A Neural Network Approach for High-Dimensional Optimal Control Applied to Multi-Agent Path Finding

Derek Onken, Levon Nurbekyan, Xingjian Li, Samy Wu Fung, Stanley Osher, and Lars Ruthotto

Abstract—**We propose a neural network approach that yields approximate solutions for high-dimensional optimal control problems and demonstrate its effectiveness using examples from multi-agent path finding. Our approach yields controls in a feedback form, where the policy function is given by a neural network (NN). Specifically, we fuse the Hamilton-Jacobi-Bellman (HJB) and Pontryagin Maximum Principle (PMP) approaches by parameterizing the value function with an NN. Our approach enables us to obtain approximately optimal controls in real-time without having to solve an optimization problem. Once the policy function is trained, generating a control at a given space-time location takes milliseconds; in contrast, efficient non-linear programming methods typically perform the same task in seconds. We train the NN offline using the objective function of the control problem and penalty terms that enforce the HJB equations. Therefore, our training algorithm does not involve data generated by another algorithm. By training on a distribution of initial states, we ensure the controls' optimality on a large portion of the state-space. Our grid-free approach scales efficiently to dimensions where grids become impractical or infeasible. We apply our approach to several multi-agent collision-avoidance problems in up to 150 dimensions. Furthermore, we empirically observe that the number of parameters in our approach scales linearly with the dimension of the control problem, thereby mitigating the curse of dimensionality.**

Index Terms—**collision avoidance, Hamilton-Jacobi-Bellman equation, high-dimensional control, multi-agent, neural networks, optimal control**

I. INTRODUCTION

Decision-making for complex systems using optimal control (OC) has become increasingly relevant yet remains challenging, especially when the state dimension is high and decisions are needed in real-time. Examples include controlling a swarm of quadcopters [1] with collision-avoidance and controlling an unmanned aerial vehicle [2]–[5] while reacting to possible wind interference during flight.

We consider real-time OC applications that lead to deterministic, finite time-horizon control problems. The speed of generating new controls is critical in these real-time problems where unexpected situations occur during deployment, e.g., wind interference [6]–[10]. While nonlinear programming (NLP) methods can provide optimal controls for fixed initial states [11], computation may be too slow for real-time applications: seconds vs milliseconds. We provide controls in a feedback form, where the policy is given by a neural network (NN). Hence, we generate approximately optimal controls in milliseconds (real-time) without having to solve an optimization problem.

Two of the most common frameworks to solve OC problems are the Pontryagin Maximum Principle (PMP) [12] and Hamilton-Jacobi-Bellman (HJB) partial differential equation (PDE) [13]. The PMP is often suitable for high-dimensional problems (Sec. III-C). A local solution method, the PMP finds the optimal policy for a single initial state, so deviations of the system from the optimal trajectory require re-computation of the solution. In contrast, the HJB approach is a global solution method suitable for real-time applications. It is based on solving the HJB PDE to obtain the value function (Sec. III-D). However, state-of-the-art HJB solvers, e.g., ENO/WENO [14], are grid-based and can suffer from the curse of dimensionality (CoD) [13], i.e., costs increase exponentially with dimension. For OC problems with a state-space dimension exceeding four, the CoD renders using grid-based HJB solvers infeasible.

We fuse the principles of the PMP and HJB methods to formulate an NN approach that is semi-global while mitigating CoD. In particular, we begin by parameterizing the value function with an NN, which circumvents CoD by approximating the solution to the HJB PDE in the underlying parameter space. Thus, our method is grid-free and suitable for high-dimensional problems. Using the PMP, we express the control in feedback form. We train the NN approximation of the value function by minimizing the expected cost on a distribution of initial states. As we minimize the cost function directly, our approach does not require generating solutions via an existing algorithm for training—i.e., it is not supervised. Training the NN on a distribution of initial states ensures the controls'

D. Onken is with the Dept of Computer Science, Emory University, Atlanta, GA, USA (derek@derekonken.com)

L. Nurbekyan and S. Osher are with the Dept of Mathematics, UCLA, Los Angeles, CA, USA (lnurbek@math.ucla.edu; sjo@math.ucla.edu)

X. Li and L. Ruthotto are with the Dept of Mathematics, Emory University, Atlanta, GA, USA (xingjian.li@emory.edu; lruthotto@emory.edu)

S. Wu Fung is with the Dept of Applied Mathematics and Statistics, Colorado School of Mines, Golden, CO, USA (swufung@mines.edu)

optimality on a large portion of the state-space; hence, our approach is semi-global. As we demonstrate, the controls are robust to moderate perturbations or shocks to the system, such as wind interference (Sec. V-B.4). The controls are obtained in a feedback form via prior offline training, so the feedback form can be applied efficiently during deployment. Lastly, we improve the NN training by adding residual penalty terms derived from the HJB PDE, similar to [15]–[17].

This paper extends a preliminary conference version of the approach [18] with more extensive and thorough experiments. Specifically, we add experiments where agents swap positions with each other and one involving a nonlinear control-affine quadcopter with complicated dynamics. Additionally, we include experiments that investigate the sensitivity of NN hyperparameters, thoroughly compare the semi-global nature of the NN model against thousands of baseline solutions, demonstrate the efficient deployment timings of the NN, and test the influence of CoD on the NN.

Our formulation is applicable to OC problems for which the underlying Hamiltonian can be computed efficiently, e.g., affine controls with convex Lagrangians. Real-world applications that fall within this scope arise in centrally controlled multi-agent systems, which are the focus of this work. These also lead to challenging high-dimensional OC problems. Indeed, for n agents in a q-dimensional space, we obtain a $d = n \cdot q$-dimensional OC problem. Therefore, even moderate n, q yield problems out of reach for traditional HJB solvers.

In our experiments, we solve a series of multi-agent OC problems whose state-space dimensions range from four ($n=2, q=2$) to 150 ($n=50, q=3$). First, we solve a two-agent corridor problem with a smooth obstacle terrain (Sec. V-B). Second, we investigate a two-agent problem where agents swap positions while avoiding hard obstacles and a 12-agent unobstructed version found in [19] (Sec. V-C). Third, we experiment with a 50-agent swarm of three-dimensional agents obstructed by rectangular prisms inspired by [1] (Sec. V-D). Finally, we solve a 12-dimensional single-agent quadcopter problem with complicated dynamics from [20] (Sec. V-E). Accompanying videos of our NN's solutions to these problems reside at `https://imgur.com/a/eWr6sUb`.

Using the corridor problem, we test our model's robustness to external shocks (random additive perturbations) that occur during deployment. We perform an example shock (Fig. 3) and compare the NN's response against a baseline method (Sec. V-A). Furthermore, we compare the solutions from the NN approach and the baseline on thousands of initial points (Fig. 5). In this example, the NN reacts approximately optimally to moderate shocks (in terms of solution quality). For large shocks, the NN learns a suboptimal control but still drives the agents towards the targets. However, the NN (trained offline) demonstrates quick online speed (Table II).

As one indicator that our approach effectively mitigates the CoD, we demonstrate empirically that increasing the state-space dimension does not lead to an exponential growth in computational costs. Specifically, we obtain approximately optimal controls by increasing the number of NN parameters linearly while keeping all other settings, including the batch size and number of optimization steps, fixed (Fig. 9). We also show that we are able to solve a 150-dimensional problem in less than one hour on a single graphics processing unit (GPU).

II. RELATED WORK

In recent years, many new numerical methods and machine learning approaches have been developed for solving high-dimensional OC problems. We discuss deterministic (Sec. II-A) and stochastic (Sec. II-B) settings separately because they differ considerably. In Sec. II-C, we survey the state-of-the-art in the application domain that motivates our experiments.

A. High-Dimensional Deterministic Optimal Control

A common difficulty in solving high-dimensional OC problems is the CoD. Exceptions are convex OC problems for which high-dimensional solvers can be devised via primal-dual methods and Hopf-Lax representation formulae [20]–[27].

Kang and Wilcox [28] alleviate the CoD by introducing a sparse grid in the state-space and use the method of characteristics to solve boundary value problems over each sparse grid point. To approximate the feedback control at arbitrary points, they interpolate the solutions of the grid using high-order polynomials. The authors solve up to six-dimensional control problems. Nakamura-Zimmerer *et al.* [29] also attempt to alleviate CoD by learning a closed-form value function. First, trajectories are generated in a similar manner as in [28]. Using a supervised learning approach, the NN is then trained to match the generated trajectories. The trajectories (training data) are generated adaptively using information about the adjoint and by combining progressive batching with an efficient adaptive sampling technique.

Bansal and Tomlin [10] solve high-dimensional reachability problems by combining the Hamilton-Jacobi-Isaacs (HJI) framework with the Deep Galerkin Method in [30]. More precisely, they approximate the value function with an NN and minimize the empirical average of the HJI residual at randomly drawn space-time points.

Our work stems from the same framework as [31], which approximates the feedback control with an NN then optimizes the control cost on a distribution of initial states. The authors also provide a theoretical analysis of OC solutions via NN approximations. We extend the framework to finite horizon problems with non-quadratic costs and parameterize the value function instead of the feedback function. This extension enables penalization of the HJB conditions, which empirically improves numerical performance for solving high-dimensional mean-field games, mean-field control, and normalizing flows [15]–[17]. We demonstrate similar advantages for OC problems considered in this work, which make similar use of NNs to parameterize the value function.

B. High-Dimensional Stochastic Optimal Control

In the seminal works [32], [33], the authors solve high-dimensional semilinear parabolic PDE problems by the method of (stochastic) characteristics. To overcome CoD, they approximate the gradient of the solution at different times by NNs and introduce a loss function that measures the deviation

from the correct terminal condition in the characteristic equations. In particular, they solve high-dimensional *stochastic* OC problems by solving the corresponding viscous HJB equation. This method recovers the gradient of the solution as a function of space and time and can be considered a global method. Nevertheless, loss functions employed in [32], [33] consider only one initial point at a time, and the generalization depends on how well the generated random trajectories fill the space. The variance of the trajectories increases as time grows. Finally, in the deterministic limit the method becomes local as there is no diffusion to enforce the trajectories to explore the whole space. Similar techniques are applied in [34], [35] based on different loss functions.

In [36], the authors solve stochastic OC problems by directly approximating controls and using the control objective as a loss function. As in [32], [33], the loss function considers a single initial point.

C. Multi-Agent Path-Finding

Multi-Agent Path-Finding (MAPF) [37]–[39] methods are methods tailored for multi-agent control problems. These methods tend to focus on collision avoidance rather than optimality. Among these are Conflict-Based Search (CBS) methods [40], [41], which are two-level algorithms. At the low level, optimal paths are found for individual agents, while at the high-level, a search is performed in a constraint tree whose nodes include constraints on time and location for a single agent. Decoupled optimization approaches [1], [42] first compute independent paths and then try to avoid collision afterwards. Other approaches phrase these as a constrained optimization problem [43]–[46]. Such methods are often combined with graph-based methods [47], sub-dimensional expansions [48], and CBS approaches [49], [50]. Another approach phrases the MAPF problem as a differential game [19]. Provided certain assumptions, this differential game strategy guarantees that the agents reach their targets while avoiding collisions. Machine learning approaches for multi-agent control have also been successfully applied in [51] where supervised learning is used to imitate non-machine-learning solutions generated by [1]. Our approach differs from these methods in that we do not have a data generation and fitting/imitation phases; instead, we directly solve for the control objective. Additionally, localization and interaction modeling techniques such as in [52] can be incorporated in our model in a straightforward manner.

III. MATHEMATICAL FORMULATION

We briefly discuss the general OC framework, derive the multi-agent control problems with collision avoidance used in the experiments, and review the theoretical foundations of the NN approach, primarily following [53, Chapters I-II].

A. Optimal Control Formulation

We consider deterministic, finite time-horizon OC problems. For a fixed time-horizon $[0, T]$, we have system dynamics

$$\partial_s \boldsymbol{z}_{t,\boldsymbol{x}}(s) = f(s, \boldsymbol{z}_{t,\boldsymbol{x}}(s), \boldsymbol{u}_{t,\boldsymbol{x}}(s)), \quad \boldsymbol{z}_{t,\boldsymbol{x}}(t) = \boldsymbol{x}, \quad (1)$$

for $t \leq s \leq T$. Here, $\boldsymbol{x} \in \mathbb{R}^d$ is the initial state, and $t \in [0, T]$ is the initial time of the system. Next, $\boldsymbol{z}_{t,\boldsymbol{x}}(s) \in \mathbb{R}^d$ is the state of the system at time $s \in [t, T]$ with initial data $(t, \boldsymbol{x})$, and $\boldsymbol{u}_{t,\boldsymbol{x}}(s) \in U \subset \mathbb{R}^a$ is the control applied at time s. The function $f \colon [0, T] \times \mathbb{R}^d \times U \to \mathbb{R}^d$ models the evolution of the state $\boldsymbol{z}_{t,\boldsymbol{x}} \colon [t, T] \to \mathbb{R}^d$ in response to the control $\boldsymbol{u}_{t,\boldsymbol{x}} \colon [t, T] \to U$.

Next, we suppose that the control $\boldsymbol{u}_{t,\boldsymbol{x}} \colon [t, T] \to U$ and the trajectory $\boldsymbol{z}_{t,\boldsymbol{x}} \colon [t, T] \to \mathbb{R}^d$ satisfying (1) yield a cost

$$\int_t^T L(s, \boldsymbol{z}_{t,\boldsymbol{x}}(s), \boldsymbol{u}_{t,\boldsymbol{x}}(s)) \, \mathrm{d}s \; + \; G(\boldsymbol{z}_{t,\boldsymbol{x}}(T)), \quad (2)$$

where $L \colon [0, T] \times \mathbb{R}^d \times U \to \mathbb{R}$ is the *running cost* or the *Lagrangian*, and $G \colon \mathbb{R}^d \to \mathbb{R}$ is the *terminal cost*. We assume that f, L, G, U are sufficiently regular (see [53, Sec. I.3, I.8-9] for a list of assumptions). The goal of the OC problem is to find an optimal control $\boldsymbol{u}_{t,\boldsymbol{x}}^*$ that incurs the minimal cost, i.e,

$$\Phi(t, \boldsymbol{x}) = \inf_{\boldsymbol{u}_{t,\boldsymbol{x}}} \left\{ \int_t^T L(s, \boldsymbol{z}_{t,\boldsymbol{x}}(s), \boldsymbol{u}_{t,\boldsymbol{x}}(s)) \, \mathrm{d}s \right.$$
$$\left. + G(\boldsymbol{z}_{t,\boldsymbol{x}}(T)) \right\} \; \text{s.t.} \; (1), \quad (3)$$

where Φ is called the *value function*. A solution $\boldsymbol{u}_{t,\boldsymbol{x}}^*$ of (3) is called an *optimal control*. Accordingly, the $\boldsymbol{z}_{t,\boldsymbol{x}}^*$ which corresponds to $\boldsymbol{u}_{t,\boldsymbol{x}}^*$ is called an *optimal trajectory*.

We also define the *Hamiltonian* of the system by

$$H(t, \boldsymbol{z}, \boldsymbol{p}) = \sup_{\boldsymbol{u} \in U} \{-\boldsymbol{p} \cdot f(t, \boldsymbol{z}, \boldsymbol{u}) - L(t, \boldsymbol{z}, \boldsymbol{u})\}$$
$$= \sup_{\boldsymbol{u} \in U} \mathcal{H}(t, \boldsymbol{z}, \boldsymbol{p}, \boldsymbol{u}), \quad (4)$$

where $\boldsymbol{p} \in \mathbb{R}^d$ is called the *adjoint state*. The Hamiltonian is a key ingredient in the Pontryagin Maximum Principle [12] (Sec. III-C) and also appears in the Hamilton-Jacobi-Bellman PDE [13] (Sec. III-D), which together form the foundation of our numerical solution approach.

B. Collision-Avoiding Multi-Agent Control Problems

While our NN approach is applicable to a broad range of OC problems, our numerical examples are motivated by centrally controlled multi-agent problems with collision avoidance. Optimal decision-making for this class of problems is complicated due to the high-dimensionality of the control problem and the interactions between the agents. These difficulties are exacerbated in the presence of random shocks and other forms of uncertainty. Here, we describe the generic set up of these problems and refer to Section V for specific instances.

We seek to control a system of n agents with initial states $x^{(1)}, \dots, x^{(n)} \in \mathbb{R}^q$. We denote the initial joint-state of the system by concatenating the agents' initial states, i.e.,

$$\boldsymbol{x} = \left(x^{(1)}, x^{(2)}, \dots, x^{(n)} \right) \in \mathbb{R}^d. \quad (5)$$

Thus, the dimension of the joint-state of the system is $d = q \cdot n$. Similarly, we denote the joint-state of the system at time s by

$$\boldsymbol{z}_{t,\boldsymbol{x}}(s) = \left(z_{t,\boldsymbol{x}}^{(1)}(s), z_{t,\boldsymbol{x}}^{(2)}(s), \dots, z_{t,\boldsymbol{x}}^{(n)}(s) \right), \quad (6)$$

where, for a fixed $s \in [t, T]$, $z_{t,\boldsymbol{x}}^{(i)}(s) \in \mathbb{R}^q$ is the state of the ith agent. Also, we represent the control as

$$\boldsymbol{u}_{t,\boldsymbol{x}}(s) = \left(u_{t,\boldsymbol{x}}^{(1)}(s), u_{t,\boldsymbol{x}}^{(2)}(s), \ldots, u_{t,\boldsymbol{x}}^{(n)}(s) \right). \quad (7)$$

Hence, both the dimension of the state and the control space are proportional to the number of agents.

In the numerical experiments, the terminal costs depend on the distance between the agents' final positions and their given target states. We denote the target joint-state of the system by the vector $\boldsymbol{y} \in \mathbb{R}^d$, obtained by concatenating the target states for all the agents as in (5), and consider the terminal cost

$$G\big(\boldsymbol{z}_{t,\boldsymbol{x}}(T)\big) = \frac{\alpha_1}{2} \|\boldsymbol{z}_{t,\boldsymbol{x}}(T) - \boldsymbol{y}\|^2. \quad (8)$$

The Lagrangian can be written as

$$L\big(s, \boldsymbol{z}, \boldsymbol{u}\big) = E\big(\boldsymbol{u}\big) + \alpha_2 Q(\boldsymbol{z}) + \alpha_3 W\big(\boldsymbol{z}\big), \quad (9)$$

where the scalar weighting parameters $\alpha_1, \alpha_2, \alpha_3$ are problem-specific and model the trade-off between the individual terms.

The first term in (9), $E\colon U \to \mathbb{R}$, is the *energy term*, which is the total consumption cost comprised of individual ones

$$E\big(\boldsymbol{u}_{t,\boldsymbol{x}}\big) = \sum_{i=1}^{n} E_i\Big(u_{t,\boldsymbol{x}}^{(i)}\Big). \quad (10)$$

In our experiments, we use $E_i(u) = \frac{1}{2}\|u\|^2 + \kappa$ with a problem-dependent constant $\kappa \in \mathbb{R}$, which simplifies the Hamiltonian computation in (4). Unlike the other terms, this first term depends explicitly on the control.

The second term in (9), $Q\colon \mathbb{R}^d \to \mathbb{R}$, models obstacles by penalizing agents at certain spatial locations (e.g., a terrain function) and decouples into

$$Q\big(\boldsymbol{z}_{t,\boldsymbol{x}}\big) = \sum_{i=1}^{n} Q_i\Big(z_{t,\boldsymbol{x}}^{(i)}\Big), \quad (11)$$

where $Q_i\colon \mathbb{R}^q \to \mathbb{R}$ models the ith agent's spatial preferences.

The third term in (9), $W\colon \mathbb{R}^d \to \mathbb{R}$, models interactions among the individual agents. For example, this term can penalize proximity among agents to avoid collisions, i.e,

$$W\big(\boldsymbol{z}_{t,\boldsymbol{x}}\big) = \sum_{j \neq i} w\Big(z_{t,\boldsymbol{x}}^{(i)}, z_{t,\boldsymbol{x}}^{(j)}\Big) \quad (12)$$

for function $w\colon \mathbb{R}^q \times \mathbb{R}^q \to \mathbb{R}$,

$$w\Big(z^{(i)}, z^{(j)}\Big) = \begin{cases} \exp\left(-\frac{\|z^{(i)} - z^{(j)}\|_2^2}{2r^2} \right), & \left\| z^{(i)} - z^{(j)} \right\|_2 < 2r, \\ 0, & \text{otherwise.} \end{cases} \quad (13)$$

Here, $r > 0$ is the radius of an agent's safety region or space bubble. While not guaranteed, this w term can in practice prevent the overlapping of the agents' space bubbles, thus avoiding collisions, when α_3 is sufficiently large. Our approach straightforwardly extends to non-symmetric interaction costs and heterogeneous agents.

We note that the presence of the terrain function Q and the interaction potential W render the objective function non-convex in $\boldsymbol{z}$. However, in our experiments, the function is strongly convex (in fact, quadratic) in $\boldsymbol{u}$, which simplifies evaluations of the Hamiltonian (4) under certain assumptions on f. Our framework can be directly applied to other choices of G, E, Q, and W as long as H can be computed efficiently.

C. The Pontryagin Maximum Principle

The Pontryagin Maximum Principle (PMP) provides a set of necessary first-order optimality conditions for the optimal control $\boldsymbol{u}_{t,\boldsymbol{x}}^*(\cdot)$ and trajectory $\boldsymbol{z}_{t,\boldsymbol{x}}^*(\cdot)$ originating from fixed initial data $(t, \boldsymbol{x})$. Since a new instance of the problem needs to be solved when the initial data change or the system's state deviates from the optimal curve, the PMP can be considered a *local* solution method.

Theorem 1 (Theorem I.6.3 [53]): Assume that $(\boldsymbol{z}_{t,\boldsymbol{x}}^*, \boldsymbol{u}_{t,\boldsymbol{x}}^*)$ is a pair of an optimal trajectory and optimal control that solve (1). Furthermore, assume that $\boldsymbol{p}_{t,\boldsymbol{x}}\colon [t, T] \to \mathbb{R}^d$ is the solution of the *adjoint equation*

$$\begin{cases} \partial_s \boldsymbol{p}_{t,\boldsymbol{x}}(s) = \nabla_{\boldsymbol{z}} \mathcal{H}\big(s, \boldsymbol{z}_{t,\boldsymbol{x}}^*(s), \boldsymbol{p}_{t,\boldsymbol{x}}(s), \boldsymbol{u}_{t,\boldsymbol{x}}^*(s)\big), \\ \boldsymbol{p}_{t,\boldsymbol{x}}(T) = \nabla_{\boldsymbol{z}} G\big(\boldsymbol{z}_{t,\boldsymbol{x}}^*(T)\big), \end{cases} \quad (14)$$

for $t \leq s \leq T$. Then

$$\boldsymbol{u}_{t,\boldsymbol{x}}^*(s) \in \underset{\boldsymbol{u} \in U}{\arg\max} \ \mathcal{H}\big(s, \boldsymbol{z}_{t,\boldsymbol{x}}^*(s), \boldsymbol{p}_{t,\boldsymbol{x}}(s), \boldsymbol{u}\big) \quad (15)$$

for almost all $s \in [t, T]$. $\diamond$

Theorem 1 provides necessary conditions, and hence does not guarantee that the computed solutions are optimal.

In general, finding $\boldsymbol{u}_{t,\boldsymbol{x}}^*, \boldsymbol{z}_{t,\boldsymbol{x}}^*, \boldsymbol{p}_{t,\boldsymbol{x}}$ that satisfy the PMP is difficult. Simultaneously solving the initial value problem (1) and the terminal value problem (14) gives the system a particularly challenging forward-backward structure [28], [54].

As we show below, the PMP can be applied more readily when the value function Φ is differentiable at $(t, \boldsymbol{x})$. First, in this case, the conditions in Theorem 1 are sufficient [55, Theorem 7.3.9]. [55, Theorems 7.3.10, 7.4.20] provide a similar result with slightly weaker assumptions. Second, as we outline below, the solution of (14) can be obtained from Φ. The following is a standing assumption throughout the paper.

Assumption 1: Assume that (15) admits a unique continuous closed-form solution

$$\boldsymbol{u}^*(s, \boldsymbol{z}, \boldsymbol{p}) = \underset{\boldsymbol{u} \in U}{\arg\max} \ \mathcal{H}\big(s, \boldsymbol{z}, \boldsymbol{p}, \boldsymbol{u}\big) \quad (16)$$

for every $s \in [t, T]$ and $\boldsymbol{z}, \boldsymbol{p} \in \mathbb{R}^d$. $\diamond$

A closed-form solution for the optimal control exists in a wide variety of OC problems [7]–[10], [31]. Importantly, the PMP can also be applied efficiently when (16) does not admit a closed-form solution but can be computed efficiently.

The next theorem states that the value function Φ contains complete information about the optimal control and we can easily recover $\boldsymbol{u}_{t,\boldsymbol{x}}^*$ and $\boldsymbol{p}_{t,\boldsymbol{x}}$ from Φ when Assumption 1 holds.

Theorem 2 (Theorem I.6.2 [53]): Assume that $\boldsymbol{u}_{t,\boldsymbol{x}}^*$ is a right-continuous optimal control and Φ is differentiable at $(s, \boldsymbol{z}_{t,\boldsymbol{x}}^*(s))$ for $t \leq s < T$. Then

$$\boldsymbol{p}_{t,\boldsymbol{x}}(s) = \nabla_{\boldsymbol{z}} \Phi\big(s, \boldsymbol{z}_{t,\boldsymbol{x}}^*(s)\big) \quad (17)$$

solves (14). Also, (15) simplifies to

$$\boldsymbol{u}_{t,\boldsymbol{x}}^*(s) = \boldsymbol{u}^* \big(s, \boldsymbol{z}_{t,\boldsymbol{x}}^*(s), \nabla_{\boldsymbol{z}} \Phi\big(s, \boldsymbol{z}_{t,\boldsymbol{x}}^*(s)\big)\big) \quad (18)$$

for almost all $s \in [t, T]$. $\diamond$

Note that enforcing or computationally verifying the differentiability condition is virtually impossible. However, in many cases including our applications, the value function is expected

to be differentiable almost everywhere. Even if Φ is not differentiable at $(t, \boldsymbol{x})$ and the optimal control is not unique, $\boldsymbol{p}_{t,\boldsymbol{x}}$ can be recovered from the super differential $\partial_{\boldsymbol{x}}^{+}\Phi$ [55, Theorem 7.3.10, 7.4.20].

Theorem 2 characterizes optimal controls in a *feedback form* (18). This means that no further optimization is necessary to find the optimal controls when the value function is known. Feedback form representations are valuable in real-world applications. If $\nabla\Phi$ can be quickly calculated, optimal controls are readily available at any point in space and time. As such, the feedback form avoids recomputing the optimal controls at new points in scenarios when sudden changes to the initial data or the system's state occur.

We can also use Assumption 1 to simplify the computation of the trajectories. Using the *envelope formula* [56, Sec. 3.1, Theorem 1], we see that

$$\begin{aligned} \nabla_{\boldsymbol{z}}\mathcal{H}\big(t,\boldsymbol{z},\boldsymbol{p},\boldsymbol{u}^{*}(t,\boldsymbol{z},\boldsymbol{p})\big) &= \nabla_{\boldsymbol{z}}H(t,\boldsymbol{z},\boldsymbol{p}) \\ \nabla_{\boldsymbol{p}}\mathcal{H}\big(t,\boldsymbol{z},\boldsymbol{p},\boldsymbol{u}^{*}(t,\boldsymbol{z},\boldsymbol{p})\big) &= \nabla_{\boldsymbol{p}}H(t,\boldsymbol{z},\boldsymbol{p}). \end{aligned} \quad (19)$$

Hence, assuming the value function is known, we can express the optimal trajectory as

$$\begin{cases} \partial_{s}\boldsymbol{z}_{t,\boldsymbol{x}}^{*}(s) = -\nabla_{\boldsymbol{p}}H\big(s,\boldsymbol{z}_{t,\boldsymbol{x}}^{*}(s),\nabla_{\boldsymbol{z}}\Phi(s,\boldsymbol{z}_{t,\boldsymbol{x}}^{*}(s))\big), \\ \boldsymbol{z}_{t,\boldsymbol{x}}^{*}(t) = \boldsymbol{x}, \end{cases} \quad (20)$$

for $s \in (t, T]$. These dynamics do not explicitly contain the control, which reduces the problem to the state variables only. Recall the optimal control can be computed via (18).

The above derivation outlines how to obtain the optimal control and trajectory from the value function under some smoothness assumptions. Once the value function Φ is known, this procedure can be applied for any initial data and also adapt the trajectory when the system is perturbed. Therefore, if Φ is computed, we have a *global* solution method. The key issue that we address next is the computation of Φ.

D. Hamilton-Jacobi-Bellman PDE

In the previous section, we reviewed that the solution to the the OC problem (3) for all initial data can be inferred from the value function Φ. In our approach, we exploit the fact that the value function satisfies the Hamilton-Jacobi-Bellman (HJB) PDE to help train our NN approximation of Φ.

Theorem 3 (Theorems I.5.1, I.6.1 [53]): Assume that the value function $\Phi \in C^{1}([0,T] \times \mathbb{R}^{d})$. Then Φ satisfies the HJB equations (also called the *dynamic programming* equations)

$$-\partial_{s}\Phi(s,\boldsymbol{z}) + H\big(s,\boldsymbol{z},\nabla_{\boldsymbol{z}}\Phi(s,\boldsymbol{z})\big) = 0, \quad \Phi(T,\boldsymbol{z}) = G(\boldsymbol{z}) \quad (21)$$

for all $(s,\boldsymbol{z}) \in [t,T) \times \mathbb{R}^{d}$. Conversely, assume that $\Psi \in C^{1}([0,T] \times \mathbb{R}^{d})$ is a solution of (21) and $\boldsymbol{u}_{t,\boldsymbol{x}}^{*}$ is such that

$$\boldsymbol{u}_{t,\boldsymbol{x}}^{*}(s) \in \arg\max_{\boldsymbol{u}\in U} \mathcal{H}\big(s,\boldsymbol{z}_{t,\boldsymbol{x}}^{*}(s),\nabla_{\boldsymbol{z}}\Psi(s,\boldsymbol{z}_{t,\boldsymbol{x}}^{*}(s)),\boldsymbol{u}\big) \quad (22)$$

for almost all $s \in [t,T]$. Then $\Psi = \Phi$, and $\boldsymbol{u}_{t,\boldsymbol{x}}^{*}$ is an optimal control. $\diamond$

The differentiability of Φ can be relaxed to differentiability almost everywhere in the framework of viscosity solutions [53, Chap. II].

The HJB PDE (21) admits robust existence, uniqueness, and stability theory in the framework of viscosity solutions because (21) is a convex constraint on Φ [57]. Well-established numerical methods, such as ENO/WENO [14], benefit from convergence guarantees when solving (21). However, these methods rely on grids and therefore are affected by the CoD. Mitigating this limitation motivates our NN approach.

We note that the PMP is the *method of characteristics* [56, Sec. 3.2] for the HJB equation (21). To be precise, we can compute Φ along the trajectory $\boldsymbol{z}_{t,\boldsymbol{x}}$ from (20) by solving

$$\begin{cases} \partial_{s}\phi_{t,\boldsymbol{x}}(s) = H\big(s,\boldsymbol{z}_{t,\boldsymbol{x}}^{*}(s),\boldsymbol{p}_{t,\boldsymbol{x}}(s)\big) \\ \qquad\qquad -\boldsymbol{p}_{t,\boldsymbol{x}}(s)\cdot\nabla_{\boldsymbol{p}}H\big(s,\boldsymbol{z}_{t,\boldsymbol{x}}^{*}(s),\boldsymbol{p}_{t,\boldsymbol{x}}(s)\big) \\ \phi_{t,\boldsymbol{x}}(T) = G(\boldsymbol{z}_{t,\boldsymbol{x}}^{*}(T)). \end{cases}$$

We then have that $\phi_{t,\boldsymbol{x}}(s) = \Phi(s,\boldsymbol{z}_{t,\boldsymbol{x}}^{*}(s))$.

IV. NEURAL NETWORK APPROACH

Our approach seeks to minimize (2) subject to (1) for initial states sampled from a probability distribution in $\mathbb{R}^{d}$ whose density we denote by ρ. Hence, it aims at solving the problem for all states along the optimal trajectories originating from those points. Since the optimal trajectories given by the PMP are characteristics of the HJB equation, our method blends these two approaches. To enable high-dimensional scalability, our method parameterizes the value function with an NN and computes the controls using (18) and (20). The NN is trained in an unsupervised fashion by minimizing the sum of the expected cost that results from the trajectories and penalty terms that enforce the HJB equations along the trajectories and at the final-time.

A. Main Formulation

We consider the semi-global version of the control problem and seek an approximately optimal control for initial states $\boldsymbol{x} \sim \rho$. We do so by approximating the value function $\Phi(\cdot)$ with an NN with parameters $\boldsymbol{\theta}$, which we denote by $\Phi(\cdot;\boldsymbol{\theta})$. Thus, we can write the controls in feedback form and the loss in terms of the parameters. In particular, we solve

$$\min_{\boldsymbol{\theta}} \ \mathbb{E}_{\boldsymbol{x}\sim\rho} \ \big\{\ell_{\boldsymbol{x}}(T) + G(\boldsymbol{z}_{0,\boldsymbol{x}}(T)) + \beta_{1}c_{\mathrm{HJt},\boldsymbol{x}}(T) \\ + \beta_{2}c_{\mathrm{HJfin},\boldsymbol{x}} + \beta_{3}c_{\mathrm{HJgrad},\boldsymbol{x}}\big\}, \quad (23)$$

subject to

$$\partial_{s}\begin{pmatrix} \boldsymbol{z}_{0,\boldsymbol{x}}(s) \\ \ell_{\boldsymbol{x}}(s) \\ c_{\mathrm{HJt},\boldsymbol{x}}(s) \end{pmatrix} = \begin{pmatrix} -\nabla_{\boldsymbol{p}}H(s,\boldsymbol{z}_{0,\boldsymbol{x}}(s),\nabla_{\boldsymbol{z}}\Phi(s,\boldsymbol{z}_{0,\boldsymbol{x}}(s);\boldsymbol{\theta}))) \\ L_{\boldsymbol{x}}(s) \\ P_{\mathrm{HJt},\boldsymbol{x}}(s) \end{pmatrix}, \quad (24)$$

where $\ell_{\boldsymbol{x}}(0) = c_{\mathrm{HJt},\boldsymbol{x}}(0) = 0$ and $s \in [0,T]$. Here, ℓ accumulates the Lagrangian cost L along the trajectories, the terms $c_{\mathrm{HJt},\boldsymbol{x}}, c_{\mathrm{HJfin},\boldsymbol{x}}, c_{\mathrm{HJgrad},\boldsymbol{x}}, P_{\mathrm{HJt},\boldsymbol{x}}$ penalize violations of the HJB, and the scalar penalty weights $\beta_{1}, \beta_{2}, \beta_{3} > 0$ are assumed to be fixed. The remainder of this section defines and discusses these terms in more detail.

TABLE I: Variables and hyperparameters inherent to the problem itself (shared for NN and baseline) and the hyperparameters tuned for the NN approach. All α values are determined relative to the α-less E term in the problem definition. The β hyperparameters are tuned relative to the α values.

| | Problem Definition | | | | | NN-specific Hyperparameters | | | | | |
	n # agents	d dim.	α_1 on G	α_2 on Q	α_3 on W	m width	β_1 on HJt	β_2 on HJfin	β_3 on HJgrad	n_t training	n_t validation	NN # Params
Corridor	2	4	100	10^4	300	32	0.02	0.02	0.02	20	50	1,311
Swap 2 [19]	2	4	300	10^6	10^5	16	1	1	3	20	50	415
Swap 12 [19]	12	24	300	-	10^5	32	5	2	5	20	50	2,196
Swarm [1]	50	150	900	10^7	25000	512	2	1	3	26	80	342,654
Quadcopter [20]	1	12	5000	-	-	128	0.1	0	0	26	50	18,576

TABLE II: NN Statistics. Training times are approximate from running on a shared NVIDIA Quadro RTX 8000 GPU. Deployment times are from running on a single 2.6 GHz Intel(R) Xeon(R) CPU E5-4627 v3 core (Sec. V-F).

| | Training | | | | Deploy Time (ms) | |
	# Iters	Batch Size	$\frac{\text{ms}}{\text{Iter}}$	Time (min)	NN Step	Baseline Estimate
Corridor	1800	1024	320	10	4.4	2899
Swap 2 [19]	4000	1024	560	37	4.5	2571
Swap 12 [19]	4000	2048	260	17	3.6	1730
Swarm [1]	6000	1024	570	57	9.6	4026
Quadcopter [20]	6000	1024	720	72	5.2	3110

The term $\ell(T)$ corresponds to the time integral in (2). To compute L at a given time, we use (4) and (19) and reformulate the Lagrangian in terms of the NN parameters $\boldsymbol{\theta}$ as

$$L_{\boldsymbol{x}}(\boldsymbol{s}) = -H\big(\boldsymbol{s}, \boldsymbol{z}_{0,\boldsymbol{x}}(\boldsymbol{s}), \nabla_{\boldsymbol{z}}\Phi(\boldsymbol{s}, \boldsymbol{z}_{0,\boldsymbol{x}}(\boldsymbol{s}); \boldsymbol{\theta})\big) \quad (25)$$
$$+ \nabla_{\boldsymbol{z}}\Phi\big(\boldsymbol{s}, \boldsymbol{z}_{0,\boldsymbol{x}}(\boldsymbol{s}); \boldsymbol{\theta}\big) \cdot \nabla_{\boldsymbol{p}}H\big(\boldsymbol{s}, \boldsymbol{z}_{0,\boldsymbol{x}}(\boldsymbol{s}), \nabla_{\boldsymbol{z}}\Phi(\boldsymbol{s}, \boldsymbol{z}_{0,\boldsymbol{x}}(\boldsymbol{s}); \boldsymbol{\theta})\big).$$

We use HJB penalty terms $c_{\text{HJt},\boldsymbol{x}}$, $c_{\text{HJfin},\boldsymbol{x}}$, and $c_{\text{HJgrad},\boldsymbol{x}}$ derived from the HJB PDE (21) as follows:

$$P_{\text{HJt},\boldsymbol{x}}(\boldsymbol{s}) =$$
$$\big|\partial_s\Phi(\boldsymbol{s}, \boldsymbol{z}_{0,\boldsymbol{x}}(\boldsymbol{s}); \boldsymbol{\theta}) - H\big(\boldsymbol{s}, \boldsymbol{z}_{0,\boldsymbol{x}}(\boldsymbol{s}), \nabla_{\boldsymbol{z}}\Phi(\boldsymbol{s}, \boldsymbol{z}_{0,\boldsymbol{x}}(\boldsymbol{s}); \boldsymbol{\theta}))\big|$$
$$c_{\text{HJfin},\boldsymbol{x}} = |\Phi(T, \boldsymbol{z}_{0,\boldsymbol{x}}(T); \boldsymbol{\theta}) - G(\boldsymbol{z}_{0,\boldsymbol{x}}(T))|$$
$$c_{\text{HJgrad},\boldsymbol{x}} = |\nabla_{\boldsymbol{z}}\Phi(T, \boldsymbol{z}_{0,\boldsymbol{x}}(T); \boldsymbol{\theta}) - \nabla_{\boldsymbol{z}}G(\boldsymbol{z}_{0,\boldsymbol{x}}(T))|. \quad (26)$$

The HJ$_{\text{t}}$ penalizer arises from the first equation in (21), whereas HJ$_{\text{fin}}$ and HJ$_{\text{grad}}$ are direct results of the final-time condition in (21) and its gradient, respectively. Penalizers prove helpful in problems similar to (23) [15]–[17], [58]. These penalizers improve the training convergence (Sec. V-B.3) without altering the solution of (23). The $P_{\text{HJt},\boldsymbol{x}}$ penalizer is accumulated along the trajectory similar to L. The scalar terms $\beta_1, \beta_2, \beta_3$ weight the importance of each HJB penalizer and are hyperparameters of the NN (Sec. IV-D, Sec. V-C.4).

B. Value Function Approximation

To enable scalability to high dimensions, we approximate the value function Φ with an NN. While our formulation supports a wide range of NNs, we design a specific model that enables efficient computation.

We parameterize the value function as

$$\Phi(\boldsymbol{s}; \boldsymbol{\theta}) = \boldsymbol{w}^\top N(\boldsymbol{s}; \boldsymbol{\theta}_N) + \frac{1}{2}\boldsymbol{s}^\top(\boldsymbol{A}^\top\boldsymbol{A})\boldsymbol{s} + \boldsymbol{b}^\top\boldsymbol{s} + c, \quad (27)$$
$$\text{where} \quad \boldsymbol{\theta} = (\boldsymbol{w}, \boldsymbol{\theta}_N, \boldsymbol{A}, \boldsymbol{b}, c).$$

Here, $\boldsymbol{s} = (\boldsymbol{x}, t) \in \mathbb{R}^{d+1}$ are the inputs corresponding to space-time, $N(\boldsymbol{s}; \boldsymbol{\theta}_N) \colon \mathbb{R}^{d+1} \to \mathbb{R}^m$ is an NN, and $\boldsymbol{\theta}$ contains the trainable weights: $\boldsymbol{w} \in \mathbb{R}^m$, $\boldsymbol{\theta}_N \in \mathbb{R}^p$, $\boldsymbol{A} \in \mathbb{R}^{\gamma \times (d+1)}$, $\boldsymbol{b} \in \mathbb{R}^{d+1}$, $c \in \mathbb{R}$, where rank $\gamma = \min(10, d)$ limits the number of parameters in $\boldsymbol{A}^\top\boldsymbol{A}$. Here, $\boldsymbol{A}$, $\boldsymbol{b}$, and c model quadratic potentials, i.e., linear dynamics; N models nonlinear dynamics.

In our experiments, for N, we use a simple two-layer residual neural network (ResNet) [59]

$$\boldsymbol{a}_0 = \sigma(\boldsymbol{K}_0\boldsymbol{s} + \boldsymbol{b}_0)$$
$$N(\boldsymbol{s}; \boldsymbol{\theta}_N) = \boldsymbol{a}_0 + \sigma(\boldsymbol{K}_1\boldsymbol{a}_0 + \boldsymbol{b}_1), \quad (28)$$

for $\boldsymbol{\theta}_N = (\boldsymbol{K}_0, \boldsymbol{K}_1, \boldsymbol{b}_0, \boldsymbol{b}_1)$ where $\boldsymbol{K}_0 \in \mathbb{R}^{m \times (d+1)}$, $\boldsymbol{K}_1 \in \mathbb{R}^{m \times m}$, and $\boldsymbol{b}_0, \boldsymbol{b}_1 \in \mathbb{R}^m$. We use the element-wise nonlinearity $\sigma(\boldsymbol{x}) = \log(\exp(\boldsymbol{x}) + \exp(-\boldsymbol{x}))$, which is the antiderivative of the hyperbolic tangent, i.e., $\sigma'(\boldsymbol{x}) = \tanh(\boldsymbol{x})$ [15], [17].

C. Numerical Implementation

We solve the ODE-constrained optimization problem (23) using the discretize-then-optimize approach [60], [61], in which we define a discretization of the ODE, then optimize on that discretization. The forward pass of the model uses a Runge-Kutta (RK) 4 integrator with n_t time steps to eliminate the constraints (24). The objective function is then computed, and automatic differentiation [62] calculates the gradient of the objective function with respect to $\boldsymbol{\theta}$. We use the ADAM optimizer [63], a stochastic subgradient method with momentum, to update the parameters $\boldsymbol{\theta}$. We iterate this process a selected maximum number of times. For the learning rate (step size) provided to ADAM, we follow a piece-wise constant decay schedule. For instance, in the experiment in Fig. 2, we divide the learning rate by 10 every 800 iterations.

To produce an NN that generalizes to the state-space, we must define initial points in a manner to promote model generalizability. We assume the initial points are drawn from a distribution with density ρ. We train the NN on one batch at a time of independent and identically distributed samples from the distribution. After training a number of iterations on that batch, we resample the distribution to define a new batch and train additional iterations on that batch. We repeat this process until we hit the maximum number of iterations. We commonly choose batches of 1024 or 2048 samples which are re-sampled every 25–100 iterations. We found no noticeable empirical difference in solution quality across those ranges. Through this process, the model uses few data points at each iteration, but does not overfit to a specific set of data points.

(a) The baseline solution for initial state x_0.

(b) For initial state x_0 (depicted), the NN learns a similar solution as the baseline. The NN approach is solved for multiple initial states.
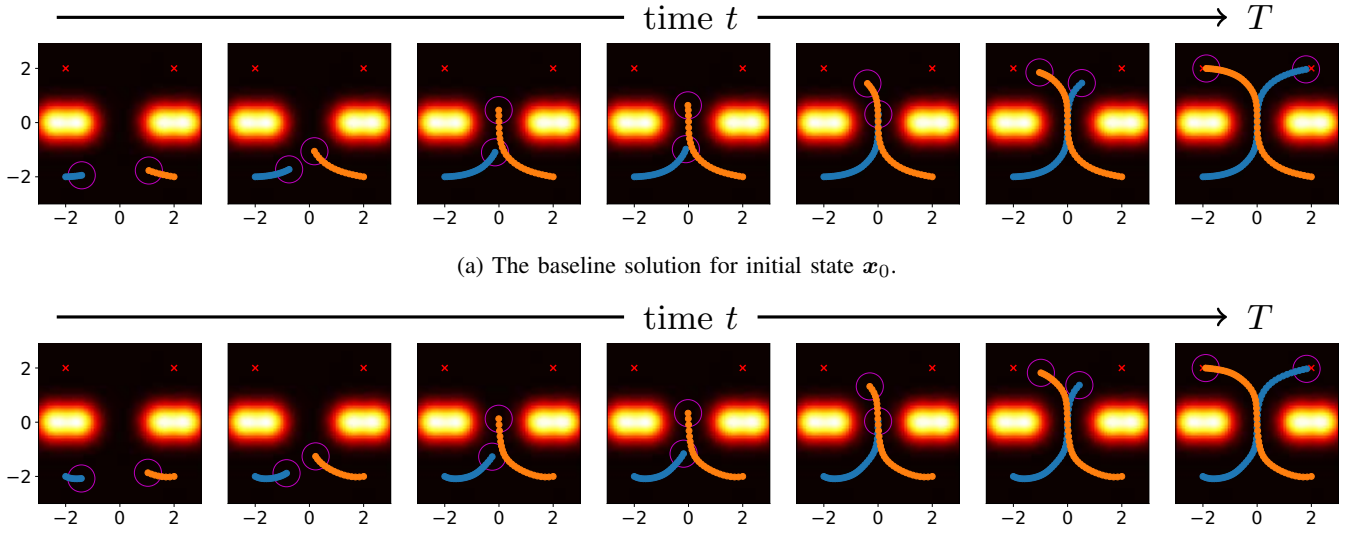
Fig. 1: Solutions for the two-agent corridor problem where two agents (orange and blue) pass in between two smooth hills. Taking the terrain into account, the agents seek shortest paths from the initial joint-state x_0 to target y (marked with red crosses) while avoiding collision with each other's space bubble (indicated by circles with radius r).

TABLE III: Comparison of solution values for the two-agent corridor problem and single instance x_0 shown in Fig. 1.

Method	$\ell + G$	ℓ	G
Baseline	61.33	61.02	0.31
NN	62.19	61.98	0.21

D. Hyperparameters

In contrast to the model parameters θ learned from the data, NN hyperparameters are values selected *a priori* to training. These include the number of time steps n_t, the ResNet width m, ResNet depth (the number of layers, tuned to equal 2), and the multipliers $\beta_1, \beta_2, \beta_3$. Additionally, each OC problem has defined $\alpha_1, \alpha_2, \alpha_3$, which both the baseline and NN use; changing these values alters the problem (Table I). For reproducibility, we include all hyperparameters and settings with a publicly available Python implementation at `https://github.com/donken/NeuralOC`. Training on a single NVIDIA Quadro RTX 8000 GPU requires between 10 and 72 minutes for the considered OC problems (Table II).

V. NUMERICAL EXPERIMENTS

We solve and analyze five OC problems and compare the NN against a baseline method described in Sec. V-A. In Sec. V-B to V-D, we present four centrally controlled multi-agent examples with dimensionality ranging from 4 to 150. In Sec. V-E, we consider a quadcopter experiment to demonstrate the NN's ability to solve problems with complicated dynamics.

A. Baseline: Optimization for a Single Initial State

For comparison with the NN approach, we provide a local solution method that solves the OC problem for a fixed initial
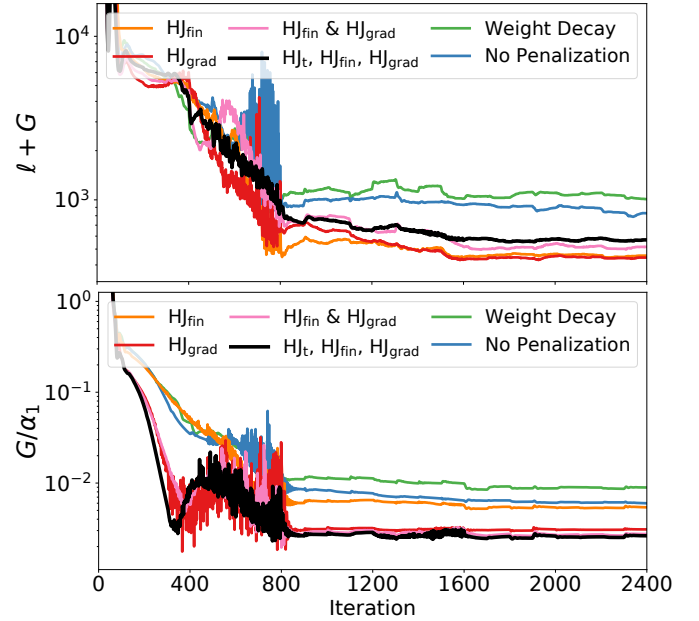

Fig. 2: For the corridor problem (Sec. V-B), we train the same model architecture six times using different combinations of the penalty terms. Using all three HJB penalizers leads to quick convergence and a low G value. Each curve is the average of three training instances.

state x_0. We consider the baseline approach's solution as the ground truth optimal solution and compute the *suboptimality* of the semi-global NN approach's solution evaluated for the initial state relative to the baseline's solution.

For the baseline, we obtain an optimization problem by applying forward Euler to the state equation and a midpoint
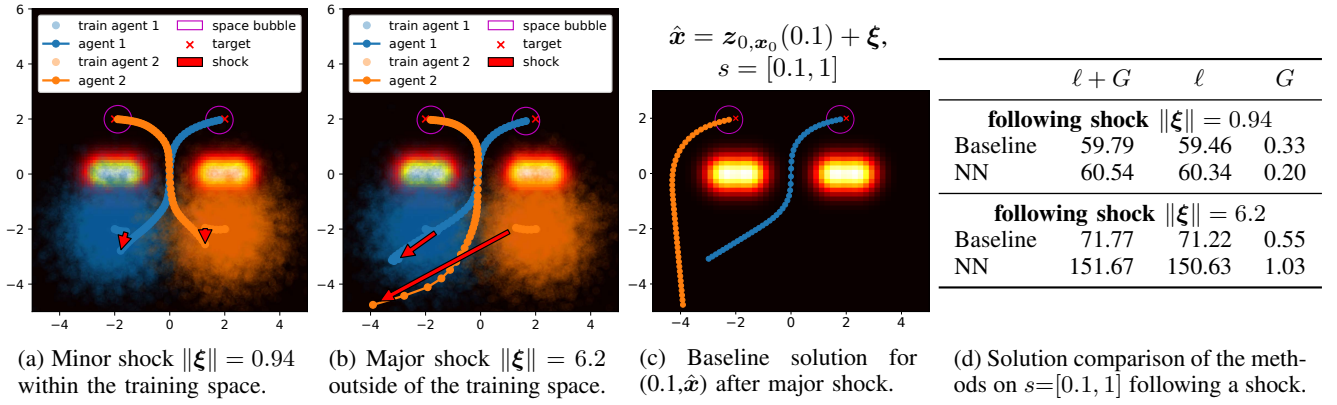
$$\hat{\boldsymbol{x}} = \boldsymbol{z}_{0,\boldsymbol{x}_0}(0.1) + \boldsymbol{\xi},$$
$$s = [0.1, 1]$$

	$\ell + G$	ℓ	G
following shock $\|\boldsymbol{\xi}\| = 0.94$			
Baseline	59.79	59.46	0.33
NN	60.54	60.34	0.20
following shock $\|\boldsymbol{\xi}\| = 6.2$			
Baseline	71.77	71.22	0.55
NN	151.67	150.63	1.03

(a) Minor shock $\|\boldsymbol{\xi}\| = 0.94$ within the training space.　(b) Major shock $\|\boldsymbol{\xi}\| = 6.2$ outside of the training space.　(c) Baseline solution for $(0.1, \hat{\boldsymbol{x}})$ after major shock.　(d) Solution comparison of the methods on $s=[0.1, 1]$ following a shock.

Fig. 3: The NN handles a shock $\boldsymbol{\xi}$ at time $s=0.1$ (depicted with red arrows) along the trajectory for the depicted corridor problem (Sec. V-B). The initial states used during training are depicted as blue and orange point clouds. It can be seen that the major shock causes the system to leave the state-space used during training.

(a) Minor shock $\|\boldsymbol{\xi}\| = 0.94$ within the training space.　(b) Major shock $\|\boldsymbol{\xi}\| = 6.2$ outside of the training space.
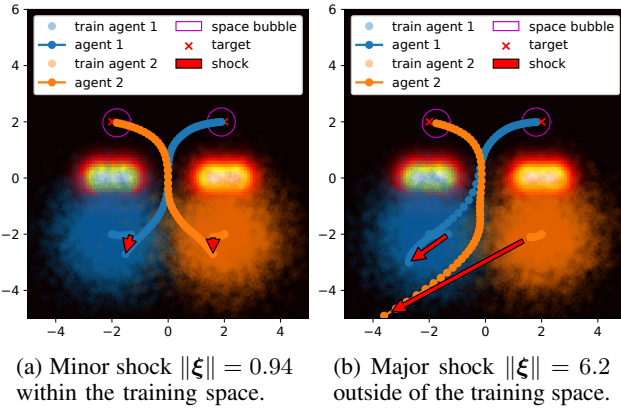
Fig. 4: We solve the corridor problem with an NN trained without HJB penalizers or weight decay. Comparable to Fig. 3, we see that the penalizers do not alter the solution.

rule to the integrals, i.e., the "direct transcription method" [11],

$$\min_{\{\boldsymbol{u}^{(k)}\}} \quad G\left(\boldsymbol{z}_{n_t}\right) + h \sum_{k=0}^{n_t-1} L\left(s_k, \boldsymbol{z}_k, \boldsymbol{u}_k\right) \tag{29}$$
$$\text{s.t.} \quad \boldsymbol{z}_{k+1} = \boldsymbol{z}_k + h\, f\left(s_k, \boldsymbol{z}_k, \boldsymbol{u}_k\right), \ \boldsymbol{z}_0 = \boldsymbol{x}_0,$$

where $h=T/n_t$. Here, we use $\boldsymbol{z}_k$ to denote $\boldsymbol{z}_{0,\boldsymbol{x}_0}(s_k)$, where time point $s_k = hk$. We use $T=1$ and $n_t=50$ and solve (29) using ADAM with initialization of the controls set as straight paths from $\boldsymbol{x}_0$ to $\boldsymbol{y}$ with small added Gaussian noise.

We arrived at these training decisions empirically. First, when solving (29) in our experiments, ADAM finds slightly more optimal solutions ($1-2\%$ more optimal) in practice than L-BFGS. Second, the initialization of the controls substantially influences the solution. As a particular example, the baseline solution depicted in Fig. 3c learns to send agent 2 around the left side of the left obstacle, resulting in the lowest value of the objective function. If initialized with controls that pass through the right of that obstacle or through the corridor, the baseline struggles to learn this optimal trajectory. As a response, we initialize the controls uniformly that lead to a straight path from $\boldsymbol{x}_0$ to $\boldsymbol{y}$. Third, we add random Gaussian noise to the initialization because doing so empirically helps avoid local minima and overall achieves better results.

B. Two-Agent Corridor Example

We design a $d=4$-dimensional problem in which two agents attempt to reach fixed targets on the other side of two hills (Fig. 1). We design the hills in such a manner that one agent must pass through the corridor between the two hills while the other agent waits. For this example, the hills use a smooth terrain, and we assess the resilience of the control to shocks.

1) Set-up: Suppose two homogeneous agents with safety radius $r=0.5$ start at $x^{(1)}=[-2,-2]^\top$ and $x^{(2)}=[2,-2]^\top$ with respective targets $y^{(1)}=[2,2]^\top$ and $y^{(2)}=[-2,2]^\top$. Thus, the initial and target joint-states are $\boldsymbol{x}_0=[-2,-2,2,-2]^\top$ and $\boldsymbol{y}=[2,2,-2,2]^\top$. We sample from ρ, which is a Gaussian centered at $\boldsymbol{x}_0$ with an identity covariance. These sampled initial positions form the training set $\boldsymbol{X}$.

The running costs depend on the spatio-temporal cost function Q_i. Throughout, obstacles are defined using the Gaussian density function with mean $\boldsymbol{\mu} \in \mathbb{R}^q$ and covariance $\boldsymbol{\Sigma} \in \mathbb{R}^{q \times q}$

$$\eta(z^{(i)}\,;\,\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{\exp\left(-\frac{1}{2}(z^{(i)}-\boldsymbol{\mu})\boldsymbol{\Sigma}^{-1}(z^{(i)}-\boldsymbol{\mu})\right)}{\sqrt{(2\pi)^d \det \boldsymbol{\Sigma}}}.$$

In this experiment, we define obstacles as

$$Q_i\left(z^{(i)}\right) = \eta\left(z^{(i)}\,;\, \begin{bmatrix} -2.5 \\ 0 \end{bmatrix}, 0.2\boldsymbol{I}\right) + \eta\left(z^{(i)}\,;\, \begin{bmatrix} 2.5 \\ 0 \end{bmatrix}, 0.2\boldsymbol{I}\right)$$
$$+ \eta\left(z^{(i)}\,;\, \begin{bmatrix} -1.5 \\ 0 \end{bmatrix}, 0.2\boldsymbol{I}\right) + \eta\left(z^{(i)}\,;\, \begin{bmatrix} 1.5 \\ 0.0 \end{bmatrix}, 0.2\boldsymbol{I}\right).$$

The energy terms are given by

$$E_i\left(u^{(i)}\right) = \frac{1}{2}\|u^{(i)}\|^2, \tag{30}$$

and the dynamics are given by $f(s, \boldsymbol{z}, \boldsymbol{u}) = \boldsymbol{u}$.

We compute the Hamiltonian (4) as

$$H(s, \boldsymbol{z}, \boldsymbol{p}) = \sup_{\boldsymbol{u} \in U}\left\{ -\boldsymbol{p}^\top \boldsymbol{u} - L\left(s, \boldsymbol{z}, \boldsymbol{u}\right) \right\}$$
$$= \sup_{\boldsymbol{u} \in U}\left\{ -\boldsymbol{p}^\top \boldsymbol{u} - E(\boldsymbol{u}) - \alpha_2 Q(\boldsymbol{z}) - \alpha_3 W(\boldsymbol{z}) \right\}. \tag{31}$$

We then can solve for the first-order necessary condition

$$0 = -\boldsymbol{p} - \nabla_{\boldsymbol{u}} E(\boldsymbol{u})$$

$$\Rightarrow \quad \boldsymbol{p} = -\nabla_{\boldsymbol{u}} \left(\sum_{i=1}^{n} \frac{1}{2} \|u^{(i)}\|^2 \right) = -\boldsymbol{u} \tag{32}$$

Using the closed-form solution for the controls (32), we rewrite the Hamiltonian as

$$H(s, \boldsymbol{z}, \boldsymbol{p}) = \|\boldsymbol{p}\|^2 - \frac{1}{2}\|\boldsymbol{p}\|^2 - \alpha_2 Q(\boldsymbol{z}) - \alpha_3 W(\boldsymbol{z})$$
$$= \frac{1}{2}\|\boldsymbol{p}\|^2 - \alpha_2 Q(\boldsymbol{z}) - \alpha_3 W(\boldsymbol{z}), \tag{33}$$

where the characteristics are given by

$$\partial_s \boldsymbol{z}_{t,\boldsymbol{x}}(s) = -\nabla_{\boldsymbol{p}} H\big(s, \boldsymbol{z}_{t,\boldsymbol{x}}(s), \boldsymbol{p}_{t,\boldsymbol{x}}(s)\big) = -\boldsymbol{p}_{t,\boldsymbol{x}}(s). \tag{34}$$

2) Results: The baseline and the NN learn to wait for one agent to pass through the corridor first, followed by the second agent (Fig. 1). The NN performs marginally worse in L values (Table III), which can be seen in the early stages of the trajectories of agent 1 (Fig. 1b). The NN achieves a slightly better G value than the baseline. Although we solve the NN by optimizing the expectation value of a set of points in the region, the NN achieves a near-optimal solution for $\boldsymbol{x}_0$.

3) Effect of the HJB Penalizers: We experimentally assess the effectiveness of the penalizers c_{HJt}, c_{HJfin}, c_{HJgrad} in (23). To this end, we define six models (various combinations of the three HJB penalizers and one model with weight decay) and train three instances of each on the corridor problem. Using the HJB penalizers results in a quicker model convergence on a hold-out validation set (Fig. 2).

HJ$_\mathbf{t}$: We enforce the PDE (21) describing the time derivative of Φ along the trajectories. Including this penalizer improves regularity and reduces the necessary number of time steps when solving the dynamics [15]–[17], [64].

HJ$_{\text{fin}}$: We enforce the final-time condition of the PDE (21). The inclusion of this penalizer helps the network achieve the target [15]. Experimentally, using HJ$_{\text{fin}}$ correlates with a slightly lower G value (Fig. 2).

HJ$_{\text{grad}}$: We enforce the transversality condition $\nabla_{\boldsymbol{z}} \Phi(T, \boldsymbol{z}(T)) = \nabla_{\boldsymbol{z}} G(\boldsymbol{z}(T))$ $\forall \boldsymbol{z}$, a consequence of the final-time HJB condition (21). Numerically, all conditions are enforced on a finite sample set. Therefore, higher-order regularization may help the generalization; i.e., achieving a better match of $\Phi(T, \cdot)$ and G for samples not used during training (the hold-out validation set). We observe the latter experimentally; HJ$_{\text{grad}}$ impacts G more than HJ$_{\text{fin}}$ (Fig. 2). Nakamura-Zimmerer *et al.* [29] similarly enforce $\nabla \Phi$ values.

4) Shocks: We use this experiment to demonstrate how our approach is robust to shocks to the system's state (Fig. 3). Consider solving the control problem for $s \in [0, T]$ as always. Then for $T = 1$, we consider a shock $\boldsymbol{\xi}$ (implemented as a random shift) to the system at time $s = 0.1$. Our method is designed to handle minor shocks that stay within the space of trajectories of the initial distribution about $\boldsymbol{x}_0$. Our model computes a trajectory to $\boldsymbol{y}$ for many initial points. Therefore, for point $\widetilde{\boldsymbol{x}} \in \boldsymbol{X}$, the model provides dynamics $f(s, \boldsymbol{z}_{\widetilde{\boldsymbol{x}}}(s), \boldsymbol{u}_{\widetilde{\boldsymbol{x}}}(s))$ before the shock. After the shock, the state picks up the trajectory of some other point $\hat{\boldsymbol{x}} \in \boldsymbol{X}$ and
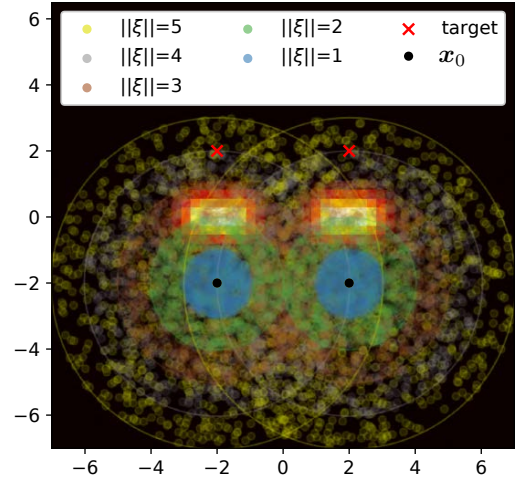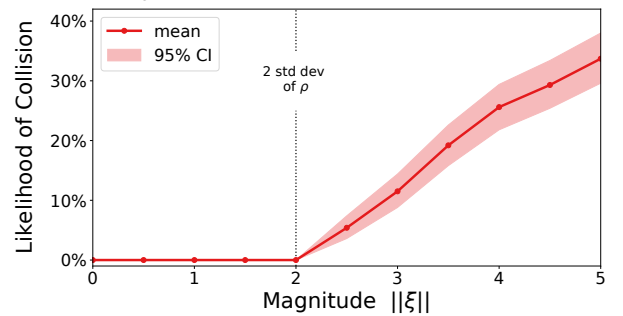

(a) The initial points $\boldsymbol{x}_0 + \boldsymbol{\xi}$ for the corridor problem sampled from the hyperspheres of radius $\|\boldsymbol{\xi}\|$.
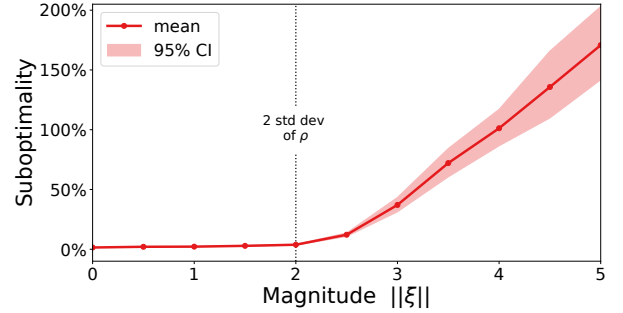
(b) The mean suboptimality of the NN's solution $\ell + G$, where the baseline solution for each initial point is considered optimal.
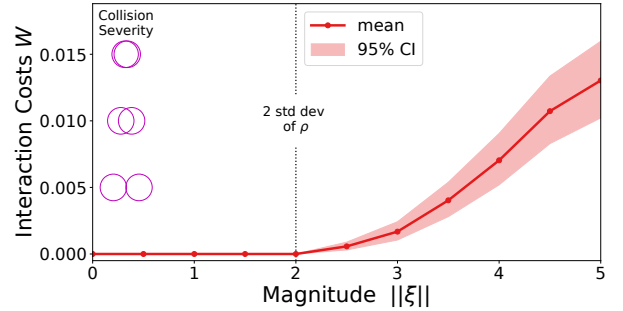
(c) NN interaction costs with comparable example collision severity of two circular agents.

(d) For initial points at each magnitude, we present the percentage of those resulting in a collision of any severity when run with the NN.

Fig. 5: We compare one NN model with 10,001 baseline models for 1000 initial points $(0, \boldsymbol{x}_0 + \boldsymbol{\xi})$ at each magnitude $\|\boldsymbol{\xi}\|$. Confidence intervals are computed via 10,000 sub-samplings of size 500 from each set of 1000 points.
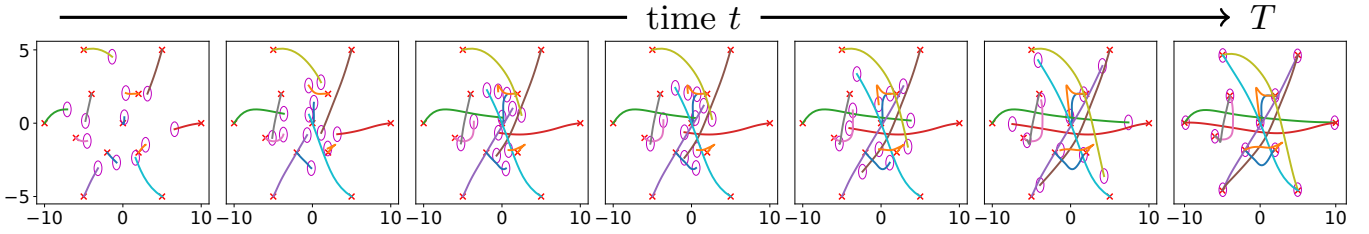
Fig. 6: Numerical results of the 12-agent swap experiment (Sec. V-C.2). The agents' targets are indicated by red crosses, and the space bubble or safety region around each agent is depicted with a circle. The agents aim to pairwise exchange their positions while avoiding each other and minimizing the length of their trajectories.
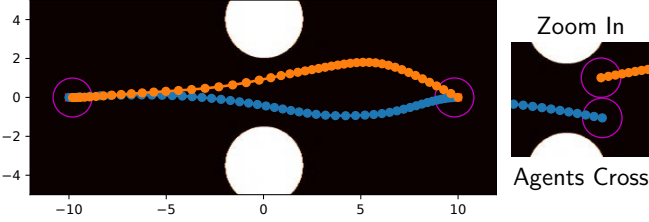

Fig. 7: Numerical results of swap experiment with hard-boundary obstacles (Sec. V-C.1). The agents seek to exchange their positions while keeping a safe distance (indicated by circle) and avoiding the obstacles (white circles). The close-up on the right shows agents at the time of minimal distance.

follows that trajectory to $\boldsymbol{y}$ (Fig. 3a). In this scenario, the total trajectory contains two portions: before the shock and after the shock. That is,

$$\boldsymbol{z}_{0,\widetilde{\boldsymbol{x}}}(0.1) = \int_{0}^{0.1} f\big(s, \boldsymbol{z}_{0,\widetilde{\boldsymbol{x}}}(s), \boldsymbol{u}_{0,\widetilde{\boldsymbol{x}}}(s)\big)\,\mathrm{d}s, \quad \text{and}$$

$$\boldsymbol{z}_{0,\hat{\boldsymbol{x}}}(1) = \int_{0.1}^{1} f\big(s, \boldsymbol{z}_{0,\hat{\boldsymbol{x}}}(s), \boldsymbol{u}_{0,\hat{\boldsymbol{x}}}(s)\big)\,\mathrm{d}s, \quad \text{where}$$

$$\boldsymbol{z}_{0,\hat{\boldsymbol{x}}}(0.1) = \boldsymbol{z}_{0,\widetilde{\boldsymbol{x}}}(0.1) + \boldsymbol{\xi},$$

respectively. We view a minor shock then as moving from one trajectory to another (Fig. 3a). The NN and baseline achieve similar results for the problem along $s=[0.1, 1]$ (Fig. 3d).

Interestingly, our model extends outside the training region (Fig. 3b). Although the vast majority of NNs cannot extrapolate, our NN still solves the control problem after a major shock, demonstrating some extrapolation capabilities. We note that the NN solves the original problem for $\boldsymbol{x}_0$ to near optimality. After a large shock, the NN still drives the agents to their targets, although suboptimally. In our example, we compare the NN's solution (Fig. 3b) with the baseline for $s=[0.1, 1]$ (Fig. 3c). The NN learns a solution where agent 2 passes through the corridor followed by agent 1. After the major shock, the NN still applies these dynamics (Fig. 3b) while the baseline finds a more optimal solution (Fig. 3c). The NN is roughly 100% less optimal in this example (Fig. 3d).

We attribute the shock robustness to the NN's semi-global nature. Experimentally, the shock robustness of our model (Fig. 3) does not noticeably differ from a model trained without penalization (Fig. 4). Since the NN is trained offline prior to deployment, it handles shocks in real-time. In contrast,

methods that solve for a single trajectory—e.g., the baseline—must pause to recompute following a shock.

5) Semi-Global Capabilities of NN model: For thorough analysis of the NN, we assess one NN's performance for many different initial conditions $(0, \boldsymbol{x}_0 + \boldsymbol{\xi})$. We sample 1000 random $\boldsymbol{\xi}$ for each magnitude $\|\boldsymbol{\xi}\| = 0.5, 1.0, \ldots, 5.0$. For each $(0, \boldsymbol{x}_0 + \boldsymbol{\xi})$, we train a baseline model and compute the suboptimality of the trained NN (Fig. 5). This experiment equivalently compares the NN and baseline on samples from concentric hyperspheres. Since a shock can be phrased as picking up a trajectory from an initial condition, testing the NN's semi-global capabilities and shock-robustness are synonymous.

We observe that the NN suboptimality grows as $\|\boldsymbol{\xi}\|$ increases (Fig. 5). Specifically, for the corridor experiment, the NN performs near optimality within $\|\boldsymbol{\xi}\| \leq 2$. Since the NN was trained on ρ which was a Gaussian about $\boldsymbol{x}_0$ with covariance $\boldsymbol{I}$. The bound $\|\boldsymbol{\xi}\| \leq 2$ then equates to being within two standard deviations of $\boldsymbol{x}_0$.

C. Multi-Agent Swap Examples

We present experiments inspired by [19], where agents swap positions while avoiding each other. All agents are two-dimensional, and the formulation mostly matches that presented in the corridor example (Sec. V-B). Specifically, we only alter $\boldsymbol{x}_0$, $\boldsymbol{y}$, and Q for the swap experiments.

1) 2-Agent Swap: We begin with two agents that swap positions with each other while passing through a corridor with hard edges. To enforce these hard edges, we enforce a space bubble around obstacles similar to how we implement multi-agent interactions (13). Therefore, we train with this space bubble but evaluate and plot the results without it. The actual obstacles (two circles with radius 2) are formulated as follows. Let $\Omega_{\mathrm{obs}} = \{z \mid \|z - \boldsymbol{\mu}_1\| < 2 \text{ or } \|z - \boldsymbol{\mu}_2\| < 2\}$, then

$$Q_i\big(z^{(i)}\big) = \begin{cases} 1, & \text{if } z^{(i)} \in \Omega_{\mathrm{obs}}, \\ 0, & \text{otherwise}, \end{cases}$$

where $\boldsymbol{\mu}_1 = \begin{bmatrix} 0 \\ 4 \end{bmatrix}$ and $\boldsymbol{\mu}_2 = \begin{bmatrix} 0 \\ -3.5 \end{bmatrix}$. However, for training, we encode this as

$$Q_{i,\mathrm{trn}}\big(z^{(i)}\big) = \begin{cases} \sum_{j=1}^{2} \eta\big(z^{(i)}; \boldsymbol{\mu}_j, \boldsymbol{I}\big), & \text{if } z^{(i)} \in \Omega_{\mathrm{obs,trn}}, \\ 0, & \text{otherwise}, \end{cases}$$

where $\Omega_{\mathrm{obs,trn}} = \{z \mid \|z - \boldsymbol{\mu}_1\| < 2.2 \text{ or } \|z - \boldsymbol{\mu}_2\| < 2.2\}$. By training with Gaussian repulsion—which has gradient

information within the obstacles—we incentivize the model to learn trajectories avoiding the obstacles. Additionally, $\Omega_{\text{obs,trn}}$ contains an obstacle radial bound ten percent more than in Ω_{obs} because we found this additional training buffer alleviates collisions during validation. We use the same obstacle definitions for the baseline and NN approaches.

For initial and target states, we choose $\boldsymbol{x}_0 = [10, 0, -10, 0]^\top$ and $\boldsymbol{y} = [-10, 0, 10, 0]^\top$. These values are a scaled down version of those in [19] for ease of visualization. For the two-agent problem, the agents successfully switch positions while avoiding each other (Fig. 7). In validation, the obstacle Q and interaction costs W are exactly 0, so we can confirm that the agents avoid collisions. Qualitatively, our method learns trajectories with shorter arclength than those in [19].

2) 12-Agent Swap: We also replicate the 12-agent case in [19]. For this experiment, six pairs of agents swap positions. Since there are no obstacles, $Q = 0$. In our setup, the problem is slightly adjusted as our semi-global approach solves for a fixed $\boldsymbol{y}$ but with initial conditions in ρ, instead of just $\boldsymbol{x}_0$. We display the solution for the single initial case $\boldsymbol{x}_0$ (Fig. 6).

3) Impact of ResNet Width: We demonstrate the influence of the ResNet width m by observing the results of models with varied width for the 12-agent swap experiment (Fig. 8a). We select several m values in the range 12–64 and train three model instances for each while controlling for the rest of the architecture. We then compute the suboptimality of the NN solution relative to the baseline (Sec. V-A) for objective function (2) of a single initial point $\boldsymbol{x}_0$ (Fig. 8a). We observe that, for the 12-agent swap experiment, the underlying manifold exists somewhere near dimension 32 as values $m \geq 32$ are relatively stagnant. We note that smaller values of m perform poorly. When $m < d$, we essentially ask the model to condense the input to a lower dimensional manifold. For the 12-agent swap problem, the $d = 24$ dimensions, though coupled, present no obvious method for reduction to a lower basis. Therefore, we observe poor model performance for $m < d$.

Based on the experiment (Fig. 8a), we use a width of $m = 32$ to balance between a small model and performance. We prefer smaller models as a model with few parameters is easier to evaluate. However, due to the GPU parallelization, different model widths in our experiment (Fig. 8a) have negligible influence on time per training iteration.

4) Hamilton-Jacobi-Bellman Penalty Hyperparameters: In general, we tune hyperparameters relative to each other and set optimizer settings based on architecture design and hyperparameter choices. Thus, in a nuanced response to the findings of Fig. 2, we find that, by training longer with an adjusted learning rate scheme, one can achieve a similar NN solution without any HJB penalizers (*cf.* Fig. 3,4). This holds because the HJB penalizers do not mathematically alter the problem.

We design experiments to demonstrate the sensitivity of the NN solution with respect to the hyperparameters $\beta_1, \beta_2, \beta_3$ (Fig. 8). We train NNs to solve the 12-agent swap experiment (Sec. V-C). We check the sensitivity of the NN solution with respect to changing one β hyperparameter while keeping all other tuned βs and hyperparameters fixed (Table I).
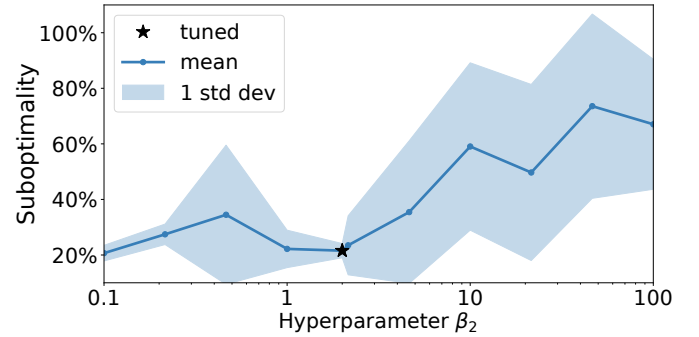
β_1: We observe best performance when $\beta_1 \in (1, 5)$ (Fig. 8b). Since β_1 weights the HJ$_t$ term, setting β_1 too high
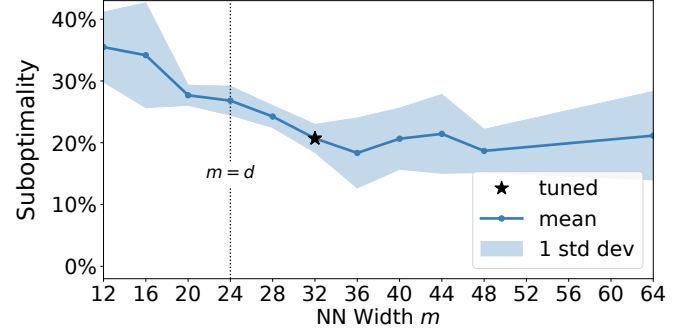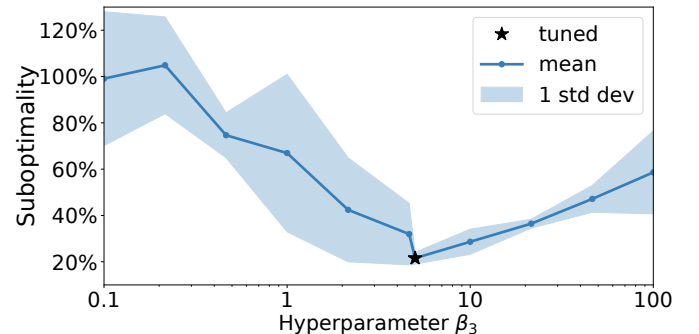

(a) Tuning ResNet width m while keeping all other settings fixed.
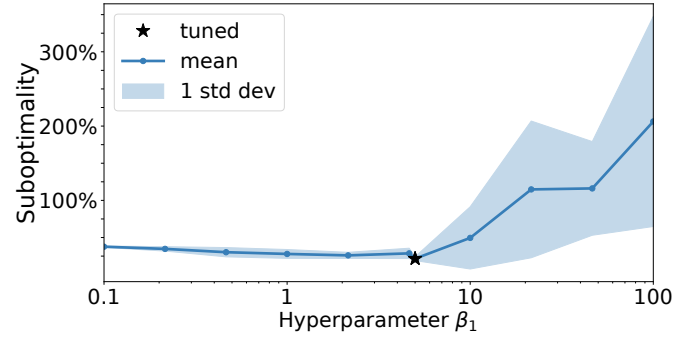
(b) Tuning the scalar hyperparameter β_1 on the HJ$_t$ term while keeping all other settings fixed. The x-axis is log-scaled.

(c) Tuning the scalar hyperparameter β_2 on the HJ$_{\text{fin}}$ term while keeping all other settings fixed. The x-axis is log-scaled.

(d) Tuning the scalar hyperparameter β_3 on the HJ$_{\text{grad}}$ term while keeping all other settings fixed. The x-axis is log-scaled.

Fig. 8: Tuned hyperparameters for the 12-agent swap experiment (Sec. V-C.2) where suboptimality is computed relative to the baseline method (Sec. V-A). Each plotted point and bounds are the mean and standard deviation of three model instances.
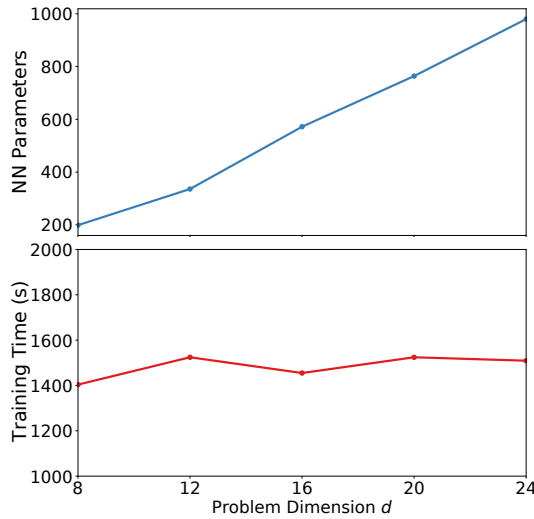
Fig. 9: The NN's number of parameters scales linearly with the problem dimension as the computational cost remains mostly unchanged, mitigating CoD. For each problem (subproblem of the 12-agent swap experiment), we train the smallest NN that achieves at least 10% suboptimality.

Fig. 10: The NN solution for the swarm with 50 agents in $\mathbb{R}^3$ (Sec. V-D). The agents avoid the prism obstacles and each other as they travel from one side of the obstacles to the other.

leads to model training that underprioritizes reaching the target which can result in very suboptimal solutions.

β_2: We observe best performance when $\beta_2 \in (1, 2)$ (Fig. 8c). Since β_2 weights the HJ_{fin} term, setting β_2 too high leads to NN training that overprioritizes fitting the Φ value at time T. Specifically, the training overprioritizes fitting Φ rather than $\nabla_{\boldsymbol{z}}\Phi$, which more directly relates to the dynamics.

β_3: We observe best performance when $\beta_3 \in (4, 10)$ (Fig. 8d). Since β_3 weights the HJ_{grad} term, setting β_3 too high leads to model training that overprioritizes the model reaching the target with less leeway in altering the trajectory for a more optimal L. Alternatively, setting β_3 too small leads to an increase in suboptimality as the model is less likely to satisfactorily reach the target.

5) Mitigating the CoD: We expand the 12-agent swap experiment to demonstrate how the NN approach mitigates the CoD (Fig. 9). We design four additional similar problems by removing agents from the original 12-agent version. Thus, we arrive at problems containing 2, 3, 4, 5, and 6 pairs of agents that swap positions. We then determine the smallest NN that is at most 10% suboptimal. We only tune the width m, which dictates the number of NN parameters, and keep all other settings, including the number of training samples and iterations, fixed. The resulting NNs follow a linear growth of number of parameters relative to the problem dimension d (Fig. 9). Due to the parallelization of the GPU, the training times of these NNs remain comparable.

D. Swarm Trajectory Planning Example

We demonstrate the high-dimensional capabilities of our NN approach by solving a 150-dimensional swarm trajectory planning problem in the spirit of [1]. The swarm problem contains 50 three-dimensional agents that fly from initial to target positions while avoiding each other and obstacles.
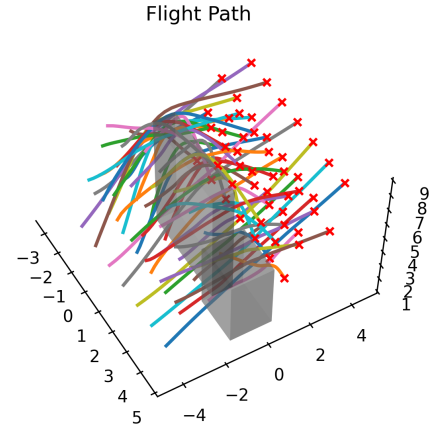
We construct Q to model two rectangular prism obstacles $[-2, 2] \times [-0.5, 0.5] \times [0, 7]$ and $[2, 4] \times [-1, 1] \times [0, 4]$. We train with Gaussian repulsion inside the obstacles similar to the swap experiment (Sec. V-C) and use the same dynamics (34). Due to the complexity of the collision avoidance, we find it beneficial to switch the weights on the HJB penalizers during training—recall that the penalizers do not alter the solution (Sec. V-B.3). For the first portion of training, we choose $\beta_1{=}2$, $\beta_2{=}1$, and $\beta_3{=}3$ (Table I); for the rest of training, we use $\beta_1{=}\beta_2{=}\beta_3{=}0$. This set-up focuses the model on solving the control problem in the first portion of training as the final-time penalizers help the agents reach their destinations. We then reduce the weights of the penalizers for optimal fine-tuning.

In validation, we observe that the values for terrain Q and interaction W are exactly 0. Thus, the NN learns to guide all agents around the obstacles and avoid collisions (Fig. 10).

E. Quadcopter Trajectory Planning Example

In this experiment from [20], a quadcopter, i.e., a multirotor helicopter, utilizes its four rotors to propel itself across space from an initial state in the vicinity of $\boldsymbol{x}_0$ to target state $\boldsymbol{y}$. We choose values $\boldsymbol{x}_0{=}[-1.5, -1.5, -1.5, 0, \ldots, 0]^\top \in \mathbb{R}^{12}$ and $\boldsymbol{y}{=}[2, 2, 2, 0, \ldots, 0]^\top \in \mathbb{R}^{12}$. Denoting gravity as g, the acceleration of a quadcopter with mass m is given by

$$\begin{cases} \ddot{x} = \frac{u}{m}\big(\sin(\psi)\sin(\varphi) + \cos(\psi)\sin(\theta)\cos(\varphi)\big) \\ \ddot{y} = \frac{u}{m}\big(-\cos(\psi)\sin(\varphi) + \sin(\psi)\sin(\theta)\cos(\varphi)\big) \\ \ddot{z} = \frac{u}{m}\cos(\theta)\cos(\varphi) - g \\ \ddot{\psi} = \tau_\psi \\ \ddot{\theta} = \tau_\theta \\ \ddot{\varphi} = \tau_\varphi \end{cases}, \tag{35}$$

where (x, y, z) is the spatial position of the quadcopter, (ψ, θ, φ) is the angular orientation with corresponding torques $\tau_\psi, \tau_\theta, \tau_\varphi$, and u is the main thrust directed out of the bottom of the aircraft [65].

The quadcopter contains highly coupled 12-dimensional dynamics, which lead to time-consuming model training despite
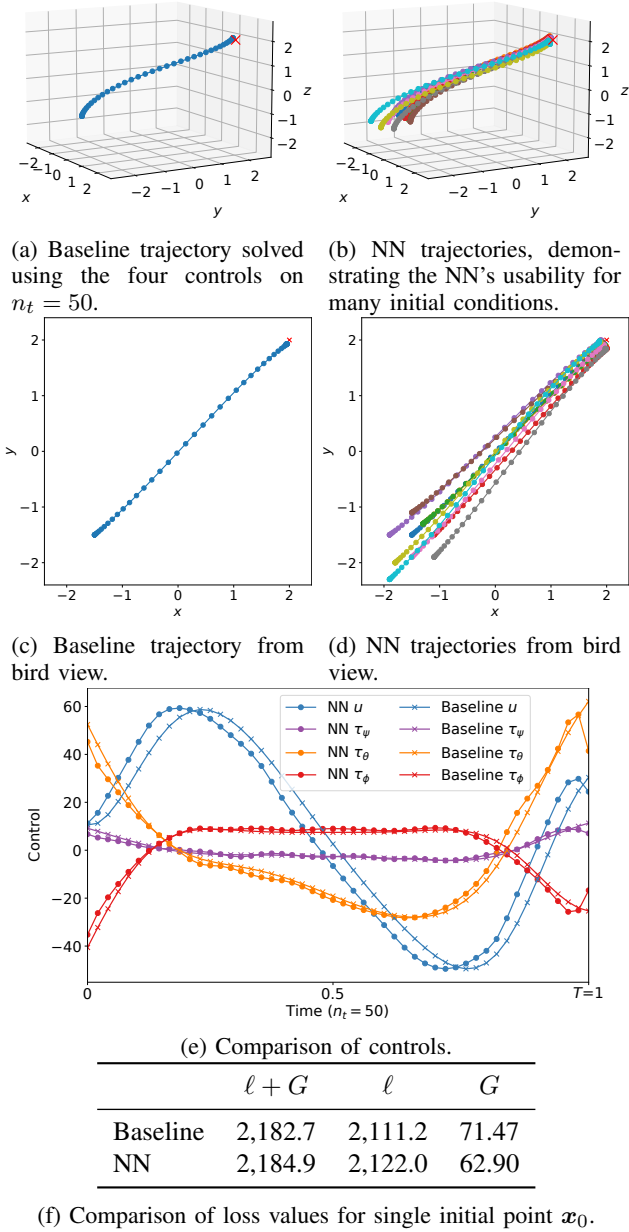
(a) Baseline trajectory solved using the four controls on $n_t = 50$.

(b) NN trajectories, demonstrating the NN's usability for many initial conditions.

(c) Baseline trajectory from bird view.

(d) NN trajectories from bird view.

(e) Comparison of controls.

	$\ell + G$	ℓ	G
Baseline	2,182.7	2,111.2	71.47
NN	2,184.9	2,122.0	62.90

(f) Comparison of loss values for single initial point $\boldsymbol{x}_0$.

Fig. 11: Quadcopter problem results and comparison.

(a) Comparison of angular values.

(b) Comparison of spatial position velocities.

(c) Comparison of angular velocities.

Fig. 12: Quadcopter comparison of the additional states as a supplement to Fig. 11.

its dimension and lack of obstacles and interactions (Table II). The HJB terminal conditions seem to offer little impact as no obstacle or interaction costs interfere with the terminal cost.

The NN approach learns similar controls (Fig. 11e) and states (Fig. 12) as the baseline method. Both methods learn a similar flight path though the NN approach learns for many initial conditions (Fig. 11). As with the corridor problem, the NN learned a solution with better terminal cost, but less optimal ℓ than the baseline (Fig. 11f).

F. Computational Cost

The offline-online paradigm of our NN approach is specifically designed for efficient deployment in real-time applications. To demonstrate this, we compare the computation of the control at time s, updating one time step to time $s + 1$

on a single CPU core for both methods. For the baseline, we compute the cost of 100 function and gradient evaluations with $n_t = 20$ for all experiments. NLP algorithms typically require sampling many initial conditions to solve these nonconvex problems. Thus, we believe 100 function-gradient evaluations is a conservative estimate of the cost to generate a trajectory with the baseline method. For the NN, we compute the approximate cost of one RK4 step; this is computed by dividing the total cost of the trajectory by the number time steps n_t. Naturally, the set-up of the real-time scenario requires online control generation for a space-time $\boldsymbol{s}$ that may not lie on the pre-computed trajectory. A local solution method, the baseline approach must recompute the entire trajectory from $\boldsymbol{s}$ to target space-time $(\boldsymbol{y}, T)$. Evaluating the NN model to obtain a control is 400x-600x faster than solving a control problem with a baseline method (Table II). To compute the full NN trajectory for comparison against the baseline, one simply multiplies the deployment timing of one NN step (Table II)

by the number of time steps n_t (Table I).

A thorough timing comparison may involve reducing the NN approach to an RK1 scheme to match the baseline. This would reduce the NN time cost in Table II by more than a factor of four. Additionally, the baseline timing used a fixed number of time steps $n_t=20$. In practice, when started later in the time-horizon, the baseline may use $n_t < 20$ and therefore perform faster. Conversely, timing the baseline with a finer discretization ($n_t > 20$) results in higher time cost.

VI. DISCUSSION

Our experiments demonstrate the effectiveness of our NN approach for solving several high-dimensional control problems arising in multi-agent collision avoidance. Problems with more complex dynamics and Lagrangians in the finite time-horizon setting are also within reach, as long as the underlying Hamiltonian can be computed efficiently (see Assumption 1). Future work will also involve experimentation with problems that render $\mathcal{H}$ non-concave in $\boldsymbol{u}$ (see Eq. 4) and extending our framework to infinite time-horizon control problems such as the ones in [66]–[68]. Future work will include extensions to situations where the terminal state is unknown or uncertain. While the focus of this work is on numerical simulations and validations, our positive results motivate the application of our technique to real-world systems.

Our approach does not require first solving for sample trajectories to generate training data and thus differs from the supervised training approaches in [28], [29]. A more similar approach to ours is deep reinforcement learning (RL) [69]. However, while RL approaches learn from observations of the dynamics and reward functions (and are thus more general), we assume known dynamics and rewards that satisfy Assumption 1. We argue that these assumptions are not too restrictive as demonstrated by our multi-agent examples with non-convex interactions and many examples in the literature, e.g., [8]. We benefit from using the model because the solution's properties—e.g., the approximated value function must satisfy the HJB equations and optimal actions can be obtained from a feedback form—inform the training process. While this makes our approach less general than RL (applicable in model-free fashion), we expect that this prior knowledge contributes to our approach's effectiveness. As part of future work, we intend to compare our method to RL approaches in terms of sample-efficiency, network choice, and robustness to hyperparameters.

Among the many RL approaches that have been applied to control problems, the perhaps closest to our method is the actor-critic framework [70], [71], which employs two neural networks to approximate the policy (actor) and the value function (critic), respectively. Notably, the weights of the two networks are not shared, and thus, we should not expect them to generally satisfy the feedback form (18). In our approach, we parameterize the value function and compute the optimal policy directly using the feedback form. In addition to requiring only one network, this potentially simplifies the training process, which we plan to investigate in future work.

In the CoD experiment, we observe linear scaling of the NN's parameters for problems of dimensions 8 to 24 (Fig. 9).

Recall that the number of parameters in a grid-based method scales exponentially with the dimension, leading to prohibitive computational complexity and memory costs. Since the NN formulation leverages the GPU parallelization and we use the same number of training samples and iterations regardless of dimension, we observe little noticeable change in the time cost across dimensions 8 to 24 (Fig. 9). Factors that influence the training time stem more from the sequential nature of solving the ODE constraints (24). In multi-agent problems, the memory scales quadratically with the number of agents due to the interaction costs W. Eventually, for a large enough dimension d, the memory costs of the model may exceed the GPU RAM, and implementation changes become necessary.

In our experiments, we show how the semi-global nature of the NN optimally solves the problem within the relevant state-space (Fig. 5). As with most machine learning approaches, our method may fail to generalize, i.e., extrapolate beyond the selected training space. Specifically, the NN often solves the control problem outside the training space, but has potential to do so suboptimally (Fig. 3b) or cause collisions (Fig. 5).

The ability of the NN to avoid collisions and the time needed to train the model depend most crucially on the number of time steps n_t selected (Sec. IV-D). Large n_t leads to high computation and training time while reducing error; meanwhile, too small n_t leads to overfitting to a refinement of the time discretization of the trajectories. A coarsely discretized approximation of the ODE constraints can result in the model unrealistically jumping over obstacles or other agents. Thus, we use large n_t for the hold-out validation set (Table I) to check for overfitting and that the agent movement is sensible.

VII. CONCLUSION

We formulate and demonstrate an NN approach for solving high-dimensional OC problems arising in multi-agent optimal control that consists of an offline and an online phase. In the offline phase, we compute an NN approximation of the control problem's value function in the relevant subset of the space-time domain. Our learning problem combines the high-dimensional scalability from the PMP and the global nature from the HJB approach. In the online phase, the NN approximation is used to compute approximately optimal controls using the feedback form in milliseconds. Our numerical experiments show the effectiveness of our approach for multi-agent problems with state dimension up to 150. Our experiments show that the obtained controls are nearly optimal relative to a baseline and that the network size and computational costs grow only moderately with the dimension of the problem. Moreover, our approach is robust to shocks and can handle complicated interaction and obstacle terms.

REFERENCES

[1] W. Hönig, J. A. Preiss, T. S. Kumar, G. S. Sukhatme, and N. Ayanian, "Trajectory planning for quadrotor swarms," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 856–869, 2018.

[2] S. J. Kim and G. J. Lim, "A real-time rerouting method for drone flights under uncertain flight time," *Journal of Intelligent & Robotic Systems*, vol. 100, pp. 1355–1368, 2020.

[3] M. ElSayed and M. Mohamed, "The uncertainty of autonomous unmanned aerial vehicles' energy consumption," in *IEEE Transportation Electrification Conference & Expo (ITEC)*, 2020, pp. 8–13.

[4] P. R. Florence, J. Carter, J. Ware, and R. Tedrake, "NanoMap: Fast, uncertainty-aware proximity queries with lazy search over local 3D data," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 7631–7638.

[5] J. Yang, C. Liu, M. Coombes, Y. Yan, and W. H. Chen, "Optimal path following for small fixed-wing uavs under wind disturbances," *IEEE Transactions on Control Systems Technology (TCST)*, vol. 29, no. 3, pp. 996–1008, 2021.

[6] I. M. Ross and F. Fahroo, "Issues in the real-time computation of optimal control," *Mathematical and Computer Modelling*, vol. 43, no. 9, pp. 1172–1188, 2006.

[7] G. Tang, W. Sun, and K. Hauser, "Learning trajectories for real-time optimal control of quadrotors," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 3620–3625.

[8] C. Sánchez-Sánchez and D. Izzo, "Real-time optimal control via deep neural networks: study on landing problems," *Journal of Guidance, Control, and Dynamics*, vol. 41, no. 5, pp. 1122–1135, 2018.

[9] M. Chen, S. Herbert, H. Hu, Y. Pu, J. Fernandez Fisac, S. Bansal, S. Han, and C. J. Tomlin, "FaSTrack: A modular framework for real-time motion planning and guaranteed safe tracking," *IEEE Transactions on Automatic Control (TAC)*, 2021.

[10] S. Bansal and C. J. Tomlin, "DeepReach: A deep learning approach to high-dimensional reachability," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 1817–1824.

[11] J. T. Betts, "Survey of numerical methods for trajectory optimization," *Journal of Guidance, Control, and Dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[12] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko, *The Mathematical Theory of Optimal Processes*, ser. Translated by K. N. Trirogoff; edited by L. W. Neustadt. Interscience Publishers John Wiley & Sons, Inc. New York-London, 1962.

[13] R. Bellman, *Dynamic Programming*. Princeton University Press, Princeton, N. J., 1957.

[14] S. Osher and C.-W. Shu, "High-order essentially nonoscillatory schemes for Hamilton–Jacobi equations," *SIAM Journal on Numerical Analysis*, vol. 28, no. 4, pp. 907–922, 1991.

[15] L. Ruthotto, S. J. Osher, W. Li, L. Nurbekyan, and S. W. Fung, "A machine learning framework for solving high-dimensional mean field game and mean field control problems," *Proceedings of the National Academy of Sciences*, vol. 117, no. 17, pp. 9183–9193, 2020.

[16] A. T. Lin, S. W. Fung, W. Li, L. Nurbekyan, and S. J. Osher, "Alternating the population and control neural networks to solve high-dimensional stochastic mean-field games," *Proceedings of the National Academy of Sciences*, vol. 118, no. 31, 2021.

[17] D. Onken, S. W. Fung, X. Li, and L. Ruthotto, "OT-Flow: Fast and accurate continuous normalizing flows via optimal transport," *AAAI Conference on Artificial Intelligence*, vol. 35, no. 10, pp. 9223–9232, 2021.

[18] D. Onken, L. Nurbekyan, X. Li, S. W. Fung, S. Osher, and L. Ruthotto, "A neural network approach applied to multi-agent optimal control," in *European Control Conference (ECC)*, 2021, pp. 1036–1041.

[19] T. Mylvaganam, M. Sassano, and A. Astolfi, "A differential game approach to multi-agent collision avoidance," *IEEE Transactions on Automatic Control (TAC)*, vol. 62, no. 8, pp. 4229–4235, 2017.

[20] A. T. Lin, Y. T. Chow, and S. J. Osher, "A splitting method for overcoming the curse of dimensionality in Hamilton–Jacobi equations arising from nonlinear optimal control and differential games with applications to trajectory generation," *Communications in Mathematical Sciences*, vol. 16, no. 7, pp. 1933–1973, 2018.

[21] J. Darbon and S. Osher, "Algorithms for overcoming the curse of dimensionality for certain Hamilton–Jacobi equations arising in control theory and elsewhere," *Research in the Mathematical Sciences*, vol. 3, no. 1, 2016.

[22] M. R. Kirchner, R. Mar, G. Hewer, J. Darbon, S. Osher, and Y. T. Chow, "Time-optimal collaborative guidance using the generalized Hopf formula," *IEEE Control Systems Letters*, vol. 2, no. 2, pp. 201–206, 2018.

[23] M. R. Kirchner, G. Hewer, J. Darbon, and S. Osher, "A primal-dual method for optimal control and trajectory generation in high-dimensional systems," in *IEEE Conference on Control Technology and Applications (CCTA)*, 2018, pp. 1583–1590.

[24] Y. Chow, J. Darbon, S. Osher, and W. Yin, "Algorithm for overcoming the curse of dimensionality for certain non-convex Hamilton-Jacobi equations, projections and differential games," in *Annals of Mathematical Sciences and Applications*, vol. 3, no. 2. International Press of Boston, 2018, pp. 369–403.

[25] Y. T. Chow, J. Darbon, S. Osher, and W. Yin, "Algorithm for overcoming the curse of dimensionality for state-dependent Hamilton-Jacobi equations," *Journal of Computational Physics*, vol. 387, pp. 376–409, 2019.

[26] C. G. Claudel and A. M. Bayen, "Lax–Hopf based incorporation of internal boundary conditions into Hamilton–Jacobi equation. Part I: Theory," *IEEE Transactions on Automatic Control (TAC)*, vol. 55, no. 5, pp. 1142–1157, 2010.

[27] ——, "Lax–Hopf based incorporation of internal boundary conditions into Hamilton-Jacobi equation. Part II: Computational methods," *IEEE TAC*, vol. 55, no. 5, pp. 1158–1174, 2010.

[28] W. Kang and L. C. Wilcox, "Mitigating the curse of dimensionality: Sparse grid characteristics method for optimal feedback control and HJB equations," *Computational Optimization and Applications*, vol. 68, no. 2, pp. 289–315, 2017.

[29] T. Nakamura-Zimmerer, Q. Gong, and W. Kang, "Adaptive deep learning for high dimensional Hamilton-Jacobi-Bellman equations," *SIAM Journal on Scientific Computing*, vol. 43, no. 2, pp. A1221–A1247, 2021.

[30] J. Sirignano and K. Spiliopoulos, "DGM: A deep learning algorithm for solving partial differential equations," *Journal of Computational Physics*, vol. 375, p. 1339–1364, Dec 2018.

[31] K. Kunisch and D. Walter, "Semiglobal optimal feedback stabilization of autonomous systems via deep neural network approximation," *ESAIM: Control, Optimisation and Calculus of Variations*, vol. 27, 2021.

[32] W. E, J. Han, and A. Jentzen, "Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations," *Communications in Mathematics and Statistics*, vol. 5, no. 4, pp. 349—-380, Nov 2017.

[33] J. Han, A. Jentzen, and W. E, "Solving high-dimensional partial differential equations using deep learning," *Proceedings of the National Academy of Sciences*, vol. 115, no. 34, pp. 8505–8510, Aug 2018.

[34] N. Nüsken and L. Richter, "Solving high-dimensional Hamilton-Jacobi-Bellman PDEs using neural networks: Perspectives from the theory of controlled diffusions and measures on path space," *Partial Differential Equations and Applications*, vol. 2, no. 4, pp. 1–48, 2021.

[35] J. Moon, "Generalized risk-sensitive optimal control and Hamilton-Jacobi-Bellman equation," *IEEE Transactions on Automatic Control (TAC)*, 2020.

[36] J. Han and W. E, "Deep learning approximation for stochastic control problems," *arXiv:1611.07422*, 2016.

[37] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. S. Kumar *et al.*, "Multi-agent pathfinding: Definitions, variants, and benchmarks," *Twelfth Annual Symposium on Combinatorial Search*, 2019.

[38] G. Jing and L. Wang, "Multiagent flocking with angle-based formation shape control," *IEEE Transactions on Automatic Control (TAC)*, vol. 65, no. 2, pp. 817–823, 2019.

[39] S. Zhao, "Affine formation maneuver control of multiagent systems," *IEEE Transactions on Automatic Control (TAC)*, vol. 63, no. 12, pp. 4140–4155, 2018.

[40] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[41] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with performance bounds," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 3260–3267.

[42] M. Erdmann and T. Lozano-Perez, "On multiple moving objects," *Algorithmica*, vol. 2, no. 1–4, 1987.

[43] A. Richards and J. P. How, "Aircraft trajectory planning with collision avoidance using mixed integer linear programming," in *American Control Conference*, vol. 3. IEEE, 2002, pp. 1936–1941.

[44] L. Blackmore and B. Williams, "Optimal manipulator path planning with obstacles using disjunctive programming," in *American Control Conference*. IEEE, 2006.

[45] R. B. Patel and P. J. Goulart, "Trajectory generation for aircraft avoidance maneuvers using online optimization," *Journal of Guidance, Control, and Dynamics*, vol. 34, no. 1, pp. 218–230, 2011.

[46] X. Zhang, A. Liniger, and F. Borrelli, "Optimization-based collision avoidance," *IEEE Transactions on Control Systems Technology (TCST)*, vol. 29, no. 3, pp. 972–983, 2021.

[47] T. Standley and R. Korf, "Complete algorithms for cooperative pathfinding problems," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2011, pp. 668–673.

[48] G. Wagner and H. Choset, "Subdimensional expansion for multirobot path planning," *Artificial Intelligence*, vol. 219, pp. 1–24, 2015.

[49] E. Boyarski, A. Felner, R. Stern, G. Sharon, O. Betzalel, D. Tolpin, and E. Shimony, "ICBS: The improved conflict-based search algorithm for

multi-agent pathfinding," in *Eighth Annual Symposium on Combinatorial Search*. Citeseer, 2015.

[50] L. Cohen, T. Uras, T. K. S. Kumar, H. Xu, N. Ayanian, and S. Koenig, "Improved solvers for bounded-suboptimal multi-agent path finding," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2016, pp. 3067–3074.

[51] B. Rivière, W. Hönig, Y. Yue, and S.-J. Chung, "GLAS: Global-to-local safe autonomy synthesis for multi-robot motion planning with end-to-end learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4249–4256, 2020.

[52] G. Shi, W. Hönig, Y. Yue, and S.-J. Chung, "Neural-Swarm: Decentralized close-proximity multirotor control using learned interactions," *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3241–3247, 2020.

[53] W. H. Fleming and H. M. Soner, *Controlled Markov Processes and Viscosity Solutions*, 2nd ed., ser. Stochastic Modelling and Applied Probability. Springer, New York, 2006, vol. 25.

[54] W. Kang, Q. Gong, and T. Nakamura-Zimmerer, "Algorithms of data development for deep learning and feedback design," *arXiv:1912.00492*, 2019.

[55] P. Cannarsa and C. Sinestrari, *Semiconcave Functions, Hamilton-Jacobi Equations, and Optimal Control*, ser. Progress in Nonlinear Differential Equations and their Applications. Boston, MA: Birkhäuser Boston, Inc., 2004, vol. 58.

[56] L. C. Evans, *Partial Differential Equations*. American Mathematical Society, 2010, vol. 19.

[57] M. G. Crandall and P.-L. Lions, "Viscosity solutions of Hamilton-Jacobi equations," *Trans. Amer. Math. Soc.*, vol. 277, no. 1, pp. 1–42, 1983.

[58] C. Finlay, J.-H. Jacobsen, L. Nurbekyan, and A. M. Oberman, "How to train your neural ODE: the world of Jacobian and kinetic regularization," in *International Conference on Machine Learning (ICML)*, 2020, pp. 3154–3164.

[59] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[60] A. Gholaminejad, K. Keutzer, and G. Biros, "ANODE: Unconditionally accurate memory-efficient gradients for neural ODEs," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2019, pp. 730–736.

[61] D. Onken and L. Ruthotto, "Discretize-optimize vs. optimize-discretize for time-series regression and continuous normalizing flows," *arXiv:2005.13420*, 2020.

[62] J. Nocedal and S. Wright, *Numerical Optimization*. Springer Science & Business Media, 2006.

[63] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

[64] L. Yang and G. E. Karniadakis, "Potential flow generator with L_2 optimal transport regularity for generative models," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[65] L. R. G. Carrillo, A. E. D. López, R. Lozano, and C. Pégard, "Modeling the quad-rotor mini-rotorcraft," in *Quad Rotorcraft Control*. Springer, 2013, pp. 23–34.

[66] Y. Jiang and Z.-P. Jiang, "Global adaptive dynamic programming for continuous-time nonlinear systems," *IEEE Transactions on Automatic Control (TAC)*, vol. 60, no. 11, pp. 2917–2929, 2015.

[67] I. Michailidis, S. Baldi, E. B. Kosmatopoulos, and P. A. Ioannou, "Adaptive optimal control for large-scale nonlinear systems," *IEEE TAC*, vol. 62, no. 11, pp. 5567–5577, 2017.

[68] V. G. Lopez, F. L. Lewis, Y. Wan, E. N. Sanchez, and L. Fan, "Solutions for multiagent pursuit-evasion games on communication graphs: Finite-time capture and asymptotic behaviors," *IEEE Transactions on Automatic Control (TAC)*, vol. 65, no. 5, pp. 1911–1923, 2019.

[69] D. P. Bertsekas, *Reinforcement Learning and Optimal Control*. Athena Scientific Belmont, MA, 2019.

[70] V. R. Konda and V. S. Borkar, "Actor-critic–type learning algorithms for Markov decision processes," *SIAM Journal on Control and Optimization*, vol. 38, no. 1, pp. 94–123, 1999.

[71] V. Konda and J. Tsitsiklis, "Actor-Critic algorithms," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 12, 1999.

Derek Onken received the M.S degree in computer science and the Ph.D. degree in computer science & informatics from Emory University, Atlanta, GA, USA in 2019 and 2021, respectively.

In 2018, he worked with Air Force Research Labs and UES Inc., and in 2019-2020, he worked for UnitedHealth Group. His research interests include the application of neural networks to optimal control, normalizing flows, image classification, and image segmentation.

Levon Nurbekyan received a Ph.D. degree in mathematics from the Technical University of Lisbon in 2012 in the framework of the UT Austin Portugal CoLab program.

He is an Assistant Adjunct Professor in the Department of Mathematics at the University of California, Los Angeles (UCLA). He previously held postdoctoral positions at McGill, KAUST, NAS of Armenia, and IST Lisbon. He was also a Senior Fellow at IPAM at UCLA and a Simons CRM Scholar-in-Residence at the University of Montreal. His research interests include mean-field games, machine learning, optimal control, optimal transportation, partial differential equations, shape optimization problems, and dynamical systems.

Xingjian Li received the B.S. degree in computational mathematics from Xiamen University, China, in 2019.

He is currently pursuing the Ph.D. degree in Computational Mathematics at Emory University. His research interests include neural ordinary differential equations, application of machine learning to optimal control, and reinforcement learning.

Samy Wu Fung received the Ph.D. degree in applied mathematics from Emory University, Atlanta, GA, USA, in 2019.

He is an Assistant Professor in the Department of Applied Mathematics and Statistics at Colorado School of Mines. Prior to joining Mines, he was an Assistant Adjunct Professor in the Department of Mathematics at UCLA. His research interests lie in the fields of inverse problems, optimization, deep learning, and mean field games.

Stanley Osher received the M.S. and Ph.D. degrees in mathematics from the Courant Institute of Mathematical Sciences, New York University, New York, NY, USA, in 1964 and 1966, respectively.

He taught at SUNY Stony Brook, Stony Brook, NY, where he became a Professor in 1975. He was with the Faculty of the University of California Los Angeles (UCLA), Los Angeles, CA, USA, in 1977. He is currently a Professor of mathematics, computer science, chemical engineering, and electrical engineering with UCLA. He is also an Associate Director of the National Science Foundation-funded Institute for Pure and Applied Mathematics with UCLA. He co-founded three successful companies, each-based largely on his own (joint) research. His research interests include information science, which includes optimization, image processing, compressed sensing and machine learning, and applications of these techniques to physics, engineering, and elsewhere.

Prof. Osher is a Fellow of the Society for Industrial and Applied Mathematics (SIAM) and the American Mathematical Society. In 2014, he received the Carl Friedrich Gauss Prize from the International Mathematics Union, regarded as the highest prize in applied mathematics. He gave the John von Neumann Lecture at the SIAM 2013 annual meeting. He has been elected to the US National Academy of Science, National Academy of Engineering, and the American Academy of Arts and Sciences. He gave a one-hour plenary address at the 2010 International Conference of Mathematicians.

Lars Ruthotto received his diploma and Ph.D. degree in mathematics from the University of Munster, in 2010 and 2012, respectively.

He is an Associate Professor of Mathematics and Computer Science at Emory University, Atlanta, GA, USA. Before joining Emory, he was a postdoctoral fellow at the University of British Columbia. His main research interests is scientific computing and its applications in machine learning and medical and geophysical imaging.