2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Neural Optimal Control using Learned System Dynamics

Selim Engin[1] and Volkan Isler[1]

***Abstract*— We study the problem of generating control laws for systems with unknown dynamics. Our approach is to represent the controller and the value function with neural networks, and to train them using loss functions adapted from the Hamilton-Jacobi-Bellman (HJB) equations. In the absence of a known dynamics model, our method first learns the state transitions from data collected by interacting with the system in an offline process. The learned transition function is then integrated to the HJB equations and used to forward simulate the control signals produced by our controller in a feedback loop.**

In contrast to trajectory optimization methods that optimize the controller for a single initial state, our controller can generate near-optimal control signals for initial states from a large portion of the state space. Compared to recent model-based reinforcement learning algorithms, we show that our method is more sample efficient and trains faster by an order of magnitude. We demonstrate our method in a number of tasks, including the control of a quadrotor with 12 state variables.

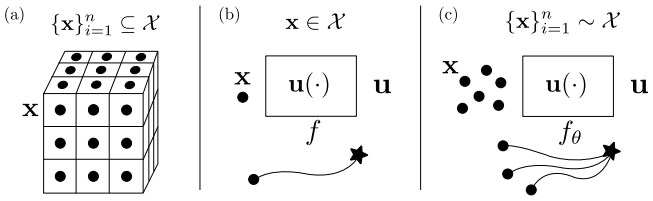

Fig. 1: An overview of different approaches for solving nonlinear optimal control problems. **(a)** Approaches that solve the HJB equation with numerical methods offer global solutions, however are restricted to lower dimensions due to discretization of the state space $\mathcal{X}$. **(b)** Trajectory optimization methods optimize the controller $\mathbf{u}(\cdot)$ for a single initial state to reach the target using known state transitions. **(c)** Our approach generates a controller trained using learned transitions f_θ, for initial states from a large portion of $\mathcal{X}$.

I. Introduction

Many robotics tasks, such as navigation and locomotion, can be formulated as optimal control problems, which are challenging to solve especially when the system has complex dynamics and the state space is high dimensional. Optimal control formulations are interesting since they allow us to express tasks as simple cost functions and leave the rest to the control algorithm.

In this paper, we study nonlinear optimal control problems in finite time horizon. One way to solve these problems is through trajectory optimization methods that use nonlinear programming, such as nonlinear model predictive control [1], which optimize the control signals to generate optimal trajectories given an initial state. However, these methods are local, meaning that they perform optimization for a single initial state, and their recomputation for other initial states taking seconds might be too restrictive for real-time applications. Updating the initial state can be needed in a variety of robotics tasks under uncertainties, due to environment disturbances and sensor limitations, for example.

Generating control signals in a global fashion is possible using the Hamilton-Jacobi-Bellman (HJB) equations, whose solution is the *value function* or the optimal cost-to-go. However, solving these equations is notoriously difficult and the state-of-the-art methods are grid-based, limiting their application in higher dimensional problems [2], [3]. Figure 1 illustrates a high-level comparison of these approaches. Numerical methods that use a grid to discretize the state space are limited to problems in lower dimensions. On the other hand, local trajectory optimization methods compute control laws from a single initial state to reach the target. Our method provides a compromise between these two approaches. Similar to local methods, it is grid-free and can be used in higher dimensional problems. Different from them, our method can generate control laws starting from initial states sampled from a large portion of the state space.

To this end, our contributions are two-fold: 1) We propose to approximate the state transitions using neural networks with sinusoidal activation functions and supervise them with numerically computed gradients of the system dynamics, 2) We present a method that integrates the learned dynamics into the HJB equations which are used to train the networks representing the controller and the value function. In our experiments, we show that our method can be used in a variety of systems in a sample efficient fashion.

II. Related Work

In this section, we review related work on various approaches for solving nonlinear optimal control problems and their relationship with our proposed method. Optimal control is concerned with finding a control law to drive a dynamical system to a goal state, such that a given cost function is minimized while satisfying the physical constraints of the system. Due to their broad applicability, there are many approaches for solving optimal control problems [4]. While systems with linear first-order constraints have closed-form optimal controllers [5], [6], their nonlinear counterparts often do not have analytic solutions [7]. Consequently, numerical methods have been commonly used to compute the controller for nonlinear systems [4].

This work was supported in part by NSF grants #1617718 and #2022894.

[1]Authors are with Department of Computer Science and Engineering, University of Minnesota, 100 Union St SE, Minneapolis, MN 55455, USA {engin003, isler}@umn.edu

Dynamic programming is a useful tool to efficiently compute optimal solutions in discrete time and space, using the Bellman equation. The continuous time analog of the Bellman equation is the HJB equation, a partial differential equation (PDE) in d state dimensions plus time. The solution of this PDE can be used to generate control inputs from arbitrary initial conditions from the state space. However, while offering global control laws, its computation can be costly when discretization of the state space is required to solve the PDE. This problem is commonly known as the curse of dimensionality. There are many studies for solving the HJB PDEs using numerical methods, such as level set methods [8] and the essentially nonoscillatory scheme [9], [10]. Yet, even the state-of-the-art numerical methods are limited to state spaces of dimension 4 to 5, since they rely on grid-based discretization of the state space [2]. Numerical methods based on HJB PDEs were shown to control Dubins vehicles in cluttered environments [11], [12] and in a differential game setting [13].

Nonlinear Model Predictive Control (NMPC) is another strategy to solve nonlinear optimal control problems. NMPC predicts the behavior of a system over some time horizon, such that the control input to the system optimizes the given cost function [14]. This approach uses standard optimization techniques [15] to solve the optimization problem starting from the initial state, and does not require space discretization. NMPCs have been successfully used for controlling nonlinear systems with complex dynamics, such as quadrotors [16], [17] and fixed-wing aircrafts [18], [19]. However, since the optimization is performed along trajectories starting from a single initial state, the controller needs to be recomputed for each new initial condition. Moreover, the original formulation requires an accurate model of the dynamics to effectively control the system [20], [21]. An alternative to NMPC is continuous-time Model-based Reinforcement Learning (CT-MBRL), which enables learning policies to solve complex tasks through learning the system's model and the policy for the agent [22], [23]. Model-based reinforcement learning methods have great potential to improve the sample complexity compared to model-free approaches [23]–[27]. Nevertheless, for simple control problems these model-based methods still require sizable interaction data during training to learn optimal policies compared to HJB-based neural controllers, as we show in our experiments (Section VI-B).

Recently, the advances in using neural networks to solve problems involving differential equations [28]–[30] led to an increasing interest in computing HJB-based controllers in a grid-free fashion, by learning the value function. One way to learn the value function is through direct supervision on the cost-to-go values [31]. If such supervision is not available, the existing HJB-based neural optimal control methods require the actual dynamics equations to analytically compute the Hamiltonian that is used to learn the value function and generate the control signals [32]–[34]. Our method builds on this line of work and uses a neural network to approximate the value function. However, different from these works, it uses learned system dynamics to integrate the outputs of the controller, and supervises the control law with the HJB PDEs computed using the learned dynamics. This is in contrast to existing approaches that derive the optimal controller analytically from the Hamiltonian based on known dynamics, whose solution requires special care when the control input is multi-dimensional and has saddle points. Additionally, our learned controller generates entire trajectories in one go instead of finding the optimal control based on the Hamiltonian at each time step, which allows fast training time performance.

Characteristics:	HJB-LSM	NMPC	CT-MBRL
Solution over the entire state space	✔	✘	✔
No space discretization	✘	✔	✔
No requirement of true dynamics	✘	✘	✔
Sample efficiency	-	✔	✘

TABLE I: A comparison of various approaches to solve nonlinear optimal control problems, highlighting the differences in their characteristics. We compare a) numerical level-set methods for solving HJB equations (HJB-LSM), b) nonlinear model predictive control (NMPC) and c) continuous-time model-based reinforcement learning (CT-MBRL).

Table I summarizes the comparison of different approaches to solve nonlinear optimal control problems in terms of whether the approach *i)* provides a global solution, *ii)* requires space discretization, *iii)* requires the true dynamics of the system, and *iv)* has good sample efficiency. Our method can generate control laws from multiple initial states that lie in a large portion of the state space, while not requiring space discretization and the true state transitions. In addition, it has better efficiency compared to recent MBRL methods.

III. Problem Statement

Our problem statement has a similar form to the classical optimal control formulations. For a fixed and finite time-horizon $[t_0, t_f]$, the state evolution is given by the following ordinary differential equation

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

where $\mathbf{x}(t) \in \mathcal{X} \subseteq \mathbb{R}^d$ and $\mathbf{u}(t) \in \mathcal{U} \subseteq \mathbb{R}^m$ denote the state and action at time $t \in [t_0, t_f]$, respectively. The state space of the system is $\mathcal{X}$, and the constrained action space is denoted by $\mathcal{U}$. Given an initial state $\mathbf{x}_0$ at time t_0, we have a continuous-time cost functional

$$J(\mathbf{x}(t_0), \mathbf{u}(\cdot), t_0) = \int_{t_0}^{t_f} L(\mathbf{x}(t), \mathbf{u}(t), t)dt + G(\mathbf{x}(t_f)) \tag{2}$$

subject to the first-order constraints given in Eq. 1, action constraints $\mathbf{u}(t) \in \mathcal{U}$, and $\mathbf{x}(t_0) = \mathbf{x}_0$. Here, $L : \mathbb{R}^d \times \mathbb{R}^m \times [t_0, t_f] \to \mathbb{R}$ is the running cost, whereas $G : \mathbb{R}^d \to \mathbb{R}$ is the terminal cost. The cost functional J and the dynamics f are assumed to be continuously differentiable in $\mathbf{x}$ and $\mathbf{u}$. Assuming its existence, our goal is to find the optimal controller $\mathbf{u}^*(t), \forall t \in [t_0, t_f]$, such that the cost functional is minimized:

$$\mathbf{u}^*(\cdot) = \arg\min_{\mathbf{u}(\cdot)\in\mathcal{U}} J(\mathbf{x}(t), \mathbf{u}(\cdot), t) \tag{3}$$

which generates the optimal state trajectory. The optimal controller also yields the value function $V : \mathbb{R}^d \times [t_0, t_f] \to \mathbb{R}$, the optimal cost starting from $(\mathbf{x}(t), t)$,

$$V(\mathbf{x}_t, t) = \min_{\mathbf{u}_{t:}} J(\mathbf{x}_t, \mathbf{u}_t, t) \tag{4}$$

Here and in the remainder of the paper, we use the $\mathbf{x}_t$ and $\mathbf{u}_t$ notation for the state and control signal at time t. We additionally use the notation $\mathbf{u}_{t:}$ to indicate the control in the time interval $[t, t_f]$, and $\mathbf{u}_{:}$ for the entire interval $[t_0, t_f]$. Therefore, $\mathbf{x}^*_{:}$ denotes the optimal state trajectory and $\mathbf{u}^*_{:}$ is the optimal control.

IV. Background

In this section, we review the Pontryagin's Maximum Principle (PMP) and the Hamilton-Jacobi-Bellman (HJB) equations that our method builds on. The PMP provides the necessary first-order conditions for the optimality of a controller. First, we need the control Hamiltonian which is given by

$$H(\mathbf{x}_t, \lambda_t, t) = \min_{\mathbf{u}_t} \left\{ L(\mathbf{x}_t, \mathbf{u}_t, t) + \lambda_t^\top \cdot f(\mathbf{x}_t, \mathbf{u}_t, t) \right\} \tag{5}$$

where λ_t is a vector of costate variables defined by the gradient of the value function with respect to the states, $\lambda_t = \nabla_{\mathbf{x}} V(\mathbf{x}_t, t)$[1]. Then, according to the PMP, the following conditions for optimality are necessary at all times t:

$$\begin{aligned} \partial \mathbf{x}^*_t / \partial t &= \nabla_\lambda H(\mathbf{x}^*_t, \lambda^*_t, t) \\ \partial \lambda^*_t / \partial t &= -\nabla_{\mathbf{x}} H(\mathbf{x}^*_t, \lambda^*_t, t) \\ \mathbf{0} &= \nabla_{\mathbf{u}} H(\mathbf{x}^*_t, \lambda^*_t, t) \end{aligned} \tag{6}$$

This means that once the value function V and its gradients $\nabla_{\mathbf{x}} V$ are known, the optimal control signals can be computed from any point from the state space. The value function satisfies the HJB equation, which involves the partial derivatives of V with respect to t and $\mathbf{x}$ as follows,

$$\partial V(\mathbf{x}_t, t)/\partial t + H(\mathbf{x}_t, \lambda_t, t) = 0 \tag{7}$$

with the boundary condition $V(\mathbf{x}_f, t_f) = G(\mathbf{x}_f)$. The HJB equation is derived using the principle of optimality. We refer the readers to [35]–[37] for their derivations.

V. Method

In this section, we describe our method for learning the dynamics of the system from data, and using the learned system dynamics to train a controller function.

A. Learning State Transitions from Data

Our method takes a model-based approach for learning the controller that minimizes the cost functional. To do so, in an offline process, we train a neural network f_θ that approximates the state transitions $\dot{\mathbf{x}}_t = f(\mathbf{x}_t, \mathbf{u}_t)$.

We use a multi-layer perceptron (MLP) $f_\theta : \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}^d$ with parameters θ and sine (sin) activations to represent the state transition function, as inspired by [30] who showed the success of periodic activation functions for fitting images and 3D shapes. To train f_θ, we first collect a dataset $\mathcal{D} = \{\mathbf{x}_i, \mathbf{u}_i\}_{i=1}^N$ of state and action pairs that are sampled uniformly from the state and action spaces. We use the ground truth transitions f to generate the desired outputs $\dot{\mathbf{x}}$ and train the network in a supervised fashion.

[1]We use the notation $\nabla_{\mathbf{x}}$ for gradients w.r.t. a vector $\mathbf{x}$, and $\partial \mathbf{x}/\partial t$ or $\dot{\mathbf{x}}$ for time derivatives of $\mathbf{x}$.

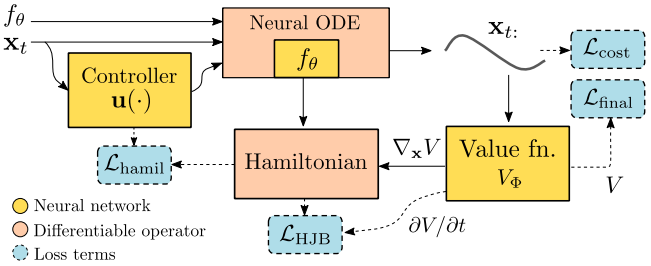

Fig. 2: **Method overview.** Our method takes as input a set of initial states $\mathbf{x}_t$, and a learned state transition function f_θ. The outputs of the controller $\mathbf{u}(\cdot)$ are integrated by a Neural ODE using f_θ to generate the state trajectories $\mathbf{x}_{t:}$ in a feedback loop. We use loss functions adapted by the HJB equation to train the controller and the value function V_Φ jointly. At test time, to evaluate the learned controller we use the actual state evolution function f.

In our method, we will use the learned transition function f_θ to compute the Hamiltonian with Eq. 5, and optimize the controller with the Hamiltonian's gradients. Therefore, in addition to the state derivatives $\dot{\mathbf{x}}$, we use the gradients of the transition function with respect to $\mathbf{x}$ and $\mathbf{u}$, $\nabla_{\mathbf{x}} f$ and $\nabla_{\mathbf{u}} f$, as supervision. Since the nonlinearities of f_θ are sinusoidal functions the gradients of the network are well-defined, and the network can approximate the transitions effectively. We use gradients of the state transitions computed using automatic differentiation to supervise the derivatives of f_θ with respect to $\mathbf{x}$ and $\mathbf{u}$.

The loss function we use for system identification is

$$\begin{aligned} \mathcal{L}_{\text{sys-id}} = \textstyle\sum_i & \left\| f_\theta(\mathbf{x}_i, \mathbf{u}_i) - f(\mathbf{x}_i, \mathbf{u}_i) \right\| \\ & + \left\| \nabla f_\theta(\mathbf{x}_i, \mathbf{u}_i) - \nabla f(\mathbf{x}_i, \mathbf{u}_i) \right\| \end{aligned}$$

where ∇f includes derivatives w.r.t. both $\mathbf{x}$ and $\mathbf{u}$.

Concurrent to our work, [38] recently showed the advantage of supervision with function gradients to learn the state transitions of a manipulator with 6 degrees of freedom (DoF). Unlike this study, our approach uses networks with sinusoidal activations which leads to better identification performance for the systems we study, as shown in our experiments.

B. The Value Function and Controller Design

We propose a learning-based strategy to generate controllers for solving control problems. Our method, which we call Control with Neural Dynamics (or CND), uses neural networks for representing both the value function and the controller. An overview of our method is shown in Figure 2.

Starting from a set of initial states sampled from a distribution, CND computes state trajectories using the learned state transitions f_θ (Section V-A) along with the control signals generated by the controller. The parameters of the

controller $\mathbf{u}(\cdot)$ are initialized uniformly at random, leading to state trajectories initially with high cost. We train the controller and the value function jointly by minimizing the cost functional and violations of the HJB equations.

The controller $\mathbf{u}(\cdot)$ takes the state at the current time $\mathbf{x}_t$ as input, and outputs the control signal $\mathbf{u}_t$. The $\mathbf{u}(\cdot)$ function is designed to be an MLP whose last layer is followed by a hyperbolic tangent $(\tanh)$ and appropriate rescaling depending on the system, to constrain the outputs be within the action space $\mathcal{U}$. We assume the stability of the system as long as $\mathbf{u}_t \in \mathcal{U}, \forall t \in [t_0, t_f]$, starting from a viable state $\mathbf{x}_0$. The controller outputs $\mathbf{u}_t$ are integrated in conjunction with f_θ by a neural ordinary differential equation (ODE) solver [28] to generate the state trajectory $\mathbf{x}_{t:}$ in a feedback loop.

CND represents the value function $V_\Phi : \mathbb{R}^d \times [t_0, t_f] \to \mathbb{R}$ also by an MLP with $\tanh$ activations and skip connections, in a similar vein with [32]. Approximating the value function using neural networks has the advantage of being grid-free, which means that the space complexity does not grow exponentially with the state dimension. Consequently, we are able to use our method for higher dimensional systems which may not be possible with existing numerical solvers.

The outputs of the value function $V_\Phi(\mathbf{x}_t, t)$, as well as its derivatives w.r.t. $\mathbf{x}$ and t are used to compute the Hamiltonian and for supervising the controller outputs using constraints from the HJB equations. The Hamiltonian is computed as

$$H = L(\mathbf{x}_t, \mathbf{u}_t, t) + \nabla_{\mathbf{x}} V_\Phi(\mathbf{x}_t, t)^\top \cdot f_\theta(\mathbf{x}_t, \mathbf{u}_t, t) \quad (8)$$

where L is the known running cost function, and $\mathbf{x}_t$ are the states dictated by the controller outputs $\mathbf{u}_t$.

In contrast to recent neural optimal control studies, such as [32], [33], our method cannot analytically derive the Hamiltonian and the optimal controller for each system, since the state transitions are assumed to be unknown. Instead, we use the loss functions described in the next section to optimize the controller and the value function.

C. Loss Functions and Training Details

Our goal is to train the value function and the controller for learning to generate optimal control signals. In our setting, there is no expert strategy or labeled controller data. Therefore, we opt for using loss functions adapted from the HJB PDEs. The first loss function we use optimizes the controller outputs by penalizing the running and terminal costs of the entire state trajectory,

$$\mathcal{L}_{\text{cost}} = \mathbb{E}_{\mathbf{x}_0 \sim \rho} \int_{t_0}^{t_f} L(\mathbf{x}_t, \mathbf{u}_t, t) dt + G(\mathbf{x}_f) \quad (9)$$

where ρ is the distribution the initial states are sampled from. Additionally, we optimize the value function for each state along the trajectory by

$$\mathcal{L}_{\text{HJB}} = \|\partial V_\Phi(\mathbf{x}_t, t)/\partial t + H\|_1 \quad (10)$$

and the boundary condition at the final time

$$\mathcal{L}_{\text{final}} = \|V_\Phi(\mathbf{x}_f, t_f) - G(\mathbf{x}_f)\|_1 \quad (11)$$

Finally, we use the gradients of the Hamiltonian w.r.t. $\mathbf{u}$ to regularize the controller

$$\mathcal{L}_{\text{hamil}} = \|\nabla_{\mathbf{u}} H\| \quad (12)$$

The total loss is then a weighted sum of these loss terms:

$$\mathcal{L} = \alpha_{\text{cost}}\mathcal{L}_{\text{cost}} + \alpha_{\text{HJB}}\mathcal{L}_{\text{HJB}} + \alpha_{\text{final}}\mathcal{L}_{\text{final}} + \alpha_{\text{hamil}}\mathcal{L}_{\text{hamil}} \quad (13)$$

where the weights are set to $\alpha_{\text{cost}} = 1, \alpha_{\text{HJB}} = 1, \alpha_{\text{final}} = 0.01, \alpha_{\text{hamil}} = 0.01$ in our experiments. We use the Adam optimizer [39] with exponentially decaying learning rate starting from 0.01 to train the networks.

VI. Experiments

In this section, we present experimental results comparing our method against the state-of-the-art. We design our experiments for investigating the following.

1) Design choices for the network and its supervision for learning the state transitions of a system
2) Optimal control performance for systems whose state transitions are not known beforehand
3) Optimal control performance for systems with known state transitions

In our experiments, we use recent MBRL algorithms, a neural optimal controller and a sampling-based motion planning algorithm to compare against our method.

A. Learning state transitions from data

In our first experiment, we analyze the performance of different strategies to approximate the state transitions $\dot{\mathbf{x}}_t = f(\mathbf{x}_t, \mathbf{u}_t)$ with a neural network f_θ, which is used as part of our method to learn the controller.

We compare strategies using different activation and loss functions to learn the state transitions of four systems: Dubins car, Acrobot, Cartpole and Quadrotor. Two of these, Acrobot and Cartpole, are well-studied systems that appear in benchmarks evaluating control and reinforcement learning algorithms [40], [41].

The other two systems, the Dubins car and Quadrotor, are commonly used motion models for robotics problems. The differential equation modeling the state evolution of a Dubins car with varying speed is

$$\dot{\mathbf{x}}_t = [\dot{x}_t,\ \dot{y}_t,\ \dot{\psi}_t]^\top = [v_t \cos(\psi_t),\ v_t \sin(\psi_t),\ \alpha_t \cdot v_t/r]^\top \quad (14)$$

where (x_t, y_t) denotes the position of the robot and ψ_t is the heading angle. The control signal includes the linear velocity $v_t \in [0, v_{max}]$ and the steering $\alpha_t \in [-1, 1]$.

The quadrotor system is controlled by its four motors. The dimension of this system's state space is 12, including the 3D positions, orientations in Euler angles and their time derivatives. We use the dynamics model given in [42].

Table II presents results comparing the design choices for the network architecture and its supervision. We compare networks with the Rectified Linear Unit (ReLU), $\tanh$ and $\sin$ activations. For each of these strategies, we use a three-layer MLP whose hidden units are followed by the corresponding activation functions. We train each of these

Strategies	Dubins Car (Dim: 3)		Acrobot (Dim: 4)		Cartpole (Dim: 4)		Quadrotor (Dim: 12)	
	Mean (s.d.) ↓	Median (iqr) ↓	Mean (s.d.) ↓	Median (iqr) ↓	Mean (s.d.) ↓	Median (iqr) ↓	Mean (s.d.) ↓	Median (iqr) ↓
ReLU (w/o ∇f)	0.0071±0.0033	0.0067±0.0043	0.7717±0.5395	0.6768±0.5464	0.0819±0.0551	0.0693±0.0611	20.8816±11.2258	18.6534±13.0919
ReLU (with ∇f)	0.0134±0.0090	0.0113±0.0094	0.6456±0.4999	0.5432±0.4726	0.0734±0.0511	0.0621±0.0543	35.9359±18.6272	30.7491±24.2143
Tanh (w/o ∇f)	0.0017±0.0009	0.0016±0.0009	0.4328±0.3702	0.3551±0.2858	0.0210±0.0111	0.0189±0.0119	8.3696±5.7249	6.8628±5.9049
Tanh (with ∇f)	0.0015±0.0003	0.0015±0.0005	0.2834±0.1800	0.2488±0.1958	0.0137±0.0067	0.0124±0.0079	18.6461±8.2238	18.6983±11.9576
Sine (w/o ∇f)	0.0014±0.0008	0.0013±0.0008	0.0673±0.0417	**0.0585±0.0414**	0.0149±0.0066	0.0140±0.0083	**1.3109±0.9937**	**1.0362±0.9085**
Sine (with ∇f)	**0.0008±0.0003**	**0.0008±0.0004**	**0.0664±0.0358**	0.0595±0.0419	**0.0101±0.0043**	**0.0096±0.0055**	15.6214±8.1264	15.8460±13.2569

TABLE II: Comparison of different activations (ReLU, Tanh and Sine) and loss functions (supervised with f and supervised with $f + \nabla f$) for learning the state transitions of the Dubins Car, Acrobot, Cartpole and Quadrotor systems. We report the mean (± standard deviation) and median (± interquantile range) errors of the predicted state transitions.

networks for 50,000 epochs using $(\mathbf{x}_t, \mathbf{u}_t)$ data sampled from the state and action spaces. The loss function used to train these networks are either with or without the $\nabla f = [\nabla_{\mathbf{x}} f, \nabla_{\mathbf{u}} f]$ supervision which is computed using PyTorch's auto-differentiation engine.

We report the average and median errors of the predicted state transitions $f_\theta(\mathbf{x}_t, \mathbf{u}_t)$. We find that neural networks with sinusoidal activation functions outperform the other tested activation functions for identifying the state transitions in all four tasks. This can be partly due to the trigonometric functions involved in all of these systems, and the angular shift-invariance that can be captured better with periodic activations. Additionally, we observe the largest gains over the other nonlinearities in the systems of Acrobot and Quadrotor more so, which is the system with the highest state space dimension among them. Using the ∇f supervision, we see further performance improvement in mean error for predicting the state transitions of the Dubins car, Acrobot and Cartpole systems. On the Quadrotor, however, it degrades the performance for all the activation functions. This may be attributed to the scale difference between the entries of the Jacobian, causing instabilities during training. Nevertheless, the performance of the sinusoidal activations for learning the state transitions is effective, and we use the learned function f_θ for generating the controllers in the following section.

B. Optimal control using learned dynamics

Next, we evaluate our method for generating a controller using learned dynamics. We compare our method against approaches from two lines of works: model-based reinforcement learning (MBRL) algorithms that learn the transitions model while optimizing the policy, and a recent neural optimal control method that derives the control law assuming known dynamics.

In our experiments, the running cost of the cost functional for our method is set to $(\mathbf{u}_t - \mathbf{u}^*)^\top R(\mathbf{u}_t - \mathbf{u}^*)$, and the terminal cost is $(\mathbf{x}_t - \mathbf{x}^*)^\top P(\mathbf{x}_t - \mathbf{x}^*)$. Here the matrices R and P are used to weight the relative importance of the cost terms. Unless noted otherwise, we use $P = I_d$ and $R = 0.01 I_m$, where I is the identity matrix. The desired state and action, $\mathbf{x}^*$ and $\mathbf{u}^*$, are given in all the methods.

The first experiment compares our method against model-based reinforcement learning methods Deep PILCO [26], PETS [43] and ENODE [23]. These MBRL methods are designed to model the transition probabilities (or the state transitions) using different approaches. Deep PILCO uses Bayesian neural networks to approximate the transitions, whereas PETS uses an ensemble of probabilistic neural networks. The most recent of them, ENODE, is designed to solve continuous time control problems by approximating the system dynamics with an ensemble of neural ODEs.

The results comparing our method against the MBRL baselines are presented in Table III, where we report the terminal error, the control inputs applied over the course of the trajectory and the number of evaluations of the learned dynamics f_θ for forward simulation during training. We found that Deep PILCO outperforms the other MBRL methods in both the Acrobot and Cartpole tasks. Our method's performance, on the other hand, is comparable to the baselines in the Acrobot environment and is better in the Cartpole task. Notably, our method requires significantly fewer number of evaluations of the learned transition function during training. This leads to much faster training time performance- whereas training the baselines take more than an hour on a Tesla V100, our method takes about 15 minutes.

Methods	Acrobot			Cartpole		
	Term. error	Control magn.	NFE	Term. error	Control magn.	NFE
Deep PILCO [26]	0.284±0.10	4.624±2.63	4789	0.146±0.20	1.168±1.17	7057
PETS [43]	3.095±0.69	6.968±0.36	12602	0.910±0.22	1.311±0.82	12602
ENODE [23]	1.133±0.52	4.934±1.83	27812	0.170±0.10	1.317±1.18	6262
Ours	1.071±0.63	5.936±0.72	1000	0.049±0.03	0.742±0.47	1000

TABLE III: Comparison of methods to solve the optimal control problems Acrobot and Cartpole without knowing the dynamics beforehand. We report the mean terminal state error, mean control magnitude along the trajectories, and the number of function evaluations used to train the controllers.

In the second experiment, we compare against NeuralOC [32], a HJB-based neural optimal controller that our method builds on. Similar to our method, NeuralOC trains the value function using penalty terms that enforce the HJB PDEs. However, unlike our method it requires the true state transitions f, and uses f to analytically derive the control law for a given system.

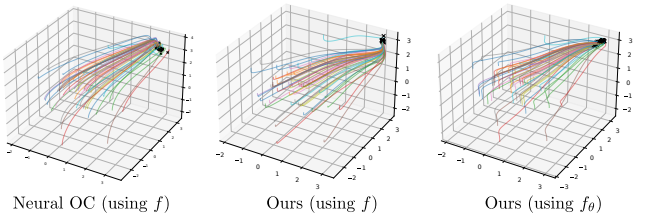

Fig. 3: Trajectories generated by NeuralOC and our method for controlling a quadrotor to reach a desired state at $[3, 3, 3]$.

We evaluate these methods on the task of controlling a quadrotor to reach a goal pose. The goal pose is set to the position $[3, 3, 3]$ in an upright orientation. The initial state

positions are sampled from a normal distribution around the origin $\mathcal{N}([0,0,0], I)$, and the rest of the state variables are initialized as 0, corresponding to an upright orientation and no initial velocities. Table IV presents quantitative results where we evaluate the methods starting from 1000 different initial states. Similar to the previous setting, we report the mean terminal state error and magnitude of control signals that generate the trajectories. To train NeuralOC, we use the hyperparameters given in [32] for the quadcopter task as is, except for the distribution the initial states are sampled from, which we enlarged. In terms of the terminal state error, we find that our method significantly outperforms NeuralOC if the transition function f is known. If the state transitions (S.T.) are unknown a priori, our method using the learned dynamics f_θ performs worse, but competitively. Remarkably, when we set the weight of the control input cost R to zero, our method trained with imperfect dynamics performs better than the baseline. Compared to all of our methods, we see that NeuralOC consumes the least energy on the control signals. Whereas our methods tend to maximize the control magnitude as long as they are within the action space. We observe this phenomenon qualitatively as well, as shown in Figure 3. The trajectories generated by NeuralOC tilt towards the goal position and start moving there, while our method trained with f learns to first go to a location under the goal state position and move upwards for better terminal state error performance. The controls generated by our method trained with f_θ, on the other hand, initially moves the quadrotor up and then move towards the goal for optimizing the terminal error at the expense of exerting larger control inputs.

Methods	S.T.	Terminal error	Control magn.
NeuralOC [32]	f	0.4601±0.1552	45.9289±17.8367
Ours	f	0.0690±0.2119	60.6403±39.7774
Ours	f_θ	0.5666±0.1131	71.3644±29.4289
Ours ($R = 0$)	f_θ	0.4161±0.1732	72.4603±18.6037

TABLE IV: Comparison of HJB-based neural controllers for controlling the quadrotor to reach a desired state. In terms of the terminal error, our method outperforms the baseline significantly when the state transitions f is known. Our method performs competitively even when the system dynamics f_θ is learned and imperfect.

C. Optimal control using known dynamics

In our final experiment, we evaluate the performance of our controller when it is trained with known state transitions. We use the Dubins car with equations of motion given by Eq. 14, in an environment containing a circular obstacle. We compare our method against RRT* [44], a variant of the rapidly-exploring random tree which is a sampling-based motion planning algorithm. Since this baseline is not a controller, but a planner, the concepts of terminal state error and magnitude of applied control are not applicable. Therefore, we instead compare the path lengths returned by RRT* and the length of the trajectories our method computes. The initial states in this experiment are sampled from a uniform distribution with lower bound $[-3.5, -3, -\pi]$ and upper bound $[-2.5, 3, \pi]$. The goal state is $[0, 0, 0]$ and the circular obstacle of radius 0.5 is located at $(-1, 0)$. We use the RRT* implementation from the PythonRobotics library [45]. The collision avoidance objective is incorporated into our controller's cost function as a penalty term, similar to [32], [34]. Table V compares the performance of our method against RRT*. Since it is an iterative algorithm whose solutions improve over time, we report two different versions of RRT* with different thresholds on the maximum number of iterations, to show how the computation time is compromised to improve the output paths. We find that our method generates trajectories with better quality in terms of the path length. Additionally, its computation time during evaluation is two orders of magnitude smaller and has better overall success rate. Figure 4 shows the qualitative performance of each method. We see that with RRT* there are unnecessary detours resulting in suboptimal output paths.

Methods	Trajectory len.	Compute time (s)	Success rate
RRT* (iter: 100)	4.8871±0.9607	6.9313	0.889
RRT* (iter: 200)	4.7343±0.8451	26.0595	0.939
Ours	4.1076±0.6916	0.0634	0.999

TABLE V: Comparison of our method against RRT* [44], a sampling-based motion planning algorithm, for controlling a Dubins vehicle to a goal state in the presence of an obstacle.

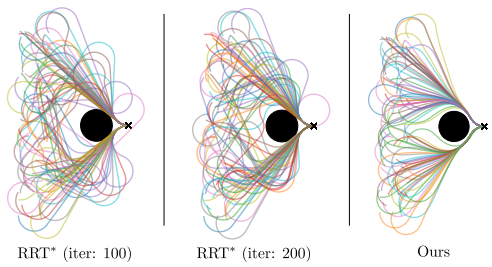

Fig. 4: Trajectories generated by RRT* and our method for controlling a Dubins car to reach a desired state in the presence of a circular obstacle.

VII. Conclusion

In this paper, we presented a method to generate control signals to drive a system to its desired state without a known dynamics model. Our method learns the state transitions to forward simulate the trajectories by fitting a neural network. Then, it uses the learned transitions along with loss functions that enforce the HJB PDEs to supervise the controller. We evaluated our method on a number of systems to demonstrate its global nature and sample efficiency. Our method assumes that the state transitions can be learned in one shot, as opposed to MBRL methods that optimize the model and the policy repeatedly until convergence. While our approach is sufficient for the control problems studied in this paper, for more complex tasks such as dexterous manipulation [46] and legged locomotion [47], continual optimization of the model may be necessary. Additionally, our method learns the controller for a fixed goal state, and requires re-optimization for new goal states, which we plan to address in future work.

REFERENCES

[1] Tor A Johansen. Introduction to nonlinear model predictive control and moving horizon estimation. *Selected topics on constrained and nonlinear control*, 1:1–53, 2011.

[2] Somil Bansal, Mo Chen, Sylvia Herbert, and Claire J Tomlin. Hamilton-jacobi reachability: A brief overview and recent advances. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 2242–2253. IEEE, 2017.

[3] Keiko Nagami and Mac Schwager. Hjb-rl: Initializing reinforcement learning with optimal control policies applied to autonomous drone racing. In *Robotics: Science and Systems*, 2021.

[4] Anil V Rao. A survey of numerical methods for optimal control. *Advances in the Astronautical Sciences*, 135(1):497–528, 2009.

[5] Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, 2002.

[6] Michael Basin and Jesus Rodriguez-Gonzalez. A closed-form optimal control for linear systems with equal state and input delays. *Automatica*, 41(5):915–920, 2005.

[7] Gao Tang and Kris Hauser. A data-driven indirect method for nonlinear optimal control. *Astrodynamics*, 3(4):345–359, 2019.

[8] 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.

[9] Stanley Osher and Chi-Wang Shu. High-order essentially nonoscillatory schemes for hamilton–jacobi equations. *SIAM Journal on numerical analysis*, 28(4):907–922, 1991.

[10] Y-T Zhang and C-W Shu. Eno and weno schemes. In *Handbook of Numerical Analysis*, volume 17, pages 103–122. Elsevier, 2016.

[11] Ryo Takei, Richard Tsai, Haochong Shen, and Yanina Landa. A practical path-planning algorithm for a simple car: a hamilton-jacobi approach. In *Proceedings of the 2010 American Control Conference*, pages 6175–6180. IEEE, 2010.

[12] Christian Parkinson, Andrea L Bertozzi, and Stanley J Osher. A hamilton-jacobi formulation for time-optimal paths of rectangular nonholonomic vehicles. In *IEEE Conference on Decision and Control (CDC)*, pages 4073–4078. IEEE, 2020.

[13] Ian M Mitchell, Alexandre M Bayen, and Claire J Tomlin. A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games. *IEEE Transactions on automatic control*, 50(7):947–957, 2005.

[14] Rolf Findeisen and Frank Allgöwer. An introduction to nonlinear model predictive control. In *21st Benelux meeting on systems and control*, volume 11, pages 119–141. Citeseer, 2002.

[15] Moritz Diehl, Hans Joachim Ferreau, and Niels Haverbeke. Efficient numerical methods for nonlinear mpc and moving horizon estimation. In *Nonlinear model predictive control*, pages 391–417. Springer, 2009.

[16] Andrea Zanelli, Greg Horn, Gianluca Frison, and Moritz Diehl. Nonlinear model predictive control of a human-sized quadrotor. In *2018 European Control Conference (ECC)*, pages 1542–1547. IEEE, 2018.

[17] Dong Wang, Quan Pan, Yang Shi, Jinwen Hu, et al. Efficient nonlinear model predictive control for quadrotor trajectory tracking: Algorithms and experiment. *IEEE Transactions on Cybernetics*, 51(10):5057–5068, 2021.

[18] Yeonsik Kang and J Karl Hedrick. Linear tracking for a fixed-wing uav using nonlinear model predictive control. *IEEE Transactions on Control Systems Technology*, 17(5):1202–1210, 2009.

[19] Thomas Stastny and Roland Siegwart. Nonlinear model predictive guidance for fixed-wing uavs using identified control augmented dynamics. In *International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 432–442. IEEE, 2018.

[20] Alberto Bemporad and Manfred Morari. Robust model predictive control: A survey. In *Robustness in identification and control*, pages 207–226. Springer, 1999.

[21] Brett T Lopez, Jean-Jacques E Slotine, and Jonathan P How. Dynamic tube mpc for nonlinear systems. In *2019 American Control Conference (ACC)*, pages 1655–1662. IEEE, 2019.

[22] Kenji Doya. Reinforcement learning in continuous time and space. *Neural computation*, 12(1):219–245, 2000.

[23] Cagatay Yildiz, Markus Heinonen, and Harri Lähdesmäki. Continuous-time model-based reinforcement learning. In *International Conference on Machine Learning*, pages 12009–12018. PMLR, 2021.

[24] Thomas M Moerland, Joost Broekens, and Catholijn M Jonker. Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712*, 2020.

[25] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning*, pages 465–472. Citeseer, 2011.

[26] Yarin Gal, Rowan McAllister, and Carl Edward Rasmussen. Improving pilco with bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, volume 4, page 25, 2016.

[27] Yuda Song and Wen Sun. Pc-mlp: Model-based reinforcement learning with policy cover guided exploration. In *International Conference on Machine Learning*, pages 9801–9811. PMLR, 2021.

[28] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 31, 2018.

[29] 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.

[30] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473, 2020.

[31] Jinwook Huh, Daniel D Lee, and Volkan Isler. Learning continuous cost-to-go functions for non-holonomic systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5772–5779. IEEE, 2021.

[32] Derek Onken, Levon Nurbekyan, Xingjian Li, Samy Wu Fung, Stanley Osher, and Lars Ruthotto. A neural network approach for high-dimensional optimal control applied to multiagent path finding. *IEEE Transactions on Control Systems Technology*, 2022.

[33] Somil Bansal and Claire J Tomlin. Deepreach: A deep learning approach to high-dimensional reachability. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1817–1824. IEEE, 2021.

[34] Michael Lutter, Boris Belousov, Kim Listmann, Debora Clever, and Jan Peters. Hjb optimal feedback control with deep differential value functions and action constraints. In *Conference on Robot Learning*, pages 640–650. PMLR, 2020.

[35] Donald E Kirk. *Optimal control theory: an introduction*. Courier Corporation, 2004.

[36] Brian DO Anderson and John B Moore. *Optimal control: linear quadratic methods*. Courier Corporation, 2007.

[37] Daniel Liberzon. *Calculus of variations and optimal control theory: a concise introduction*. Princeton university press, 2011.

[38] Youngho Kim, Hoosang Lee, and Jeha Ryu. Learning an accurate state transition dynamics model by fitting both a function and its derivative. *IEEE Access*, 10:44248–44258, 2022.

[39] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[40] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[41] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[42] Luis Rodolfo García Carrillo, Alejandro Enrique Dzul López, Rogelio Lozano, and Claude Pégard. Modeling the quad-rotor mini-rotorcraft. In *Quad Rotorcraft Control*, pages 23–34. Springer, 2013.

[43] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in Neural Information Processing Systems*, 31, 2018.

[44] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[45] Atsushi Sakai, Daniel Ingram, Joseph Dinius, Karan Chawla, Antonin Raffin, and Alexis Paques. PythonRobotics: a python code collection of robotics algorithms. 2018.

[46] Anusha Nagabandi, Kurt Konolige, Sergey Levine, and Vikash Kumar. Deep dynamics models for learning dexterous manipulation. In *Conference on Robot Learning*, pages 1101–1112. PMLR, 2020.

[47] Yu Sun, Wyatt L Ubellacker, Wen-Loong Ma, Xiang Zhang, Changhao Wang, Noel V Csomay-Shanklin, Masayoshi Tomizuka, Koushil

Sreenath, and Aaron D Ames. Online learning of unknown dynamics for model-based controllers in legged locomotion. *IEEE Robotics and Automation Letters*, 6(4):8442–8449, 2021.