Robotics: Science and Systems 2022
New York City, NY, USA, June 27-July 1, 2022

Meta Value Learning for Fast Policy-Centric Optimal Motion Planning

Siyuan Xu and Minghui Zhu
Pennsylvania State University
Email: {spx5032, muz16}@psu.edu

***Abstract*—This paper considers policy-centric optimal motion planning with limited reaction time. The motion planning queries are determined by their goal regions and cost functionals, and are generated over time from a distribution. Once a new query is requested, the robot needs to quickly generate a motion planner which can steer the robot to the goal region while minimizing a cost functional. We develop a meta-learning-based algorithm to compute a meta value function, which can be fast adapted using a small number of samples of a new query. Simulations on a unicycle are conducted to evaluate the developed algorithm and show the anytime property of the proposed algorithm.**

I. INTRODUCTION

Motion planning is a fundamental problem and it aims to steer a mobile robot from an initial state to a goal set, while satisfying dynamic constraints and environmental rules. It is well-known that the problem is computationally challenging, especially when the problem's dimension is high. For example, the warehouseman's problem is shown to be PSPACE-hard [9]. Sampling-based algorithms, e.g. Probabilistic RoadMap (PRM) [15] and Rapidly-exploring Random Tree (RRT) [22], are particularly efficient in finding feasible paths in high-dimensional spaces. Optimal motion planning seeks a collision-free path which minimizes a given cost functional, and is computationally harder than feasible motion planning. RRT* [13] is shown to be both computationally efficient and asymptotically optimal. Paper [14] extends RRT* to handle non-holonomic dynamical systems, and SST* [23] relaxes the use of steering function in [14].

The above motion planners are path-centric, i.e., searching for a collision-free path connecting a given initial state and a goal. Afterwards a feedback controller for path following, e.g., model predictive control (MPC) method [3], or trajectory optimization [16], is synthesized to track the path. As pointed out in [32], motion model inaccuracies, random disturbances, and control signal saturation inevitably introduce additional errors, making tracking precomputed paths difficult. In contrast, policy-centric motion planners compute feedback control policies over the whole state space. Policy-centric optimal motion planning of kinodynamic systems is essentially optimal control of nonlinear dynamic systems, e.g., unicycle robots and quadcopter, subject to state and input constraints. Notice that this class of control problems is known to be computationally challenging and suffers from the curse-of-dimensionality [2]. Papers [19, 20] leverage Fast Marching Method (FMM) [29], a numerical solver of Hamilton-Jacobi-Bellman (HJB) equations, to compute the optimal value function on a simplicial complex rather than a state grid. Repairing Fast Marching Method (ReFMM) in [31, 32] proposes a repairing modification of FMM to adaptively refine the simplicial complex mesh, and is shown to be asymptotically optimal and fast in replanning during the mesh refinement. Several methods, e.g., MPC, have been proposed to trade optimality for scalability. MPC uses a sequence of finite-horizon optimal control problems online to approximate an infinite-horizon optimal control problem. Paper [7] employs the MPC method with the probing feature to plan a globally convergent trajectory, and employs another controller to track the trajectory.

Meta-learning or learning-to-learn has a potential to develop fast algorithms for policy-centric optimal motion planning. Meta-learning is oriented to multi-task learning scenarios and employs a learning model to extract task-agnostic knowledge from a family of related tasks and harness the knowledge to improve the learning of new tasks from the family. Meta-learning is shown to be effective in addressing many conventional challenges of deep learning, including data and computation bottlenecks, as well as generalization [11]. It has been demonstrated that meta-learning leads to state-of-the-art performance on few-shot image classification, few-shot regression and few-step policy fine-tuning of reinforcement learning (RL) [6, 8]. In meta-reinforcement learning (MRL) [8], the dynamic systems are modeled by Markov Decision Processes (MDPs) and motion planning is formulated as an RL problem and solved by REINFORCE [27] and trust-region policy optimization (TRPO) [28]. When the meta policy is trained, its adaptation to a new task only requires a few steps of gradient descent. In [6, 8], MDPs are limited to discrete time space and the value functions are characterized by Bellman equations. In kinodynamic motion planning, it is a standard practice that dynamic systems of robots, e.g., unicycle, Dubins car, and quadcopter, are modeled as differential equations in continuous time, state, and control spaces [21, 22]. Correspondingly, HJB equations, continuous counterparts of Bellman equations, are used to characterize value functions.

Contributions. The paper develops the first meta-learning approach which can efficiently solve policy-centric motion planning of nonlinear robot dynamics in continuous time, state, and control domains. In particular, we study the problem where a robot faces a family of motion planning queries. Each query is characterized by a goal region and a running cost functional, which is drawn from a known probability distribution. Once a specific query is requested, the robot needs to quickly generate

a motion planner which steers it to the goal while minimizing its running cost functional. First, we leverage the set-valued method in [5] to efficiently approximate the optimal value function and the optimal controller of each specific query on a coarse state grid. Second, the obtained optimal value functions are used to train a meta value function modeled by a deep neural network (DNN). The meta training aims to minimize the distance between the optimal value functions and the DNN with one-step gradient descent subject to a hard constraint derived from the HJB equation. Third, we introduce an anytime adaptation law which quickly updates the meta value function once a new query is requested. The update only requires the optimal values of a small number of states for the new query and the data are efficiently generated by the causality-free method. Simulations on a unicycle robot show that the algorithm is able to return a near-optimal control policy quickly and keeps improving policy optimality if more time is given. Our key insights are as follows. The meta value learning approach identifies an optimal initialization of the DNN training for the whole query family. The meta training searches for an initial DNN model which has the minimal cost after running a few steps of query-specific training. So this initialization tends to be closer to a good parameter of the query which is more difficult to be learned, i.e., requiring more steps of training, so that a few steps of training on both easy and difficult queries leads to a low expected cost.

Notations. Define the distance from a point $x \in X$ to a set $A \subseteq X$ as $d(x, A) \triangleq \inf\{\|x - a\| \mid a \in A\}$. The unit ball centered at 0 is denoted as $\mathcal{B}$. The δ expansion of a set A is defined as $A + \delta\mathcal{B} \triangleq \{x \mid d(x, A) \leq \delta\}$ for some $\delta \geq 0$. For a set X and a mapping f, we define the image set $f(X) \triangleq \{f(x) \mid x \in X\}$.

II. Problem Formulation

A. System model

Consider a mobile robot whose dynamic system is governed by the following differential equation:

$$\dot{x}(t) = f(x(t), u(t)), \tag{1}$$

where $x(t) \in X$ is the state of the robot and $u(t) \in U$ is its control. Let $X \subseteq \mathbb{R}^n$ and $U \subseteq \mathbb{R}^m$ be the state space and the set of all possible control values, respectively. The goal region $X_G \subseteq X$. Assume that f is Lipschitz continuous in both variables. The running cost functional $L(x, u)$ satisfies $L(x, u) = 0$ when $x \in X_G$, and $L(x, u) > 0$ when $x \in X \setminus X_G$.

B. Policy-centric optimal motion planning

Given a goal region X_G and a running cost functional L, a policy-centric planner is synthesized to steer the robot to reach the goal region with minimum cost. The task is formulated as the following optimal control problem:

$$\begin{cases} \min\limits_{u(\cdot)\in\Pi} & \int_0^\infty L(x(t), u(t))dt, \\ \text{s.t.} & \dot{x}(t) = f(x(t), u(t)), \\ & x(0) = x,\ x(t) \in X,\ \forall t \geq 0. \end{cases} \tag{2}$$

Here the set of time-invariant state feedback control policies is defined as $\Pi \triangleq \{u(\cdot) : X \to U\}$. The minimal value of (2) is the optimal value function denoted as $V^*(x; X_G, L)$ and the optimal solution is the optimal controller denoted as $u^*(x; X_G, L)$. Since L and f are time-invariant, so is $V^*(x; X_G, L)$. Assume that $V^*(x; X_G, L)$ is continuously differentiable in X for any X_G and L.

C. Fast policy-centric motion planning

In the fast policy-centric motion planning problem, the robot is required to visit a set of goal regions which are revealed sequentially in a time-varying environment. We consider a family of motion planning queries and each query is characterized by a query configuration, i.e., the pair of the goal region $X_G^{(i)}$ and the running cost functional $L^{(i)}$. The pair of $(X_G^{(i)}, L^{(i)})$ is drawn from a probability distribution $p(X_G, L)$. We assume that the query configuration distribution $p(X_G, L)$ is given in advance. For example, in the dynamic vehicle routing problem [4], a vehicle travels to provide on-site service while minimizing the expected waiting time of the demands, where the inter-arrival times follow a Poisson process and the routing demands for service follow a time-invariant spatio–temporal Poisson process. Our objective is to develop an approach which can quickly generate a nearly optimal solution of (2) once a specific query is requested.

III. Main Result

In this section, we develop a meta-learning approach to solve the fast optimal motion planning problem. First, we efficiently approximate the optimal value function and the optimal controller for each query on a coarse state grid by the set-valued method in [5] and deep learning is employed to interpolate the optimal value function and the optimal controller in the continuous state space (Section III-A). Second, we learn the meta value function DNN by minimizing the distance between the discretized optimal value functions obtained in the first step and the DNN obtained by applying one-step gradient descent to the meta parameter (Section III-B). Third, we introduce the anytime adaptation approach which quickly updates the meta value function in response to a newly requested query (Section III-C). The adaptation only requires the optimal values of a small number of states for the new query and the data are efficiently generated by the causality-free method in [12, 25].

A. Query-specific optimal motion planning

Assume that the query is specified. For notational simplicity, we drop X_G and L in this section. For any admissible control policy [1] $u \in \Pi$, assume the following cost function is continuously differentiable:

$$V^u(x) \triangleq \int_0^\infty L(x(t), u(x(t)))dt, \quad x(0) = x. \tag{3}$$

Define the Hamiltonian function as

$$H\left(x, u, \nabla_x V^u\right) \triangleq L(x, u) + \nabla_x V^u(x)^T f(x, u). \tag{4}$$

With the Hamiltonian (4), the HJB equation is given by:

$$\min_{u\in\Pi} H\left(x,u,\nabla_x V^*\right) = H\left(x,u^*,\nabla_x V^*\right) = 0. \tag{5}$$

The optimal controller minimizes the Hamiltonian, i.e.,

$$u^*(x) = \arg\min_{u\in\Pi} H\left(x,u,\nabla_x V^*\right). \tag{6}$$

To solve (5), a widely used idea is to discretize the system and perform dynamic programming on a discrete state grid. However, dynamic programming suffers from the curse-of-dimensionality [2], i.e., the computational complexity exponentially increases as the size of the state grid. In what follows, we develop a new two-step solution. First, we use the set-valued method in [5] to obtain a discrete approximation of V^* in a coarse state grid X^d. Second, a DNN is employed to interpolate the discrete optimal value function in continuous state space X. Then an approximate optimal controller is computed by solving (6), where V^* is replaced by the interpolated optimal value function.

1) Set-valued approximation: The set-valued method in [5] is used to solve problem (2). Specifically, system (1) is approximated by a discretized set-valued dynamic system, and value iteration is executed until reaching a fixed point.

The set-valued method assumes that the state and input spaces are compact. To satisfy the requirement, we choose compact sets $\bar{X}$ which contains all the optimal trajectories and $\bar{U}$ which contains all the optimal control values. Then the state and control are constrained in compact sets $\bar{X}\subset X$ and $\bar{U}\subset U$, respectively. Since $\bar{X}$ and $\bar{U}$ are compact and f is Lipschitz, then $M \triangleq \max_{x\in\bar{X},u\in\bar{U}}\|f(x,u)\|$ exists.

The set-valued approximation algorithm is stated as follows. First, discretize X into X^d with spatial resolution h. Second, construct the differential inclusion $\dot{x}(t)\in F(x(t)),\forall t\geq 0$ to represent the dynamic system (1), where $F(x)\triangleq\left\{f(x,u)\mid u\in\bar{U}\right\}$. Because $f(x,u)$ is Lipschitz with respect to both variables, the set-valued map $F(x)$ is Lipschitz and the Lipschitz constant is denoted by l. Third, discretize the differential inclusion in the time and space domains and obtain the following discretized set-valued dynamic system:

$$x_{n+1}\in\begin{cases}(x_n+\varepsilon F(x_n)+\alpha\mathcal{B})\cap X^d \\ \qquad \text{if } d(x_n,X_G) > M\varepsilon + h \\ x_n, \text{ otherwise}\end{cases}$$

where $\varepsilon > 2h$ is the temporal resolution. The ball $\alpha\mathcal{B}$ represents perturbations on the dynamics and ensures the image set of any x is nonempty and the set-valued dynamic system is well-defined. Fourth, run value iteration (7) until reaching a fixed point, which is the optimal value function on X^d:

$$\begin{aligned}v_{n+1}(x) = \min_{u\in\bar{U}}\{&\varepsilon L(x,u)+v_n(x')\mid \\ &x'\in(x+\varepsilon f(x,u)+\alpha\mathcal{B})\cap X^d\}.\end{aligned} \tag{7}$$

The solution of (7) is the optimal control at state x.

The algorithm returns the optimal value function $\hat{V}$ and optimal control $\hat{u}$ of the discretized set-valued dynamic system on the state grid X^d. These values are included in set $\mathcal{D}^{tr}$ defined as follows:

$$\mathcal{D}^{tr}\triangleq\left\{X^d;\left(\hat{V}(X^d),\hat{u}(X^d)\right)\right\}. \tag{8}$$

2) DNN interpolation: Next, we employ $\mathcal{D}^{tr}$ to train a DNN which interpolates $\hat{V}$ and $\hat{u}$ in X. By [10], NNs can approximate any piecewise differentiable function with arbitrary accuracy and their derivatives can approximate the generalized derivatives of the function. Let V^{NN}: $X\to\mathbb{R}$ an approximation of V^* and it is represented by a NN as: $V^{NN}(x)\triangleq g_M\circ g_{M-1}\circ\cdots\circ g_1(x)$, where each layer $g_m(\cdot)$ is defined as $g_m(y)\triangleq\sigma_m\left(W_m y+b_m\right)$. Here W_m and b_m are the weights and bias vectors, respectively. In addition, $\sigma_m(\cdot)$ is a nonlinear activation function, e.g., ReLU or sigmoid. In this paper, we use ReLU for all the hidden layers. Typically, the final layer $g_M(\cdot)$ is a linear function, so $\sigma_M(\cdot)$ is the identity function. Let θ denote the collection of the parameters of V^{NN}, i.e. $\theta\triangleq\{W_m,b_m\}_{m=1}^M$. So $V^{NN}(x)$ is parameterized by θ and denoted as $V^{NN}(x;\theta)$.

The NN V^{NN} aims to interpolate $\hat{V}$ in X. To achieve so, the NN is trained by minimizing the following distance to $\hat{V}$ over the training data set D^{tr}:

$$\mathcal{L}_v\left(\theta,\mathcal{D}^{tr}\right)\triangleq\sum_{x\in X^d}\left[\hat{V}(x)-V^{NN}(x;\theta)\right]^2. \tag{9}$$

From (6), one can see that the optimal control u^* is determined by $\nabla_x V^*$. The approximate optimal control $\hat{u}$ in the set-valued approximation in Section III-A1 provides extra supervision for V^{NN} and is expected to improve the rudimentary loss function (9). We denote the mapping in (6) as $h:\mathbb{C}(X)\to\mathbb{C}(X)$, such that $u^*(x)=h(\nabla_x V)(x)$. The training of V^{NN} can be viewed as minimizing the loss function (9) subject to the following hard constraint:

$$\hat{u}(x) = h\left(\nabla_x V^{NN}(x;\theta)\right),\quad \forall x\in X^d. \tag{10}$$

Analogously, the underlying physics described by general nonlinear PDEs are included as hard constraints in the physics-informed NN approaches [26]. HJB equations are included as hard constraints when NNs are trained to solve optimal control [25, 30]. We define the violation of the constraint (10) as follows:

$$\mathcal{L}_u\left(\theta,\mathcal{D}^{tr}\right)=\sum_{x\in X^d}\left\|\hat{u}(x)-h\left(\nabla_x V^{NN}(x;\theta)\right)\right\|^2. \tag{11}$$

Consider the following Lagrangian:

$$\mathcal{L}\left(\theta,\mathcal{D}^{tr}\right)\triangleq\mathcal{L}_v\left(\theta,\mathcal{D}^{tr}\right)+\mu\cdot\mathcal{L}_u\left(\theta,\mathcal{D}^{tr}\right), \tag{12}$$

where $\mu\geq 0$ is the Lagrangian multiplier. Let θ^* be a global minimizer of the Lagrangian (12). The interpolated optimal value function is given by $V^{NN}(\ ;\theta^*)$ and the interpolated optimal controller $u^{NN}:X\to U$ is given by

$$u^{NN}(x)=h(\nabla_x V^{NN}(x;\theta^*)).$$

B. Meta value learning

The approach in Section III-A can solve the optimal motion planning problem if a query is given. However, it is not fast enough to handle a new query in real time. Instead, in this section, we develop the offline meta value learning which uses the set-valued approximation to generate the training data, the DNN structure to represent the optimal value function and the Lagrangian (12) to evaluate the models.

As stated in Section II-C, we consider a family of queries which are drawn from the query distribution $p(X_G, L)$. We can minimize $\mathcal{L}$ in (12) to get V^{NN} and u^{NN} for each query. In order to plan new queries faster, we hope the DNN training in the new queries can be accelerated by starting from a good initial DNN parameter and consuming less training data. To do so, we want to find a meta parameter θ_{meta} for V^{NN}, such that one step update from θ_{meta} with small K-shot data minimizes the expected loss over the query family. In particular, the meta value training is formulated as the optimization problem:

$$\begin{aligned}\min_{\theta} \mathbb{E}_{(X_G^{(i)},L^{(i)})\sim p}&\left[\mathbb{E}_{\mathcal{D}_i^K}\left[\mathcal{L}^{(i)}\left(\theta^{(i)})\right)\right]\right] \\ =\mathbb{E}_{(X_G^{(i)},L^{(i)})\sim p}&\left[\mathbb{E}_{\mathcal{D}_i^K}\left[\mathcal{L}^{(i)}\left(\theta-\alpha\nabla_\theta\mathcal{L}\left(\theta,\mathcal{D}_i^K\right)\right)\right]\right],\end{aligned} \tag{13}$$

where $\mathcal{L}$ is given in (12) and the query-specific loss function $\mathcal{L}^{(i)}(\theta) = \mathcal{L}\left(\theta, \mathcal{D}_i^{tr}\right)$. The data set $\mathcal{D}_i^{tr}$ is given in (8) for query configuration $(X_G^{(i)}, L^{(i)})$ and the set $\mathcal{D}_i^K$ is a K-combination sampled from $\mathcal{D}_i^{tr}$. The parameter $\theta^{(i)}$ is produced by one step gradient descent $\theta^{(i)} = \theta - \alpha\nabla_\theta\mathcal{L}\left(\theta, \mathcal{D}_i^K\right)$ with the randomly sampled K-shot data $\mathcal{D}_i^K$, then is evaluated by the whole query-specific data set $\mathcal{D}_i^{tr}$. So for each query and arbitrarily sampled K-shot data, the solution of one-step gradient descent from θ_{meta} has a low expected error, where θ_{meta} is the global minimizer of (13).

The meta training process is shown in Algorithm 1. First, in line 1, we sample a set of motion planning queries $\mathcal{T}$ from the given query distribution $p(X_G, L)$. Then, for each sampled query $(X_G^{(i)}, L^{(i)})$, the training data set $\mathcal{D}_i^{tr} \triangleq \left\{X^d; \left(\hat{V}_i(X^d), \hat{u}_i(X^d)\right)\right\}$ is generated by the set-valued approximation in Section III-A1. Second, we use the following empirical minimization to approximate the expected minimization (13):

$$\min_{\theta} \sum_{(X_G^{(i)},L^{(i)})\in\mathcal{T}} \sum_{\mathcal{D}_i^K} \left[\mathcal{L}^{(i)}\left(\theta-\alpha\nabla_\theta\mathcal{L}\left(\theta,\mathcal{D}_i^K\right)\right)\right]. \tag{14}$$

To solve (14), in each iteration, we sample a batch of queries from the query family, then sample a batch of data from the whole training data set $\mathcal{D}_i^{tr}$ for each query to minimize the loss function, as the stochastic gradient descent (SGD) method does. In each optimization iteration, a batch of queries $\mathcal{T}_k$ are sampled from $\mathcal{T}$, then inner optimization (lines 9-11) and outer optimization (line 13) are employed. In inner optimization, for each query $(X_G^{(i)}, L^{(i)})$, we sample a K-shot data set $\mathcal{D}_i^{(1)} = \left\{X_1; \left(\hat{V}_i(X_1), \hat{u}_i(X_1)\right)\right\}$ from the $\mathcal{D}_i^{tr}$, where $X_1 \subset X^d$ and $|X_1| = K$, and compute the one step K-shot gradient descent

Algorithm 1 Meta value learning

Input: $p(X_G, L)$: query configurations distribution
Input: α, β : hyperparameters, X^d: state gird
1: Sample a set of queries $\mathcal{T}$ from $p(X_G, L)$
2: **for all** $(X_G^{(i)}, L^{(i)}) \in \mathcal{T}$ **do**
3: Compute $\hat{V}_i(X^d)$ and $\hat{u}_i(X^d)$ for query $(X_G^{(i)}, L^{(i)})$
4: **end for**
5: Randomly initialize θ_0
6: **while** $k \leq P$ **do**
7: Sample a subset of queries $\mathcal{T}_k$ from $\mathcal{T}$
8: **for all** $(X_G^{(i)}, L^{(i)}) \in \mathcal{T}_k$ **do**
9: Sample K-shot data set $\mathcal{D}_i^{(1)}$ for query $(X_G^{(i)}, L^{(i)})$
10: Update the parameter for query $(X_G^{(i)}, L^{(i)})$ as (15)
11: Sample the data set $\mathcal{D}_i^{(2)}$
12: **end for**
13: Update meta parameter θ_k by (16)
14: $k \leftarrow k + 1$
15: **end while**
16: **return** $\theta_{meta} = \theta_k$

parameter $\theta_k^{(i)}$ from the meta parameter θ_k:

$$\theta_k^{(i)} = \theta_k - \alpha\nabla_\theta\mathcal{L}\left(\theta, \mathcal{D}_i^{(1)}\right)|_{\theta=\theta_k}. \tag{15}$$

In outer optimization, we evaluate and optimize the performance of the parameter after one-step gradient descent in inner optimization. First, for each query, we need another data set $\mathcal{D}_i^{(2)}$ to evaluate the performance, where $\mathcal{D}_i^{(2)} = \left\{X_2; \left(\hat{V}_i(X_2), \hat{u}_i(X_2)\right)\right\}$ is also sampled from $\mathcal{D}_i^{tr}$ and $X_2 \subset X^d$. Then the meta parameter θ_k is updated along the gradient of the loss function in (14) for one step:

$$\begin{aligned}\theta_{k+1} =&\theta_k - \frac{\beta}{|\mathcal{T}_k|}\nabla_\theta\Big(\sum_{(X_G^{(i)},L^{(i)})\in\mathcal{T}_k}\mathcal{L}(\theta- \\ &\alpha\nabla_\theta\mathcal{L}(\theta,\mathcal{D}_i^{(1)}),\mathcal{D}_i^{(2)}))|_{\theta=\theta_k} \\ =&\theta_k - \frac{\beta}{|\mathcal{T}_k|}\sum_{(X_G^{(i)},L^{(i)})\in\mathcal{T}_k}(I-\alpha\nabla^2\mathcal{L}(\theta_k,\mathcal{D}_i^{(1)})) \\ &\nabla\mathcal{L}(\theta_k^{(i)},\mathcal{D}_i^{(2)}),\end{aligned} \tag{16}$$

where β is the learning rate of meta training. The output of Algorithm 1 is the meta value function $V^{NN}(\cdot, \theta_{meta})$.

C. Anytime adaptation

When a new query is requested, the meta value function needs to be updated to the query-specific value function in an anytime manner, i.e., a feasible solution is quickly obtained and its optimality improves over time. Algorithm 1 returns the meta value function $V^{NN}(\cdot, \theta_{meta})$, also indicates the one-step parameter adaptation to the new query in (15).

1) K-shot data generation: To execute (15) for a new query, we need to obtain the K-shot data of the optimal value function for the query. Anytime adaptation requires that K is small and data points are generated sequentially.

However, because the set-valued method in Section III-A1 simultaneously computes the approximate optimal value of all states in a state grid, it cannot solve the problem within the time limit, and then is not suitable for generating K-shot data.

The causality-free method [12, 25] is a computational method for optimal control by solving high-dimensional HJB equations. Unlike the set-valued method, the causality-free method does not approximate the optimal value function V^* and the optimal controller u^* on a state grid. Instead, a two-point BVP derived from the Pontryagin's Minimum Principle (PMP) is solved at each point on the state grid. It is worthy to highlight that these BVPs can be solved independently, making the algorithm causality-free. As a result, the causality-free method is more suitable than the set-valued method to generate the K-shot data. In Section IV-D, we compare the set-valued method and the causality-free method by simulations.

In order to use the causality-free method, we modify (2) into the following fixed final time unconstrained optimal control problem:

$$\tilde{V}^*(t,x) = \min_{u(\cdot)\in\Gamma} \left\{ S(x(t_f)) + \int_0^{t_f} L(x(t),u(t))dt \right\}, \\ \text{s.t. } \dot{x}(t) = f(x(t),u(t)),\ x(0) = x, \tag{17}$$

where the set of time-varying state feedback control policies is defined as $\Gamma \triangleq \{u(\cdot) : \mathbb{R}_{\geq 0} \times X \to U\}$, and t_f is sufficiently large. Unlike (2), the optimal value function $\tilde{V}^*(t,x)$ is time-varying, and so is the optimal controller $\tilde{u}^*(t,x)$. Then the Hamiltonian function is defined as $\mathcal{H}(t,x,\lambda,u) \triangleq L(x,u) + \lambda^T f(x,u)$, where $\lambda : [0,t_f] \to \mathbb{R}^n$ is the costate. The optimal control $\tilde{u}^*(t,x)$ satisfies the following Hamiltonian minimization condition:

$$\tilde{u}^*(t,x) = \arg\min_{u\in\Gamma} \mathcal{H}(t,x,\nabla\tilde{V}_x^*(t,x),u).$$

The causality-free method exploits a two-point BVP, which is derived from the PMP:

$$\begin{cases} \dot{x}(t) = f\left(t,x,u(t,x,\lambda)\right), \quad x(0) = x_0 \\ \dot{\lambda}(t) = -\nabla\mathcal{H}_x\left(t,x,\lambda,u(t,x,\lambda)\right), \lambda\left(t_f\right) = \frac{dS}{dx}\left(x\left(t_f\right)\right) \\ \dot{v}(t) = -L\left(x,u(t,x,\lambda)\right), \quad v\left(t_f\right) = S\left(x\left(t_f\right)\right) \\ \nabla\mathcal{H}_u\left(t,x,\lambda,u(t,x,\lambda)\right) = 0. \end{cases} \tag{18}$$

Given any initial state x_0, the BVP (18) returns the solution, say $v(t)$, $x(t)$ and $u(t)$, which satisfies two equations

$$\tilde{V}^*(t,x(t)) = v\left(t\right), \quad \tilde{u}^*(t,x(t)) = u(t). \tag{19}$$

In (19), the optimal value function $\tilde{V}^*(t,x(t))$ and the opti-mal controller $\tilde{u}^*(t,x(t))$ in (17) along $x(t)$ can be obtained. Note that $x(t)$ is the optimal state trajectory, $u(t)$ and $v(t)$ are the control and the value along $x(t)$, respectively. The BVPs with different initial states are solved independently and return the corresponding trajectories under the optimal controller. Hence, the method is called causality-free.

We choose the terminal cost $S(x)$ in (17) such that it is high when the robot is far from the goal region and decreases exponentially to 0 when the robot approaches to the goal region. So the robot is penalized if it moves away from the goal region. In this way, the optimal value function $V^*(x)$ in (2) and the optimal controller $u^*(x)$ in (6) are approximated by $\tilde{V}^*(t,x(t))$ and $\tilde{u}^*(t,x(t))$, respectively.

To solve the BVP (18), we implement the BVP solver introduced in [17], which is an iterative method based on a three-stage discretization. However, the algorithm convergence is local, i.e., it needs to start from a neighborhood of the solution of (18). Fortunately, in the beginning (line 3) of Algorithm 1, $\hat{V}(X^d)$ and $\hat{u}(X^d)$ are computed for a number of training queries by the set-valued method in Section III-A1. When a new query is requested, we can select one of the training queries that is close to the new query. Then the state trajectory, the value function and the controller computed by the set-valued method are used as the initial guess for (18).

2) Anytime adaptation: Anytime adaptation shown in Algorithm 2 adapts the meta value function $V^{NN}(\cdot,\theta_{meta})$ to the optimal value function of a new query (X_G^*, L^*). The algorithm iteratively executes the two phases: data generation and parameter adaptation. Data generation in each iteration constantly extends the data pool for following parameter adaptation. In the n-th iteration, in line 5-6, we sample a state x_n from X uniformly and solve BVP (18) of query (X_G^*, L^*)

Algorithm 2 Anytime adaptation

Input: (X_G^*, L^*): the new query configuration
Input: meta training parameter θ_{meta} and meta step size α
1: $n = 0$, $\theta_0 = \theta_{meta}$
2: $S_J = \emptyset$
3: **while** $n < N$ **do**
4: $\mathcal{S}_D = \emptyset$
Data generation
5: Sample a state x_n from X
6: Compute the trajectory J_n by solving BVP (18) of query (X_G^*, L^*) with initial state x_n
7: $S_J = S_J \cup J_n$
8: **for all** $J \in S_J$ **do**
9: Sample data with time interval Δt from J and obtain the data set $\mathcal{D}$
10: $\mathcal{S}_D = \mathcal{S}_D \cup \mathcal{D}$
11: **end for**
12: Data set $\mathcal{D}_g$ with states sampled from the goal region
13: $\mathcal{S}_D = \mathcal{S}_D \cup \mathcal{D}_g$
14: Sample the K_n-shot data set $\mathcal{D}_K$ from $\mathcal{S}_D$
Parameter adaptation
15: $j = 0$
16: $\theta_0^* = \theta_n$
17: **for** $j < M$ **do**
18: $\theta_{j+1}^* = \theta_j^* - \alpha\nabla_\theta\mathcal{L}\left(\theta_j^*, \mathcal{D}_K\right)$
19: $j \leftarrow j + 1$
20: **end for**
21: $\theta_n = \theta_M^*$
22: $u^{NN}(x) = h(\nabla_x V^{NN}(x;\theta_n))$
23: $n \leftarrow n + 1$
24: **end while**
25: **return** u^{NN}

with initial state x_n. Then the solver returns the trajectory denoted as $J_n = \left\{\tilde{V}^*(t, x_n(t)), \tilde{u}^*(t, x_n(t))\right\}$ by (19) and adds it into the trajectory set S_J. In line 8-14, we obtain a K-shot data set from S_J. First, we sample the data of $\tilde{V}(t, x(t))$ and $\tilde{u}(t, x(t))$ along each trajectory in S_J with sampling unit Δt. Denote the data set as $\mathcal{S}_D$. Further, we sample several states in the goal region X_G^*, construct the data set $\mathcal{D}_g$ with $\tilde{V}(t, x(t)) = 0$ and $\tilde{u}(t, x(t)) = 0$ for each state, and include $\mathcal{D}_g$ in data set $\mathcal{S}_D$ as an extension of the training supervision. Notice that in each iteration, the number of states sampled in X_G^* is required to keep the ratio of $|\mathcal{D}_g|$ to $|\mathcal{S}_D|$ constant. It is clear that this data extension does not consume any time. Then, the data set denoted as $\mathcal{D}_K$ which includes K_n data points, is randomly sampled from $\mathcal{S}_D$ and applied to anytime adaptation, where K_n is proportional to n. In line 15-19, query-specific parameter adaptation is done through gradient descent. For each iteration n, M times gradient descent steps are performed and start from the parameter θ_{n-1} in the last iteration. In line 22, the control policy is updated in each iteration and in the anytime manner.

Algorithm 2 can quickly adapt the meta parameter to the query-specific parameter and hence rapidly plan the motion. Firstly, the parameter adaptation requires doing a few steps of gradient descent from the meta parameter, whose computational complexity is much lower than that of usual DNN training starting from a random initial parameter. Secondly, the adaptation requires a small number of data points of a new query, which are efficiently generated by the causality-free method. Moreover, in the causality-free method, optimal values of states can be solved independently. As a result, the data generation can be accelerated by parallel computing.

Once reaction time is reached, Algorithm 2 terminates and the returned controller is executed. If more reaction time is given, more adaptation data are generated to extend the data pool for parameter update. By sampling from the expanded data pool, the dispersion of the K-shot data set keeps increasing. Thus, the optimal values in these states provide an exhaustive supervision for the whole space in parameter adaptation. In this way, Algorithm 2 keeps improving on the optimality of the control policy.

IV. Simulation

In this section, we evaluate the proposed meta-learning method. All simulations are executed on a 4.10 GHz Intel Core i5 CPU with 4 total cores. The set-valued method in Section III-A1 is executed in MATLAB. Algorithm 1 and 2 are written by Python and its PyTorch library. The ReFMM algorithm in [32] and the MPC method are also written by Python.

A. Problem parameters

The robot is modeled as a unicycle and its kinematic model is given by $\dot{p}_x(t) = \cos(\beta(t))v(t)$, $\dot{p}_y(t) = \sin(\beta(t))v(t)$ and $\dot{\beta}(t) = \omega(t)$, where $x(t) = [p_x(t), p_y(t), \beta(t)]^T$ is the robot's state with $[p_x(t), p_y(t)]^T \in [-5, 5] \times [-5, 5]$ being its position and $\beta(t) \in (-\pi, \pi]$ being its orientation, and $u(t) = [v(t), \omega(t)]^T$ is the robot's control with $v(t) \in \mathbb{R}$ and $\omega(t) \in \mathbb{R}$ being its linear and angular velocities, respectively.

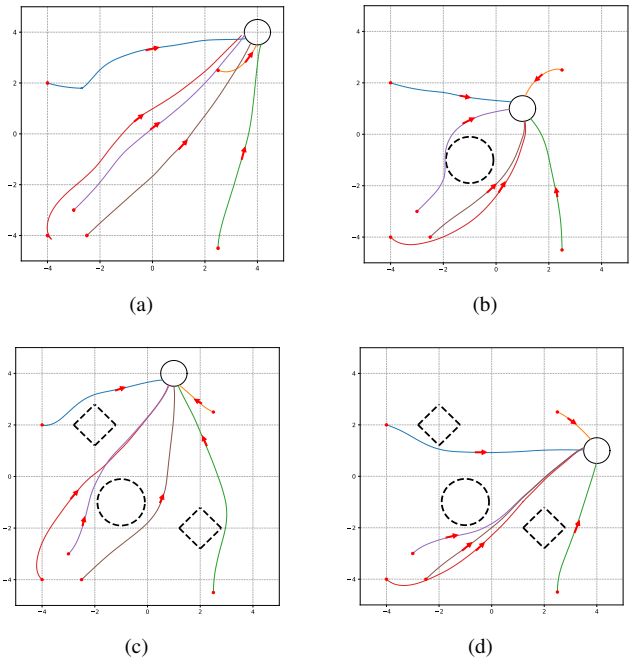


Fig. 1. Robot trajectories. Four queries are moving to $[4, 4]$ without obstacles in Fig 1(a), moving to $[1, 1]$ with an obstacle in Fig 1(b), moving to $[1, 4]$ and $[4, 1]$ with three obstacles in Fig 1(c) and 1(d). The robot starts from initial states $[-4, 2, -0.1\pi]$, $[2.5, 2.5, -0.125\pi]$, $[2.5, -4.5, -0.5\pi]$, $[-4, -4, -0.25\pi]$, $[-3, -3, 0.25\pi]$, $[-2.5, -4, 0.25\pi]$, respectively.

Consider the scenario where the goal region is a ball centered at x_g with radius r_g. The obstacles are balls or rectangles and each is centered at x_k^o with radius r_k^o. The running cost functional is defined as $L(x, u) = Q(x) + u^T u + \phi(x)$, where $Q(x) = \max\{\frac{2}{1+e^{-3(||x-x_g||_2^2 - r_g^2)}} - 1, 0\}$ penalizes the distance to the goal region, and $\phi(x) = \sum_k e^{-4(||x-x_k^o||_{m_k}^2 - r_k^{o2})}$ increases exponentially as the robot approaches the obstacles. In $\phi(x)$, $m_k = 2$ when the obstacle is a circle and $m_k = 1$ when it is a rectangle.

Consider the query family where the goal radius is $r_g = 0.5$, the goal center x_g follows a uniform distribution over $[0, 5] \times [0, 5]$. There is no obstacle when $x_g \in (2.5, 5] \times (2.5, 5]$, a circle obstacle is centered at $x_1^o = [-1, -1]$ when $x_g \in [0, 2.5) \times [0, 2.5)$, a circle obstacle and two rectangle obstacles are centered at $x_1^o = [-1, -1]$, $x_2^o = [-2, 2]$, $x_3^o = [2, -2]$, respectively, when $x_g \in [2.5, 5] \times [0, 2.5] \cup [0, 2.5] \times [2.5, 5]$. Each query configuration (X_G, L) is defined in the last paragraph and follows a distribution $p(X_G, L)$.

B. Meta value learning and adaptation

Algorithm 1 adopts a DNN which consists of an input layer of size 4, followed by 3 hidden layers of size 256 with ReLU nonlinearities and an output layer of size 1. Notice that the optimal value function V^* is periodic in β. Thus, the input β is replaced by two inputs $\sin(\beta)$ and $\cos(\beta)$. Then, the inputs of the DNN are p_x, p_y and $\sin(\beta)$, $\cos(\beta)$. In Algorithm 1,

TABLE I
TOTAL COSTS

Goal	$[1,1]$	$[1,4]$	$[4,4]$	$[4,1]$
Set-valued method (fine grid)	17.3	20.1	24.0	20.1
Set-valued method (coarse grid)	17.6	20.3	24.4	20.3
ReFMM (fine grid)	17.4	20.1	24.1	20.1
ReFMM (coarse grid)	17.9	21.1	24.9	20.9
MPC	21.9	23.8	28.2	24.3
TRPO	19.6	22.1	27.2	22.0
DNN interpolation	17.8	20.2	24.2	20.3
Anytime adaptation	17.9	20.6	24.9	20.5

TABLE II
AVERAGE TIME CONSUMPTION

Approach		Elapsed time
Set-valued method (fine grid)		36 min
Set-valued method (coarse grid)		4.5 min
ReFMM (fine grid)		17 min
ReFMM (coarse grid)		1.9 min
MPC		0.2 s (each instant)
TRPO		3.7 min
Meta value training	Data generation	4.5 h
	Meta training	2 h
Anytime adaptation ($n=8$)	Data generation	6.823 s
	Parameter adaptation	0.043 s
Anytime adaptation (Parallel computing)	Data generation	2.753 s
	Parameter adaptation	0.043 s

$\mathcal{L}_u$ in the Lagrangian $\mathcal{L}$ of (12) includes the gradient of V^{NN} with respect to x. Both of finite difference approximation and automatic differentiation [24] by some existing libraries, e.g. PyTorch, are feasible. For our case, the finite difference approximation is more efficient,

In Algorithm 1, we randomly sample 60 queries. For each query, we generate $118,000$ data points by the set-valued approximation in Section III-A1, where $h = 0.175$ and $\varepsilon = 0.4$. Set the adaptation data number as $K = 256$, the meta learning rate $\alpha = 0.001$ and the Lagrange multiplier $\mu = 0.1$. In each iteration, we sample 10 queries as a mini-batch and use the Adam method [18] to optimize the parameters. In Algorithm 2, we set $K = 256$, $N = 64$, $M = 5$ and $K_n = 32n$. The terminal cost is chosen as $S(x) = e^{2(||x-x_g||_2^2 - r_g^2)}$.

Recall that the data generation in Algorithm 2 can be accelerated by parallel computing. In the simulations, we implement Algorithm 2 by multiprocessing, where 1, 2 and 4 processes are invoked by the program, respectively.

C. Competitors

We compare our method with five competitors. First, we use the set-valued method in Section III-A1 on increasingly refined state grids as [5]. Table I and II show the results on a fine state grid (row 1) and a coarse state grid (row 2), which contain $955,000$ states and $118,000$ states, respectively. Note that the set-valued method is asymptotically optimal, then the total costs of the controllers approach the minimal cost when the state grids are sufficiently dense. Here, we use the set-valued method on the fine state grid (row 1) as our benchmark.

Secondly, we employ ReFMM in [32] on increasingly refined state simplices, and show the results on a fine state simplicial mesh (row 3) and a coarse state simplicial mesh (row 4), which contain $96,000$ simplices and $12,000$ simplices, respectively. The ReFMM method is also asymptotically optimal.

The third competitor is MPC. We use the discrete-time kinodynamic model of the unicycle robot with time interval 0.1 second and solve a 20-horizon optimal control problem at each time instant. A quadratic function of the state is employed as the terminal cost.

Fourthly, we employ a widely used RL algorithm TRPO [28] to train a control policy model. The same dynamics is set up as the MPC method in the training.

Finally, we compute the DNN interpolation controller shown in Section III-A2. Note that the training data sets for both DNN interpolation in Section III-A2 and Algorithm 1 are obtained by the set-valued method on the coarse state grid with $118,000$ states.

D. Simulation results

We test the meta-learning approach and the competitors on four queries which are drawn from the query distribution in Section IV-A: (i) $x_g = [4,4]$, no obstacle. (ii) $x_g = [1,1]$, $x_1^o = [-1,-1]$, $r_1^o = 1$, $m_1 = 2$. (iii) $x_g = [1,4]$, $x_1^o = [-1,-1]$, $r_1^o = 1$, $m_1 = 2$, $x_2^o = [-2,2]$, $x_3^o = [2,-2]$, $r_2^o = r_3^o = 0.8$, $m_2 = m_3 = 1$. (iv) $x_g = [4,1]$, the obstacles are set as (iii). Fig. 1 shows examples of the robot trajectories of the four queries, where the arrows represent the moving directions of the robot. Collisions are avoided when the obstacles are present in the environments.

The total costs of the controllers where the robot starts from initial state $[-4,-4,-0.25\pi]$ are shown in Table I and the average computation time of each component of these algorithms is shown in Table II. Here we show the results of that, 8 trajectories are generated, i.e., the computation time of Algorithm 2 when it stops at iteration $n = 8$. The ReFMM method (row 3 and 4) and the set-valued method (row 1 and 2), return nearly optimal solutions. The average costs of our method when $n = 8$ (Table I row 8), ReFMM on the fine grid (row 3) and ReFMM on the coarse grid (row 4) exceed the benchmark (row 1) by 4%, 4%, and 0.5%, respectively, showing the near optimality of the controller returned by the meta-learning approach. MPC (row 5) is inherently suboptimal

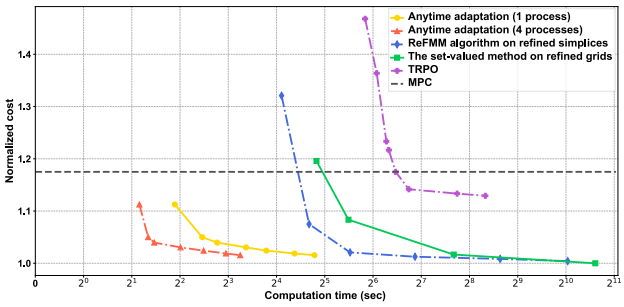
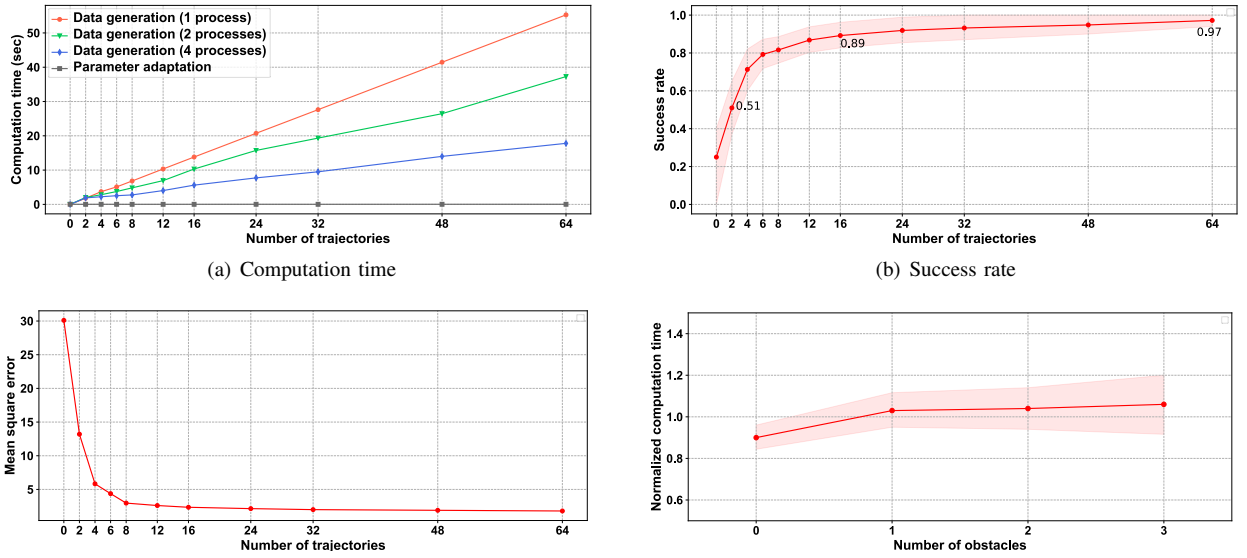


Fig. 2. Relation between solution optimality and computation time



(a) Computation time

(b) Success rate

(c) Approximation error

(d) Normalized computation time

Fig. 3. Anytime property of Algorithm 2

and its cost exceeds the benchmark by 20%. The cost by TPRO (row 6) exceeds the benchmark by 11%. On the other hand, the reaction time of Algorithm 2 is less than 7 seconds when $n = 8$. When the trajectories are computed in parallel (4 processes invoked), the computation time reduces to 2.8 seconds. The fast online planning comes at a cost of long offline meta value training time in Algorithm 1, which is about 6.5 hours. Table II also shows that the data generation dominates the computation time of Algorithm 2. In addition, MPC takes 0.2 seconds to solve the 20-horiozon optimal control problem at each time instant and the total online computation time is proportional to the traveling time. On the other hand, other methods only perform offline computation once to synthesize controllers. So we do not compare the computation times of MPC and other methods in Table II and Fig. 2. As mentioned, its optimality is the lowest. The computation times of other competitors are minutes.

Fig. 2 shows the normalized costs of the algorithms, i.e., the total costs relative to that of the benchmark (the set-valued method on the fine state grid shown in row 1 of Table I), as allowable computation times increase. In the figure, the robot starts from initial states $[-4, -4, -0.25\pi]$, $[-2.5, -4, 0.25\pi]$, $[-4, 2, -0.1\pi]$ and aims to reach goal point $[4, 4]$. Notice that each curve starts from the earliest time that the method can compute a feasible controller, which can successfully steer the robot from all of the initial states to the goal. It is shown that the meta-learning method finds a feasible controller in 4 seconds (a single process) and in 2 seconds (4 processes) which are much shorter than those of the ReFMM algorithm and the set-valued method (about 15 seconds and 30 seconds, respectively). In addition, the meta-learning method provides nearly optimal controllers (the total cost exceeding the benchmark within 5%) in 6 seconds (a single process) and in 3 seconds (4 processes). The ReFMM algorithm and the set-valued method spend about 30 seconds and 1.5 minutes, respectively, to reach the accuracy level 5%. It takes the longest time for TRPO to search for a feasible policy (1 minute) and the cost does not reduce after 3 minutes.

We run Algorithm 2 on 20 queries which are sampled from the query family. Fig. 3 verifies the anytime property of Algorithm 2. Notice that the algorithm keeps generating data trajectories as the iteration number increases, and the number of trajectories is equal to the iteration number. Fig. 3(a) shows the average computation time over the queries linearly increases in the iteration number, and the parallel computing accelerates the data generation. In Algorithm 2, the data generation is much more time-consuming than the parameter adaptation, then the algorithm is accelerated as more processors are used. We evaluate the success rate of the controllers by the percentage that the robot reaches the goal region in finite time, where the initial states are uniformly sampled from the free space. The success rate shown in Fig. 3(b) logarithmically increases as the iteration number. Fig. 3(c) shows that the approximation error of $V^{NN}(\cdot, \theta_n)$ by Algorithm 2 exponentially decreases over the iteration number. Overall, Fig. 3(a)-3(c) shows the anytime property of Algorithm 2. More specifically, it takes 2 iterations (about 1 second) to identify a controller whose success rate is about 50% and the success rate exceeds 90% if 16 iterations (about 7 seconds) are conducted. Fig. 3(d) shows the normalized computation time

TABLE III
COMPARISON OF TWO METHODS FOR DATA GENERATION

Approach	Average time per state	Number of states	Total time
Set-valued method	0.0023s	$118,000$	278s
Causality-free method	0.0168s	256	4.3s

of Algorithm 2, i.e., computation time relative to the average, slightly increases as the number of obstacles. It indicates that Algorithm 2 is scalable with respect to the number of obstacles.

Finally, Table III compares the computation time of the set-valued method in Section III-A1 and the causality-free method in Section III-C1. The average computation time per state of the set-valued method is 0.0023s on the $118,000$ states which are uniformly distributed over the state space. The average time of the causality-free method is 0.0168s on 256 states which concentrate on 8 motion trajectories (n=8). The set-valued method is suitable for data generation of meta training due to its low average computation time. But if it is applied to the 256 states, the approximation error is too large. On the other hand, the causality-free method quickly computes the exact optimal values of a small number of states, and thus is more suitable for anytime adaptation.

V. Conclusions

We propose a meta-learning-based algorithm for fast policy-centric motion planning. The proposed method explores an approach of policy-centric planning by the meta-learning algorithm, which trades long offline training time for fast online planning. A meta value function model is learned for a motion planning query family, which can quickly adapt to a new query from the family. Our method is evaluated against state-of-the-art policy-centric motion planning methods in simulations. In the simulations, a robot is required to quickly compute controllers, which can drive the robot to different goals and under various environments (no obstacle, one obstacle, and 3 obstacles are included, respectively). The simulation results verify the effectiveness of the algorithm and assess its anytime property.

VI. Acknowledgments

This work was partially supported by the grants NSF CNS 1830390 and ECCS 1846706.

References

[1] Murad Abu-Khalaf and Frank L. Lewis. Nearly optimal control laws for nonlinear systems with saturating actuators using a neural network HJB approach. *Automatica*, 41(5):779–791, 2005. URL https://doi.org/10.1016/j.automatica.2004.11.034.

[2] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957. URL https://press.princeton.edu/books/paperback/9780691146683/dynamic-programming.

[3] Alex Brooks, Tobias Kaupp, and Alexei Makarenko. Randomised MPC-based motion-planning for mobile robot obstacle avoidance. In *IEEE International Conference on Robotics and Automation*, pages 3962–3967, 2009. URL https://doi.org/10.1109/ROBOT.2009.5152240.

[4] Francesco Bullo, Emilio Frazzoli, Marco Pavone, Ketan Savla, and Stephen L. Smith. Dynamic Vehicle Routing for Robotic Systems. *Proceedings of the IEEE*, 99 (9):1482–1504, 2011. URL https://ieeexplore.ieee.org/document/5954127.

[5] Pierre Cardaliaguet, Marc Quincampoix, and Patrick Saint-Pierre. Set-valued numerical analysis for optimal control and differential games. In *Stochastic and differential games*, pages 177–247. Springer, 1999. URL https://link.springer.com/chapter/10.1007%2F978-1-4612-1592-9_4.

[6] Alireza Fallah, Kristian Georgiev, Aryan Mokhtari, and Asuman Ozdaglar. On the Convergence Theory of Debiased Model-Agnostic Meta-Reinforcement Learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 3096–3107. Curran Associates, Inc., 2021. URL https://papers.nips.cc/paper/2021/hash/18085327b86002fc604c323b9a07f997-Abstract.html.

[7] Morteza Farrokhsiar, Graham Pavlik, and Homayoun Najjaran. An integrated robust probing motion planning and control scheme: A tube-based MPC approach. *Robotics and Autonomous Systems*, 61(12):1379–1391, 2013. URL https://www.sciencedirect.com/science/article/pii/S0921889013001401?via%3Dihub.

[8] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135, 2017. URL http://proceedings.mlr.press/v70/finn17a.html.

[9] John E Hopcroft, Jacob Theodore Schwartz, and Micha Sharir. On the Complexity of Motion Planning for Multiple Independent Objects; PSPACE-Hardness of the Warehouseman's Problem. *The International Journal of Robotics Research*, 3(4):76–88, 1984. URL https://journals.sagepub.com/doi/10.1177/027836498400300405.

[10] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural networks*, 3(5):551–560, 1990. URL https://doi.org/10.1016/0893-6080(90)90005-6.

[11] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. URL https://www.computer.org/csdl/journal/tp/5555/01/09428530/1twaJR3AcJW.

[12] Wei Kang and Lucas Wilcox. A causality free computational method for HJB equations with application to rigid body satellites. In *Guidance, Navigation, and Control Conference*, page 2009, 2015. URL https://arc.aiaa.org/doi/abs/10.2514/6.2015-2009.

[13] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011. URL https://journals.sagepub.com/doi/10.1177/0278364911406761.

[14] Sertac Karaman and Emilio Frazzoli. Sampling-based optimal motion planning for non-holonomic dynamical systems. In *IEEE International Conference on Robotics and Automation*, pages 5041–5047, 2013. URL https://ieeexplore.ieee.org/abstract/document/6631297.

[15] Lydia Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996. URL https://doi.org/10.1109/70.508439.

[16] Matthew Kelly. An introduction to trajectory optimization: How to do your own direct collocation. *SIAM Review*, 59(4):849–904, 2017. URL https://epubs.siam.org/doi/pdf/10.1137/16M1062569.

[17] Jacek Kierzenka and Lawrence F Shampine. A BVP solver based on residual control and the Maltab PSE. *ACM Transactions on Mathematical Software*, 27(3):299–316, 2001. URL https://dl.acm.org/doi/10.1145/502800.502801.

[18] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations*, 2015. URL http://arxiv.org/abs/1412.6980.

[19] Prashanth Konkimalla and Steven M LaValle. Efficient computation of optimal navigation functions for nonholonomic planning. In *Proceedings of the First Workshop on Robot Motion and Control*, pages 187–192, 1999. URL https://ieeexplore.ieee.org/abstract/document/791074.

[20] Steven M LaValle. Numerical computation of optimal navigation functions on a simplicial complex. *Robotics: The Algorithmic Perspective*, pages 339–350, 1998. URL http://lavalle.pl/papers/Lav98b.pdf.

[21] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006. URL https://www.cambridge.org/us/catalogue/catalogue.asp?isbn=0521862051.

[22] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001. URL https://journals.sagepub.com/doi/10.1177/02783640122067453.

[23] Yanbo Li, Zakary Littlefield, and Kostas E Bekris. Asymptotically optimal sampling-based kinodynamic planning. *The International Journal of Robotics Research*, 35(5):528–564, 2016. URL https://journals.sagepub.com/doi/full/10.1177/0278364915614386.

[24] Bart van Merriënboer, Olivier Breuleux, Arnaud Bergeron, and Pascal Lamblin. Automatic differentiation in ML: where we are and where we should be going. In *the 32nd International Conference on Neural Information Processing Systems*, pages 8771–8781, 2018. URL https://proceedings.neurips.cc/paper/2018/file/770f8e448d07586afbf77bb59f698587-Paper.pdf.

[25] Tenavi Nakamura-Zimmerer, Qi Gong, and Wei Kang. Adaptive Deep Learning for High-Dimensional Hamilton–Jacobi–Bellman Equations. *SIAM Journal on Scientific Computing*, 43(2):A1221–A1247, 2021. URL https://epubs.siam.org/doi/abs/10.1137/19M1288802?journalCode=sjoce3.

[26] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

[27] Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992. URL https://ieeexplore.ieee.org/document/6796337.

[28] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015. URL https://proceedings.mlr.press/v37/schulman15.html.

[29] James A Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, 93(4):1591–1595, 1996. URL https://www.jstor.org/stable/38628.

[30] Yuval Tassa and Tom Erez. Least Squares Solutions of the HJB Equation With Neural Network Value-Function Approximators. *IEEE Transactions on Neural Networks*, 18(4):1031–1041, 2007. URL https://ieeexplore.ieee.org/document/4267720.

[31] Dmitry S Yershov and Emilio Frazzoli. Asymptotically optimal feedback planning: FMM meets adaptive mesh refinement. In *Algorithmic Foundations of Robotics XI*, pages 695–710. Springer, 2015. URL https://link.springer.com/chapter/10.1007/978-3-319-16595-0_40.

[32] Dmitry S Yershov and Emilio Frazzoli. Asymptotically optimal feedback planning using a numerical Hamilton-Jacobi-Bellman solver and an adaptive mesh refinement. *The International Journal of Robotics Research*, 35(5):565–584, 2016. URL https://journals.sagepub.com/doi/full/10.1177/0278364915602958.