initial trajectory.

In the second stage of trajectory planning, optimization-based algorithms, such as the L1SCP algorithm, can be invoked using the command [trajectory_s, isfeasible] =

PlanL1SCPTrajectory(initial_trajectory, collision_choice). This command produces the final trajectory and a flag indicating the feasibility of the trajectory. Finally, the VisualizeStaticResults(trajectory_s) program is called to visualize the planning results.

For collision checking, the CheckByMap program in the CheckCollision folder can be utilized. When verifying the feasibility of generated path segments in PlanAStarPath(), the CheckDynamics program can be employed for vehicle kinematics checking.

In the function PlanL1SCPTrajectory(), apart from configuring the initial trajectory (through "initial_trajectory"), it is also necessary to configure the approach of transforming obstacles into convex feasible sets (through "collision_choice"). When collision_choice is set to 1, the FindBox program in the FindCorridor directory is invoked, and the feasible region for each trajectory point is a rectangle. When collision_choice is set to 2, the FindConvexSets program is called, and the feasible region for each trajectory point is a polygon. Both of these collision avoidance methods transform the original non-convex collision-avoidance constraints into linear inequal constraints. In contrast to the PlanL1SCPTrajectory() function, the function PlanL2SCPTrajectory() employs the L2 norm to handle kinematic constraints.

D. Comparison of Trajectory Planners for a Holonomic AGV and an Intelligent Vehicle

This subsection aims to demonstrate the variations among similar trajectory planning methods applied to different entities through simulation experiments, primarily focusing on exploring the disparities between their applications for an autonomous driving vehicle and a holonomic AGV.

In motion planning for a holonomic AGV, the decision variables are the 2-dim position at discretized moments, i.e., $[p_x(k), p_y(k)]^{\mathrm{T}}, k = 1,2, \dots$. However, a nonholonomic vehicle has to consider kinematic constraints, thus expanding the decision variables to $[p_x(k), p_y(k), \theta(k), v(k), \varphi(k)]^{\mathrm{T}}$. In the conducted simulation experiment, a time step of 1 and a total of 60 discrete trajectory points are employed, resulting in 120 decision variables for the holonomic AGV and 300 decision variables for the intelligent vehicle. Comparative simulation results are depicted in Fig. 13, which indicates that CFS finds a smooth path for an holonomic AGV, but the AGV body suffers from a drast change at the beginning moment. While this does not impact the holonomic AGV's performance, it is unsuitable for an intelligent vehicle as it violates the vehicle kinematic rules.

The trajectory planning process varies for different application objects due to their presence of distinct decision variables and constraints, resulting in diverse trajectory shapes and solution times. To compare the solution times of methods applicable to different objects, we conducted experiments within a 60m×60m space, incorporating randomly generated obstacles ranging from 0 to 20.

The comparison involved optimization-based methods, namely CFS vs. SLiFS, search-based methods with primitives, namely A* vs. Hybrid A* search, and search-based methods with random sampling, namely RRT* vs. CL-RRT*. The former is for the holonomic AGV, and the latter is for the intelligent vehicle. The results, depicted in Fig. 14, reveal that methods applied to the vehicle (solid line) take significantly longer to solve compared to methods for the holonomic AGV (dashed line). This disparity in solution times highlights the increased complexity associated with trajectory planning for an autonomous driving vehicle relative to a holonomic AGV.

TABLE XVI. SCENE PARAMETRIC SETTINGS

Parameter	Description	Settings
S_x, S_y	Scale of trajectory planning sceme (m)	60, 60
R_x, R_y	Resolution of occupancy gridmap (m)	0.5, 0.5
N_x, N_y	Number of discretized grids in occupancy gridmap	120, 120

TABLE XVII. EGO-VEHICLE GEOMETRIC PARAMETRIC SETTINGS

Parameter	Description	Settings (m)
L_L	Vehicle length	4.689
L_B	Vehicle width	1.942
L_W	Vehicle wheelbase	2.800
L_F	Vehicle front hang	0.960
L_R	Vehicle rear hang	0.929

TABLE XVIII. EGO-VEHICLE KINEMATIC PARAMETRIC SETTINGS

Parameter	Description	Settings
$v_{\max}, v_{\min}$	Upper and lower bounds of $v(t)$ (m/s)	2.5, -2.5
$a_{\max}, a_{\min}$	Upper and lower bounds of $a(t)$ (m/s^2)	1, -1
$\phi_{\max}, \phi_{\min}$	Upper and lower bounds of $\phi(t)$ (rad)	0.75, -0.75
$\omega_{\max}, \omega_{\min}$	Upper and lower bounds of $\omega(t)$ (rad/s)	0.5, -0.5
$r_{\min}$	Minimum turning radius of the ego vehicle, i.e., $L_W/\tan(\phi_{\max})$ (m)	3.0056

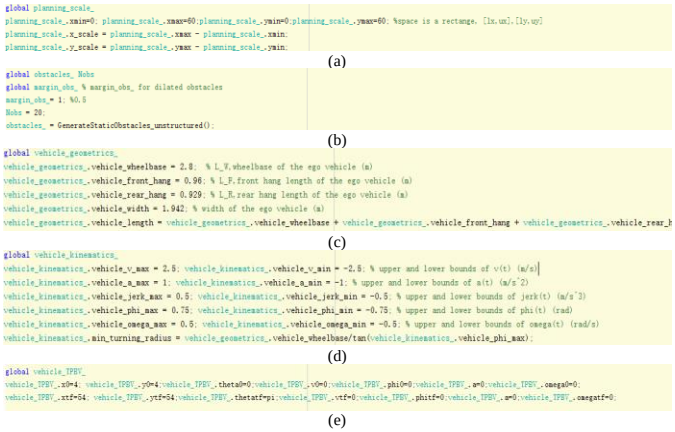


Fig. 12. Parameter settings in SetTrajectoryTask() program: (a) environment size setup; (b) random obstacle generation; (c) vehicle shape parameter configuration; (d) vehicle kinematic parameter configuration; (e) state restrictions of ego vehicle at the initial and terminal moments.

E. Comparison of Solution Runtime of Trajectory Planners with an Ascending Obstacle Number

Except for varying the number of obstacles from 0 to 20, all other parameters in the experimental scenario remain consistent with those in Appendix D. The initial and final states of the vehicle, as well as the positions of the obstacles, are randomly generated. The results presented below are based on 100 repetitions of the experiment.

The results show that trajectory planners run fastest in C++ than other coding environments. This is mainly due to the huge advantages that the compiled language (C++) has over interpreted languages (Python and MATLAB), and the better

memory management in C++. It should be noted that the optimization-based methods primarily spend time on the optimizer solution. In C++ and MATLAB, we use cplexqp [65] as the optimizer. There is not much difference in the time taken by cplexqp between the two languages. Due to limited support for cplexqp in Python, we use the Python version of solver.qp to solve the optimization problem, which will result in increased solution times for optimization-based methods in Python.

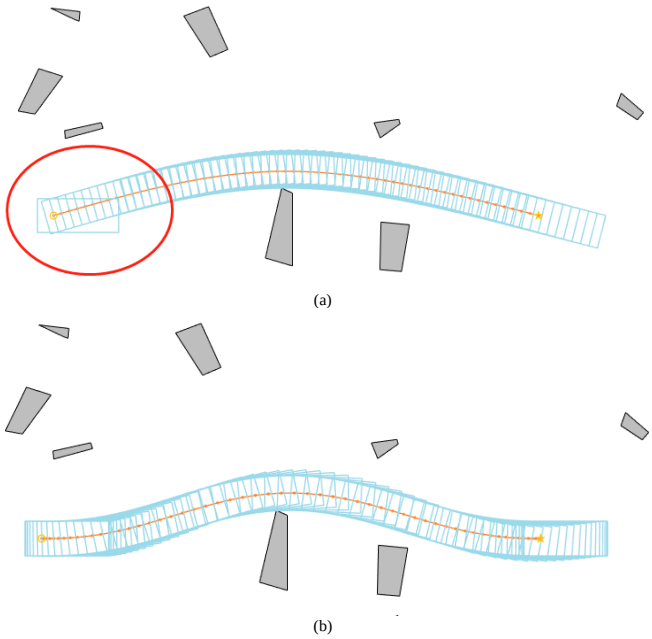


Fig. 13. Comparison of optimization-based methods for finding trajectories applicable to a holonomic AGV and an intelligent vehicle: (a) trajectories and footprints obtained by the CFS algorithm applied to a holonomic AGV; (b) trajectories and footprints obtained by the SLiFS algorithm for an intelligent vehicle.

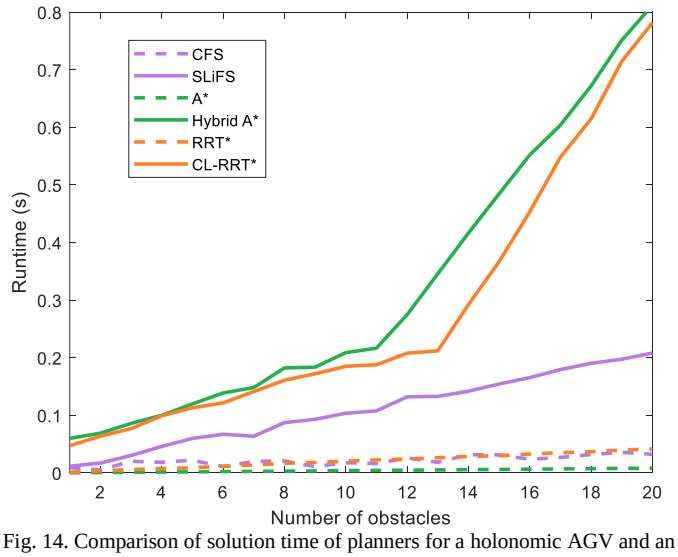
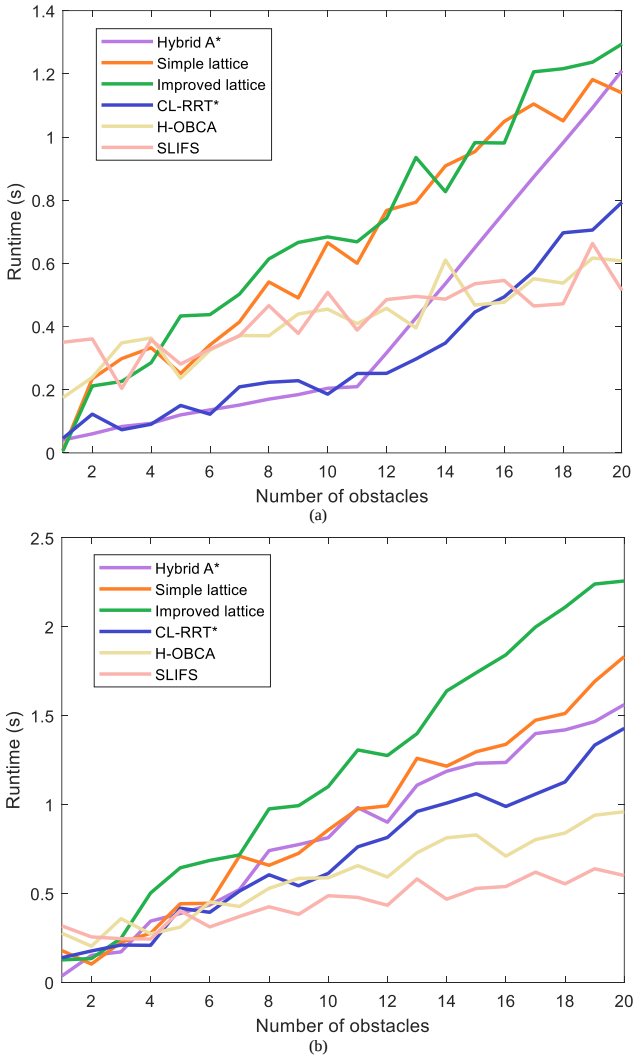
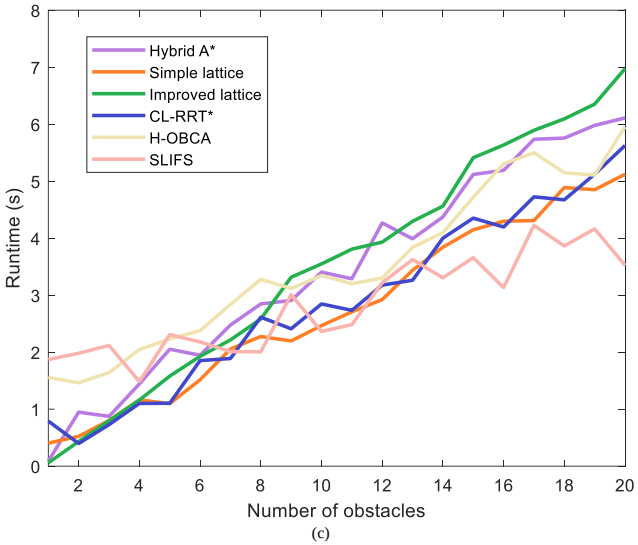


Fig. 14. Comparison of solution time of planners for a holonomic AGV and an intelligent vehicle (in C++ release mode).

As seen from Fig. 15, the CL-RRT* algorithm and the hybrid A* search algorithm exhibit shorter solution times with fewer obstacles, but longer solution times with more obstacles. Conversely, the algorithm demonstrates a shorter solution time as the number of obstacles increases. This phenomenon may stem from the reduced frequency of collision checks in environments with fewer obstacles, leading to more efficient node expansion and improved tree search efficiency. Conversely, in environments with many obstacles, the increased collision checks hinder the efficiency of node expansion, increasing the solution time of these planners.

Besides, in narrow environments, the decrease in effective expansion nodes requires generating a large number of motion primitives, increasing the solution time. By contrast, the SLiFS algorithm utilizes a two-stage approach incorporating a coarse search and fine optimization framework. The inclusion of coarse motion primitives in narrow environments helps mitigate time consumption issues. Moreover, the coarse initial trajectory effectively guides the subsequent numerical optimization, resulting in a reduction in overall solution time. Hence, we recommend employing the two-stage SLiFS algorithm to deal with complex unstructured environments.





(c)

Fig. 15. Variation of solution time with an increasing number of obstacles for unstructured scenarios trajectory planners. The solution time is illustrated for three programming languages: (a) C++; (b) MATLAB; and (c) Python.

REFERENCES

[1] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Trans. Intell. Veh.*, vol. 1, no. 1, pp. 33–55, Mar. 2016.

[2] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 4, pp. 1135–1145, Apr. 2016.

[3] S. Aradi, "Survey of deep reinforcement learning for motion planning of autonomous vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 2, pp. 740–759, Feb. 2022.

[4] D. Delling, P. Sanders, D. Schultes, and D. Wagner, "Engineering route planning algorithms," in *Algorithmics of large and complex networks*. Springer, 2009, pp. 117–139.

[5] L. Claussmann, M. Revilloud, D. Gruyer, and S. Glaser, "A review of motion planning for highway autonomous driving," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 5, pp. 1826–1848, May 2020.

[6] S. Dixit *et al.*, "Trajectory planning and tracking for autonomous overtaking: State-of-the-art and future prospects," *Annu. Rev. Control*, vol. 45, pp. 76–86, Jan. 2018.

[7] F. Tian *et al.*, "Trajectory planning for autonomous mining trucks considering terrain constraints," *IEEE Trans. Intell. Veh.*, vol. 6, no. 4, pp. 772–786, Dec. 2021.

[8] B. Li, Z. Yin, Y. Ouyang, Y. Zhang, X. Zhong, and S. Tang, "Online trajectory replanning for sudden environmental changes during automated parking: A parallel stitching method," *IEEE Trans. Intell. Veh.*, vol. 7, no. 3, pp. 748–757, Sep. 2022.

[9] J. Pan, L. Zhang, and D. Manocha, "Collision-free and smooth trajectory computation in cluttered environments," *Int. J. Robot. Res.*, vol. 31, pp. 1155–1175, Sep. 2012.

[10] J. Reeds and L. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pac. J. Math.*, vol. 145, no. 2, pp. 367–393, Oct. 1990.

[11] M. O'Connell *et al.*, "Neural-fly enables rapid learning for agile flight in strong winds," *Sci. Robot.*, vol. 7, no. 66, p. eabm6597, May 2022.

[12] X. Zhou *et al.*, "Swarm of micro flying robots in the wild," *Sci. Robot.*, vol. 7, no. 66, p. eabm5954, May 2022.

[13] P. Foehn, A. Romero, and D. Scaramuzza, "Time-optimal planning for quadrotor waypoint flight," *Sci. Robot.*, vol. 6, no. 56, p. eabh1221, Jul. 2021.

[14] Y. Guo, D. Yao, B. Li, Z. He, H. Gao, and L. Li, "Trajectory planning for an autonomous vehicle in spatially constrained environments," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 10, pp. 18326–18336, Oct. 2022.

[15] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," in *Proc. 2015 IEEE Intell. Veh. Symp. (IV)*, Seoul, Korea (South), 2015, pp. 1094–1099.

[16] D. Schramm, M. Hiller and R. Bardini, *Vehicle Dynamics: Modeling and Simulation*, Berlin, Germany:Springer, 2014.

[17] M. Althoff, M. Koschi and S. Manzinger, "CommonRoad: Composable benchmarks for motion planning on roads", in *Proc. 2017 IEEE Intell. Veh. Symp. (IV)*, Los Angeles, CA, USA, 2017, pp. 719–726.

[18] B. Li, Y. Ouyang, L. Li, and Y. Zhang, "Autonomous driving on curvy roads without reliance on frenet frame: A cartesian-based trajectory planning method," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 9, pp. 15729–15741, Sep. 2022.

[19] J. Chen, W. Zhan, and M. Tomizuka, "Autonomous driving motion planning with constrained iterative LQR," *IEEE Trans. Intell. Veh.*, vol. 4, no. 2, pp. 244–254, Jun. 2019.

[20] K. Bergman, O. Ljungqvist, and D. Axehill, "Improved path planning by tightly combining lattice-based path planning and optimal control," *IEEE Trans. Intell. Veh.*, vol. 6, no. 1, pp. 57–66, Mar. 2021.

[21] D. Zheng and P. Tsiotras, "Accelerating kinodynamic RRT* through dimensionality reduction", in *Proc. 2021 IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Prague, Czech Republic, 2021, pp. 3674–3680.

[22] D. J. Webb and J. van den Berg, "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics," in *Proc. 2013 IEEE Int. Conf. Robot. Autom. (ICRA)*, Karlsruhe, Germany, 2013, pp. 5054–5061.

[23] C. Rösmann, W. Feiten, T. Wösch, F. Hoffmann, and T. Bertram, "Efficient trajectory optimization using a sparse model," in *Proc. 2013 Eur. Conf. Mob. Robots (ECMR)*, Barcelona, Spain, 2013, pp. 138–143.

[24] B. Li and Z. Shao, "Precise trajectory optimization for articulated wheeled vehicles in cluttered environments," *Adv. Eng. Softw.*, vol. 92, pp. 40–47, Feb. 2016.

[25] Y. Rasekhipour, A. Khajepour, S.-K. Chen, and B. Litkouhi, "A potential field-based model predictive path-planning controller for autonomous road vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 5, pp. 1255–1267, May 2017.

[26] W. Liu and Z. Li, "Comprehensive predictive control method for automated vehicles in dynamic traffic circumstances," *IET Intell. Transp. Syst.*, vol. 12, no. 10, pp. 1455–1463, Dec. 2018.

[27] N. Ceccarelli, M. Di Marco, A. Garulli, A. Giannitrapani, and A. Vicino, "Path planning with uncertainty: A set membership approach," *Int. J. Adapt. Control Signal Process.*, vol. 25, no. 3, pp. 273–287, Mar. 2011.

[28] A. Takahashi, T. Hongo, Y. Ninomiya, and G. Sugimoto, "Local path planning and motion control for AGV in positioning," in *Proc. IEEE/RSJ International Workshop on Intelligent Robots and Systems. (IROS '89) 'The Autonomous Mobile Robots and Its Applications*, Tsukuba, Japan, 1989, pp. 392–397.

[29] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Practical search techniques in path planning for autonomous driving," *Ann Arbor*, vol. 1001, no. 48105, pp. 18–80, 2008.

[30] C. Ericson, *Real-Time Collision Detection*, Boca Raton, FL, USA:CRC Press, 2004.

[31] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning", *The Annual Research Report*, 1998.

[32] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proc. 1985 IEEE Int. Conf. Robot. Autom. (ICRA)*, MO, USA, 1985, pp. 500–505.

[33] E. Gilbert and D. Johnson, "Distance functions and their application to robot path planning in the presence of obstacles," *IEEE J. Robot. Autom.*, vol. 1, no. 1, pp. 21–30, 1985.

[34] T. T. Mac, C. Copot, D. T. Tran, and R. De Keyser, "Heuristic approaches in robot path planning: A survey," *Robot. Auton. Syst.*, vol. 86, pp. 13–28, Dec. 2016.

[35] X. Luo, L. Li, L. Zhao, and J. Lin, "Dynamic intra-cell repositioning in free-floating bike-sharing systems using approximate dynamic programming," *Transp. Sci.*, vol. 56, no. 4, pp. 799–826, Jul. 2022.

[36] W. Liu, Z. Li, L. Li, and F.-Y. Wang, "Parking like a human: A direct trajectory planning solution," *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 12, pp. 3388–3397, Dec. 2017.

[37] Y.-L. Lin, L. Li, X.-Y. Dai, N.-N. Zheng, and F.-Y. Wang, "Master general parking skill via deep learning," in *Proc. 2017 IEEE Intell. Veh. Symp. (IV)*, Los Angeles, CA, USA, 2017, pp. 941–946.

[38] Y. Guan *et al*., "Integrated decision and control: Toward interpretable and computationally efficient driving intelligence," *IEEE Trans. Cybern.*, vol. 53, no. 2, pp. 859–873, Feb. 2023.

[39] L. Chen, X. Hu, W. Tian, H. Wang, D. Cao, and F.-Y. Wang, "Parallel planning: A new motion planning framework for autonomous driving," *IEEE/CAA J. Automatica Sinica*, vol. 6, no. 1, pp. 236–246, Jan. 2019.

[40] C. Liu, C.-Y. Lin, and M. Tomizuka, "The convex feasible set algorithm for real time optimization in motion planning," *SIAM J. Control Optim.*, vol. 56, no. 4, pp. 2712–2733, Jan. 2018.

[41] X. Zhang, A. Liniger, A. Sakai, and F. Borrelli, "Autonomous parking using optimization-based collision avoidance," in *Proc. 2018 IEEE Conf. Decis. Control (CDC)*, Miami, FL, USA, 2018, pp. 4327–4332.

[42] J. Wang, M. Q.-H. Meng, and O. Khatib, "EB-RRT: Optimal motion planning for mobile robots," *IEEE Trans. Autom. Sci. Eng.*, vol. 17, no. 4, pp. 2063–2073, Oct. 2020.

[43] Y. Shimizu, W. Zhan, L. Sun, J. Chen, S. Kato and M. Tomizuka, "Motion planning for autonomous driving with extended constrained iterative LQR", *Dynamic Systems and Control Conference*, vol. 84270, pp. V001T12A001, 2020.

[44] B. Li *et al*., "Optimization-based trajectory planning for autonomous parking with irregularly placed obstacles: A lightweight iterative framework," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 8, pp. 11970–11981, Aug. 2022.

[45] F. Ulbrich, D. Goehring, T. Langner, Z. Boroujeni, and R. Rojas, "Stable timed elastic bands with loose ends," in *Proc. 2017 IEEE Intell. Veh. Symp. (IV)*, 2017, pp. 186–192.

[46] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *J. Field Robot.*, vol. 37, no. 3, pp. 362–386, Apr. 2020.

[47] X. Zhang, A. Liniger, and F. Borrelli, "Optimization-based collision avoidance," *IEEE Trans. Control Syst. Technol.*, vol. 29, no. 3, pp. 972–983, May 2021.

[48] Y. Guo, D. Yao, B. Li, H. Gao, and L. Li, "Down-sized initialization for optimization-based unstructured trajectory planning by only optimizing critical variables", *IEEE Trans. Intell. Veh.*, vol. 8, no.1, pp.709–720, Jan. 2023.

[49] B. Li, L. Li, T. Acarman, Z. Shao, and M. Yue, "Optimization-based maneuver planning for a tractor-trailer vehicle in a curvy tunnel: A weak reliance on sampling and search," *IEEE Robot. Autom. Lett.*, vol. 7, no. 2, pp. 706–713, Apr. 2022.

[50] E. G. Tsardoulias, A. Iliakopoulou, A. Kargakos, and L. Petrou, "A review of global path planning methods for occupancy grid maps regardless of obstacle density," *J. Intell. Robot. Syst.*, vol. 84, no. 1–4, pp. 829–858, Dec. 2016.

[51] M. C. Lin and S. Gottschalk, "Collision detection between geometric models: A survey," in *Proc. IMA Conf. Math. Surf.*, 1998, pp. 602–608.

[52] B. Li, K. Wang, and Z. Shao, "Time-optimal maneuver planning in automatic parallel parking using a simultaneous dynamic optimization approach," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 11, pp. 3263–3274, Nov. 2016.

[53] B. Li and Z. Shao, "A unified motion planning method for parking an autonomous vehicle in the presence of irregularly placed obstacles," *Knowl.-Based Syst.*, vol. 86, pp. 11–20, Sep. 2015.

[54] I. Papadimitriou and M. Tomizuka, "Fast lane changing computations using polynomials," in *Proc. 2003 Ame. Control Conf. (ACC)*, Denver, CO, USA, 2003, pp. 48–53 vol.1.

[55] S. Boyd, S. P. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge, U.K.:Cambridge Univ. Press, 2004.

[56] R. He *et al.*, "TDR-OBCA: A reliable planner for autonomous driving in free-space environment," in *Proc. 2021 Ame. Control Conf. (ACC)*, New Orleans, LA, USA, 2021, pp. 2927–2934.

[57] J. Chen, T. Liu, and S. Shen, "Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments," in *Proc. 2016 IEEE Int. Conf. Robot. Autom. (ICRA)*, Stockholm, 2016, pp. 1476–1483.

[58] F. Gao, W. Wu, Y. Lin, and S. Shen, "Online safe trajectory generation for quadrotors using fast marching method and Bernstein basis polynomial," in *Proc. 2018 IEEE Int. Conf. Robot. Autom. (ICRA)*, Brisbane, QLD, Australia, 2018, pp. 344–351.

[59] W. Ding, L. Zhang, J. Chen, and S. Shen, "Safe trajectory generation for complex urban environments using spatio-temporal semantic corridor," *IEEE Robot. Autom. Lett.*, vol. 4, no. 3, pp. 2997–3004, Jul. 2019.

[60] Z. Zhu, E. Schmerling, and M. Pavone, "A convex optimization approach to smooth trajectories for motion planning with car-like robots," in *Proc. 2015 54th IEEE Conf. Decis. Control (CDC)*, Osaka, 2015, pp. 835–842.

[61] Z. Zhang *et al.*, "A guaranteed collision-free trajectory planning method for autonomous parking," *IET Intell. Transp. Syst.*, vol. 15, no. 2, pp. 331–343, 2021.

[62] M. Vahedi and A. F. van der Stappen, "Caging polygons with two and three fingers," *Int. J. Robot. Res.*, vol. 27, no. 11–12, pp. 1308–1324, Nov. 2008.

[63] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: gradient optimization techniques for efficient motion planning," in *Proc. 2009 IEEE Int. Conf. Robot. Autom. (ICRA)*, Kobe, Japan, 2009, pp. 489–494.

[64] C. Sun, Q. Li, B. Li, and L. Li, "A successive linearization in feasible set algorithm for vehicle motion planning in unstructured and low-speed scenarios," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 4, pp. 3724–3736, Apr. 2022.

[65] S. Nickel, C. Steinhardt, H. Schlenker, W. Burkart and M. Reuter-Oppermann, "Ibm ilog cplex optimization studio" in Angewandte Optimierung mit IBM ILOG CPLEX Optim. Studio, Gabler, Berlin, Heidelberg, Germany:Springer, pp. 9–23, 2021.

[66] B. Cohen and M. Likhachev. (2009). *The Search Based Planning Library (SBPL)* [Online]. Available: http://www.ros.org/wiki/sbpl

[67] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robot. Autom. Mag.*, vol. 19, no. 4, pp. 72–82, Dec. 2012.

[68] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *Proc. 2011 IEEE Int. Conf. Robot. Autom. (ICRA)*, Shanghai, China, 2011, pp. 4569–4574.

[69] J. Pan and D. Manocha. (2011). *The Fast Collision Library (FCL)* [Online]. Available: http://www.ros.org/wiki/fcl_ros

[70] M. Buehler, K. Iagnemma and S. Singh, *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*, vol. 56. Springer, 2009.

[71] J. Sun, H. Zhang, H. Zhou, R. Yu, and Y. Tian, "Scenario-based test automation for highly automated vehicles: A review and paving the way for systematic safety assurance," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 9, pp. 14088–14103, Sep. 2022.

[72] B. Li *et al*., "Online competition of trajectory planning for automated parking: Benchmarks, achievements, learned lessons, and future perspectives," *IEEE Trans. Intell. Veh.*, vol. 8, no. 1, pp. 16–21, Jan. 2023.

[73] P. Polack, F. Altché, B. d'Andréa-Novel, and A. de La Fortelle, "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?," in *Proc. 2017 IEEE Intell. Veh. Symp. (IV)*, Los Angeles, CA, USA, 2017, pp. 812–818.

[74] F. Tian, Z. Li, F.-Y. Wang, and L. Li, "Parallel learning-based steering control for autonomous driving," *IEEE Trans. Intell. Veh.*, vol. 8, no. 1, pp. 379–389, Jan. 2023.

[75] B. Li, T. Acarman, Y. Zhang, L. Zhang, C. Yaman, and Q. Kong, "Tractor-trailer vehicle trajectory planning in narrow environments with a progressively constrained optimal control approach," *IEEE Trans. Intell. Veh.*, vol. 5, no. 3, pp. 414–425, Sep. 2020.

[76] M. Yue, X. Hou, X. Zhao, and X. Wu, "Robust tube-based model predictive control for lane change maneuver of tractor-trailer vehicles based on a polynomial trajectory," *IEEE Trans. Syst. Man Cybern. Syst.*, vol. 50, no. 12, pp. 5180–5188, Dec. 2020.

[77] O. Ljungqvist, N. Evestedt, D. Axehill, M. Cirillo, and H. Pettersson, "A path planning and path-following control framework for a general 2-trailer with a car-like tractor," *J. Field Robot.*, vol. 36, no. 8, pp. 1345–1377, 2019.

[78] L. Li and F.-Y. Wang, "Parking guidance system for front wheel steering vehicles using trajectory generation," in *Proc. 2003 IEEE International Conference on Intelligent Transportation Systems (ITSC)*, Shanghai,

China, 2003, pp. 1770–1775.

[79] F. Tian, F.-Y. Wang, and L. Li, "Enhancing feedback steering controllers for autonomous vehicles with deep monte carlo tree search," *IEEE Robot. Autom. Lett.*, vol. 7, no. 4, pp. 10438–10445, Oct. 2022.

[80] B. Li, Y. Ouyang, X. Li, D. Cao, T. Zhang, and Y. Wang, "Mixed-integer and conditional trajectory planning for an autonomous mining truck in loading/dumping scenarios: A global optimization approach," *IEEE Trans. Intell. Veh.*, vol. 8, no. 2, pp. 1512–1522, Feb. 2023.

[81] A. Wang, A. Jasour, and B. C. Williams, "Non-gaussian chance-constrained trajectory planning for autonomous vehicles under agent uncertainty," *IEEE Robot. Autom. Lett.*, vol. 5, no. 4, pp. 6041–6048, Oct. 2020.

[82] Y. Guo, H. Xu, D. Yao and L. Li, "A real-time optimization-based trajectory planning method in dynamic and uncertain environments," in *Proc. 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, Rhodes, Greece, 2020, pp. 1–6.

[83] A. Artuñedo, J. Villagra, J. Godoy, and M. D. del Castillo, "Motion planning approach considering localization uncertainty," *IEEE Trans. Veh. Technol.*, vol. 69, no. 6, pp. 5983–5994, Jun. 2020.

[84] S. Glaser, B. Vanholme, S. Mammar, D. Gruyer, and L. Nouvelière, "Maneuver-based trajectory planning for highly autonomous vehicles on real road with traffic and driver interaction," *IEEE Trans. Intell. Transp. Syst.*, vol. 11, no. 3, pp. 589–606, Sep. 2010.

[85] W. Sun, S. Patil, and R. Alterovitz, "High-frequency replanning under uncertainty using parallel sampling-based motion planning," *IEEE Trans. Robot.*, vol. 31, no. 1, pp. 104–116, Feb. 2015.

[86] J. Van Den Berg, S. Patil, and R. Alterovitz, "Motion planning under uncertainty using iterative local optimization in belief space," *Int. J. Robot. Res.*, vol. 31, no. 11, pp. 1263–1278, Sep. 2012.

[87] L. Li, Y. Lin, N. Zheng, and F.-Y. Wang, "Parallel learning: A perspective and a framework," *IEEE/CAA J. Automatica Sinica*, vol. 4, no. 3, pp. 389–395, 2017.

[88] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robot. Autom. Mag.*, vol. 4, no. 1, pp. 23–33, Mar. 1997.

[89] B. Li *et al.*, "Adaptive Pure Pursuit: A Real-Time Path Planner Using Tracking Controllers to Plan Safe and Kinematically Feasible Paths," *IEEE Trans. Intell. Veh.*, vol. 8, no. 9, pp. 4155–4168, Sep. 2023.

[90] Z. Zhou *et al.*, "A reliable path planning method for lane change based on hybrid PSO-iACO algorithm," in *Proc. 2021 6th International Conference on Transportation Information and Safety (ICTIS)*, Wuhan, China, 2021, pp. 1253–1258.

[91] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, Jul. 1968.

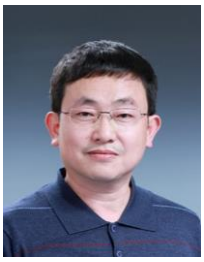
[92] S. Teng *et al.*, "Motion Planning for Autonomous Driving: The State of the Art and Future Perspectives," *IEEE Trans. Intell. Veh.*, vol. 8, no. 6, pp. 3692–3711, Jun. 2023.

[93] B. Li *et al.*, "Embodied Footprints: A Safety-Guaranteed Collision-Avoidance Model for Numerical Optimization-Based Trajectory Planning," *IEEE Trans. Intell. Transp. Syst.*, early access, Oct. 25, 2023, doi: 10.1109/TITS.2023.3316175.

Yuqing Guo received her B.S. degree from the Harbin Institute of Technology, China, in 2018, and her Ph.D. degree in July 2023 from Tsinghua University, China. Her current research interests include autonomous driving, cooperative driving, and trajectory planning.

Zelin Guo received his B.S. degree in Vehicle Engineering from Zhejiang University, China, in June 2023. He is pursuing a Master's degree in automation science at Tsinghua University currently. His research focuses on unmanned systems with special emphasis on intelligent vehicle decision making, motion planning, and simulation.

Yazhou Wang received his B.S. degree in Vehicle Engineering from Hunan University, China, in July 2021. Currently, he is pursuing his Master's degree at the College of Mechanical and Vehicle Engineering in the same institution. His research focuses on intelligent vehicle systems, with an emphasis on decision making, trajectory planning, control, and software engineering aspects.



Danya Yao received the B.S., M.S., and Ph.D. degrees from Tsinghua University, Beijing, China, in 1988, 1990, and 1994, respectively. He is currently a Full Professor with the Department of Automation, Tsinghua University. His research interests include intelligent detection technology, system engineering, mixed traffic flow theory, and intelligent transportation systems.

Bai Li (S'13–M'18) received his B.S. degree in 2013 from Beihang University, China, and his Ph.D. degree in 2018 from Zhejiang University, China. From Nov. 2016 to June 2017, he visited the University of Michigan (Ann Arbor), USA as a joint training Ph.D. student. He is currently an associate professor at the College of Mechanical and Vehicle Engineering, Hunan University, China. Before teaching in Hunan University, he worked in JDX R&D Center of Automated Driving, JD Inc., China from 2018 to 2020 as an algorithm engineer. Prof. Li has been the first author of more than 80 journal/conference papers and two books in numerical optimization, motion planning, and robotics. He was a recipient of the International Federation of Automatic Control (IFAC) 2014–2016 Best Journal Paper Prize from Engineering Applications of Artificial Intelligence. He received the 2022 Best Associate Editor Award of IEEE TRANSACTIONS ON INTELLIGENT VEHICLES. He is currently an Associate Editor for IEEE TRANSACTIONS ON INTELLIGENT VEHICLES. His research interest is motion planning methods for autonomous driving.

Li Li (F'17) is currently a Professor with the Department of Automation, Tsinghua University, Beijing, China, working in the fields of artificial intelligence, complex systems, intelligent transportation systems, and intelligent vehicles. He has published over 160 SCI-indexed international journal articles and over 70 international conference papers as a first/corresponding author. He is a member of the Editorial Advisory Board for the Transportation Research Part C: Emerging Technologies, and a member of the Editorial Board for the Transport Reviews and ACTA Automatica. He serves as an Associate Editor of IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS and IEEE TRANSACTIONS ON INTELLIGENT VEHICLES.