Kinodynamic FMT* with Dimensionality Reduction Heuristics and Neural Network Controllers

Dongliang Zheng* and Panagiotis Tsiotras[†]

Georgia Institute of Technology, Atlanta, GA, 30332-0150

This paper proposes a new sampling-based kinodynamic motion planning algorithm, called FMT*PFF, for nonlinear systems. It exploits the novel idea of dimensionality reduction using partial-final-state-free (PFF) optimal controllers. With the proposed dimensionality reduction heuristic, the search space is restricted within a smaller subspace, thus achieving faster convergence compared to a regular kinodynamic FMT*. The dimensionality reduction heuristic can be viewed as a sampling strategy preserving asymptotic optimality when combined with uniform full-state sampling. Another feature of FMT*PFF is the ability to deal with a steering function with inexact steering, which is vital when using learning-based steering functions. Learning-based methods allow us to solve the steering problem for nonlinear systems efficiently. However, learning-based methods often fail to reach the exact goal state. For general nonlinear systems, we train a neural network controller using supervised learning to generate the steering commands. We show that FMT*PFF with a learning-based steering function is efficient and generates dynamically feasible motion plans. We compare our algorithm with previous algorithms and show superior performance using various simulations.

I. Introduction

Motion planning, as a fundamental component of robot autonomy, has been studied extensively in the last three decades to increase its efficiency and capability. Efficiency means faster convergence to better solutions. Efficient planning algorithms are crucial for robots with limited computation power capability for replanning in changing environments. Capability means dealing with more complicated planning problems. High-dimensional state space, nonlinear system dynamics, and cluttered environments with nonconvex obstacles still pose great challenges for efficient kinodynamic planning despite recent advances in the field.

Sampling-based motion planning algorithms, such as PRM [1] and RRT [2], have been developed to solve planning problems in high-dimensional continuous state spaces by incrementally building a graph/tree through the search space. The optimal sampling-based planning RRT* algorithm [3] almost surely converges asymptotically to the optimal solution. RRT* is well-suited for planning in high-dimensional spaces and obstacle-rich environments. Many applications of RRT* have been studied in recent years [4].

For motion planning of dynamical systems, sampling-based optimal kinodynamic planning algorithms (SBKMP) such as Kinodynamic RRT* [5] and Kinodynamic FMT* [6] have been developed and consider differential constraints. SBKMP requires any two points sampled in the planning space to be connected with an optimal trajectory. For robots with differential constraints, the optimal trajectory between two states is obtained by solving a two-point boundary value problem (TPBVP), which is a non-trivial undertaking for complex nonlinear systems. The solution to this local TPBVP is also referred to as the steering function. Simulation-based methods such as SST [7] avoid solving the TPBVP by using random control sampling and simulation. However, without the local optimal edges (trajectories) provided by the steering function, the convergence of SST to a good solution is slow.

Solving TPBVPs efficiently for nonlinear systems is one of the bottlenecks of kinodynamic RRT*. Thus, researchers have looked into more efficient ways to solve these TPBVPs. A steering function based on LQR is used in [8]. A fixed-final-state free-final-time controller for linear systems that optimally connects any pair of states is introduced in [9]. While an analytical solution of the TBPVP for linear systems is available, considering general nonlinear system dynamics is difficult. Learning-based methods have the potential to solve TPBVP efficiently. To deal with nonlinear dynamics, steering functions based on supervised learning and reinforcement learning are developed in [10] and [11]

---

*PhD candidate, School of Aerospace Engineering.

[†]David and Andrew Lewis Chair Professor, School of Aerospace Engineering and Institute for Robotics and Intelligent Machines; AIAA Fellow.

respectively, and integrated with a kinodynamic RRT* algorithm. In [12, 13], a goal-conditioned state-feedback neural network controller for nonlinear systems is trained and used for solving the TPBVPs for RRT* problem formulations.

Another limitation of RRT* is the slow convergence rate of the solution to the optimal one, which is especially evident for the kinodynamic planning case where the sampling space is not just the configuration space but the full state space. Heuristic and informed sampling methods have been developed to improve the convergence rate. Informed RRT* [14] focuses sampling to an informed subset that could potentially provide a better solution. Exploiting the benefit of ordered search, FMT* [15] and BIT* [16] are shown to find better solutions faster than RRT*. However, most of these methods only consider the geometric planning problem. Existing work on heuristics for improving the convergence of kinodynamic motion planning is rather limited [17, 18]. In our previous work [19], the Kino-RRT* algorithm with a dimensionality reduction heuristic is developed, in which the heuristic is obtained by solving a partial final-state free (PFF) optimal control problem. Instead of sampling the full state space, Kino-RRT* only samples part of the state space while the rest of the states are selected by the PFF optimal controller. By sampling in the reduced state space and utilizing the PFF optimal controller, Kino-RRT* shows faster convergence. An analytical solution for the PFF optimal control problem for linear systems is also derived in [19].

In this paper, we propose the FMT*PFF, which is built on our previous works [12] and [19]. We extend the dimensionality reduction heuristic to learning-based planners and train neural network controllers to solve the PFF optimal control problem. Compared to [12], the dimensionality reduction heuristic is used in FMT*PFF to improve the convergence rate of the algorithm. Also, training a PFF neural network controller is simpler than the set-to-set controller in [12]. Compared to [19], solving PFF using supervised learning allows us to deal with nonlinear system dynamics more efficiently. Furthermore, while [12] and [19] are based on the RRT* algorithm, FMT*PFF is based on the FMT* algorithm to benefit from FMT* algorithm's ordered search.

The contributions of the paper are as follows: First, we propose a dimensionality reduction heuristic for accelerating sampling-based kinodynamic planning for nonlinear systems. Second, we propose a neural network controller for solving the partial final-state free (PFF) optimal control problem. The proposed PFF neural network controller is used as the steering function in sampling-based kinodynamic motion planning algorithms; The FMT*PFF algorithm is developed for planning with learning-based steering functions that cannot achieve exact steering. Finally, we provide extensive simulations and comparisons with previous methods demonstrating the better performance of our method.

II. Preliminaries

A. Problem Statement

We consider the optimal kinodynamic motion planning problem formulated as the following optimal control problem (OCP),

$$\min_{u, t_{\mathrm{f}}} \quad J = \int_0^{t_{\mathrm{f}}} c(x, u) \, \mathrm{d}\tau, \tag{1a}$$

$$\text{s.t.} \quad \dot{x} = f(x, u), \tag{1b}$$

$$x(0) = x_s, \; x(t_{\mathrm{f}}) = x_g, \tag{1c}$$

$$u \in U, \; x \in X_{\mathrm{free}}, \; \forall t \in [0, t_{\mathrm{f}}]. \tag{1d}$$

where $x \in X \subset \mathbb{R}^{n_x}$ is the state, $u \in U \subset \mathbb{R}^{n_u}$ is the control input, x_s and x_g are the initial state and goal state, respectively. The free space is denoted by $X_{\mathrm{free}} \subset X$, where at each $x \in X_{\mathrm{free}}$, the system does not collide with any obstacles in the environment. Finally, $c(x, u)$ is a cost function that we aim to minimize.

The goal of the optimal kinodynamic motion planning problem is to find a control trajectory $u(t)$, $t \in [0, t_{\mathrm{f}}]$, such that the solution state trajectory $x(t)$ is obstacle-free, reaches the goal state, and minimizes a cost.

B. Partial-Final-State-Free Optimal Controller

Sampling-based motion planning algorithms such as RRT* and FMT* solve the problem (1a)-(1d) by growing a tree. The state space is approximated by random samples. The transition between samples is achieved using optimal steering functions. The sampled nodes and the connections between nodes define a graph. A trajectory tree is obtained by searching over this graph.

In kinodynamic RRT* and FMT*, the edge between two sampled states, x_a and x_b, is constructed using a steering function which is the solution of a TPBVP given by

$$\min_{u,t_\mathrm{f}} \quad J = \int_0^{t_\mathrm{f}} c(x,u)\,\mathrm{d}\tau, \tag{2a}$$

$$\text{s.t.} \quad \dot{x} = f(x,u), \tag{2b}$$

$$x(0) = x_a, \; x(t_\mathrm{f}) = x_b, \tag{2c}$$

$$u \in U, \; \forall t \in [0,t_\mathrm{f}]. \tag{2d}$$

Note that the obstacle constraint in (1d) is removed in this TPBVP.

In our proposed FMT*PFF, instead of sampling the full state x from the full state space X, we sample a partial state $\bar{x}$ from a state space of reduced dimensionality $\bar{X}$. Let $x = [x_1^\top \; x_2^\top]^\top$, where $x_1 \in \mathbb{R}^{n_1}$, $x_2 \in \mathbb{R}^{n_2}$, and $n_1 + n_2 = n_x$. We introduce the partial-final-state-free (PFF) optimal control problem as follows

$$\min_{u,t_\mathrm{f}} \quad J = \int_0^{t_\mathrm{f}} c(x,u)\,\mathrm{d}\tau, \tag{3a}$$

$$\text{s.t.} \quad \dot{x} = f(x,u), \tag{3b}$$

$$x(0) = x_a, \; x_1(t_\mathrm{f}) = \bar{x}_c, \tag{3c}$$

$$u \in U, \; \forall t \in [0,t_\mathrm{f}]. \tag{3d}$$

Compared to (2c), instead of fixing the state $x(t_\mathrm{f})$, only $x_1(t_\mathrm{f})$ is fixed and $x_2(t_\mathrm{f})$ is free in (3c).

Note that, after solving the problem (3a)-(3d), we obtain the full state trajectory which includes the full final state $x(t_\mathrm{f})$. Thus, the PFF optimal controller chooses the remaining free final state to minimize the cost. If we set this full final state $x(t_\mathrm{f})$ as the terminal state in the terminal constraint in problem (2a)-(2d) and solve the problem (2a)-(2d), we will get the same trajectory as in problem (3a)-(3d). Therefore, the partial state sampling and the PFF optimal controller work as an intelligent heuristic for state-space sampling.

For linear systems and quadratic cost functions, the analytical solution of the problem (3a)-(3d) is derived in [19]. For general nonlinear systems and cost functions, solving (3a)-(3d) efficiently is the bottleneck for sampling-based kinodynamic planning algorithms. Methods that integrate a numerical solver within a sampling-based planner to solve (2a)-(2d) have been studied [20]. However, the long computation time makes them unacceptable for practical use. In this paper, we use the PFF optimal controller as the steering function, and we train a neural network controller to solve the PFF optimal control problem. We show that the proposed FMT*PFF algorithm with a neural network controller is efficient and generates dynamically feasible motion plans.

III. The FMT*PFF Algorithm

The main differences between FMT*PFF and the original kinodynamic FMT* are: a) state sampling is performed in a reduced state space; b) the use of the PFF optimal controller as the steering function. Furthermore, in our case, the PFF optimal controller is approximated using a neural network.

The FMT*PFF algorithm is given by Algorithm 1 and a graphical illustration is given in Figure 1. We use $\bar{x}$ to represent a partial state in the reduced state space and $\bar{X}$ to represent the partial state set. We use x to represent a (full)) state in the (full) state space, and X to represent the set of (full) states. Some primitive procedures are given as follows.
Sampling: The sampling procedure SamplePFF(m) randomly samples m partial states in the *reduced* state space. The sampled partial states are collision-free in the corresponding reduced state space. For example, for a robot whose state space includes the position space and the velocity space, SamplePFF samples positions of the robot that are collision-free.
Near Nodes: The function Near($x, \bar{X}$) returns all the partial states in $\bar{X}$ that are contained in a ball of radius r centered at x. The function Near($X, \bar{x}$) returns all the states in X that are contained in a ball of radius r centered at $\bar{x}$. One simple implementation of the distance function is the Euclidean distance in the partial state space.
Collision Checking: The function CollisionFree(τ) takes a trajectory τ (an edge segment) as an input and returns true if and only if τ lies entirely in the collision-free space.
Cost: The procedure Cost(x) returns the cost-to-come from the root node to x.
Segment Cost: The procedure SegCost($x_i, \bar{x}_j$) returns the cost to go from x_i to $\bar{x}_j$. This cost is obtained by solving the
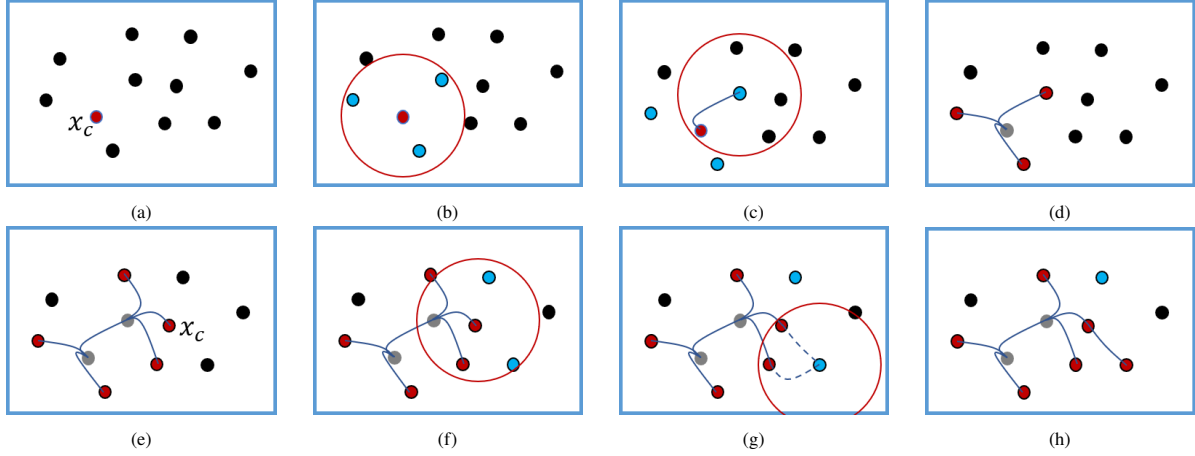
Fig. 1 An illustration of the FMT*PFF algorithm. (a) m new samples, shown as black dots, are sampled in the reduced state space using SamplePFF. The starting state x_s (red dot) is added to the tree. Since it is the only state in the tree (and the set X_{open}), x_c is set to be x_s. (b) The neighboring states, shown in blue, are found. (c) Find the best parent in the tree for all neighboring states. The partial final-state free (PFF) controller is used for steering. (d) Completed One iteration. (e) At some iteration of the algorithm, the state x_c with the minimum cost-to-come in X_{open} is selected. (f) The neighboring states, shown in blue, are found. (g) Find the best parent in the tree for a neighboring state using SteerPFF. (h) The best parent is found. The new state x_{new} is added to the tree.

Algorithm 1: FMT*PFF

1   $v_s.x \leftarrow x_s;\ v_s.\bar{x} \leftarrow \bar{x}_s;$
2   $V \leftarrow \{v_s\};\ E \leftarrow \emptyset;\ G \leftarrow (V, E);$
3   $\bar{X}_{\text{unvisited}} \leftarrow \{\bar{x}_g\} \cup \text{SamplePFF}(m);$
4   $X_{\text{open}} \leftarrow \{x_s\};$
5   $x_c \leftarrow x_s;\ \bar{x}_c \leftarrow \bar{x}_s;$
6   **while** $\bar{x}_c \neq \bar{x}_g$ **do**
7      $X_{\text{open,new}} \leftarrow \emptyset;$
8      $\bar{X}_{\text{near}} \leftarrow \text{Near}(x_c, \bar{X}_{\text{unvisited}});$
9   **foreach** $\bar{x}_{\text{near}} \in \bar{X}_{\text{near}}$ **do**
10        $X_{\text{near}} \leftarrow \text{Near}(X_{\text{open}}, \bar{x}_{\text{near}});$
11        $x_{\text{parent}} = \underset{x \in X_{\text{near}}}{\text{argmin}}(\text{cost}(x) + \text{SegCost}(x, \bar{x}_{\text{near}}));$
12        $(\tau, x_{\text{new}}) \leftarrow \text{SteerPFF}(x_{\text{parent}}, \bar{x}_{\text{near}});$
13   **if** $\text{CollisionFree}(\tau)$ **then**
14          $v.x \leftarrow x_{\text{new}};\ v.\bar{x} \leftarrow \bar{x}_{\text{near}};$
15          $V \leftarrow V \cup \{v\};$
16          $E \leftarrow E \cup \{(v(x_{\text{parent}}), v)\};$
17          $\bar{X}_{\text{unvisited}} \leftarrow \bar{X}_{\text{unvisited}} \setminus \bar{x}_{\text{near}};$
18          $X_{\text{open,new}} \leftarrow X_{\text{open,new}} \cup \{x_{\text{new}}\};$
19      $X_{\text{open}} \leftarrow X_{\text{open}} \cup X_{\text{open,new}} \setminus \{x_c\};$
20   **if** $X_{\text{open}} = \emptyset$ **then**
21   **return** Failure;
22      $x_c = \underset{x \in X_{\text{open}}}{\text{argmin}}(\text{cost}(x));$
23      $\bar{x}_c \leftarrow v(x_c).\bar{x}$
24   **return** $G;$

PFF optimal control problem with boundary conditions x_i and $\bar{x}_j$. We can train a neural network to predict this edge cost.

Steering: The procedure $\mathsf{SteerPFF}(x_i, \bar{x}_j)$ solves the TPBVP using the PFF optimal controller, and it returns a trajectory τ that starts from x_i and ends at $\bar{x}_j$.

In Algorithm 1, every vertex v is associated with a state $v.x$ and the corresponding partial state $v.\bar{x}$. We initialize the tree in Line 1-2. In Line 3, the m partial states and the goal are added to the unvisited set $\bar{X}_{\text{unvisited}}$, which is a set of partial states that have not been added to the tree. X_{open} is the set of states that have already been added to the tree. They are the frontier nodes of the tree that will be extended next. x_c is the state in X_{open} that has the minimum cost-to-come (Line 22) and $\bar{x}_c$ is the partial state associated with x_c. These are initialized in Line 5. If $\bar{x}_c = \bar{x}_g$, the solution has been found. Otherwise, we try to extend the tree.

In Line 8, the neighboring partial states of x_c in $\bar{X}_{\text{unvisited}}$, $\bar{X}_{\text{near}}$, are found. For every $\bar{x}_{\text{near}}$ in $\bar{X}_{\text{near}}$, we try to find its best parent in X_{open} (Line 9-18). In Line 10, the neighboring state of $\bar{x}_{\text{near}}$ in X_{open}, X_{near}, are found. The best parent for $\bar{x}_{\text{near}}$ that results in the minimum cost-to-come for $\bar{x}_{\text{near}}$ is obtained in Line 11. The trajectory τ and the full state x_{new} are obtained in Line 12 using the PFF steering function. If τ is collision-free, the new vertex v and new edge are added to the tree, and the sets $\bar{X}_{\text{unvisited}}$ and $X_{\text{open,new}}$ are updated (Line 13-18). $v(x_{\text{parent}})$ denotes the vertex associated with x_{parent} and $v(x_c).\bar{x}$ denotes the partial state associated with x_c.

A. A Neural Network Controller for PFF Optimal Control

The proposed FMT*PFF can directly work with the analytical solution of the PFF optimal control problem, a numerical solver for the PFF optimal control problem, and a learning-based steering function for the PFF optimal control problem. In this section, we describe how to train the neural network controller for computing the PFF optimal control.

We first generate the training data. For a dynamical system, we use numerical optimization solvers [21] to solve the PFF optimal control problem given by (3a)-(3d) offline. Since we are interested in steering the system from different initial states to the partial final state, we generate optimal trajectories with different initial states. The initial states are uniformly sampled from an initial state set. We choose the partial final state to be the position, and let the remaining state (heading angle, velocity, etc.) be free.

We utilize the translational invariance of the trajectories. If the goal is not the origin, we can translate the goal position to the origin and translate the position of the starting state accordingly. After solving the optimal control problem with the translated boundary condition, we obtain the trajectory of the original problem by translating the solution trajectory back. Therefore, we choose the goal position to be at the origin (zero). Translational invariance regarding the position is common for systems such as double integrators, cars, and UAVs.

Note that we only need to steer the system to nearby states given by the Near function in FMT*PFF. Thus, a "local" PFF optimal controller is sufficient. This makes the learning-based controller using a neural network well-suited for this task since we can only generate finite training data. When using the neural network for prediction, it is important to make sure we are using it for interpolation instead of extrapolation. Extrapolation will diminish prediction accuracy. For this purpose, the offline training data should cover the neighborhood used in the Near function.

Each trajectory returned by the numerical solver contains a sequence of control inputs and a sequence of states indexed by time. We combine all the data points from all offline trajectories to form the final training dataset. Each data point is a tuple (x_i, u_i). The goal of the neural network is to mimic the structure of the optimal controller. The neural network controller, $\mathsf{NNController} : x_i \to u_i$, is a state feedback controller mapping from the current state x_i to the current control u_i to be applied.

After training the neural network controller, it is used for online control and online trajectory generation. The $\mathsf{SteerPFF}$ in Algorithm 1 is obtained by applying the neural network controller to obtain a trajectory. Given a novel initial state, which should be inside the initial state set used in training data generation, we repetitively apply the neural network control commands and simulate the system dynamics to obtain the trajectory that steers the system to the goal state, where the goal position is the origin. We also train a cost-to-go neural network that predicts the segment cost, $\mathsf{SegCost} : x_1 \to J$, where x_1 is the initial state, J is the cost of the trajectory from x_1 to the goal returned by the numerical solver.
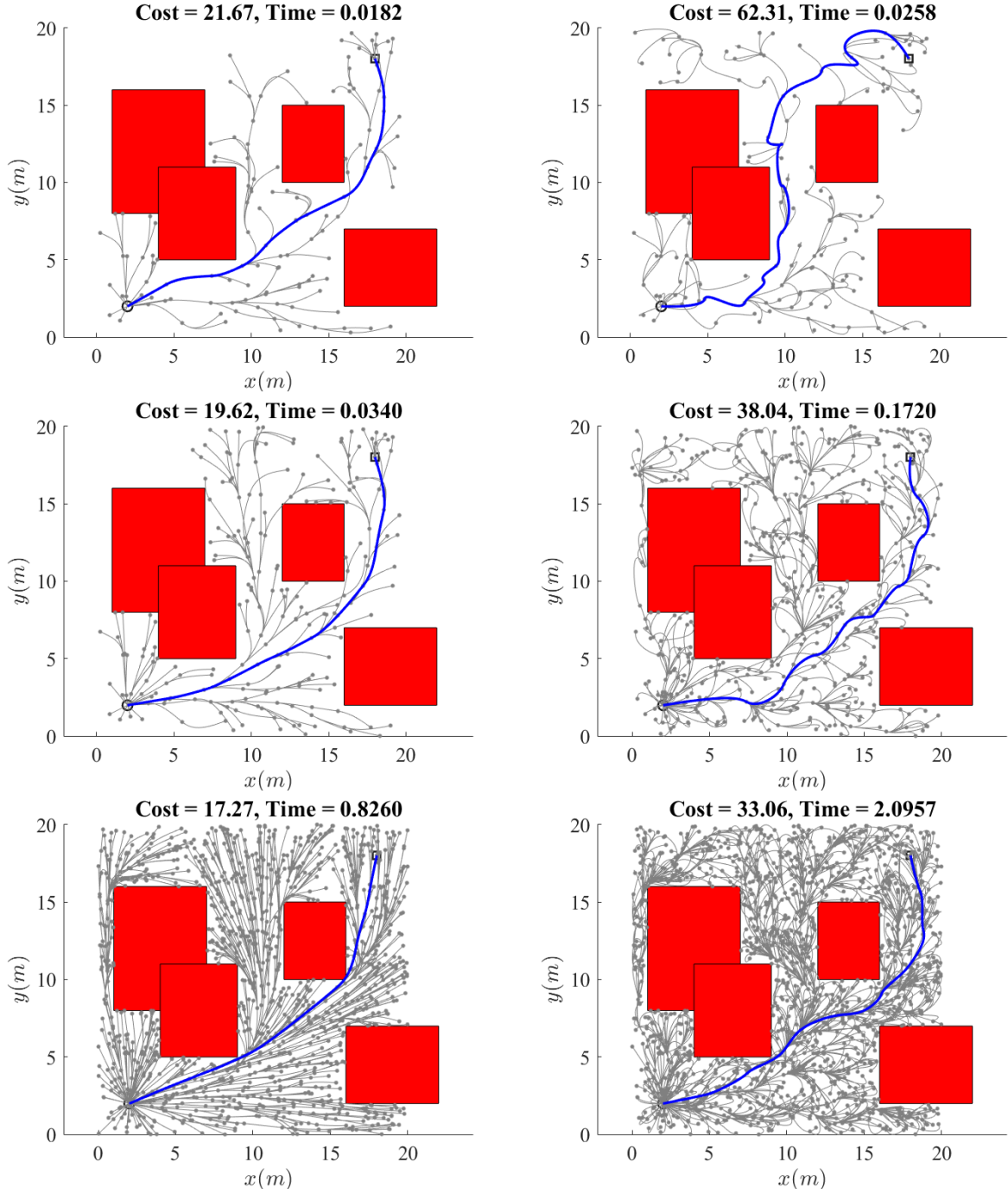
Fig. 2 FMT*PFF planning results. In each figure, the blue line is the solution from the current tree. The figures in the first column show the planning results of our proposed FMT*PFF algorithm with an increasing number of samples. The figures in the second column are the planning results of the original kinodynamic FMT*.

IV. Empirical Evaluation

We evaluate the proposed method using two examples. In the first example, we compare FMT*PFF with FMT* and Kino-RRT using a 2D double integrator model. The analytical solution of the PFF optimal controller for linear systems is used. In the second example, FMT*PFF is used for planning for a car model. Since the car model is nonlinear, a neural network controller is trained to solve the PFF optimal control problem and then incorporated into FMT*PFF. Our proposed method is general in that it can deal with various systems with nonlinear dynamics, providing that the PFF optimal controller is obtainable using learning-based methods. Learning a controller to solve the steering function for sampling-based motion planning itself is a research problem. Future work includes considering more nonlinear systems such as UAVs.

A. 2D Double Integrator

We first compare FMT*PFF with the kinodynamic FMT* [6] and the Kino-RRT* [19]. For this purpose, we consider a linear system and use the analytical solution for (2a)-(2d) and (3a)-(3d). The difference between FMT*PFF and the kinodynamic FMT* is that kinodynamic FMT* samples the full state-space and solves (2a)-(2d) for steering functions while FMF*PFF samples in the reduced state-space and solves (3a)-(3d) for steering functions. For comparison, we set the tuning parameters of the algorithms, such as the neighborhood radius r, to be the same. Both Kino-RRT* and FMT*PFF use the PFF optimal control, but they are based on RRT* and FMT*, respectively.

The state of the 2D double integrator is given by $x = [p^\top \ v^\top]^\top$, where $p = [x_1 \ x_2]^\top$ is the position and $v = [x_3 \ x_4]^\top$ is the velocity. The control input is acceleration. The system dynamics is given by $\dot{x} = Ax + Bu$, where

$$A = \begin{bmatrix} 0 & I_2 \\ 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ I_2 \end{bmatrix}.$$

The cost function $c(x, u) = 1 + u^\top Ru$, where $R = I_2$.

The position is uniformly sampled within the boundary of the environment. The free final state of the PFF controller is the velocity. Thus, FMT*PFF only samples the position space. For the kinodynamic FMT* algorithm, the velocity is uniformly sampled in $v \in [-2, 2]^2$ m/s^2. Note that a larger interval for the velocity essentially requires searching in a larger state space, which will result in slower convergence. However, if the sampling velocity interval is too small, the search is confined to a small state space that may not contain the optimal solution.

The planning results of the FMT*PFF algorithm and the Kinodynamic FMT* algorithm are given in Figure 2. By sampling in the reduced state space and using a PFF optimal controller, FMT*PFF reduced the dimensionality of the planning problem. Thus, FMT*PPF finds a better trajectory from the beginning and continues to find better solutions given the same amount of planning time. For Kinodynamic FMT*, the probability of sampling good velocities to decrease the cost is low.

The solution cost vs planning time comparison is given in Figure 3. We can see from Figure 3 that FMT*PFF also has better convergence performance compared to Kino-RRT*. This is because FMT*PFF uses ordered search while Kino-RRT* uses unordered random search. One drawback of FMT*PFF is that it is not an anytime algorithm. Still, the benefit of FMT*PFF becomes more clear when planning with learning-based steering functions that cannot achieve exact steering, since FMT*PFF does not use a rewiring procedure.

B. A Simple Car Model

A kinematic car model is modeled by the equations

$$\dot{x} = v\cos(\theta), \tag{4a}$$
$$\dot{y} = v\sin(\theta), \tag{4b}$$
$$\dot{\theta} = u_1, \tag{4c}$$
$$\dot{v} = u_2, \tag{4d}$$

where (x, y) is the position, θ is the heading angle, v is the speed, u_1 and u_2 are the control inputs.

We first generate offline training data by solving PFF optimal control problems using numerical optimization solvers. The offline trajectory examples used for training are shown in Figure 4. We sample initial states from an initial state set. The sampling intervals of x, y, θ, and v are $x \in [-4, 4]$ m, $y \in [-4, 4]$ m, $\theta \in [-\pi, \pi]$ rad, and
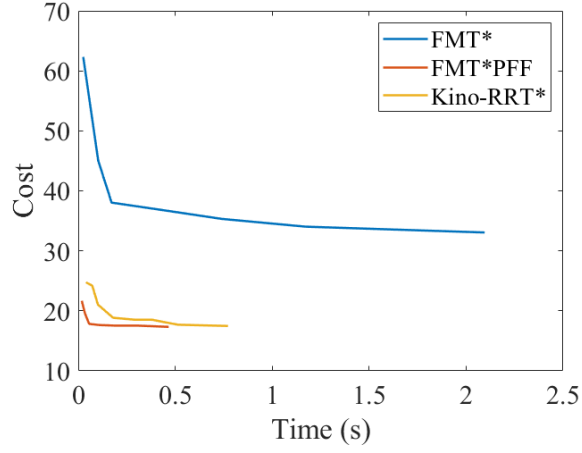
Fig. 3 Comparison of FMT*, FMT*PFF, and Kino-RRT* algorithm. Compared with FMT*, the dimensionality reduction heuristic in Kino-RRT* and FMT*PFF improve their performance significantly. FMT*PFF also performs better than Kino-RRT* due to its ordered search.
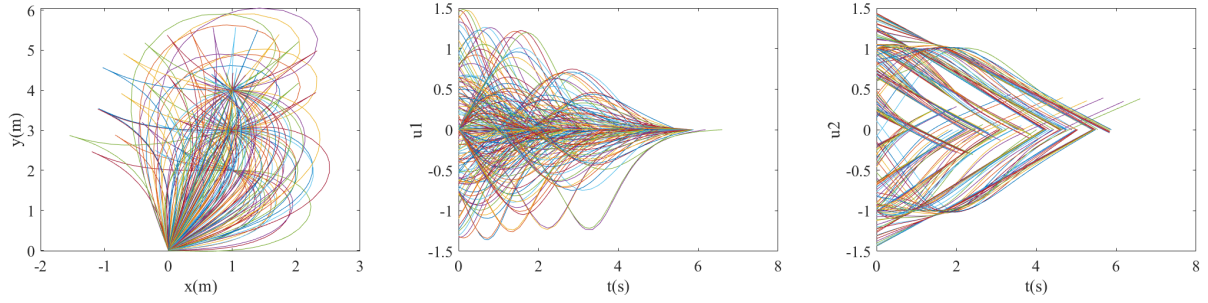

Fig. 4 Offline generated trajectories for neural network training. The first figure gives the trajectories of the position. The last two figures show the control trajectories.
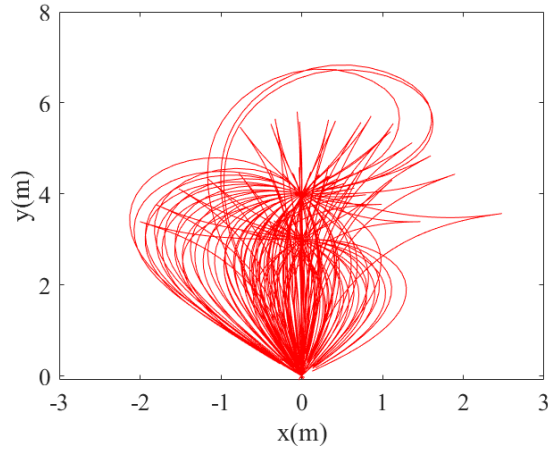

Fig. 5 Trajectories generated using the neural network controller. The initial states are sampled from the initial state set. The goal of the learning-based steering function is to steer the system to reach the goal position (the origin). By repeatedly applying the state-feedback neural network controller and forward simulating the system dynamics, the system reaches a neighborhood of the goal.

$v \in [-2, 2]$ m/s, respectively. The reduced state space is the position space and θ and v are free states. Thus, the goal state is $(x, y, \theta, v) = [0\ 0\ \text{free free}]$. For this example, 10,400 trajectories are generated. State-action pairs from the trajectories were used for neural network training.
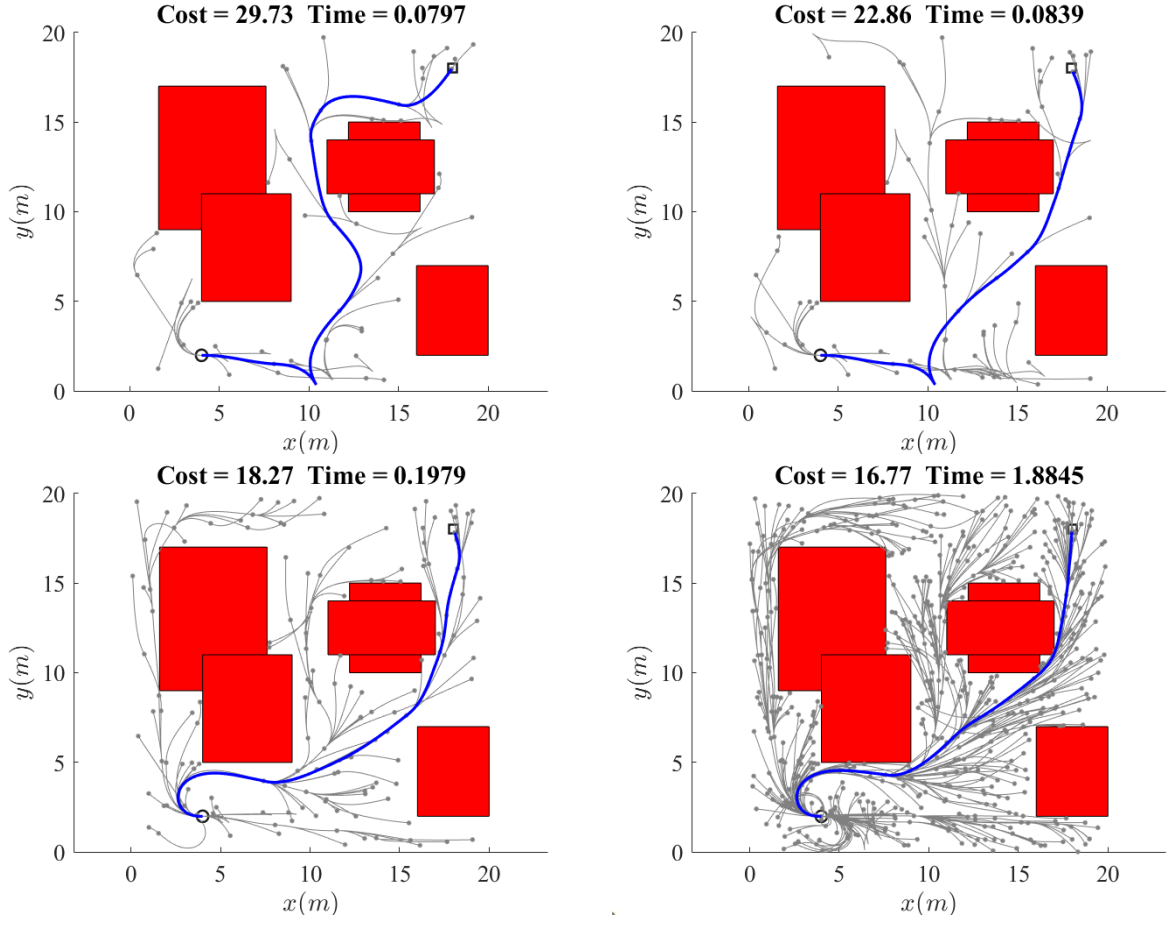

Fig. 6   FMT*PFF planning results for the car example. The blue line is the solution in the current tree.
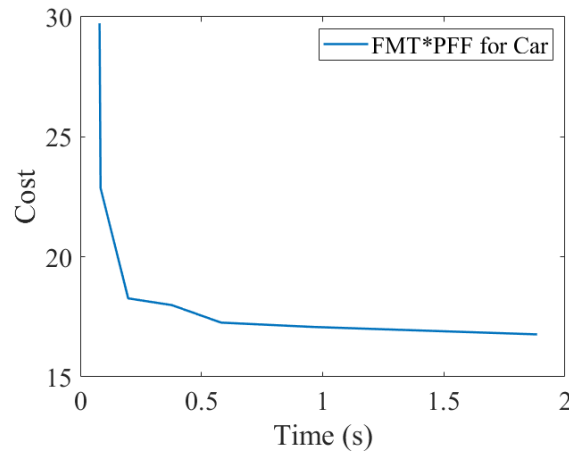

Fig. 7   Performance of FMT*PFF with a neural network controller for the car example.

After training the neural network, we tested the neural network controller using randomly sampled initial states.

9

Some example trajectories obtained using the neural network controller are shown in Figure 5. Since the goal of the neural network controller is to steer the system to reach the origin, one index is the error between the end position of the trajectories and the origin. 1000 trajectories corresponding to sampled novel initial states are generated using the neural network controller. 98% of the resulting trajectory reached a 0.3 neighborhood of the origin. If this end position error is too large (greater than 0.3), the connection is not successful, and this edge will not be added to the tree. Note that FMT*PFF does not require exact steering, which makes it suitable for learning-based steering functions. The final trajectory obtained from the FMT*PFF algorithm is smooth and satisfies the differential constraint, while the trajectory from [12] may have small gaps between edges.

The planning results of FMT*PFF using the neural network controller are shown in Figure 6 and Figure 7. Figure 6 shows the trees with a different number of samples and Figure 7 gives the cost vs time performance.

Finally, we use FMT*PFF to plan trajectories for the car model in various environment settings. We randomly sample the number, size, and location of the obstacles. We also vary the starting point and goal point of the car. The planning results are given in Figure 8. The FMT*PFF algorithm finds dynamically feasible trajectories using the neural network controller.

V. Conclusion

We propose the FMT*PFF algorithm for optimal kinodynamic motion planning for nonlinear systems. The key idea is the use of a partial-final-state-free (PFF) optimal controller to reduce the dimensionality of the sampling space and accelerate kinodynamic motion planning. FMT*PFF planning in the reduced state space and has faster convergence. By training a neural network model of the PFF optimal controller, FMT*PFF can plan trajectories for nonlinear systems in cluttered environments with nonconvex obstacles. FMT*PFF can deal with learning-based steering functions that can not achieve exact steering because no rewire is needed. We show that FMT*PFF is efficient and generates dynamically feasible motion plans. Through numerical simulations and comparison with previous works, FMT*PFF are shown to have better cost-time performance.

References

[1] Kavraki, L. E., Svestka, P., Latombe, J. C., and Overmars, M. H., "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, Vol. 12, No. 4, 1996, pp. 566–580.

[2] LaValle, S. M., and Kuffner Jr, J. J., "Randomized kinodynamic planning," *The International Journal of Robotics Research*, Vol. 20, No. 5, 2001, pp. 378–400.

[3] Karaman, S., and Frazzoli, E., "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, Vol. 30, No. 7, 2011, pp. 846–894.

[4] Gammell, J. D., and Strub, M. P., "Asymptotically Optimal Sampling-Based Motion Planning Methods," *Annual Review of Control, Robotics, and Autonomous Systems*, Vol. 4, 2021, pp. 1–25.

[5] Karaman, S., and Frazzoli, E., "Optimal kinodynamic motion planning using incremental sampling-based methods," *49th IEEE Conference on Decision and Control*, Atlanta, GA, 2010, pp. 7681–7687.

[6] Schmerling, E., Janson, L., and Pavone, M., "Optimal sampling-based motion planning under differential constraints: the drift case with linear affine dynamics," *54th IEEE Conference on Decision and Control (CDC)*, Osaka, Japan, 2015, pp. 2574–2581.

[7] Li, Y., Littlefield, Z., and Bekris, K. E., "Asymptotically optimal sampling-based kinodynamic planning," *The International Journal of Robotics Research*, Vol. 35, No. 5, 2016, pp. 528–564.

[8] Perez, A., Platt, R., Konidaris, G., Kaelbling, L., and Lozano-Perez, T., "LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics," *IEEE International Conference on Robotics and Automation*, Saint Paul, MN, 2012, pp. 2537–2542.

[9] Webb, D. J., and Van Den Berg, J., "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics," *IEEE International Conference on Robotics and Automation*, Karlsrühe, Germany, 2013, pp. 5054–5061.

[10] Wolfslag, W. J., Bharatheesha, M., Moerland, T. M., and Wisse, M., "RRT-CoLearn: Towards kinodynamic planning without numerical trajectory optimization," *IEEE Robotics and Automation Letters*, Vol. 3, No. 3, 2018, pp. 1655–1662.
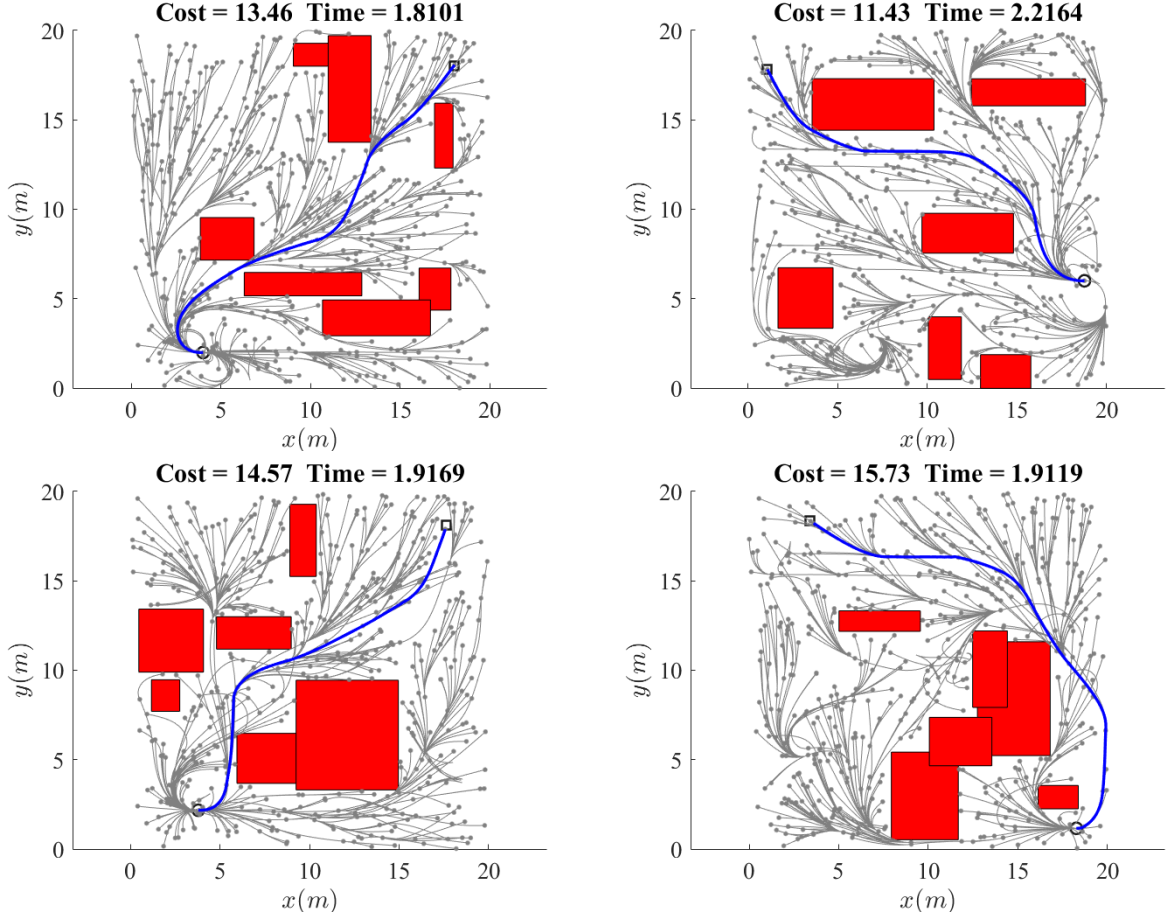
Fig. 8 Planning results for the car in different environments.

[11] Chiang, H. T. L., Hsu, J., Fiser, M., Tapia, L., and Faust, A., "RL-RRT: Kinodynamic motion planning via learning reachability estimators from RL policies," *IEEE Robotics and Automation Letters*, Vol. 4, No. 4, 2019, pp. 4298–4305.

[12] Zheng, D., and Tsiotras, P., "Sampling-based Kinodynamic Motion Planning Using a Neural Network Controller," *AIAA Scitech Forum*, 2021, p. 1754.

[13] Zheng, D., and Tsiotras, P., "Near-optimal finite-time feedback controller synthesis using supervised and unsupervised learning," *AIAA Scitech Forum*, 2021, p. 1950.

[14] Gammell, J. D., Srinivasa, S. S., and Barfoot, T. D., "Informed RRT: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 2997–3004.

[15] Janson, L., Schmerling, E., Clark, A., and Pavone, M., "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," *The International Journal of Robotics Research*, Vol. 34, No. 7, 2015, pp. 883–921.

[16] Gammell, J. D., Barfoot, T. D., and Srinivasa, S. S., "Batch Informed Trees (BIT*): Informed asymptotically optimal anytime search," *The International Journal of Robotics Research*, Vol. 39, No. 5, 2020, pp. 543–567.

[17] Paden, B., Varricchio, V., and Frazzoli, E., "Verification and synthesis of admissible heuristics for kinodynamic motion planning," *IEEE Robotics and Automation Letters*, Vol. 2, No. 2, 2017, pp. 648–655.

[18] Yi, D., Thakker, R., Gulino, C., Salzman, O., and Srinivasa, S., "Generalizing informed sampling for asymptotically-optimal sampling-based kinodynamic planning via markov chain monte carlo," *IEEE International Conference on Robotics and Automation*, Brisbane, Australia, 2018, pp. 7063–7070.

[19] Zheng, D., and Tsiotras, P., "Accelerating Kinodynamic RRT* Through Dimensionality Reduction," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 3674–3680.

[20] Xie, C., van den Berg, J., Patil, S., and Abbeel, P., "Toward asymptotically optimal motion planning for kinodynamic systems using a two-point boundary value problem solver," *IEEE International Conference on Robotics and Automation*, Seattle, WA, 2015, pp. 4187–4194.

[21] Patterson, M. A., and Rao, A. V., "GPOPS-II: A MATLAB software for solving multiple-phase optimal control problems using hp-adaptive Gaussian quadrature collocation methods and sparse nonlinear programming," *ACM Transactions on Mathematical Software*, Vol. 41, No. 1, 2014, pp. 1–41.