Parking Trajectory Planning Using Multi-resolution State Roadmaps

Yuichi Tazaki, *Member, IEEE,* Hiroyuki Okuda, *Nonmember,* and Tatsuya Suzuki *Member, IEEE*

***Abstract*—This paper presents a trajectory planning method for automated parking that makes use of multi-resolution roadmaps. The proposed roadmap generation algorithm takes as an input manually designed geometric information of a parking lot consisting of parking space layout, obstacle configuration, and a set of geometric features called guidelines. It constructs a roadmap by dividing the guidelines in multiple resolutions until it achieves enough coverage over the set of safe trajectories satisfying vehicle kinematics, curvature limit, and collision-avoidance. Using this roadmap, a complex parking trajectory composed of both forward and reverse motions can be computed with small online computation cost. The proposed method is evaluated in both numerical simulations and real experiments.**

***Index Terms*—Autonomous Parking, Trajectory Planning, Multi-resolution Roadmaps**

I. Introduction

Parking is a difficult task for many drivers because maneuvering a vehicle having non-holonomic characteristics in narrow space between obstacles to a desired parking space requires high driving skill. For this reason, congestion and collision accidents occur frequently at parking lots. Therefore, autonomous parking function of automated cars is expected to reduce the effort of human drivers and to realize safe and efficient parking system.

A variety of trajectory planning methods have been applied to parking tasks. A method used in [1], [2], [3] plans a parking trajectory in two steps. In the first step, a collision-free path is generated without considering the non-holonomic constraint of the vehicle, and in the second step, a trajectory that approximately tracks that path subject to the non-holonomic constraint is generated. If the so-called small-space controllability property is satisfied, then it is mathematically guaranteed that a feasible trajectory is always found in the second step. A drawback of decomposition into two steps is that collision avoidance must be considered with conservativeness in the first step in order to leave some collision-free margin for path deformation carried out in the second step.

Another type of methods expresses complex parking trajectories by concatenating primitive trajectory segments, and uses search methods to generate a connected trajectory to a desired parking space. Search methods used in the literature are A* search [4], rapidly-exploring random trees (RRT) [5], [6], and state lattice [7]. Computational cost is considered to be a major issue of this approach. In fact, it was reported in [4] that discretization of steering operation had to be limited to low resolution in order to complete the path planning in a practical amount of time.

Digital map is one of the key elements of autonomous driving technology [8]. A digital map contains various information of traffic environment that is useful for realizing autonomous driving. It includes point-cloud data of obstacles, geometric shapes of roads and buildings, topological information of lanes and intersections, and location of traffic signs. To the authors' knowledge, study on digital maps particularly suited to autonomous parking is limited. The following are some existing studies that utilize off-line computed map-like information for parking. A method that makes use of a trajectory database was proposed in [9]. Here, a large set of simple trajectories that drives a vehicle to different goal configurations was pre-computed and stored in a database. It was reported that a database for a particular parking environment consists of 50 million records. The efficiency of path planning largely depends on the search speed of the database engine, but it seems hard to generate a long trajectory composed of multiple primitive trajectories using this method. In [4], a SLAM (simultaneous localization and mapping) method developed in mobile robotics was used for generating a map of a parking lot. A map generated in this manner is useful for self-localization and for recognition of other parked cars, but it does not contain information that is directly useful for path planning. In [10], a method for extracting a map of a parking lot from satellite images was proposed. A map generated by this method contains parking space layout and the topological skeleton of the parking lot. Maps having graph-like structure is considered to be well-suited to trajectory planning, but topological information alone is insufficient for generating trajectories under complex collision avoidance and non-holonomic constraints.

This paper proposes a trajectory planning method for automated parking that makes use of a graph-like digital map. The proposed map representation is called state roadmap in this paper because it is essentially a roadmap (a map in the form of a graph) defined in the state space of the vehicle. A state roadmap of a parking lot is generated off-line, and stored in the memory of the trajectory planner. Once a state roadmap is generated, it can be used repeatedly for many different parking requests. A fundamental difference of the proposed method from existing methods that make use of trajectory database such as [9] is that a map generated by the proposed method is not a mere collection of trajectory segments. It has a graph-like structure so that various graph-search methods can

Y. Tazaki is with the Department of Mechanical Engineering, Kobe University, Rokkodai-cho, Nada-ku, Kobe, Hyogo, Japan e-mail: tazaki@mech.kobe-u.ac.jp

H. Okuda and T. Suzuki are with the Department of Mechanical Science and Engineering, Nagoya University, Furo-cho, Chikusa-ku, Nagoya, Aichi, Japan e-mail: (h_okuda,t_suzuki)@nuem.nagoya-u.ac.jp

be used for planning long and complex parking trajectories efficiently. The proposed map is also different from simple topological maps because it is constructed in consideration of geometric feasibility such as collision avoidance and curvature limit. Therefore, from any connected path of a state roadmap, a physically realizable trajectory can be instantiated.

This paper is based on the earlier work [11], in which the authors proposed a parking trajectory generation method based on state roadmaps. This paper makes some important extensions to this method, and evaluates the method in new simulation and real experiments.

Improved trajectory representation The trajectory representation used in [11] has some range of orientations for which no trajectory can be defined. In the improved representation, a trajectory can be defined for almost arbitrary combination of orientations, except for a few singular cases. In addition the new formulation supports clothoids and arcs, while the previous formulation supported clothoids only.

Introduction of guidelines The previous method has a limitation that a number of via-points have to be laid out manually in the parking environment, and the spacing of via-points has to be determined empirically. This paper extends the method to support guidelines in addition to via-points. Guidelines can be placed along the center-lines of lanes and parking spaces, and therefore it requires less tuning than via-points.

Switching active constraints at plan-time A new feature presented in this paper is that constraints can be independently made active or inactive at plan-time without modifying the pre-computed state roadmap. Thanks to this new feature, the planner is now able to generate shorter trajectories by letting the vehicle travel across vacant parking spaces.

A new index for determining appropriate partition resolution In the algorithm presented in [11], the user needed to specify the maximum partition depth, but appropriate partition depth was problem dependent. In the revised algorithm, the quality of partitioning is evaluated by an index called ambiguity ratio. Based on this index, the algorithm automatically terminates the partitioning process when a desired ambiguity ratio is achieved.

The organization of the rest of this paper is as follows. In Section II, parking is formulated as a general trajectory planning problem. After introducing geometric objects called guidelines and explicit representation of trajectories, a more specific guideline-based path planning is formulated. In Section III, state roadmap is introduced as a graph-like map that consists of partition trees of guidelines and a collection of transitions between partitioned intervals. Next, an algorithm that automatically generates a state roadmap by iteratively refining the partition trees is presented. Numerical evaluation results as well as the results of parking experiments using a real electric vehicle are shown in Section IV. Concluding remarks are given in Section V.

II. Formulation of Trajectory Planning Problem

A. General Formulation of Parking Trajectory Planning

In this paper, the following simple kinematic model of a car with front steering is considered.

$$\begin{aligned} \dot{p_{\mathrm{x}}}(s) &= v(s)\cos(\theta(s)), \\ \dot{p_{\mathrm{y}}}(s) &= v(s)\sin(\theta(s)), \\ \dot{\theta}(s) &= v(s)u(s). \end{aligned} \tag{1}$$

Here, s denotes the travel distance that takes 0 at the starting point. The derivatives in the above equations are taken with respect to s. The symbols $p(s) = [p_{\mathrm{x}}(s), p_{\mathrm{y}}(s)]^{\mathsf{T}}$, $\theta(s)$, $u(s)$, and $v(s) \in \{+1, -1\}$ denote the position, the orientation, the turning curvature, and the moving direction at s, respectively. The position of the vehicle is represented by the position of the center of the rear axle. The configuration of the vehicle is defined as $x(s) = [p_{\mathrm{x}}(s), p_{\mathrm{y}}(s), \theta(s)]^{\mathsf{T}}$, and the set of possible configurations is denoted by $X \subset \mathbb{R}^3$. The control inputs to the vehicle are $u(s)$ and $v(s)$. Parking maneuver is normally executed in low speed, at which dynamic characteristics such as tyre slippage and body inertia are negligible. This justifies the use of the simple kinematic model described above for parking trajectory planning.

We assume that full geometric information of a parking lot is known a priori. Several obstacles are placed in the parking lot, where each obstacle is expressed as a convex polygon P^{o}. The set of all obstacles is denoted by $\mathcal{O}$. The shape of the vehicle is approximated by a bounding convex polygon. The bounding polygon of the vehicle at the configuration x is denoted by $P^{\mathrm{v}}(x)$. The curvature of a trajectory must not exceed the maximum allowable curvature $C^{\max}$:

$$|u(s)| \leq C^{\max} \tag{2}$$

The parameter $C^{\max}$ is determined by the wheelbase and the maximum steering angle of the vehicle. Moreover, the vehicle should not collide with any obstacle. That is,

$$P^{\mathrm{v}}(x(s)) \cap P^{\mathrm{o}} = \varnothing \tag{3}$$

must hold for all $P^{\mathrm{o}} \in \mathcal{O}$.

Based on this setup, a general parking trajectory planning problem can be formulated as follows.

Problem 1 (Parking Trajectory Planning): Given a start configuration $\mathsf{x}^{\mathrm{s}} \in X$, a goal configuration $\mathsf{x}^{\mathrm{g}} \in X$, find $x(s)$, $u(s)$, and $v(s)$ ($s \in [0, S]$) that satisfies the vehicle kinematics (1) and the curvature limit (2) for all $s \in [0, S]$, and the collision avoidance (3) for all $P^{\mathrm{o}} \in \mathcal{O}$, $s \in [0, S]$.

Since this problem is hard to solve directly, we formulate a more specific planning problem in Subsection II-D, after introducing guidelines in Subsection II-B and explicit trajectory representation in Subsection II-C.

B. Guidelines

Several *guidelines* are placed in the parking lot. These guidelines are straight line segments that are used for guiding the vehicle from its initial configuration to a specified parking space. A guideline is denoted by p (it is in serif font not to be confused with vehicle position p) and the set of guidelines

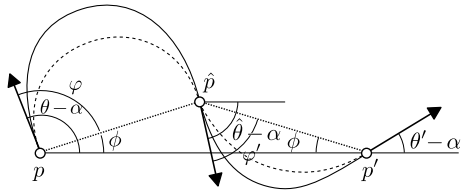


Fig. 1. Illustration of trajectory representation. Clothoid curves are depicted by solid lines and arcs by dashed lines.

is denoted by $\mathcal{P}$. Each guideline $\mathsf{p} \in \mathcal{P}$ is defined by a tuple $(c_\mathsf{p}, l_\mathsf{p}, \theta_\mathsf{p})$, where c_p, l_p, and θ_p denote the center, the length, and the orientation of the guideline p. When the vehicle is on a guideline, its orientation must match the orientation of that guideline. The set of configurations of the vehicle on a guideline p is expressed as the image of a function $f_\mathsf{p} : [0,1] \mapsto X$ defined as

$$f_\mathsf{p}(\mathsf{v}) = \begin{bmatrix} c_\mathsf{p} + l_\mathsf{p}(\mathsf{v} - \frac{1}{2}) \begin{bmatrix} \cos\theta_\mathsf{p} \\ \sin\theta_\mathsf{p} \end{bmatrix} \\ \theta_\mathsf{p} \end{bmatrix} \tag{4}$$

The parameter v determines the position of the vehicle on the guideline whereas the orientation of the vehicle is fixed to θ_p. Apart from this definition, one can consider other parametrization of configurations such as viapoints, which was used in [11]. Moreover, connectivity of the guidelines is defined by a set $\mathcal{L} \subseteq \mathcal{P} \times \mathcal{P}$. The vehicle is allowed to move from p to p' directly if and only if $(\mathsf{p}, \mathsf{p}') \in \mathcal{L}$. Note that $\mathcal{L}$ may include self-loops; $(\mathsf{p}, \mathsf{p}) \in \mathcal{L}$. This allows a vehicle to change its position on a guideline before moving to another guideline.

C. Trajectory Representation

A number of different types of curves have been used in the existing studies. Examples are combination of arcs and straight lines [12], [13], quintic polynomials [14], [15], η^3 polynomials [16], and solutions to sinusoidal steering input [17]. The method proposed in this paper uses curves composed of clothoids or arcs. The curvature of a clothoid changes linearly and takes 0 at each endpoint. The vehicle can therefore track a clothoid by smooth steering. A limitation of clothoid is that it cannot express curves having rapid change of curvature, which is often required for parking. An arc has constant curvature and therefore it is suited to expressing steep turns. Its disadvantage is that the curvature may be discontinuous at endpoints and this may cause non-smoothness of steering. By supporting both clothoids and arcs, the planner is able to take the benefit of these two types of curves.

Trajectories are classified into two major types: *forward* and *reverse*. Moreover, each major type is divided into two minor types: *clothoid* and *arc*. Thus there are four different types for total: forward-clothoid, forward-arc, reverse-clothoid, and reverse-arc.

The derivation of a forward trajectory is illustrated in Fig. 1. Let $\alpha = \tan^{-1}((p'_\mathrm{y} - p_\mathrm{y})/(p'_\mathrm{x} - p_\mathrm{x}))$ be the angle of the line p-p'. The figure is rotated by $-\alpha$ so that p and p' lay on the same horizontal line. First, draw two lines, one from p in a direction denoted by ϕ, and another from p' in the direction $-\phi+\pi$. Let $\hat{p}$ be the point of intersection of these two lines. Moreover, let $\hat{\theta}$ be the tangential direction of a trajectory at $\hat{p}$. It can be shown that if ϕ and $\hat{\theta}$ are defined as $\phi = (\theta - \theta')/4$ and $\hat{\theta} = -(\theta + \theta')/2$, respectively, then for each of the two line segments, p-$\hat{p}$ and $\hat{p}$-p', the tangential direction at the endpoints with respect to the line segment have the same magnitude and the opposite sign. By utilizing this property, one can define two continuous curves, one connecting p and $\hat{p}$, and the other connecting $\hat{p}$ and p', where the type of the curve is either clothoid or arc depending on the minor type of the trajectory. A reverse trajectory is simply obtained by flipping the moving direction of a forward trajectory.

There are some singular cases in which some trajectory type is undefined in the way described above. First, neither forward nor reverse trajectory can be defined when $p = p'$, $\theta \neq \theta'$. Second, forward trajectory cannot be defined if $\theta - \alpha = \theta' - \alpha = \pm\pi$. Third, reverse trajectory cannot be defined if $\theta - \alpha = \theta' - \alpha = 0$. As explained above, for a pair of initial and terminal configurations and a label specifying the trajectory type, a continuous trajectory connecting the two configurations is uniquely determined, except for the singular cases.

Note that the state roadmap method proposed in this paper is not limited to this specific trajectory representation, but it requires that a curve connecting two configurations must deform smoothly with respect to continuous variation of end configurations. This is because the feasibility checking mechanism of our method requires constraint error functions to be continuous functions.

D. Guideline-based Formulation of Parking Trajectory Planning

Consider a tuple $(\mathsf{p}, \mathsf{v}, \mathsf{p}', \mathsf{v}', \mathsf{w})$ where $(\mathsf{p}, \mathsf{p}') \in \mathcal{L}$, $\mathsf{v} \in [0,1]$, $\mathsf{v}' \in [0,1]$, and $\mathsf{w} \in W$. Let us call this tuple a *transition*. A transition defines a trajectory segment connecting two configurations $f_\mathsf{p}(\mathsf{v})$ and $f_{\mathsf{p}'}(\mathsf{v}')$ with the trajectory type w. Moreover, a *path* is defined as a series of transitions $\{(\mathsf{p}_i, \mathsf{v}_i, \mathsf{p}'_i, \mathsf{v}'_i, \mathsf{w}_i)\}_{[1:N]}$ such that $\mathsf{p}'_i = \mathsf{p}_{i+1}$ and $\mathsf{v}'_i = \mathsf{v}_{i+1}$ for all $i \in [1 : N-1]$.

Every trajectory segment defined by a transition must satisfy a set of constraints. Detailed description of constraints is given in the Appendix. A constraint is denoted by a symbol c, and the set of all constraints is denoted by $\mathcal{C}$. A transition $(\mathsf{p}, \mathsf{v}, \mathsf{p}', \mathsf{v}', \mathsf{w})$ is *feasible* with respect to a constraint $c \in \mathcal{C}$ if the trajectory segment defined by this transition satisfies c, or otherwise it is *infeasible* with respect to c. A path is feasible with respect to c if all of its transitions are feasible with respect to c, or otherwise it is infeasible. Moreover, a *feasible set* $\mathcal{F}(\mathsf{p}, \mathsf{p}', \mathsf{w}, c)$ is a set of $(\mathsf{v}, \mathsf{v}') \in [0,1]^2$ such that a transition $(\mathsf{p}, \mathsf{v}, \mathsf{p}', \mathsf{v}', \mathsf{w})$ is feasible with respect to c.

Each constraint can be independently made active or inactive. The set of active constraints is denoted by $\mathcal{C}^\mathrm{a} \subseteq \mathcal{C}$. One typical usage of this feature is to take into account the occupancy state of each parking space in path planning. See Section IV for examples. This feature may also be used for handling vehicles having different dimensions and maneuverability. More concretely, one may define multiple curvature

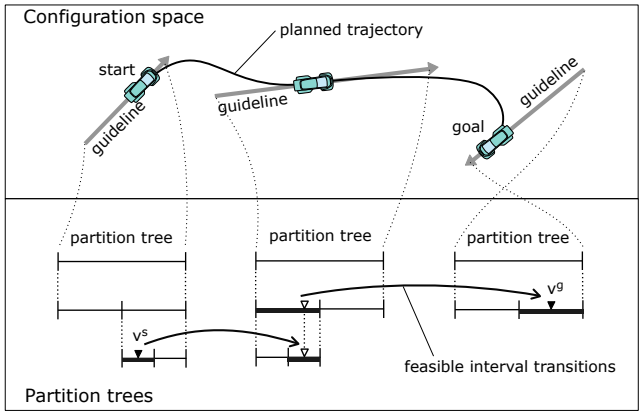


Fig. 2. Illustration of path planning using state roadmap. Each guideline is associated with a partition tree. In the roadmap construction phase, partition trees and feasible transitions (shown by black arrows) between intervals are computed and stored in the state roadmap. In the path planning phase, a series of feasible transitions connecting start and the goal configurations is computed by graph search. By choosing specific intermediate configurations inside the intervals (triangular markers), one can obtain a trajectory that connects the start and the goal configurations.

limit constraints with different values of maximum curvature or define multiple collision avoidance constraints based on different bounding polygons, and activate one of them that is compatible with a specific vehicle. Note that $\mathcal{C}$ must be known at the roadmap construction phase, but $\mathcal{C}^{\mathrm{a}}$ can be specified arbitrarily at the path planning phase.

Now, a guideline-based path planning problem is stated as follows:

Problem 2 (Guideline-based Path Planning): Given a set of guidelines $\mathcal{P}$ and their connections $\mathcal{L}$, $(\mathsf{p}^{\mathrm{s}}, \mathsf{v}^{\mathrm{s}}) \in \mathcal{P} \times [0,1]$, $(\mathsf{p}^{\mathrm{g}}, \mathsf{v}^{\mathrm{g}}) \in \mathcal{P} \times [0,1]$, and a set of active constraints $\mathcal{C}^{\mathrm{a}} \subseteq \mathcal{C}$, find a path $\{(\mathsf{p}_i, \mathsf{v}_i, \mathsf{p}'_i, \mathsf{v}'_i, \mathsf{w}_i)\}_{i\in[1:N]}$ such that

$$(\mathsf{p}_1, \mathsf{v}_1) = (\mathsf{p}^{\mathrm{s}}, \mathsf{v}^{\mathrm{s}}),\ (\mathsf{p}'_N, \mathsf{v}'_N) = (\mathsf{p}^{\mathrm{g}}, \mathsf{v}^{\mathrm{g}}),$$
$$(\mathsf{v}_i, \mathsf{v}'_i) \in \mathcal{F}(\mathsf{p}_i, \mathsf{p}'_i, \mathsf{w}_i, c)\ \ \forall i \in [1:N], c \in \mathcal{C}^{\mathrm{a}}.$$

Note that in Problem 2, guidelines $\mathcal{P}$ and their connections $\mathcal{L}$ are given. In order to derive a complete trajectory planning algorithm to Problem 1, one needs to consider a guideline design problem, which is formulated as follows.

Problem 3 (Guideline Design): Given $\mathsf{x}^{\mathrm{s}} \in X$ and $\mathsf{x}^{\mathrm{g}} \in X$ for which Problem 1 has a solution, find a set of guidelines $\mathcal{P}$ and their connections $\mathcal{L}$, $(\mathsf{p}^{\mathrm{s}}, \mathsf{v}^{\mathrm{s}}) \in \mathcal{P} \times [0,1]$, $(\mathsf{p}^{\mathrm{g}}, \mathsf{v}^{\mathrm{g}}) \in \mathcal{P} \times [0,1]$ such that $\mathsf{x}^{\mathrm{s}} = f_{\mathsf{p}^{\mathrm{s}}}(\mathsf{v}^{\mathrm{s}})$, $\mathsf{x}^{\mathrm{g}} = f_{\mathsf{p}^{\mathrm{g}}}(\mathsf{v}^{\mathrm{g}})$, and Problem 2 with $\mathcal{P}$, $\mathcal{L}$, $(\mathsf{p}^{\mathrm{s}}, \mathsf{v}^{\mathrm{s}})$, and $(\mathsf{p}^{\mathrm{g}}, \mathsf{v}^{\mathrm{g}})$ has a solution.

Developing a method that solves Problem 3 systematically is considered to be very hard. So in this paper, we consider that guideline layout is designed manually. The state roadmap method presented in the next section is a resolution-complete planner for Problem 2.

III. State Roadmap Method

A. Overview of the State Roadmap Method

Problem 2 is a hybrid optimization problem of continuous variables $\{(\mathsf{v}_i, \mathsf{v}'_i)\}$ and discrete variables N, $\{(\mathsf{p}_i, \mathsf{p}'_i)\}$, and $\{\mathsf{w}_i\}$. Such a problem generally requires high computational cost that cannot meet the strict time requirement of automotive applications. One important observation is that parking lot is in most case a quasi-static environment. That is, the geometric shape of the parking lot is fixed, and so long as simultaneous parking of multiple cars is not considered, the occupancy state of each parking space is also unchanging while a vehicle travels toward the desired parking space. This motivates us to utilize pre-computation techniques for reducing online path planning cost.

The abstract procedure of the state roadmap method is illustrated in Fig.2. Each guideline is associated with a partition tree of intervals with multiple resolution levels. In the off-line roadmap construction phase, a transition is defined between one interval to another if the product of these intervals is included in the associated feasible set. This process and bisection of intervals are done in a coarse-to-fine manner so that the computation and storage cost of the state roadmap is reduced. In the path planning phase, a series of transitions that connects the start and the goal configurations is found by means of graph search. Finally, a feasible path, which is a solution to Problem 2, is obtained by selecting specific intermediate configurations inside the intervals. Once a state roadmap is constructed, it can be used repeatedly for many different parking requests. Furthermore, the occupancy of each parking space can be taken into account at the planning phase without regenerating or modifying the state roadmap.

B. Definition of State Roadmap

A partition tree $\mathcal{V}_{\mathsf{p}}$ of $\mathsf{p} \in \mathcal{P}$ is a collection of sub-intervals of $[0,1]$. Each element $[\mathsf{v}] \in \mathcal{V}_{\mathsf{p}}$ may have one or more child elements. The set of child elements of $[\mathsf{v}]$ is denoted by $\mathsf{child}([\mathsf{v}])$, and they form a partition of $[\mathsf{v}]$. An element of $\mathcal{V}_{\mathsf{p}}$ with no child element is called a leaf element. The set of all leaf elements of $\mathcal{V}_{\mathsf{p}}$ forms a partition of $[0,1]$. See Fig. 3 for the illustration of partition trees.

Consider a tuple $\sigma = (\mathsf{p}, [\mathsf{v}], \mathsf{p}', [\mathsf{v}'], \mathsf{w})$, where $(\mathsf{p}, \mathsf{p}') \in \mathcal{L}$, $[\mathsf{v}] \in \mathcal{V}_{\mathsf{p}}$, $[\mathsf{v}'] \in \mathcal{V}_{\mathsf{p}'}$, and $\mathsf{w} \in W$. Let us call this tuple an *interval transition*. The feasibility of an interval transition takes one of the following three states: *feasible*, *infeasible*, and *ambiguous*. For a constraint $c \in \mathcal{C}$, an interval transition $\sigma = (\mathsf{p}, [\mathsf{v}], \mathsf{p}', [\mathsf{v}'], \mathsf{w})$ is:

$$\begin{cases} \text{feasible} & \text{if } [\mathsf{v}] \times [\mathsf{v}'] \subseteq \mathcal{F}(\mathsf{p}, \mathsf{p}', \mathsf{w}, c), \\ \text{infeasible} & \text{if } [\mathsf{v}] \times [\mathsf{v}'] \cap \mathcal{F}(\mathsf{p}, \mathsf{p}', \mathsf{w}, c) = \varnothing, \\ \text{ambiguous} & \text{otherwise} \end{cases}$$

A state roadmap is defined as a tuple $(\{\mathcal{V}_{\mathsf{p}}\}_{\mathsf{p}\in\mathcal{P}}, \{\Sigma_c^{\mathrm{f}}\}_{c\in\mathcal{C}}, \{\Sigma_c^{\mathrm{nf}}\}_{c\in\mathcal{C}}, \{\Sigma_c^{\mathrm{a}}\}_{c\in\mathcal{C}})$. Here, Σ_c^{f} is a set of interval transitions that is feasible with respect to $c \in \mathcal{C}$, whereas Σ_c^{nf} and Σ_c^{a} store infeasible and ambiguous interval transitions, respectively. An *interval path* of a state roadmap is a series of interval transitions $\{\sigma_i\}_{i\in[1:N]} = \{(\mathsf{p}_i, [\mathsf{v}]_i, \mathsf{p}'_i, [\mathsf{v}']_i, \mathsf{w}_i)\}_{i\in[1:N]}$ that satisfies $\sigma_i \in \Sigma_c^{\mathrm{f}}$ for all $i \in [1:N]$ and $c \in \mathcal{C}^{\mathrm{a}}$, $\mathsf{p}'_i = \mathsf{p}_{i+1}$ and $[\mathsf{v}']_i \cap [\mathsf{v}]_{i+1} \neq \varnothing$ for all $i \in [1:N-1]$.

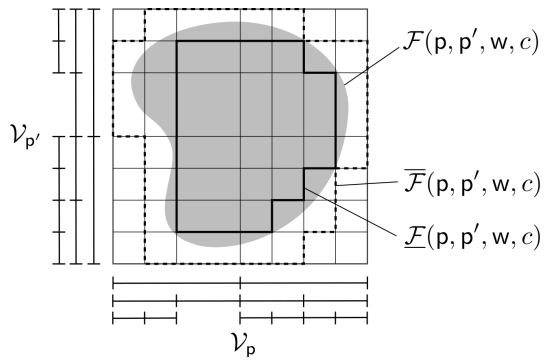


Fig. 3. Inner and outer approximations of a feasible set

An important property of state roadmap is that the sets Σ_c^{f} and Σ_c^{a} define inner and outer approximations of the feasible set in the following manner.

$$\underline{\mathcal{F}}(\mathsf{p},\mathsf{p}',\mathsf{w},c) = \bigcup_{(\mathsf{p},[\mathsf{v}],\mathsf{p}',[\mathsf{v}'],\mathsf{w})\,\in\,\Sigma_c^{\mathrm{f}}} [\mathsf{v}]\times[\mathsf{v}'], \tag{5}$$

$$\overline{\mathcal{F}}(\mathsf{p},\mathsf{p}',\mathsf{w},c) = \bigcup_{(\mathsf{p},[\mathsf{v}],\mathsf{p}',[\mathsf{v}'],\mathsf{w})\,\in\,\Sigma_c^{\mathrm{f}}\cap\Sigma_c^{\mathrm{a}}} [\mathsf{v}]\times[\mathsf{v}'] \tag{6}$$

The inner approximation $\underline{\mathcal{F}}$ is given by the union of $[\mathsf{v}]\times[\mathsf{v}']$ of all feasible interval transitions, while the outer approximation $\overline{\mathcal{F}}$ is given by the union of all feasible and ambiguous ones. Figure 3 illustrates the relation of the feasible set and its inner and outer approximations. It is intuitive from the figure that the gap between the two approximations becomes smaller as the partition becomes finer. Thus we use the following as an index of approximation quality.

$$\begin{aligned}\mathrm{vol}^{\mathrm{a}}(\mathsf{p},\mathsf{p}',\mathsf{w},c) &= \mathrm{vol}(\overline{\mathcal{F}}(\mathsf{p},\mathsf{p}',\mathsf{w},c)) - \mathrm{vol}(\underline{\mathcal{F}}(\mathsf{p},\mathsf{p}',\mathsf{w},c))\\ &= \sum_{(\mathsf{p},[\mathsf{v}],\mathsf{p}',[\mathsf{v}'],\mathsf{w})\in\Sigma_c^{\mathrm{a}}} \mathrm{vol}([\mathsf{v}]\times[\mathsf{v}']).\end{aligned} \tag{7}$$

Since the volume of $[0,1]^2$ is 1, the above index can be seen as the ratio of the volume of the ambiguous region over the entire region $[0,1]^2$. Thus we call this index ambiguity ratio. The algorithm presented next refines the partition trees until the ambiguity ratio becomes smaller than a prescribed constant ϵ for all $(\mathsf{p},\mathsf{p}',\mathsf{w},c)$.

It is intuitively clear that $\underline{\mathcal{F}}(\mathsf{p},\mathsf{p}',\mathsf{w},c)$ converges to $\mathcal{F}(\mathsf{p},\mathsf{p}',\mathsf{w},c)$ in the limit $\epsilon\to 0$. Problem 2 is to find a series of transitions that are included in the feasible set $\mathcal{F}$, while the state roadmap method computes a series of interval transitions included in $\underline{\mathcal{F}}$. From this observation, we can see that the state roadmap method is a resolution complete planner for Problem 2.

C. Construction of State Roadmap

The pseudo-code of the state roadmap generation algorithm is shown in Algorithm 1. The symbol $A \leftarrow B$ means that B is substituted to A, while $A \leftarrow^{\mathrm{op}} B$ is equivalent to $A \leftarrow A\,\mathrm{op}\,B$. For example, $A \leftarrow^{+} B$ is equivalent to $A \leftarrow A + B$. This algorithm iteratively refines the partition trees and generates new interval transitions until the ambiguity ratio (7) becomes smaller than ϵ for all $(\mathsf{p},\mathsf{p}',\mathsf{w},c)$. In line 3, the partition tree $\mathcal{V}_{\mathsf{p}}$ associated with each guideline $\mathsf{p}\in\mathcal{P}$ is initialized. Here, INITIALPARTITION equally partitions $[0,1]$ so that the size of each sub-interval, when map to the associated guideline, is not greater than the initial resolution ρ^{ini}. In lines 4-5, the check queue $\mathcal{Q}$ is initialized. The check queue $\mathcal{Q}$ stores pairs of an interval transition σ and a constraint c for which feasibility check is to be performed. Feasibility check of each (σ,c) in the queue is performed in lines 11-20. The procedure CHECKFEASIBILITY(σ,c) checks the feasibility state of σ with respect to c. The detail of this procedure is explained in the next subsection. In lines 22-34, If an interval transition $\sigma=(\mathsf{p},[\mathsf{v}],\mathsf{p}',[\mathsf{v}'],\mathsf{w})$ is ambiguous and the ambiguity ratio is greater than ϵ for some $c\in\mathcal{C}$, then $[\mathsf{v}]$ and $[\mathsf{v}']$ are bisected, and new interval transitions are created. The procedure BISECT$(\mathsf{p},[\mathsf{v}],\rho)$ bisects $[\mathsf{v}]$ into two sub-intervals if the size of $[\mathsf{v}]$, when mapped to the guideline

Algorithm 1 State Roadmap Construction

1: Input $\mathcal{P},\mathcal{L},W,\mathcal{C},\rho^{\mathrm{ini}},\epsilon$
2: Output $(\{\mathcal{V}_{\mathsf{p}}\}_{\mathsf{p}\in\mathcal{P}},\{\Sigma_c^{\mathrm{f}}\}_{c\in\mathcal{C}},\{\Sigma_c^{\mathrm{nf}}\}_{c\in\mathcal{C}},\{\Sigma_c^{\mathrm{a}}\}_{c\in\mathcal{C}})$
3: $\mathcal{V}_{\mathrm{p}} \leftarrow$ INITIALPARTITION$(\mathsf{p},\rho^{\mathrm{ini}})$ **for each** $\mathsf{p}\in\mathcal{P}$
4: $\mathcal{Q}\leftarrow\{((\mathsf{p},[\mathsf{v}],\mathsf{p}',[\mathsf{v}'],\mathsf{w}),c)\,|$
5: $\quad(\mathsf{p},\mathsf{p}')\in\mathcal{L},[\mathsf{v}]\in\mathcal{V}_{\mathsf{p}},[\mathsf{v}']\in\mathcal{V}_{\mathsf{p}'},\mathsf{w}\in W,c\in\mathcal{C}\}$
6: $l\leftarrow 0$, $\rho\leftarrow\rho^{\mathrm{ini}}$
7: **while** $\mathcal{Q}\neq\varnothing$ **do**
8: **for each** $(\mathsf{p},\mathsf{p}',\mathsf{w},c)\in\mathcal{L}\times W\times\mathcal{C}$ **do**
9: $\mathrm{vol}^{\mathrm{a}}(\mathsf{p},\mathsf{p}',\mathsf{w},c)\leftarrow 0$
10: **end for**
11: **for** $(\sigma,c)\in\mathcal{Q}$ **do**
12: CHECKFEASIBILITY(σ,c)
13: **if** σ is feasible w.r.t. c **then**
14: $\Sigma_c^{\mathrm{f}}\leftarrow\Sigma_c^{\mathrm{f}}\cup\sigma$
15: **else if** σ is infeasible w.r.t. c **then**
16: $\Sigma_c^{\mathrm{nf}}\leftarrow\Sigma_c^{\mathrm{nf}}\cup\sigma$
17: **else**
18: $\mathrm{vol}^{\mathrm{a}}(\mathsf{p},\mathsf{p}',\mathsf{w},c)\leftarrow^{+}\mathrm{vol}([\mathsf{v}]\times[\mathsf{v}'])$
19: **end if**
20: **end for**
21: $\mathcal{Q}'\leftarrow\varnothing$
22: **for** $(\sigma=(\mathsf{p},[\mathsf{v}],\mathsf{p}',[\mathsf{v}'],\mathsf{w}),c)\in\mathcal{Q}$ **do**
23: **if** σ is ambiguous w.r.t. c **then**
24: **if** $\mathrm{vol}^{\mathrm{a}}(\mathsf{p},\mathsf{p}',\mathsf{w},c)\le\epsilon$ **then**
25: $\Sigma_c^{\mathrm{a}}\leftarrow\Sigma_c^{\mathrm{a}}\cup\sigma$
26: **else**
27: $\mathsf{child}([\mathsf{v}])\leftarrow$ BISECT$(\mathsf{p},[\mathsf{v}],\rho)$
28: $\mathsf{child}([\mathsf{v}'])\leftarrow$ BISECT$(\mathsf{p}',[\mathsf{v}'],\rho)$
29: **for each** $\{[\hat{\mathsf{v}}],[\hat{\mathsf{v}}']\}\in\mathsf{child}([\mathsf{v}])\times\mathsf{child}([\mathsf{v}'])$ **do**
30: $\mathcal{Q}'\leftarrow\mathcal{Q}'\cup((\mathsf{p},[\hat{\mathsf{v}}],\mathsf{p}',[\hat{\mathsf{v}}'],\mathsf{w}),c)$
31: **end for**
32: **end if**
33: **end if**
34: **end for**
35: $\mathcal{Q}\leftarrow\mathcal{Q}'$
36: $l\leftarrow l+1$, $\rho\leftarrow\rho/2$
37: **end while**

p, is greater than ρ. Otherwise, it simply duplicates $[\mathsf{v}]$. After bisection, newly created interval transitions are inserted in the check queue. The algorithm terminates when the check queue becomes empty.

D. Conservative Evaluation of Constraint Error

Each constraint is quantified by a *constraint error function*. A constraint error function of a constraint $c \in \mathcal{C}$ is a scalar-valued function E_c which takes a negative value if the constraint is satisfied, or a positive value if the constraint is violated. In other words, the sub-level set of E_c is the feasible set of c. The concrete definition of each type of constraints is given in the Appendix. For an interval transition $\sigma = (\mathsf{p}, [\mathsf{v}], \mathsf{p}', [\mathsf{v}'], \mathsf{w})$ and a constraint $c \in \mathcal{C}$, consider the maximum and the minimum values of E_c taken by transitions represented by σ:

$$\begin{aligned} \underline{E}_c(\sigma) &= \min_{\mathsf{v}\in[\mathsf{v}],\mathsf{v}'\in[\mathsf{v}']} E_c(\mathsf{p},\mathsf{v},\mathsf{p}',\mathsf{v}',\mathsf{w}), \\ \overline{E}_c(\sigma) &= \max_{\mathsf{v}\in[\mathsf{v}],\mathsf{v}'\in[\mathsf{v}']} E_c(\mathsf{p},\mathsf{v},\mathsf{p}',\mathsf{v}',\mathsf{w}). \end{aligned} \tag{8}$$

An interval transition σ is feasible if $\overline{E}_c(\sigma) < 0$, infeasible if $\underline{E}_c(\sigma) > 0$, or otherwise ambiguous. It is generally hard to compute the exact values of $\underline{E}_c$ and $\overline{E}_c$. Therefore, conservative approximation of these functions are computed by evaluating the constraint error at the extreme points of $[\mathsf{v}]\times[\mathsf{v}']$. Let Θ be the set of four extreme points of $[\mathsf{v}] \times [\mathsf{v}']$, and let δ be the diameter of $[\mathsf{v}] \times [\mathsf{v}']$. Then an under-estimate of $\underline{E}_c$ and an over-estimate of $\overline{E}_c$ are given by

$$\begin{aligned} &\min_{(\mathsf{v},\mathsf{v}')\in\Theta} E_c(\mathsf{p},\mathsf{v},\mathsf{p}',\mathsf{v}',\mathsf{w}) - L_c\delta/2, \\ &\max_{(\mathsf{v},\mathsf{v}')\in\Theta} E_c(\mathsf{p},\mathsf{v},\mathsf{p}',\mathsf{v}',\mathsf{w}) + L_c\delta/2, \end{aligned} \tag{9}$$

respectively. Here, L_c is the Lipschitz constant of $E_c(\mathsf{p},\mathsf{v},\mathsf{p}',\mathsf{v}',\mathsf{w})$ seen as a function of (v,v'). In this manner, feasibility cheking can be executed in constant time regardless of the length of intervals. For longer intervals, however, checking becomes more conservative and thus more likely to judge them ambiguous.

E. Trajectory Planning Using State Roadmap

Thanks to the directed graph structure of state roadmap, one can find a path connecting a start configuration x^{s} and a goal configuration x^{g} using a graph search algorithm. The first step is to define a set of start nodes $\mathcal{N}^{\mathrm{s}}$ and the set of goal nodes $\mathcal{N}^{\mathrm{g}}$. These sets must be defined differently according to the configuration of x^{s} and x^{g} relative to the guidelines. In the 'on-guideline' case, x^{s} (x^{g}) is on one of the guidelines. In this case, $\mathcal{N}^{\mathrm{s}}$ ($\mathcal{N}^{\mathrm{g}}$) is defined as follows.

$$\begin{aligned} \mathcal{N}^{\mathrm{s}} &= \{(\mathsf{p},[\mathsf{v}]) \,|\, \exists \mathsf{v} \in [\mathsf{v}] \text{ s.t. } f_{\mathsf{p}}(\mathsf{v}) = \mathsf{x}^{\mathrm{s}}\}, \\ \mathcal{N}^{\mathrm{g}} &= \{(\mathsf{p},[\mathsf{v}]) \,|\, \exists \mathsf{v} \in [\mathsf{v}] \text{ s.t. } f_{\mathsf{p}}(\mathsf{v}) = \mathsf{x}^{\mathrm{g}}\}. \end{aligned}$$

In the 'off-guideline' case, x^{s} (x^{g}) is not on any of the guidelines. In this case, $\mathcal{N}^{\mathrm{s}}$ ($\mathcal{N}^{\mathrm{g}}$) is defined as

$$\begin{aligned} \mathcal{N}^{\mathrm{s}} &= \{(\mathsf{p}^{\mathrm{s}},[0,1])\}, \\ \mathcal{N}^{\mathrm{g}} &= \{(\mathsf{p}^{\mathrm{g}},[0,1])\}. \end{aligned}$$

Here, p^{s} (p^{g}) is a special guideline generated upon path planning that has zero length and whose position and orientation are matched to x^{s} (x^{g}). At the same time, connections between p^{s}, p^{g} with other guidelines $\mathsf{p} \in \mathcal{P}$ are temporarily generated. Next, Algorithm 1 is executed for these connections.

Typical graph search algorithms require the following functions to be implemented: pre, post, and cost. The functions pre and post return the predecessor set and the successor set of a specified graph node, respectively, and the function cost returns the cost of transition from one node to another. Each pair $(\mathsf{p},[\mathsf{v}])$, $\mathsf{p} \in \mathcal{P}$, $[\mathsf{v}] \in \mathcal{V}_{\mathsf{p}}$ is treated as a graph node, for which the pre and post sets are defined as follows.

$$\begin{aligned} \mathsf{pre}((\mathsf{p}',[\mathsf{v}'])) = \{(\mathsf{p},[\mathsf{v}]) \,|\, \exists\sigma = (\mathsf{p},[\mathsf{v}],\mathsf{p}',[\hat{\mathsf{v}}],\mathsf{w}) \in \Sigma^{\mathrm{f}}_{\mathcal{C}^{\mathrm{a}}} \\ \text{s.t. } [\mathsf{v}'] \cap [\hat{\mathsf{v}}] \neq \varnothing\}, \\ \mathsf{post}((\mathsf{p},[\mathsf{v}])) = \{(\mathsf{p}',[\mathsf{v}']) \,|\, \exists\sigma = (\mathsf{p},[\hat{\mathsf{v}}],\mathsf{p}',[\mathsf{v}'],\mathsf{w}) \in \Sigma^{\mathrm{f}}_{\mathcal{C}^{\mathrm{a}}} \\ \text{s.t. } [\mathsf{v}] \cap [\hat{\mathsf{v}}] \neq \varnothing\}. \end{aligned} \tag{10}$$

Here, $\Sigma^{\mathrm{f}}_{\mathcal{C}_{\mathrm{a}}} = \bigcap_{c\in\mathcal{C}^{\mathrm{a}}} \Sigma^{\mathrm{f}}_c$. Note that the computation of these sets requires very small cost, because Σ^{f}_c for each $c \in \mathcal{C}$ is pre-computed and stored in the state roadmap. Exceptions are transitions from p^{s} and transitions to p^{g} for the off-guideline case. The feasibility of these transitions must be checked in the path planning phase as mentioned above.

The transition cost from $(\mathsf{p},[\mathsf{v}])$ to $(\mathsf{p}',[\mathsf{v}'])$ is defined as

$$\mathsf{cost}((\mathsf{p},[\mathsf{v}]),(\mathsf{p}',[\mathsf{v}']),\mathsf{w}) = \max_{\mathsf{v}\in[\mathsf{v}],\mathsf{v}'\in[\mathsf{v}']} S(\mathsf{p},\mathsf{v},\mathsf{p}',\mathsf{v}',\mathsf{w}) \tag{11}$$

where S gives the length of a curve defined by the transition $(\mathsf{p},\mathsf{v},\mathsf{p}',\mathsf{v}',\mathsf{w})$. The $\max$ operation in the right-hand-side indicates that the travel distance is over-estimated by the maximum possible value over all transitions that can be instantiated from the interval transition $(\mathsf{p},[\mathsf{v}],\mathsf{p}',[\mathsf{v}'],\mathsf{w})$.

Having implemented $\mathcal{N}^{\mathrm{s}}$, $\mathcal{N}^{\mathrm{g}}$, pre, post, and cost, any graph search algorithm that requires these routines can be used. Among them, the bidirectional search algorithm is used in this paper. It grows two search trees, one from the start node and the other from the goal node, until these two trees meet at some node. The bidirectional search is known to find a path faster than other one-directional search methods such as Dijkstra's method.

If a feasible interval path $\{(\mathsf{p}_i,[\mathsf{v}]_i,\mathsf{p}'_i,[\mathsf{v}']_i,\mathsf{w}_i)\}_{i\in[1:N]}$ is found successfully, a feasible path $\{(\mathsf{p}_i,\mathsf{v}_i,\mathsf{p}'_i,\mathsf{v}'_i,\mathsf{w}_i)\}_{i\in[1:N]}$ is instantiated from it by setting

$$\begin{aligned} &\mathsf{v}_1 \in [\mathsf{v}]_1, \mathsf{v}'_N \in [\mathsf{v}']_N, \\ &\mathsf{v}'_i = \mathsf{v}_{i+1} \in [\mathsf{v}']_i \cap [\mathsf{v}]_{i+1} \ \forall i \in [1:N-1]. \end{aligned}$$

Note that any choice of $\mathsf{v}'_i (= \mathsf{v}_{i+1})$ within the range $[\mathsf{v}']_i \cap [\mathsf{v}]_{i+1}$ makes feasible transitions. We simply choose the middle point of $[\mathsf{v}']_i \cap [\mathsf{v}]_{i+1}$.

IV. Evaluation

A. Numerical Example

This section shows simulation results. All computation was done on a computer with Intel Xeon CPU (3.5 GHz) and 16GB memory. The algorithm was implemented as a single-thread C++ program and run on Windows operating system.

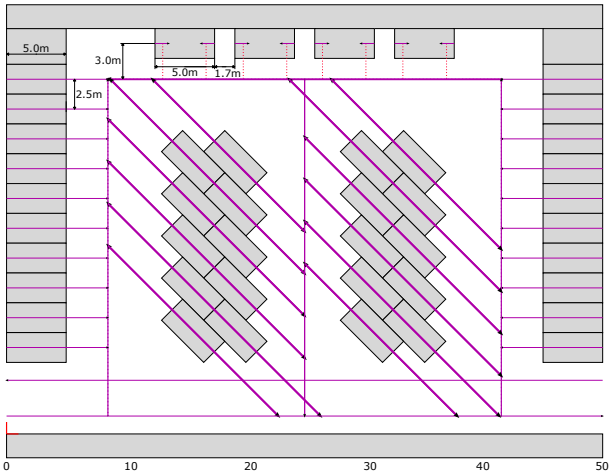


Fig. 4. Parking environment for simulation. Gray rectangles are obstacles. Purple lines are guidelines, where each of the thicker lines depict a pair of guidelines pointing to the opposite directions. Red dotted lines indicate connections between guidelines in the parallel parking spaces and the nearby long guidelines.

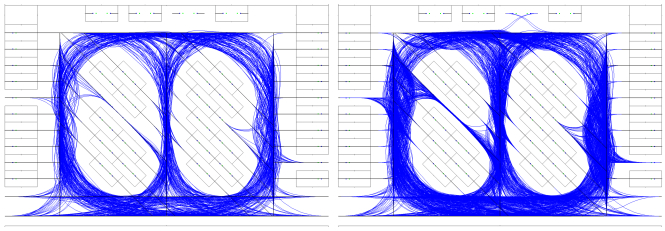
TABLE I
PARAMETER SETTING

description	symbol	value
rear axle to front	—	3.6 [m]
rear axle to rear end	—	0.9 [m]
car width	—	1.7 [m]
tread	—	1.7 [m]
wheelbase	—	2.5 [m]
minimum turning radius	—	5.2 [m]
equivalent maximum curvature	$C^{\max}$	0.27 [m^{-1}]
singular position margin	$L^{\min}$	0.1 [m]
singular angle margin	$\varphi^{\max}$	90 [deg]
discretization for collision check	N^{check}	20
initial resolution	ρ^{ini}	8[m]
cutoff ambiguity ratio	ϵ	0.5, 0.2

A parking lot environment used for simulation evaluation is shown in Fig. 4. There are 48 obstacles, 130 guidelines, and 666 connections of guidelines in this environment. The connection set $\mathcal{L}$ is defined as follows. First, all pairs of intersecting guidelines are included in $\mathcal{L}$. Second, self-loops for all guidelines are included. Third, guidelines for parallel parking spaces are connected with the nearby guidelines, as shown in Fig. 4.

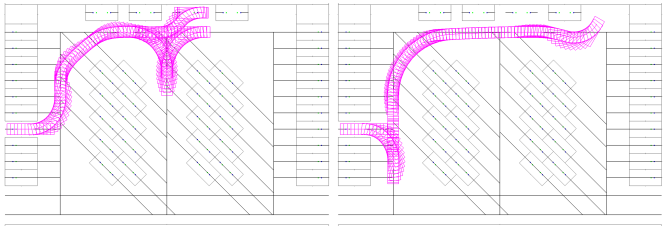
The parameter setting is shown in Table I. The specification of the vehicle considered here is based on conventional cedan-type car. The minimum turning radius of a typical cedan-type car is 5.2[m] if it is measured by the distance from the turning center to the outer front wheel. Assuming that the tread is 1.7[m] and the wheelbase is 2.5[m], the equivalent distance from the turning center to the center of the rear axle is approximately 3.7[m], which gives the maximum curvature of $0.27[\mathrm{m}^{-1}]$.

The generated state roadmap with $\epsilon = 0.2$ is shown in Figs. 5(a)(b). Here, feasible interval transitions at different resolution levels are shown separately. For visualization, each interval transition is represented by a trajectory that is instantiated from that interval transition by choosing the center values of the intervals. The number of intervals and interval transitions are summarized in Table III.

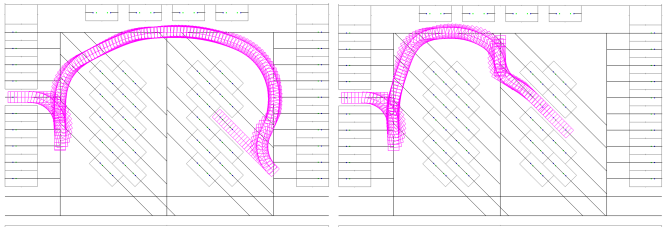


(a) 1[m] resolution (b) 1/4[m] resolution

Fig. 5. Visualization of generated state roadmap ($\epsilon = 0.2$). Gray boxes depict active obstacles, and black lines depict guidelines. Blue curves depict feasible transitions with respect to active constraints. Some obstacles are set as inactive to show some characteristic transitions.



(a) parallel parking (b) parking from an off-guideline configuration



(c) detouring occupied parking spaces (d) traveling across vacant spaces

Fig. 6. Parking trajectories from different initial configurations generated from a state roadmap with $\epsilon = 0.2$. (c) and (d) show parking trajectories for the same start and goal configurations with different active/inactive states of obstacles. In (c) the vehicle takes a longer path to avoid occupied parking spaces, while in (d) the vehicle takes a shorter path by traveling across a vacant parking space.

Trajectory planning results from different initial configurations are shown in Figs. 6(a)-(d). As shown in these figures, using the same state roadmap, parking trajectories for various initial and goal configurations under different active/inactive states of obstacles can be generated.

Table II summarizes several computational characteristics of the proposed method in different settings and those of well-known path planners, the RRT planner and the State Lattice planner [7]. In the 'uniform' setting, roadmaps are generated by partitioning the guidelines in uniform resolutions, 1[m] and 0.5[m], while in the 'multi-res' setting, roadmaps are generated by Algorithm 1 with $\epsilon = 0.5$ and $\epsilon = 0.2$. The implementation of the RRT planner used here is fairly basic except that it grows two search trees, one forward from the initial configuration and another backward from the goal configuration. It outputs a path when the two search trees meet at some configuration. The implementation of the state-lattice

TABLE II
SUMMARY OF COMPUTATION COST, DATA SIZE, AND SUCCESS RATE OF PATH PLANNERS

		uniform		multi-res		RRT	Lattice A*
		1[m]	$1/2$[m]	$\epsilon = 0.5$	$\epsilon = 0.2$		
Roadmap generation time [s]		150	609	9	32	-	-
Roadmap data size [MB]		103	376	61	100	-	-
Planning time (on guideline) [ms]	ave.	30.7	136.9	4.3	26.1	-	-
	max.	126.0	457.5	17.7	88.9	-	-
Planning time (off guideline) [ms]	ave.	379.5	839.4	129.7	222.9	14.5	446.6
	max.	541.4	1322.3	174.2	333.4	> 10[s]	> 10[s]
Success rate (on guideline) [%]		100	100	71.5	100	-	-
Success rate (off guideline) [%]		96.9	97.7	79.6	96.7	99.6	98,3
Cost of planned path (on guideline)	ave.	65.0	55.8	109.6	69.1	-	-
Cost of planned path (off guideline)	ave.	59.6	53.0	94.8	64.7	128.6	81.2

TABLE III
SIZE OF STATE ROADMAPS GENERATED WITH $\epsilon = 0.5$ AND $\epsilon = 0.2$

resolution	# of intervals		# of interval transitions	
	$\epsilon = 0.5$	$\epsilon = 0.2$	$\epsilon = 0.5$	$\epsilon = 0.2$
8[m]	240	240	26328	26328
4[m]	172	172	14038	16480
2[m]	336	336	55316	64132
1[m]	664	664	54538	167722
1/2[m]	16	1328	256	284500
1/4[m]	32	1612	512	51520
1/8[m]	64	64	1280	1408

planner is based on [7]. The translational resolution of the lattice is set as $1/4[m]$, whereas the angular resolution is set as $\pi/8$. The control set is designed so that the out-degree of each lattice node is atmost 70. The state lattice planner performs forward A* search from the initial configuration, until the nearest node to the goal configuration is found. The technique of weighted A* search is used to find a feasible but possiblly suboptimal path quickly. The RRT and the state lattice planners do not make use of guidelines.

It is observed that the 'multi-res' cases require smaller computation time for roadmap generation compared to the 'uniform' cases, while setting smaller ϵ requires greater computation time. The multi-resolution setup also contributes to reducing the data size of generated state roadmaps.

Computation time and success rate of trajectory planning are also compared. Planning time and success rate listed in the table are calculated from the results of 1000 trajectory planning queries with randomly sampled initial and goal configurations. The active/inactive state of each obstacle was also switched at random with 50% probability at every trial. In the 'on guideline' case, both initial and goal configurations are randomly sampled from the guidelines. In the 'off guideline' case, initial configurations are randomly sampled from the entire collision-free configuration space, while the goal configurations are sampled from the guidelines. Success rate is the ratio of queries for which a planner successfully found a feasible trajectory over all 1000 queries. If a planner spent more than 10 seconds to find a path in a single trial, then that trial was considered unsuccessful. The multi-resolution roadmap planner with $\epsilon = 0.2$ achieved 100.0% success rate for the 'on-guideline' case, and 96.7% for the 'off-guideline' case. In most of the unsuccessful cases, the initial configuration were nearly orthogonal to the nearest guideline, and the vehicle was unable to steer itself to any of the nearby guidelines without violating the curvature limit and collision avoidance constraints. The multi-resolution roadmap planner showed small (less than 30[ms] in average) planning time for the 'on-guideline' cases. For the 'off-guideline' cases, it showed greater planning time because transitions from the initial configuration needs on-line feasibility checking. The planning time of the roadmap planner is also stable in the sense that that worst-case planning time is not significantly greater than the average. The RRT planner showed 99.6% success rate, thanks to its probabilistic completeness. Its planning time was small in average but highly unstable; for some queries it took considerably larger planning time than the average. In fact, the success rate is not 100% because in a few trials, the planning time exceeded 10 seconds. Moreover, the cost of paths produced by the RRT planner were generally greater than those produced by the roadmap planners. This is not surprising because the RRT planner outputs a first feasible path found regardless of its cost. The state lattice planner took long planning time in average, even though a weighted heuristic was used. This is mainly because the heuristic does not accurately estimate the true cost to the goal, especially when feasible paths to the goal are complicated due to obstacles inbetween.

The proposed roadmap-based planner needs further improvement for raising the success rate for the 'off-guideline' cases to 100%. One possible idea is to make use of a probabilistically complete planners such as RRT for generating a partial trajectory from the initial configuration to a first guideline. Another idea worth investigating is to place a temporary guideline at plan-time along the initial configuration of the vehicle. This is expected to expand the reachable range of the vehicle and thereby increase the possibility to find a connected path to the desired parking space.

B. Test Results on an Electric Vehicle

An experimental autonomous parking system was constructed for a small-sized electric vehicle. The parking lot environment for this experiment is shown in Fig. 7, together with the recorded trajectories of the vehicle in three parking trials. The dimensions of this parking lot is set to fit the size of the vehicle which is much smaller than the conventional

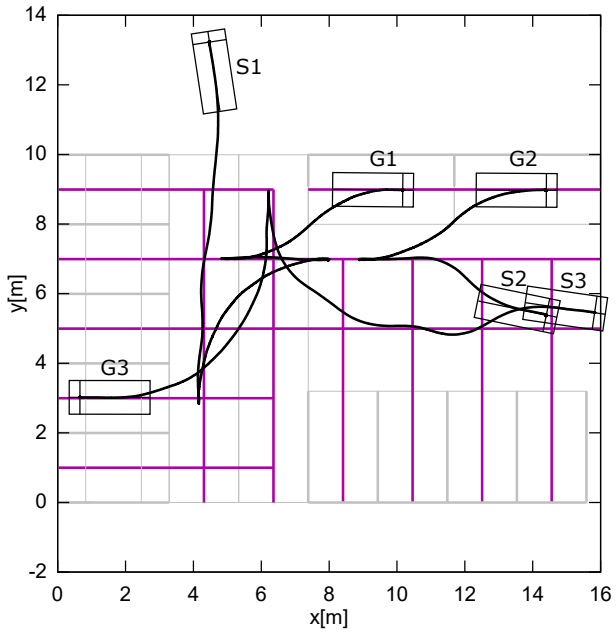


Fig. 7. Parking results of a real vehicle. Gray lines are obstacles and purple lines are guidelines. The actual traces of the vehicle obtained from GNSS are shown in black lines. The start and goal configurations of three trials are marked Si and Gi ($i = 1, 2, 3$).

cedan cars considered in the simulation experiment. A state roadmap for this parking lot was computed off-line and stored in the memory of the base-station computer. The base station receives the global position/heading of the vehicle via wireless network, generates a parking path using the state roadmap, and sends back the generated path to the vehicle's controller. In Fig. 7, the actual traces of the vehicle obtained from the GNSS receiver are depicted by solid lines. For every trial, the starting configuration of the vehicle was set apart from the guidelines, while the goal parking space was manually selected. The planner successfully generated parking paths for all trials.

V. Conclusions

In this paper, a new parking trajectory planning method that uses multi-resolution state roadmap was presented. The generation procedure of state roadmap is automated except for the design of guidelines. Once a state roadmap is generated, it can be used for planning parking trajectories starting from different initial configurations, not necessarily on the guidelines. Moreover, the combination of active constraints can be set arbitrarily at plan-time, and feasible trajectories that satisfy active constraints is planned accordingly. It was confirmed in numerical experiments that feasible trajectories starting from randomly selected initial configurations are obtained at the success ratio close to 100%, if the ambiguity ratio is set small enough. From the theoretical point of view, however, it is important to clarify the relation between the layout of guidelines, the ambiguity ratio, and the success rate of trajectory planning. Topics for future research includes application to more complex parking situations in which multiple cars perform parking simultaneously.

Acknowledgements

The authors would like to thank Hiroshi Fuji and Jingyu Xiang for their contribution during the early stage of this work.

References

[1] Lefebvre, Lamiraus, Pradalier, Fraichard: "Obstacle Avoidance for Car-like Robots: Integration and Experimentation on Two Robots", ICRA 2004.
[2] C. Pradalier, J. Hermosillo, C. Koike, C. Braillon, P. Bessiere and C. Laugier: "The Cycab: A Car-like Robot Navigating Autonomously and Safely Among Pedestrians", Robotics and Autonomous Systems, Vol.50, No.1, pp.51-68, 2005.
[3] B. Muller, J. Deutscher, and S. Grodde: "Continuous Curvature Trajectory Design and Feedforward Control for Parking a Car", IEEE Transactions on Control Systems Technology, Vol.15, No.3, pp.541-553, 2007.
[4] R. Kummerle, D. Hahnel, D. Dolgov, S. Thrun, W. Burgard: "Autonomous Driving in a Multi-Level Parking Structure", IEEE International Conference on Robotics and Automation, pp.3395-3400, 2009.
[5] L. Han, H. Do, S. Mita: "Unified Path Planner for Parking an Autonomous Vehicle based on RRT", IEEE International Conference on Robotics and Automation, pp.5622-5627, 2011.
[6] Q.H. Do, S. Mita, K. Yoneda: "A Practical and Optimal Path Planning for Autonomous Parking Using Fast Marching Algorithm and Support Vector Machine", IEICE Transactions on Information and Systems, Vol.E96-D, No.12, pp.2795-2804, 2013.
[7] M. Likhachev, D. Ferguson: "Planning Long Dynamically Feasible Maneuvers for Autonomous Vehicles", International Journal of Robotics Research, Vol.28, No.8, pp.933-945, 2009.
[8] J. Ziegler et al: "Making Bertha Drive - An Autonomous Journey on a Historic Route", IEEE Intelligent Transportation Systems Magazine, Vol.6, No.2, pp.8-20, 2014.
[9] C. Pradalier, S. Vaussier and P. Corke: "Path Planning for a Parking Assistance System: Implementation and Experimentation", Australasian Conference on Robotics and Automation (ACRA 2005), 2005.
[10] Y. Seo, C. Urmson, D. Wettergreen, J. Lee: "Building Lane Graphs for Autonomous Parking", IEEE/RSJ International Conference on Intelligent Robots and Systems, pp.6052-6057, 2010.
[11] H. Fuji, J. Xiang, Y. Tazaki and T. Suzuki: "Trajectory Planning for Automated Parking Using Multi-resolution State Roadmap Considering Non-holonomic Constraints": IEEE Intelligent Vehicles Symposium, pp.407-413, 2014.
[12] G. Casalino, A. Bicchi and C. Santilli: "Planning Shortest Bounded-Curvature Paths for a Class of Nonholonomic Vehicles Among Obstacles", Journal of Intelligent and Robotic Systems, Vol.16, No.4, pp. 387-405, 1996.
[13] M.F. Hsieh and U. Ozguner: "A Parking Algorithm for an Autonomous Vehicle", IEEE Intelligent Vehicles Symposium, pp.1155-1160, 2008.
[14] J. Xu, G. Chen, M. Xie: "Vision-Guided Automatic Parking for Smart Car", IEEE Intelligent Vehicles Symposium, pp.725-730, 2000.
[15] D. Lyon: "A Min-Time Analysis of Three Trajectories with Curvature and Nonholonomic Constraints Using a Parallel Parking Criterion", JSME International Journal Series C Mechanical Systems, Machine Elements and Manufacturing, Vol.46, No.4, pp.1523-1530, 2003.
[16] G. Lini: "Multi-optimization of η^3 Splines for Autonomous Parking", IEEE Conference on Decision and Control and European Control Conference, pp.6367-6372, 2011.
[17] I.E. Paromtchik, C. Laugier: "Autonomous Parallel Parking of a Nonholonomic Vehicle", IEEE Intelligent Vehicles Symposium, pp.13-18, 1996.

A. Detailed Description of Constraints

In the following, concrete definitions of constraints and associated constraint error functions are given. For a transition $(\mathsf{p}, \mathsf{v}, \mathsf{p}', \mathsf{v}', \mathsf{w})$, the end configurations are denoted by $\mathsf{x} = \begin{bmatrix} p \\ \theta \end{bmatrix} = f_{\mathsf{p}}(\mathsf{v})$, $\mathsf{x}' = \begin{bmatrix} p' \\ \theta' \end{bmatrix} = f_{\mathsf{p}'}(\mathsf{v}')$, and a continuous curve connecting x and x' with the trajectory type w is denoted by $x(s), u(s)$, $s \in [0, S]$.

Minimum margin from singular points As mentioned in Section II-C, there exist some singular combinations of

configurations for which trajectory of some type is undefined. Moreover, if configurations are near singular, the shape of a trajectory connecting those configurations may become highly sensitive to slight perturbation of the configurations, which is undesirable from the view point of interval-based constraint checking. For this reason, we would like to have some margin from the singular configurations:

$$\|p - p'\| \geq L^{\min}, \tag{12}$$

$$\left|\frac{3\theta + \theta'}{4}\right| \leq \varphi^{\max}, \left|\frac{\theta + 3\theta'}{4}\right| \leq \varphi^{\max} \tag{13}$$

The first constraint requires that p and p' must be apart from each other by at least $L^{\min}$. The second constraint requires that the angles depicted by φ and φ' in Fig. 1 must not be greater than $\varphi^{\max}$. By setting $\varphi^{\max} < \frac{\pi}{2}$, one can avoid singular configurations mentioned earlier ($\theta = \theta' = \pm\pi$ for forward, and $\theta = \theta' = 0$ for reverse). In addition, we need to set $\varphi^{\max} < \frac{\pi}{3}$ to have clothoid curves well-defined. The constraint error function of (12) and (13) are defined as follows, respectively.

$$E_c(\mathsf{p}, \mathsf{v}, \mathsf{p}', \mathsf{v}', \mathsf{w}) = L^{\min} - \|p - p'\|, \tag{14}$$

$$E_c(\mathsf{p}, \mathsf{v}, \mathsf{p}', \mathsf{v}', \mathsf{w}) = \max\left(\left|\frac{3\theta + \theta'}{4}\right| - \varphi^{\max}, \left|\frac{\theta + 3\theta'}{4}\right| - \varphi^{\max}\right) \tag{15}$$

Curvature limit A constraint error function for the curvature limit constraint (2) is defined as

$$E_c(\mathsf{p}, \mathsf{v}, \mathsf{p}', \mathsf{v}', \mathsf{w}) = \max_{s \in [0, S]} (|u(s)| - C^{\max}) \tag{16}$$

Since each trajectory consists of clothoids or arcs, the peek of the curvature can be calculated analytically. Therefore, checking these peeks is necessary and sufficient for verifying the maximum curvature constraint.

Collision avoidance For each obstacle $P^{\mathrm{o}} \in \mathcal{O}$, a collision avoidance constraint is defined. A constraint error function for the collision avoidance constraint (3) for $P^{\mathrm{o}} \in \mathcal{O}$ is defined as

$$E_c(\mathsf{p}, \mathsf{v}, \mathsf{p}', \mathsf{v}', \mathsf{w}) = \max_{s \in [0, S]} d(P^{\mathrm{v}}(x(s)), P^{\mathrm{o}}) \tag{17}$$

Here, d takes two polygons and returns in real values the depth of penetration. If the polygons are apart, then it returns the clearance between them in negative values. This function can be implemented by popular collision-checking algorithms such as SAT (separation axis theorem) and GJK. Since it is generally hard to compute the $\max$ operation over a continuous range $[0, S]$, this part is approximated by the maximum over N^{check} discrete sample points of $[0, S]$.

PLACE PHOTO HERE

Yuichi Tazaki Yuichi Tazaki was born in Kanagawa, Japan, in 1980. He received the Dr.Eng degree in control engineering from Tokyo Institute of Technology, Tokyo, Japan, in 2008. From 2007 to 2009, he was a research fellow of Japan Society for the Promotion of Science. In 2008, he was a guest scientist of Honda Research Institute Europe, Germany. From 2009 to 2016, he was an assistant professor of Nagoya University, Japan. From 2016, he has been an associate professor of Kobe University, Japan. His research interests include autonomous vehicles, mobile robots, and humanoid robots. He is a member of IEEE, SICE and The Robotics Society of Japan.

PLACE PHOTO HERE

Hiroyuki Okuda Hiroyuki Okuda was born in JAPAN, in 1982. He received the B.E. and M.E. degrees in Advanced Science and Technology from Toyota Technological Institute, JAPAN in 2005 and 2007, respectively. He received the Ph.D degree in Mechanical Science and Engineering from Nagoya University, JAPAN in 2010. From 2012 to 2016, he was an assistant professor of the Green Mobility Collaborative Research Center in Nagoya University. Currently, he is an assistant professor of the Department of Mechanical System Engineering of Nagoya University. His research interests are in the areas of system identification of hybrid dynamical system and its application to modeling of human behavior and design of human centered assistance system and automatic control system. Dr. Okuda is a member of the IEEE, IEEJ, SICE and JSME.

PLACE PHOTO HERE

Tatsuya Suzuki Tatsuya Suzuki was born in Aichi, JAPAN, in 1964. He received the B.S., M.S. and PhD degrees in Electronic Mechanical Engineering from Nagoya University, JAPAN in 1986, 1988 and 1991, respectively. From 1998 to 1999, he was a visiting researcher of the Mechanical Engineering Department of U.C.Berkeley. Currently, he is a Professor of the Department of Mechanical System Engineering of Nagoya University, Vice Research Leader of Center of Innovation, Nagoya Univ. (Nagoya Univ.-COI), JST, and Principal Investigator in JST, CREST. He won the best paper award in International Conference on Autonomic and Autonomous Systems 2017 and the outstanding paper award in International Conference on Control Automation and Systems 2008. He also won the journal paper award from IEEJ, SICE and JSAE in 1995, 2009 and 2010, respectively. His current research interests are in the areas of analysis and design of human-centric mobility systems and integrated design of transportation and energy management systems. Dr. Suzuki is a member of the SICE, ISCIE, IEICE, JSAE, RSJ, JSME, IEEJ and IEEE.