MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

A two-stage RRT path planner for automated parking

Wang, Y.; Jha, D.; Akemi, Y.

TR2017-121 August 2017

Abstract

Path planning for automated parking remains challenged by the demand to balance general parking scenarios and computational efficiency. This paper proposes a two-stage rapid-exploring random tree (RRT) algorithm to improve the computational efficiency. At first the proposed algorithm performs space exploration and establishes prior knowledge, represented as waypoints, using cheap computation. Secondly a waypoint-guided RRT algorithm, with a sampling scheme biased by the waypoints, constructs a kinematic tree connecting the initial and goal configurations. Numerical study demonstrates that the two-stage algorithm achieves at least 2X faster than the baseline one-stage algorithm.

IEEE Conference on Automation and Science Engineering

This work may not be copied or reproduced in whole or in part for any commercial purpose. Permission to copy in whole or in part without payment of fee is granted for nonprofit educational and research purposes provided that all such whole or partial copies include the following: a notice that such copying is by permission of Mitsubishi Electric Research Laboratories, Inc.; an acknowledgment of the authors and individual contributions to the work; and all applicable portions of the copyright notice. Copying, reproduction, or republishing for any other purpose shall require a license with payment of fee to Mitsubishi Electric Research Laboratories, Inc. All rights reserved.

Copyright © Mitsubishi Electric Research Laboratories, Inc., 2017
201 Broadway, Cambridge, Massachusetts 02139

A two-stage RRT path planner for automated parking

Yebin Wang, Devesh K. Jha, and Yukiyasu Akemi

***Abstract*— Path planning for automated parking remains challenged by the demand to balance general parking scenarios and computational efficiency. This paper proposes a two-stage rapid-exploring random tree (RRT) algorithm to improve the computational efficiency. At first the proposed algorithm performs space exploration and establishes prior knowledge, represented as waypoints, using cheap computation. Secondly a waypoint-guided RRT algorithm, with a sampling scheme biased by the waypoints, constructs a kinematic tree connecting the initial and goal configurations. Numerical study demonstrates that the two-stage algorithm achieves at least 2X faster than the baseline one-stage algorithm.**

I. Introduction

Path planning arises in a variety of applications such as autonomous vehicles, robotics, manufacturing, pharmaceutical drug design, and computer animation [1]. Many planning algorithms have been proposed, including, to name a few, graph-based A* [2] and D* [3]; continuous approaches involving navigation function and potential fields [4]; more recently, sampling-based algorithms such as probabilistic roadmaps (PRM) [5], expansive-space trees [6], [7], rapidly-exploring random trees (RRT) [1], [8], optimal variants RRT* and PRM* [9], [10], and anytime RRT [11], [12].

Path planning for automated parking, characterized by a tight free space, attracts interests from both academia and automotive industry due to its tremendous social impacts in the foreseeable future. Such efforts lead to a fruitful of results. For instance, [13], [14] consider specific parking scenarios, where the vehicle starts from a fixed location and assumes a fixed geometry of its environment. Such specialized methods achieve fast path generation, but is difficult to generalize. In [15], [16], path planning is tackled by solving general numerical optimization problems, which however have no guarantee of completeness. Work [17] exploits RRT to deal with general parking tasks, but exhibits unsatisfactory computational efficiency. Work [18] proposes a space exploration guided heuristic search method which relies on a circle-path connecting the initial and goal configurations. Because the existence of such a circle-path is more restrictive, this algorithm is not complete. The status quo remains challenged by the conflicting requirements on planning algorithms: computational efficiency and capability of dealing with generic parking tasks.

Y. Wang and D. K. Jha are with Mitsubishi Electric Research Laboratories, Cambridge, MA 02139, USA (email: yebinwang@ieee.org,jha@merl.com).

Y. Akemi is with the Advanced Technology R&D Center, Mitsubishi Electric Corporation, 8-1-1, Tsukaguchi-honmachi, Amagasaki City, 661–8661, Japan (email: Akemi.Yukiyasu@bc.MitsubishiElectric.co.jp).

This paper proposes a two-stage RRT path planning algorithm to determine a kinematically admissible and collision-free path between initial and goal configurations. At first, based on geometries of the environment and the vehicle, and initial/goal configurations, the path planner establishes prior knowledge in the form of a plausible region which, with a high probability, contains feasible paths. Here the plausible region is represented by a neighborhood of a geometric and collision-free path, which is further abstracted by a set of milestone waypoints between initial/goal configurations. Second, the path planner, biased by waypoints to draw samples from the plausible region, constructs a kinematically admissible and collision-free path. The entire algorithm essentially explores the configuration space and determines admissibility to system kinematics (or named after drivability) separately. Such a treatment leads to prevailing algorithms, e.g., decomposition-based approach [19]. With prior knowledge unnecessarily in the form of a geometric path, our work can be viewed as a randomized extension of [19]. The multi-stage framework is motivated by the following facts

1) Sampling-based approaches highly depend on the sampling schemes. The more knowledge or heuristics exploited, the more efficient sampling is.
2) The prior knowledge or space exploration can be obtained by much cheaper operation than the construction of kinematically admissible paths. As an example, given an obstacle-free environment, it takes about 10X longer of time to construct a Reed-Shepp (RS) path between two configurations than a geometric one.

In the contrast, one-stage algorithms solely rely on computationally intensive operations to accomplish space exploration and ensure kinematical admissibility. Overall, the proposed algorithm provides better tradeoff between computational efficiency and completeness. However, one could expect that the proposed algorithm, inclined to search paths in the plausible region, may be stuck in local optimum compared with one-stage algorithms. With the plausible region represented by a collision-free and geometric path, the feasibility of the path planning problem is resolved, and thus the proposed algorithm bears certain anytime property [11]. Our work is different from anytime RRT [11] in that feasibility and optimality are explicitly addressed in two separate stages, with cheap operation dedicated to feasibility.

This paper is organized as follows. Section II introduces system dynamics and the path planning problem. In Section III we analyze an one-stage RRT algorithm and present the two-stage RRT algorithm. Simulation results are provided in Section IV to verify the proposed algorithm. Section V

completes this paper with conclusion and future work.

Notation: A tree $\mathcal{T}$ is a union of a node set $V \subset \mathcal{C}_{free}$ and an edge set E, i.e., $\mathcal{T} = (V, E)$. An edge in E represents a collision-free path between two nodes. A tree $\mathcal{T}$ is termed a geometric tree, if its edges are geometric paths; with kinematically admissible edges, $\mathcal{T}$ is a kinematic tree. A bidirectional RRT tree is a union of $\mathcal{T}_s(V_s, E_s)$ and $\mathcal{T}_g(V_g, E_g)$. Specifically, a start tree $\mathcal{T}_s(V_s, E_s)$ has the initial configuration X_0 as its root, and a goal tree $\mathcal{T}_g(V_g, E_g)$ has the goal configuration as its root. For a finite set V, $|V|$ denotes the number of its elements.

II. Preliminary

For completeness, this section reviews key concepts in path planning. Details can be found in [20]–[23].

A. System Models

Consider a control-affine dynamical system

$$\dot{X} = f(X) + g(X)u, \tag{1}$$

where $X \in \mathcal{X} \subset \mathbb{R}^n$ is state, $u \in \mathcal{U} \subset \mathbb{R}^m$ the control, f a smooth vector field or the drift, and $g = [g_1^\top, \cdots, g_m^\top]^\top$ with g_i a smooth vector field. System (1) is symmetric if $\mathcal{U}$ is symmetric with respect to the origin $X = 0$. An admissible *trajectory* X_t is a solution of system (1) with given initial and final conditions and $u \in \mathcal{U}$. A *configuration* of system (1) is a complete specification of the position of every points of that system. The *configuration space* $\mathcal{C} \subset \mathbb{R}^n$ is a compact set representing all possible configurations of the system. A collision-free configuration space $\mathcal{C}_{free}$ is the set of configurations at which the vehicle has no intersection with obstacles in the environment. Denote the collision configuration space $\mathcal{C}_{obs} = \mathcal{C} \backslash \mathcal{C}_{free}$. An admissible *path* is the image of an admissible trajectory on the configuration space $\mathcal{C}$.

Without loss of generality, this paper considers a front wheel drive vehicle, which is illustrated by Fig. 1. Its dynamics are given by the following fifth order model [21]

$$\begin{aligned} \dot{x} &= \cos(\theta)\cos(\phi)v \\ \dot{y} &= \sin(\theta)\cos(\phi)v \\ \dot{\theta} &= \sin(\phi)\frac{v}{l} \\ \dot{v} &= a_1 \\ \dot{\phi} &= a_2, \end{aligned} \tag{2}$$

where (x, y) is the coordinates of the midpoint A of the rear wheels, θ the vehicle orientation, v the velocity of the front wheels, ϕ the angle between the front wheels and the vehicle orientation, a_1 the translational acceleration, a_2 the steering angular velocity, and l the distance between (x, y) and the midpoint B of the front wheels. Both a_1 and a_2, as control inputs, are subject to constraints: $|a_i| \leq a_{i\max}$ for $i = 1, 2$. Also, a mechanical constraint $|\phi| \leq \phi_{\max}$ limits the minimum turning radius and the curvature of a path. It is desirable to perform path planning based on system (2) because it implicitly enforces continuous differentiability of the velocity and the steering angle. This is however non-trivial, and thus most of literature performs path planning based on a continuous curvature car model [24]–[26] or the Reed-Shepp's car model.

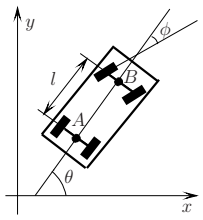

Fig. 1. The vehicle schematics

With the (v, ϕ)-dynamics omitted, system (2) is reduced to the RS car model

$$\begin{aligned} \dot{x} &= \cos(\theta)u_1 \\ \dot{y} &= \sin(\theta)u_1 \\ \dot{\theta} &= u_2, \end{aligned} \tag{3}$$

where $u_1 = \cos(\phi)v$ is the velocity along the car orientation, and $u_2 = u_1 \tan(\phi)/l$ is the steering control.

B. Path Planning Problem

For system (3), its kinematic state space $X = (x, y, \theta)^\top$ coincides with the configuration space, i.e., $\mathcal{C} = \mathcal{X}$. Given geometries of obstacles and the vehicle, the configuration spaces $\mathcal{C}_{obs}$ and $\mathcal{C}_{free}$ are well-defined. Path planning for automated parking corresponds to the following problem.

Problem 2.1: Given an initial configuration $X_0 \in \mathcal{C}_{free}$ and a goal configuration $X_f \in \mathcal{C}_{free}$, and the system kinematics (3), find a collision-free and admissible path $\mathcal{P}_t$ which

1) starts at X_0 and ends at X_f;
2) lies in the collision-free configuration space $\mathcal{C}_{free}$;
3) satisfies the system kinematics.

Let J be a cost function that assigns to each non-trivial path a non-negative cost. The optimal path planning is to find a collision-free and admissible path $\mathcal{P}_t^* : [0, 1] \to \mathcal{C}_{free}$ that minimizes the cost function J.

For brevity, a path satisfying system kinematics is admissible; if additionally the path is collision-free, we call it a collision-free and admissible path. Computation of a shortest admissible path between two configurations for system (3), also known the RS path, has been disclosed in [27].

A path planner solving Problem 2.1 is *complete*, meaning that within finite time, it always finds a solution when one exists, or indicates failure otherwise. Particularly, sampling-based planners involve in *probabilistic completeness*, which means that the probability of finding a solution converges to 1 as time goes to infinity.

C. One-Stage RRT Planner

The one-stage RRT planner for Problem 2.1 is a combination of the bidirectional RRT (bi-RRT) algorithm [23] and the RS steering [27]. It consists of two parts: constructs

a kinematic tree connecting X_0 with X_f; and determines a collision-free and admissible path according to the kinematic tree. The tree construction algorithm is presented below for comparison with the proposed algorithm.

Algorithm 1: One-Stage RRT Planner

$V_s \leftarrow X_0$, $E_s \leftarrow \emptyset$; $V_g \leftarrow X_f$, $E_g \leftarrow \emptyset$;
$k \leftarrow 1$, $flag \leftarrow false$;
$\mathcal{T}_s \leftarrow (V_s, E_s)$; $\mathcal{T}_g \leftarrow (V_g, E_g)$;
while $k < K$ **and not** $flag$ **do**
  $X_{\text{rand}} \leftarrow \texttt{Sample}(\mathcal{C})$;
  $flagS \leftarrow false$;
  $(\mathcal{T}_s, flagS) \leftarrow \texttt{Extend}(\mathcal{T}_s, X_{\text{rand}}, flagS)$;
  $(\mathcal{T}_g, flagG) \leftarrow \texttt{Extend}(\mathcal{T}_g, X_{\text{rand}}, flagS)$;
  $flag \leftarrow flagS$ **and** $flagG$;
return $(\mathcal{T}_s, \mathcal{T}_g, flag)$;

A kinematic bi-RRT tree $\mathcal{T} = \mathcal{T}_s \bigcup \mathcal{T}_g$ is constructed by Algorithms 1-3, where two kinematic RRT trees $\mathcal{T}_s$ and $\mathcal{T}_g$ grow toward each other until connected. The `Sample` procedure generates a collision-free sample $X_{\text{rand}} \in \mathcal{C}_{free}$ according to a uniform sampling scheme. The `Extend` grows the tree $\mathcal{T}$ toward x_{rand}. The configuration X_{rand} is added to $\mathcal{T}$ only if it can be connected to the nearest configuration $X_{\text{nearest}} \in V$ through a collision-free and admissible edge (path). Given a configuration X_1 and a node set V, the `Nearest` returns the configuration $X_{\text{nearest}} \in V$ that is closest to X_1, and stays inside its r-neighborhood defined as $V(X_1) : \{Y | Y \in V, d(X_1, Y) \leq r\}$, i.e.,

$$\texttt{Nearest}(V, X_1) \triangleq \text{argmin}_{Y \in V(X_1)} d(X_1, Y),$$

where $d(\cdot, \cdot)$ is a distance function. Given two configurations $X_i, X_j \in \mathcal{C}_{free}$, the `Connect` procedure determines if there exists a collision-free RS path between X_i and X_j. For X_i and X_j, the `ReedShepp` returns a set of admissible paths $\mathcal{P}$, where each element $\mathcal{P}_i \in \mathcal{P}$ represents an RS path between X_i and X_j. The $\texttt{CollisionFree}(\mathcal{P}_i)$ determines whether $\mathcal{P}_i$ is collision-free. In the `Connect`, $\mathcal{P}_i$ is picked from $\mathcal{P}$ in the order of descending lengths, and $i_{\max}$ controls how many RS paths will be tested for collision.

Algorithm 2: $\texttt{Extend}(\mathcal{T}, X_{\text{rand}}, flagS)$ in Algorithm 1

$flag \leftarrow false$; $(V, E) \leftarrow \mathcal{T}$;
$X_{\text{nearest}} \leftarrow \texttt{Nearest}(V, X_{\text{rand}})$;
if $\texttt{Connect}(X_{\text{nearest}}, X_{\text{rand}})$ **then**
  $flag \leftarrow true$;
  if not $flagS$ **then**
    $V \leftarrow V \cup \{X_{\text{new}}\}$;
  $E \leftarrow E \cup \{(X_{\text{nearest}}, X_{\text{new}})\}$;
$\mathcal{T} \leftarrow (V, E)$;
return $(\mathcal{T}, flag)$

Remark 2.2: In Algorithm 1, the one-stage algorithm terminates the tree construction once two trees $\mathcal{T}_s$ and $\mathcal{T}_g$ connected. This can be replaced by other criterion to improve quality of paths, for instance, imposing a threshold on the number of nodes added to both trees. In such a case, the output of the algorithm is a graph. For the sake of notation brevity, this work does not distinguish these two cases, and assumes the algorithm outputs a tree. □

Algorithm 3: $\texttt{Connect}(X_{\text{nearest}}, X_{\text{rand}})$ in Algorithm 2

$i \leftarrow 0, i_{\max}$;
$\mathcal{P} \leftarrow \texttt{ReedShepp}(X_{\text{nearest}}, X_{\text{rand}})$;
for $\mathcal{P}_i \in \mathcal{P}$ **and** $i \leq i_{\max}$ **do**
  $flag \leftarrow \texttt{CollisionFree}(\mathcal{P}_i)$;
  if $flag$ **then**
    break;
return $flag$

III. Main Results

We first expose limitations of the one-stage algorithm, and then present a two-stage RRT planner.

A. Analysis of the One-Stage Algorithm

As pointed out in many literature, e.g. [22], computational efficiency of sampling-based algorithms heavily depend on the sampling scheme. An ideal scheme avoids sampling configurations which are unlikely to be part of the collision-free and admissible path. This is impossible due to lack of prior knowledge. A sampling scheme failing to incorporate or being lack of prior knowledge necessarily leads to low efficiency of path planning. The one-stage algorithm takes the uniform sampling scheme due to lack of prior knowledge about $\mathcal{C}_{free}$. This is illustrated in Fig. 2, where the one-stage algorithm samples and rejects a configuration X_{rand} if it is far from $\mathcal{T}$. Let Ω_0, the region inside the gray dash line, denote a neighborhood of $\mathcal{T}_s$; Ω_f, bounded by the gray solid line, be the neighborhood of $\mathcal{T}_g$. Any sample outside of Ω_0 or Ω_f will be rejected. Denote Ω the plausible region bounded by the solid blue line. It is clear that the distance function $d(\cdot, \cdot)$, employed to reject potentially useless samples, could not incorporate knowledge about the plausible region Ω. Consequently, configurations which belongs to Ω_0 but not Ω be tested and added to the tree. This unnecessarily increases the size of the tree, and ultimately slows down its construction. The one-stage algorithm tends to construct a tree spanning the entire space $\mathcal{C}_{free}$.

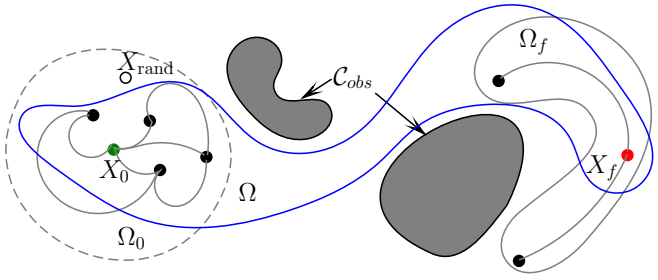

Fig. 2. One-stage algorithm: tree construction

In the one-stage algorithm, space exploration and insurance of drivability (or kinematic admissibility) are conducted simultaneously. This results in two remarkable drawbacks.

The first is low computational efficiency. The space exploration produces a tree $\mathcal{T}$ as an approximation of $\mathcal{C}_{free}$. In the course of the tree construction, the space exploration incrementally and iteratively tests whether a sample configuration $X_{\rm rand}$ can be reached from the existing explored space, represented by the tree $\mathcal{T}$. Reachability from $\mathcal{T}$ to $X_{\rm rand}$ is determined by the steering method. The one-stage algorithm, due to its one-stage nature, uses RS steering, which corresponds to lines 1-6 in Algorithm 3. As shown later, it is more effective to use Line steering in the space exploration, because it is much more computationally efficient than the RS steering. The second drawback is lack of flexibility allowing buildup of knowledge during the space exploration phase, and exploitation of the knowledge in the next phase. In fact, because the drivability phase is more computationally expensive, one would like to fulfill this task using as much as information possible to reduce waste of computation resources. The one-stage algorithm however does not offer such possibility because, with simultaneous execution of both phase, no extra information can be exploited in the drivability phase, compared with the exploration phase.

B. Main Algorithm

The two-stage RRT planning algorithm addresses limitations of the one-stage algorithm by

- Utilize a two-stage framework which allows space exploration and determination of drivability to progress at different speeds or sequentially. Thus a faster space exploration stage can establish knowledge to be employed in the slower stage for computational efficiency.
- Propose the use of cheap computation to accomplish fast space exploration, which typically involves a large number of low probability events (successful connection between two configurations). With a waypoint-guided sampling scheme, the determination of drivability involves computationally expensive but high probability events (successful kinematic connection between two configurations).

The two-stage RRT planner is given by Algorithm 4. The first stage, space exploration, is realized in the procedure `Waypoint`, while the rest account for the second stage.

Algorithm 4: Waypoint-Guided bi-RRT Planning

```
1 W ← ∅;
2 W ← Waypoint(X_0, X_f);
3 (V_s, E_s, flag) ← WGRRT(W);
4 if flag then
5 |   ValueIteration(V_s, E_s);
6 |   P ← Compute(V_s, E_s);
7 |   P ← Smooth(P);
8 return P;
```

1) Comparison of RS and Line Steering: We compare computational efficiency of two steering methods, without considering obstacles: RS and Line, which construct an RS path and a straight line, respectively. Performing 100,000 randomly generated steering tasks, and normalizing the computation time for Line steering to 1, we learn that the RS steering is 11 times slower.

Table I shows computational efficiency of RS and Line steering methods with collision detection taken into account. The computation time of Line steering has been normalized to 1. The second column in Table I shows the computation time of the RS steering for different values of $i_{\max}$.[1] The RS steering could take up to 18 times longer than the Line steering to compute a collision-free path, depending on the value of $i_{\max}$. The third and forth columns in Table I list normalized probability of which the RS steering can successfully return a collision-free and admissible path for two configurations, where the probability of which the Line steering can construct a collision-free and geometric path is normalized to 1.[2] Specifically, the third column corresponds to the case when $r = 1$ is used in the procedure `Nearest`, whereas the forth column corresponds to the case $r = 5$. Table I shows that with a larger r, the probability of constructing a collision-free path is lower. Also, the Line steering more likely results in a collision-free path than the RS steering except the case $i_{\max} = 48$. Note that Table I is obtained by conducting 100,000 simulations for a given environment map and initial configuration. Although the data could vary according to distinct environment maps and initial configurations, no significant difference is expected.

TABLE I

COMPUTATIONAL EFFICIENCY: WITH OBSTACLES

$i_{\max}$	Computation	Probability (r=1)	Probability (r=5)
1	2.02	0.82	0.83
2	2.88	0.89	0.90
3	4.01	0.93	0.93
48	17.6	1.08	1.09

2) Space Exploration: The `Waypoint` procedure generates a plausible region which is represented by a set of waypoints $W = \{W_i, 1 \leq i \leq L\}$, based on the initial and goal configurations X_0, X_f and the environment map. Without loss of generality, $W_1 = X_0$ and $W_L = X_f$. Each waypoint W_i represents a collision-free configuration in $\mathcal{C}_{free}$. More importantly, every two adjacent waypoints are connected through a collision-free path, which might not be admissible to the system kinematics (3).

Waypoint generation is supposed to be achieved through computationally efficient operations. A simple choice is to use a geometric path planner, deterministic or randmized.

[1]The parameter $i_{\max} \in \mathbb{N}$ in Algorithm 3 specifies the number of admissible RS paths for collision detection. It is well-understood that an RS path between any two configurations possibly admits 48 driving patterns [27], and thus $1 \leq i_{\max} \leq 48$. With $i_{\max} = 1$, we only perform collision detection for the shortest admissible path. On the other hand, $i_{\max} = 48$ implies that we enumerate all admissible RS paths for collision detection. It is noteworthy that the number of driving patterns can be further reduced to 46 [28], in which case $1 \leq i_{\max} \leq 46$.

[2]The probability of the Line steering for $r = 1$ and $r = 5$ is 0.1973 and 0.1939, respectively. Probability in general is less than 1. In Table I, normalized probabilities for cases $i_{\max} = 48$ are greater than 1. This means the RS steering with $i_{\max} = 48$ is more likely to construct a collision-free and admissible path than the Line steering.

As an example, a standard RRT geometric planner can be used, ignoring the vehicle kinematics (3). Different from the one-stage algorithm, the geometric RRT planner uses Line steering in Algorithm 3 instead of the RS steering. Unsurprisingly, the geometric RRT planner is much more computationally efficient to sweep through the configuration space, and determinine whether a collision-free configuration X_{rand} is reachable or not.

3) Waypoint-Guided RRT: Given the waypoint set W, the `WGRRT` procedure constructs a kinematic tree connecting X_0 and X_f in the plausible region. It can have different implementations. For example, the `WGRRT` procedure can enumerate all adjacent pairs of waypoints $(W_i, W_{i+1}), 1 \leq i \leq L-1$, and constructs a bidirectional RRT tree between each pair of (W_i, W_{i+1}). In the end, the `WGRRT` procedure returns a kinematic tree $\mathcal{T}$ connecting W_1 all the way to W_L by unioning all bidirectional RRT trees.

Algorithm 5 illustrates another implementation of the `WGRRT` procedure used in simulation. At the kth iteration, the `BiRRT` procedure constructs a kinematic tree connecting $W_i, 1 \leq i \leq k+1$ sequentially. After the $L-1$th iteration, the `WGRRT` procedure finishes the tree construction with all waypoints connected. This procedure is further illustrated by Fig. 3, which corresponds to the iteration $k = 2$. In Fig. 3, waypoints are represented by red nodes and a kinematic tree $\mathcal{T}_1$ between X_0 and W_2 has been constructed. Next, in the 2nd iteration, the `WGRRT` procedure constructs a kinematic tree connecting the tree $\mathcal{T}_1$ with the waypoint W_3.

Algorithm 5: The `WGRRT` Procedure

$k \leftarrow 1$, $L \leftarrow \texttt{Length}(W)$, $flag \leftarrow true$, $P \leftarrow \emptyset$;
$V_s \leftarrow X_0$; $E_s \leftarrow \emptyset$;
while $k < L$ **and** $flag$ **do**
  $V_g \leftarrow W_{k+1}$; $E_g \leftarrow \emptyset$;
  if $\texttt{BiRRT}(V_s, E_s, V_g, E_g)$ **then**
    $V_s \leftarrow V_s \cup V_g$;
    $E_s \leftarrow E_s \cup E_g$;
  else
    $flag \leftarrow false$;
    break;
  $k \leftarrow k + 1$;
return $(V_s, E_s, flag)$;

Fig. 3 also illustrates how the two-stage algorithm improves computational efficiency. Assume that the region inside the red dash line Ω_s characterizes the prior knowledge or the plausible region. The region Ω_s defines where collision-free and admissible paths stay and the sampling scheme should focus on. The gray dash line defines the region close to the kinematic tree $\mathcal{T}_1$. Compared with sampling from Ω_0, sampling X_{rand} from $\Omega_s \bigcap \Omega_0$ is more likely to connect with W_3 within a certain time.

A key component of the `WGRRT` is how to utilize waypoints in the sampling scheme in `BiRRT`. We illustrate the sampling scheme in Algorithm 5 by exemplifying the scenario given in Fig. 3. In the course of growing the tree $\mathcal{T}_1$ toward W_3, we sample a new collision-free configuration according to

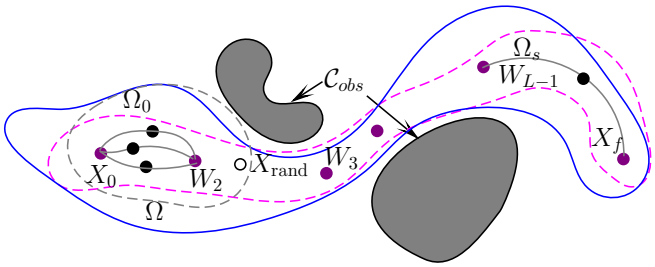

Fig. 3. Two-stage algorithm: tree construction

the following Gaussian distribution

$$p(X_{\text{rand}} = X) \sim \mathcal{N}(\mu, \sigma),$$

where $\mu = \lambda_2 W_2 + \lambda_3 W_3$ with $0 < \lambda_2, \lambda_3 < 1$ and $\lambda_2 + \lambda_3 = 1$, and $\sigma = \text{diag}(\alpha_x, \alpha_y, \alpha_\theta) \|W_3 - \mu\|$. Constants $\alpha_x, \alpha_y, \alpha_\theta$ are positive.

4) Value Iteration: Given the kinematic tree $\mathcal{T}$, a number of graph-based search algorithms such A* and D* can be applied to find a collision-free and admissible path from X_0 to X_f. These algorithms are computationally efficient, but the path optimality depends on how good the heuristic cost-to-go function approximates the optimal value (cost-to-go) function. One remedy of sub-optimality is to apply approximate dynamic programming to the tree $\mathcal{T}$ and obtain the value function $v : V \to \mathbb{R}^+$. This is done in the procedure `ValueIteration`, which performs value iteration over V to estimate the value function from every configuration of V to the goal configuration. The value iteration is guaranteed to converge, as the number of iterations goes to infinity [29].

Algorithm 6: The `ValueIteration` Procedure

for $X_i \in V$ **do**
  $V_a(X_i) \leftarrow \texttt{Adjacent}(V, X_i)$;
  if $\text{Card}(V_a(X_i)) > 1$ **then**
    $v(X_i) \leftarrow \min_{Y \in V_a(X_i)} (\texttt{cost}(X_i, Y) + v(Y))$;

It is noteworthy that the number of iterations taken to ensure the convergence increases at least proportionally to $|V|$, and the number of arithmetic operations increases at least quadratically. In other words, value iteration could be expensive. We propose to first trim the tree by removing all leaf-nodes, and then perform value iteration over the trimmed tree. A leaf-node is connected to only one node. It is noteworthy that throughout extensive simulations, the two-stage RRT algorithm typically ends up with a tree, which contains a higher percentage of leave nodes than that by the one-stage RRT algorithm. Hence, this further facilitates value iteration based path generation over A* and D*.

Given a kinematic tree $\mathcal{T}$, the initial configuration X_0 and the value function v over V, the `Compute` procedure implements policy update and returns a kinematic path $\mathcal{P}_t$ connecting X_0 and X_f. Given the fact that RRT planners result in fragmental paths, the `Smooth` procedure smooths the path $\mathcal{P}_t$. The policy update for a known value function

is standard and can be found in [29]. A number of literature also talked about smoothing of a given path [5]. Details about these two procedures are therefore omitted here.

Remark 3.1: The effectiveness of the proposed algorithm relies on the quality of waypoints, which is influenced by the distance function. This issue is similar to A* which is limited by the quality of heuristic costs, and thus can be alleviated by similar techniques applied to treat A*. □

C. Analysis

The two-stage algorithm can be viewed as a randomized extension of decomposition-based path planning algorithms, e.g. [19], and thus its analysis resorts to similar tools. This section is included for completeness. We introduce controllability and small-time controllability, which are essential to ensure the completeness for a certain class of path planners [21].

Definition 3.2: [21] System (1) is locally controllable from X if the set of points reachable from X by an admissible trajectory contains a neighborhood of X. It is small-time controllable from X if the set of points reachable from X before a given time T contains a neighborhood of X for any T.

The following Theorem gives necessary and sufficient conditions to verify whether a driftless system is controllable.

Theorem 3.3: A symmetric system without the drift is small-time controllable from X if and only if the rank of the vector space spanned by the family of vector fields g_i together with all their Lie brackets in n at X.

Theorem 3.4: [21, Thm. 3.1] For symmetric small-time controllable systems, the existence of a collision-free and admissible path between two given configurations is equivalent to the existence of any collision-free path between these two configurations.

It has been established that system (3) is symmetric, driftless, and small-time controllable [21]. Hence, Theorem 3.4 is applicable to system (3), and thus shows that the existence of a collision-free and admissible path is equivalent to the existence of a collision-free geometric path . Since the waypoint set W implies the existence of a collision-free path between X_0 and X_f, and the RRT algorithm to generate the set W is probabilistic complete, so is the proposed algorithm.

IV. Case Studies

We compare the one-stage algorithm with the proposed algorithm by conducting simulation in Matlab®2016b, and for multiple environment maps. This section shows simulation results for the environment depicted in Fig. 4, where the map boundary is represented by a rectangle of $20m \times 12m$, and all obstacles are rectangles with their boundaries in bold lines. The initial and goal configurations are $X_0 = (-3.65, 8, 0)$ and $X_f = (5.25, 2, \pi/2)$, represented by gray and red dot, respectively. The vehicle has a length of 4.85m, width 1.81m, and minimum turning radius 4.4m. Both algorithms use the same values of parameters: $r = 6, i_{\max} = 3$; and top when the number of nodes added to both $\mathcal{T}_s$ and $\mathcal{T}_g$ reaches 100. We perform 10000 simulations and have the average computation time: $2.96sec$ for the one-stage algorithm, versus $1.42sec$ for the two-stage algorithm.

Detailed results of both algorithms are illustrated in Figs. 4-7. Specifically, Fig. 4 plots waypoints as a result of space exploration, where each node comes with an arrow indicating the orientation of the vehicle. Fig. 5 plots the kinematic tree constructed by the procedure `WGRRT`, where the values next to nodes are obtained by the procedure `ValueIteration`, and edges merely indicate the existence of kinematic paths. Fig. 6 gives vehicle positions along the collision-free and kinematic path, where green $\times$ markers form the entire path. Fig. 7 visualizes the kinematic tree constructed by the one-stage algorithm. Comparing with the two-stage algorithm, one notices that the one-stage algorithm results in a much larger tree.

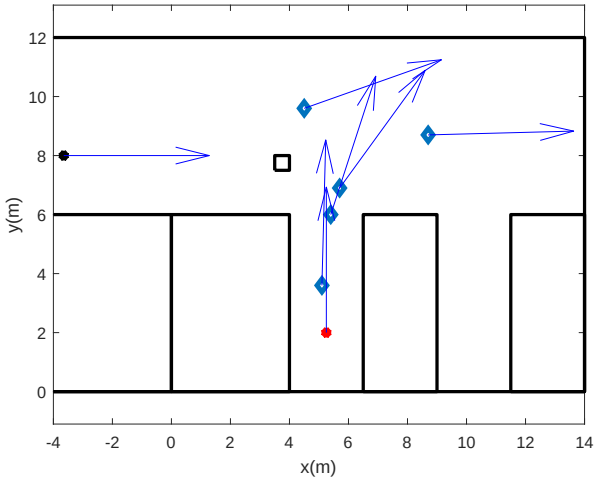

Fig. 4. Two-stage algorithm: waypoint

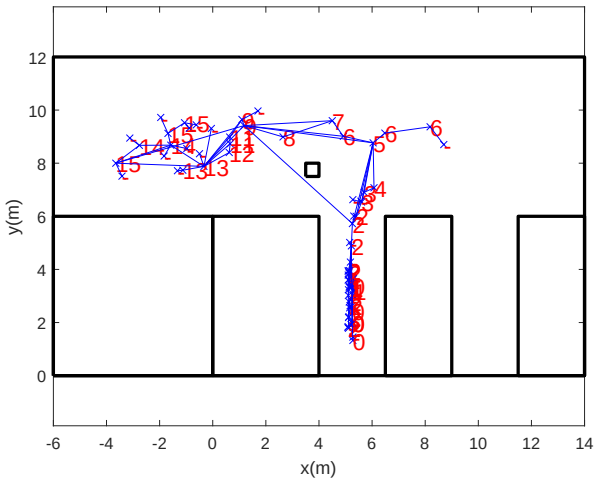

Fig. 5. Two-stage algorithm: kinematic RRT tree

V. Conclusion and Future Work

This paper proposes a two-stage rapid-exploring random tree (RRT) algorithm to better balances general parking scenarios and computational efficiency. Baseline one-stage

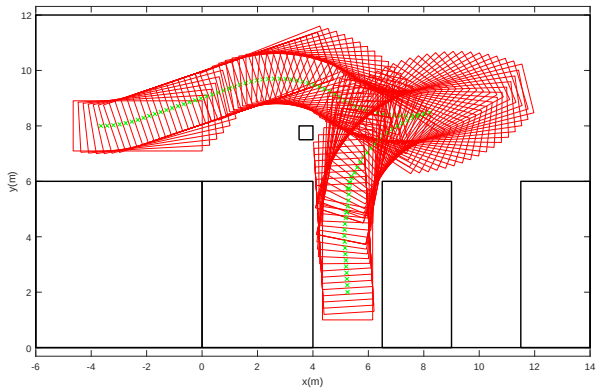

Fig. 6. Two-stage algorithm: kinematic path

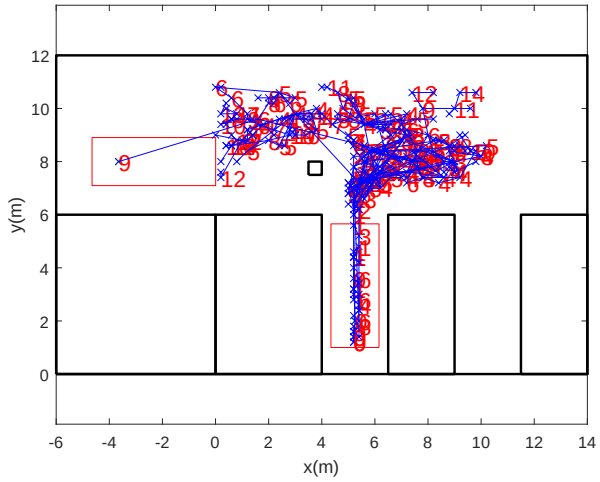

Fig. 7. One-stage algorithm: kinematic RRT tree

RRT algorithm suffers low efficiency because: 1) it conducts space exploration and determination of drivable paths simultaneously, where the former could be achieved more efficiently; 2) its simultaneous execution framework does not allow incremental buildup beforehand and explicit exploitation of prior knowledge in the later computationally intensive stage. Differently, the proposed algorithm splits the space exploration and the determination of drivable paths, where the former establishes prior knowledge by cheap computation, and the latter exploits the prior knowledge for computational efficiency. Numerical study demonstrated that the two-stage algorithm speeds up the one-stage algorithm at least 2 times. The proposed algorithm trades the optimality for computational efficiency though. Future work includes alternative space exploration techniques for efficiency, as well as methods to incorporate drivability metric into the space exploration stage for optimality.

REFERENCES

[1] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *Int. J. Robot Res.*, vol. 20, no. 5, pp. 378–400, 2001.

[2] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[3] A. Stentz, "Optimal and efficient path planning for unknown and dynamic environments," *International Journal of Robotics and Automation*, vol. 10, pp. 89–100, 1993.

[4] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. Robot Res.*, vol. 5, no. 1, pp. 417–431, 1986.

[5] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Automat.*, vol. 12, no. 4, pp. 566–580, Aug 1996.

[6] D. Hsu, "Randomized single-query motion planning in expansive spaces," Ph.D. dissertation, Stanford University, 2000.

[7] D. Hsu, R. Kindel, J. C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *Int. J. Robot Res.*, vol. 21, no. 3, pp. 233–255, 2002.

[8] J. J. Kuffner and S. M. LaValle, "RRT connect: An efficient approach in single-query path planning," in *Proc. 2000 ICRA*, San Francisco, CA, May 2000, pp. 995–1001.

[9] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot Res.*, vol. 30, no. 7, pp. 846–894, 2011.

[10] ——, "Sampling-based optimal motion planning for non-holonomic dynamical systems," in *Proc. 2013 ICRA*, Karlsruhe, Germany, May 2013, pp. 5041–5047.

[11] D. Ferguson and A. Stentz, "Anytime RRTs," in *Proc. 2006 IROS*, 2006, pp. 5369–5375.

[12] J. van den Berg, D. Ferguson, and J. Kuffner, "Anytime path planning and replanning in dynamic environments," in *Proc. 2006 ICRA*, 2006, pp. 2366–2371.

[13] I. E. Paromtchik and C. Laugier, "Motion generation and control for parking an autonomous vehicle," in *Proc. 1996 ICRA*, Apr. 1996, pp. 3117–3122.

[14] ——, "Autonomous parallel parking of a nonholonomic vehicle," in *Proc. of the 1998 IEEE Intelligent Vehicles Symposium*, Sep. 1996, pp. 13–18.

[15] K. Kondak and G. Hommel, "Computation of time optimal movements for autonomous parking of non-holonomic mobile platforms," in *Proc. 2001 ICRA*, Seoul, Korea, May. 2001, pp. 2698–2702.

[16] G. Lini, A. Piazzi, and L. Consolini, "Multi-optimization of η^3 for autonomous parking," in *Proc. 50th CDC*, Orlando, FL, Dec. 2011, pp. 6367–6372.

[17] L. Han, Q. H. Do, and S. Mita, "Unified path planner for parking an autonomous vehicle based on RRT," in *Proc. 2011 ICRA*, Shanghai, China, May 2011, pp. 5622–5627.

[18] C. Chen, M. Rickert, and A. Knoll, "Path planning with orientation-aware space exploration guided heuristic search for autonomous parking and maneuvering," in *2015 IEEE Intelligent Vehicles Symposium*, Jun. 2015, pp. 1148–1153.

[19] J.-P. Laumond, P. E. Jacobs, M. Taïx, and R. M. Murray, "A motion planner for nonholonomic mobile robots," *IEEE Trans. Robot. Automat.*, vol. 10, no. 5, pp. 577–593, Oct. 1994.

[20] J.-C. Latombe, *Robot Motion Planning*. Boston, MA: Kluwer, 1991.

[21] J.-P. Laumond, S. Sekhavat, and F. Lamiraux, *Guidelines in nonholonomic motion planning for mobile robots*. Springer, 1998.

[22] H. M. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementation*. Cambridge, MA: MIT Press, 2005.

[23] S. M. LaValle, *Planning algorithms*. Cambridge, UK: Cambridge University Press, 2006.

[24] A. Scheuer and C. Laugier, "Planning sub-optimal and continuous-curvature paths for car-like robots," in *Proc. 1998 IROS*, 1998, pp. 25–31.

[25] F. Lamiraux and J.-P. Laumond, "Smooth motion planning for car-like vehicles," *IEEE Trans. Robot. Automat.*, vol. 17, no. 4, pp. 498–502, Aug. 2001.

[26] T. Fraichard and A. Scheuer, "From Reeds and Shepp's to continuous-curvature paths," *IEEE Trans. Robot.*, vol. 20, no. 6, pp. 1025–1035, Dec. 2004.

[27] J. A. Reeds and L. A. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific Journal of Mathematics*, vol. 145, no. 2, pp. 367–393, 1990.

[28] H. Sussmann and G. Tang, "Shortest paths for the reeds-shepp car: A worked out example of the use of geometric techniques in nonlinear optimal control," Dept. of Mathematics, Rutgers University, Piscataway, NJ, Tech. Rep., 1991, technical Report SYNCON 91-10.

[29] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. Athena Scientific Belmont, 1995.