Differentiable Learning of Scalable Multi-Agent Navigation Policies

Xiaohan Ye[1,2], Zherong Pan[1], Xifeng Gao[1], Kui Wu[1], Bo Ren[3†]

***Abstract*—We present an end-to-end differentiable learning algorithm for multi-agent navigation policies. Compared with prior model-free learning algorithms, our method leads to a significant speedup via the gradient information. Our key innovation lies in a novel differentiability analysis of the optimization-based crowd simulation algorithm via the implicit function theorem. Inspired by continuum multi-agent modeling techniques, we further propose a kernel-based policy parameterization, allowing our learned policy to scale up to an arbitrary number of agents without re-training. We evaluate our algorithm on two tasks in obstacle-rich environments, partially labeled navigation and evacuation, for which loss functions can be defined making the entire task learnable in an end-to-end manner. The results show that our method can achieve more than one order of magnitude speedup over model-free baselines and readily scale to unseen target configurations and agent sizes.**

I. Introduction

Multi-agent systems find various robotic applications and there has been an ever-increasing interest in the automatic learning of multi-agent behaviors. Complex behaviors can be automatically acquired, e.g., by Multi-Agent Reinforcement Learning (MARL) algorithms, through a gazillion of reward- or curiosity-guided random behavior explorations. On the other hand, the overwhelming computational overhead of offline learning makes it difficult to develop and experiment with new solutions.

In this paper, we focus on collision-free, multi-agent navigation, a common sub-task of high-level collective behaviors. Learning-based navigation algorithms use RL [1, 2, 3] or MARL [4, 5] to combine the merits of centralized and decentralized algorithms. The learned policies are informed of the local configuration of other robots and/or the global environment and infer a control signal. However, all these methods inherit the drawback of their underlying learning algorithms. Specifically, (MA)RL requires a huge number of environment-interacting experiences, and supervised learning algorithms rely on a dataset of groundtruth control signals.

To alleviate the computational burden of offline training, multiple recent efforts propose the use of model-based algorithms using learned [6, 7, 8] or differentiable dynamics [9, 10, 11]. These methods have achieved orders of magnitude speedup over their model-free counterparts, especially in the locomotive and manipulation control of complex dynamic systems. However, deriving differentiable dynamics is a non-trivial task by involving non-differentiable steps or stiff behaviors leading to an unstable differentiation scheme.

Main Result: We propose the first differentiable and scalable learning method for collision-free multi-agent navigation policies. We first conduct a differentiability analysis on two existing local navigation algorithms, Optimal Reciprocal Collision Avoidance (ORCA) [12] and Implicit Crowds (IC) [13]. We show that both algorithms lead to ill-defined gradient information. Instead, we derive a new algorithm based on a weaker assumption on collision-free paths, with well-defined gradient information. We then derive a compact and scalable parameterization of navigation policies. Inspired by continuum multi-agent modeling [14, 15], we propose to control the agents using a boundary-aligned, divergence-free velocity field. We further borrow ideas from vortex-based sparse velocity representation [16] and parameterize the velocity field using a sparse set of Lagrangian kernel functions. By using the parameters of kernel functions as the low-dimensional action space, our navigation policy can control an arbitrary number of agents with an invariant representation. Complementary to the general neural network analysis approach [17, 18], our action space representation improves the AI explainability in a domain-specific manner. Our kernel functions inherently summarize the key information of the underlying velocity field. We train our controller to solve two tasks: partially labeled navigation and evacuation, where our method achieves at least one order of magnitude speedup over model-based counterparts and the learned policies can be readily generalized to unseen target configurations and agent sizes.

II. Related Work

We review relevant research directions in multi-agent navigation and differentiable physics.

Multi-agent navigation has garnered long-lasting research interest over the past decades. The focus has shifted from theoretical hardness analysis [19], centralized robot routing algorithm [20], all the way to decentralized real-time navigation algorithms [12, 21, 22] and learning-based semi-centralized navigation policy search [2, 1]. Compared to centralized methods, local navigation algorithms suffer from deadlock configurations [23, 24] and can fail to provide completeness guarantees. Heuristic algorithms [25, 26] have been provided over the years to improve the success rate empirically. In particular, fluid dynamics have been used to model large agent crowds as a continuum flow [14, 15, 27].

Learning-based local navigation [2, 1] can infer control signals in real-time while accounting for global information via rich observations and inter-agent communications. However, all existing navigation policy search algorithms incur a

Manuscript received: November 4, 2022; Revised January 18, 2023; Accepted February 13, 2023.

This paper was recommended for publication by Editor Ashis Banerjee upon evaluation of the Associate Editor and Reviewers' comments.

† indicates corresponding author. [1]LightSpeed Studios, Tencent ({xiaohanye, zrpan, xifgao, kwwu}@global.tencent.com). [2]College of Computer Science, Nankai University (2120200463@mail.nankai.edu.cn). [3]College of Computer Science, Nankai University (rb@nankai.edu.cn).

Digital Object Identifier (DOI): see top of this page.

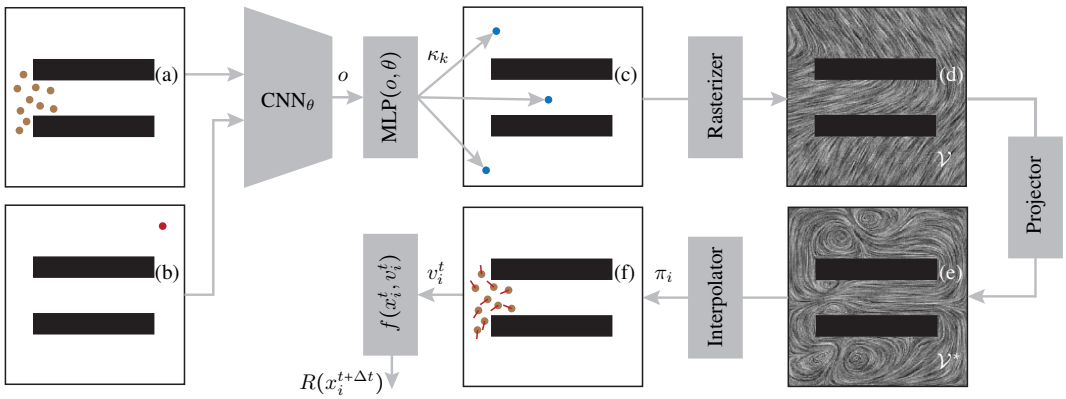

Fig. 1: An illustration of our centralized policy representation for both training and test time. Our policy π takes two images, the global environmental map (a) and the target configuration (b), as inputs. Latent features are extracted using a CNN to form the observation o. The kernel estimator is a fully-connected MLP that estimates the parameters of a small set of velocity kernels κ_k (c). The velocity field $\mathcal{V}$ generated by these kernels (d) is extracted by a rasterizer. This velocity field is not divergence-free, and we then extract the divergence-free component $\mathcal{V}^*$ (e) via a projector. The final preferred moving directions π_i are then extracted by interpolating $\mathcal{V}^*$ (f). The interpolated control signal is fed into the differential collision-free navigation simulation function f. Finally, the new agent locations are feed into the reward signal $R(x_i^{t+\Delta t})$.

high offline training cost. For example, the average RL training cost is around 10 hours on average, as reported by [1]. For the class of navigation tasks considered in this work, we show that our differentiable learning method not only allows faster training but also saves inference costs by deploying a single policy for all the agents.

Differentiable physics has recently arisen as a promising direction of high-dimensional manipulation. Indeed, the development of stochastic, large-scale non-smooth optimization [28] enables optimization-based control of ultra-high-dimensional, non-smooth dynamic systems. This include fluid bodies [11], stoke flows [29, 30], deformable solids [31], and thin shells [9]. The key idea is to use only first-order gradient information computed via the adjoint method while using the stochastically approximate, bundled gradient instead of the exact gradient in the face of non-smooth dynamics. The differentiability of a dynamic system significantly depends on the underlying discretization scheme. Certain discretization schemes, such as explicit Euler integration, incur stiff dynamic systems, leading to gradient explosion. Therefore, existing differential models are based on more stable discretization schemes such as projective dynamics [32, 33] and (semi-)implicit time integrators [9]. Recently, Karamouzas *et al.* [13] proposed an implicit time-integration scheme for guaranteed collision-free multi-agent navigation, based on which we propose a weaker assumption on collision-free paths that is more amenable to differentiability.

III. Problem Formulation & Background

In this section, we formulate the problem of multi-agent collision-free navigation tasks. We assume a group of N holonomic agents, in which the ith agent is located at $x_i^t \in \mathbb{R}^2$ at time instance t. All the agents are assumed to have the same spherical shape with a unified radius of r. We further introduce a set of M static obstacles, with the jth obstacle taking up volume $o_j \subset \mathbb{R}^2$. The problem of multi-agent navigation is to search for a discrete trajectory over a horizon of T steps, such that any pair of two agents do not overlap at any time instance, i.e.:

$$\forall \alpha \in [t, t+\Delta t] : \begin{cases} \text{dist}(x_i^\alpha, x_j^\alpha) \geq 2r \\ \text{dist}(x_i^\alpha, o_j) \geq r \end{cases}, \tag{1}$$

where Δt is the timestep size. Since the agents are assumed to be holonomic, their dynamics are dominated by collision-free constraints and the control signal. A common choice [12, 13] represents the control signal of x_i as the preferred moving velocity, denoted as v_i^t. The next position $x_i^{t+\Delta t}$ can then be predicted using a multi-agent simulation function $f(x_i^t, v_i^t) = x_i^{t+\Delta t}$. The goal of navigation is encoded in a reward function $R(x_i^t)$ summed over all the agents and time instances. A navigation policy for the ith agent could be modeled as a function $\pi_i(o, \theta) = v_i^t$, where o is the observation of the local and global features around the agent, and θ are the decision variables that parameterizes π_i. The goal of multi-agent navigation is then formulated as the following Markov Decision Process (MDP):

$$\underset{\theta}{\text{argmax}}\ \mathbb{E}_{\tau_i \sim (f, \pi_i), x_i^0 \sim \mathcal{S}} \left[\sum_{x_i^t \in \tau_i} \gamma^t R(x_i^t) \right], \tag{2}$$

where $\mathcal{S}$ is the initial distribution of agent configurations, τ_i is the trajectory of x_i generated by the state transition function f and the policy π_i, and γ is the discount factor. Recent learning-based methods [1, 2, 34] considered variants of Equation (2) with different choices of f, R, π_i and algorithms for solving the stochastic optimization. We discuss these choices in more detail below.

In all previous works, the transition function f is implemented using the ORCA algorithm [12] during the training phase. This algorithm assumes each agent's trajectory within a time interval $[t, t + \Delta t]$ is a straight line. Under this assumption, a subset of collision-free straight-line trajectories can be specified by locally linear constraints between pairs of neighboring agents, which can then be satisfied by solving a number of small-sized linear programming problems. In

terms of the reward function, prior works [1, 2] considered the labeled navigation task where the reward signal measures the distance between each agent and their goal positions. Li *et al.* [35] and Ji *et al.* [34] considered labeled and unlabeled navigation tasks, respectively, but they optimize θ via imitation learning, and the reward is some discrepancy measure against the expert policy. Finally, the design of π_i poses a key point of departure among different works. Prior works [1] assumed a decentralized setting where agents receive only the local sensing information as the observation o. Tan *et al.* [2] assumed a centralized setting where agents can receive a snapshot of the global map and the locations of all other agents. Other works [35, 34] assume a semi-centralized setting where agents can communicate with others in a small vicinity. Despite these differences, all prior works formulate the learning problem as a model-free algorithm, i.e., the learning algorithm updates the policy parameter θ using reward signal on each trajectory τ_i, treating the underlying dynamics f as a black box.

IV. Differentiable & Scalable Policy Search

We describe our differentiable and scalable policy search algorithm as outlined in Figure 1. Our method consists of novel designs of functions f, R, and π_i, such that the gradient information could be brought all the way back through the entire trajectory τ to efficiently update θ. Our design further allows the use of a small network to control an arbitrary number of agents in a centralized manner.

A. Differentiable Multi-Agent Simulator

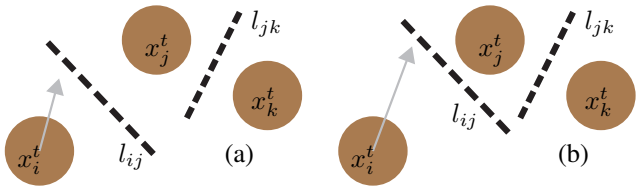

Fig. 2: ORCA confines three agents $x_{i,j,k}^t$ to one side of linear constraints denoted as l_{ij}, l_{jk} (a). If the preferred velocity of x_i^t, arrow v_i^t, is larger, the constraint l_{ij} will be pushed toward x_j (b), but this can never influence x_k unless x_k is in the direct neighborhood of x_i. This phenomenon limits the range of gradient propagation.

To bring the gradient information through function f, we need a differentiable collision-free multi-agent simulator. At the early stage of this research, we tried to use an auto-differentiation system [36] to re-implement the ORCA algorithm. Note the linear programming algorithm in ORCA is locally differentiable [37]. However, we find that the model-based learning algorithm cannot efficiently optimize policies using this gradient information. This is because ORCA and its variants [38, 39, 40] confine each agent to independent feasible subdomains so that linear programming problems can be solved separately. Although this method significantly lowers the computational overhead, it prevents gradient information from propagating to neighboring agents, as illustrated in Figure 2.

Instead, we turn to the more recently proposed IC algorithm [13]. This method formulates the agent simulation as an implicit time integration problem, identifying the predicted agent position as the minimizer of the following energy:

$$x_1^{t+\Delta t}, \cdots, x_N^{t+\Delta t} \triangleq \underset{x_1,\cdots,x_N}{\operatorname{argmin}} E(x_i, x_i^t, v_i) \tag{3}$$

$$E(x_i, x_i^t, v_i) \triangleq \sum_i \frac{1}{2\Delta t^2} \|x_i - x_i^t - v_i^t \Delta t\|^2 + \sum_{i \neq j} U(x_i, x_j),$$

where $U(x_i, x_j)$ is a stiff potential function defined as:

$$U(x_i, x_j) \triangleq \frac{1}{\|x_i - x_j\| - 2r}, \tag{4}$$

which tends to infinity when $\|x_i, x_j\| \to 2r$. By maintaining the solution in the feasible domain via a line-search, $x_i^{t+\Delta t}$ are guaranteed to satisfy Equation (1) at the time instance $t+\Delta t$. Time integrating the dynamic system with the potential function U can suffer from stability issues because U can generate extremely large forces. Therefore, Karamouzas *et al.* [13] uses the implicit scheme and reformulates the time integrator as an optimization Equation (1). This method is way more amenable to a differentiable system. By not fixing the decision variables in the constraint, the gradient information can be readily propagated to faraway agents.

Algorithm 1: Differentiable Crowd Simulation: $f(x_i^t, v_i^t)$

Input: $x_i^t, v_i^t, \epsilon_g > 0, \gamma \in (0,1)$
Output: $x_i^{t+\Delta t}$

$x_i^{t+\Delta t} \leftarrow x_i^t$
while $\|\partial E / \partial x_i^{t+\Delta t}\|_\infty > \epsilon_g$ **do**
  Find search direction d_i using Newton's method
  $\alpha \leftarrow 1$
  while True **do**
    if Segments $x_i^{t+\Delta t} \leftrightarrow x_i^{t+\Delta t} + \alpha d_i$ intersects **then**
      $\alpha \leftarrow \gamma\alpha$
    else if $E(x_i^{t+\Delta t}) \geq E(x_i^{t+\Delta t} + \alpha d_i)$ **then**
      $\alpha \leftarrow \gamma\alpha$
    else
      $x_i^{t+\Delta t} \leftarrow x_i^{t+\Delta t} + \alpha d_i$
      Break
  Return $x_i^{t+\Delta t}$

However, satisfying Equation (1) at $t + \Delta t$ alone is not enough, as Equation (1) should be satisfied at any time instance $\alpha \in [t, t + \Delta t]$, which could be achieved via continuous collision check. To this end, Karamouzas *et al.* [13] considered an alternative definition of distance, aka., the minimal distance along a straight line between x_i^t and x_i, leading to the following new definition of U':

$$U'(x_i, x_j) \triangleq \frac{1}{\underset{\alpha \in [0,1]}{\operatorname{argmin}} \|(1-\alpha)(x_i - x_j) + \alpha(x_i^t - x_j^t)\| - 2r}.$$

Unfortunately, it is well-known that the distance between line segments is non-differentiable [41] at nearly parallel configurations, again hindering the well-definedness of gradient information. To resolve this problem, we consider replacing the straight-line assumption with the following weaker assumption on collision-free paths:

Assumption 4.1: Agents x_i^α are collision-free at any $\alpha \in [t, t+\Delta t]$ if there exists a piecewise linear trajectory satisfying Equation (1).

Based on Assumption 4.1, we propose an "inconsistent" optimizer where the potential energy U considers a single time instance $t + \Delta t$, while the line-search algorithm takes care of the entire time period $[t, t + \Delta t]$ by ensuring that each search step represents a collision-free linear sub-trajectory. As a result, the entire trajectory generated by the optimizer of Equation (3) is exactly piecewise linear. Our key observation is that such "inconsistent" optimizer belongs to the class of primal interior point method [42] and is guaranteed to converge to the locally optimal solution of Equation (3) and generate a collision-free trajectory. By relying on U instead of U', our objective function is twice-differentiable at any point in its feasible domain. We summarize this method in Algorithm 1. Note our weaker assumption is more difficult to be deployed on a physical hardware because the control signal needs to change multiple times within a single control loop, but this is less important during the training phase. We further borrow ideas from [41] and use the following locally supported U:

$$U(x_i, x_j) = \frac{1}{2}\sum_{l=1}^{\infty} l^3 \max\left[0, \frac{l_0}{l^{\frac{1}{4}}} - (\|x_i - x_j\| - 2r)\right]^2, \quad (5)$$

where l_0 is the support range parameter. Unlike Equation (4) that is non-zero throughout the domain, the above-defined U is zero when $\|x_i - x_j\| \geq 2r + l_0$, so we could compute the double summation in Equation (3) efficiently using local neighborhood search with a radius of $2r + l_0$ and all the agent pairs with distance larger than $2r + l_0$ can be excluded from consideration. Finally, the gradient information can be computed using the implicit function theorem as:

$$\frac{\partial x^{t+\Delta t}}{\partial\left(x^t\, v^t\right)} = -\left[\frac{\partial^2 E}{\partial x^{t+\Delta t 2}}\right]^{-1} \frac{\partial^2 E}{\partial x^{t+\Delta t}\partial\left(x^t\, v^t\right)},$$

where we denote x, v without subscript as a vector concatenating x, v of all agents.

B. Kernel-Based Policy Parameterization

Aside from a differentiable navigation simulator, the key to efficient and robust training lies in a compact parameterization of policy π_i. Indeed, a redundant parameterization is prone to over-fitting or requires an excessively large dataset or amount of explorations. Our policy parameterization is inspired by the continuum model of crowd motions [14, 15, 27]. These methods model agents as infinitesimal material particles immersed in a velocity field denoted by $\mathcal{V}$. It has been shown that the collision-free constraints can be approximated by the requirement that $\mathcal{V}$ is divergence-free or volume-preserving [14, 15], i.e., $\nabla \cdot \mathcal{V} = 0$. Such representation has two advantages. First, divergence-free constraints are linear, which can be efficiently enforced by fast linear system solvers. Without using learnable parameters, divergence-free constraints can already prevent a considerable portion of local, inter-agent collisions or boundary penetrations. As a result, a neural network policy could focus on solving high-level tasks. Second, several prior works [43, 44] have demonstrated that a divergence-free vector field can be sparsely represented as a set of kernel functions. Indeed, a vector field is smooth except at a set of singular points.

In order to represent a centralized and scalable control policy, we propose to learn a sparse representation of the velocity field. We assume there are K kernel functions denoted as $\kappa_k(p, \phi)$ with $k = 1, \cdots, K$, where $p \in \mathbb{R}^2$ is a spatial location and ϕ is a small set of parameters. Two kinds of kernel functions have been proven effective in generating structured motions of immersed particles.Rotating motions are generated by the following kernel:

$$\kappa(p, \phi) \triangleq \beta \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} (p - p_0) \exp\left(-\alpha \|p - p_0\|^2\right) \;\; \phi \triangleq \left(\alpha\, \beta\, p_0\right),$$

while directional motions can be generated by defining:

$$\kappa(p, \phi) \triangleq d \exp\left(-\alpha \|p - p_0\|^2\right) \quad \phi \triangleq \left(\alpha\, d\, p_0\right). \quad (6)$$

Here β and d control the strength of swirl and motion velocity, respectively. Both functions are called kernel functions that attenuate quickly with distance from a kernel center x_0 with a parameter α. However, since robots are mostly moving along a fixed direction during navigation tasks, we choose only to use directional kernels. We speculate that curl kernels could be useful in other tasks as well. The velocity field can then be defined as an accumulation of the K kernels:

$$\mathcal{V}(p) \triangleq \sum_{k=1}^{K} \kappa(p, \phi_k). \quad (7)$$

In practice, the accumulated velocity field $\mathcal{V}$ is then rasterized onto a dense grid. To further enforce the divergence-free condition, we solve a discrete Poisson's equation using a staggered grid discretization as proposed in [45], which is denoted as:

$$\mathcal{V}^* \triangleq \underset{\mathcal{V}^*}{\operatorname{argmin}} \frac{1}{2} \int_x \|\mathcal{V}^*(p) - \mathcal{V}(p)\|^2 \quad \text{s.t.} \nabla \cdot \mathcal{V}^* = 0, \quad (8)$$

where we always enforce the Dirichlet boundary condition that the normal velocity on the boundary is zero. Equation (8) involves solving a large linear system. However, for a fixed obstacle setup, the left-hand side of the linear system is fixed and can be pre-factorized. As a result, solving for $\mathcal{V}^*$ at both training and inference time can be very efficient. Finally, the preferred velocity of each agent is derived by interpolating $\mathcal{V}^*$ at x_i^t. As a remarkable feature, all the above derivations do not involve any learnable parameters and only expose a small number of kernel parameters ϕ_k to be determined. We propose to use a fully connected neural network to infer $\phi_1, \cdots, \phi_K$ using a small Multi-Layer Perceptron (MLP) from observations:

$$\text{MLP}(o, \theta) = \left(\phi_1 \cdots \phi_K\right). \quad (9)$$

In summary, the definition of our compact policy parameterization of π_i combines Equation (6),7,8,9. In practice, MLP outputs less than 100 parameters to define an entire velocity field, which could be used to control an arbitrary number of agents without re-training.

Layer	Kernel	Stride	#Filters/#Neuron	Activation
Conv_1	(7,7)	1	8	MaxPool.+ReLU
Conv_2	(7,7)	1	12	MaxPool.+ReLU
Conv_3	(5,5)	1	16	MaxPool.+ReLU
Conv_4	(3,3)	1	20	MaxPool.+ReLU
MLP_1	/	/	128	ReLU
MLP_2	/	/	5K	Sigmoid

TABLE I: Parameters for the MLP and CNN.

Since our policy is centralized, the observation must contain global information, which could be the global map in [2]. We find our policy achieves the best performance using two global maps, one encoding the obstacle with agent configurations and the other encoding the target configuration. The observation consists of features extracted from these two images using two CNNs of the same structure. Our network architecture is summarized in Table I.

To highlight the advantage of our policy parameterization, we compare the inference-time complexity of several learning-based multi-agent navigation methods [2, 1, 35]. All these methods concatenate two kinds of networks to form the control policy: CNN for (local or global) feature encoding and MLP for action prediction, while some technique [35] optionally involves a Graph Neural Network (GNN) to enable inter-agent communication. The detailed parameterization of each network is task-specific, so we only compare the number of inferences for each type of network during a single timestep. We further distinguish local feature encoder (LCNN) and global feature encoder (GCNN). Encoding local features is much cheaper than global ones, using small images [35] or 1D convolution of LiDAR data [2]. As indicated in Table II on page 5, our method is the only one that incurs a inference-time complexity independent of N and utilizes global information.

C. Policy Search Algorithm

As long as the reward signal R is differentiable, our learning architecture could bring gradient information through each and every component to facilitate self-supervised, model-based learning. To train robust control policies, however, we still need to feed the learning algorithm with enough trajectories sampled according to the current functions f, π_i. To stabilize training, we propose to maintain a replay buffer similar to Q-learning, but instead of storing the entire transition tuple $< x_i^t, v_i^t, x_i^{t+\Delta t}, R(x_i^{t+\Delta t}) >$, we only need to store the current state x_i^t. During each iteration of training, we sample a batch $\mathcal{B}$ of x_i^t and optimize R over a receding horizon of H timesteps, i.e.:

$$\theta \leftarrow \theta + \alpha_{\text{lr}} \nabla_\theta \sum_{x_i^t \in \mathcal{B}} \sum_{h=1,\cdots,H} \gamma^t R(x_i^{t+h\Delta t}). \tag{10}$$

The replay buffer is constructed by sampling an initial configuration $x_i^0 \in \mathcal{S}$ and running the current policy π_i to generate trajectories τ. To minimize the distribution bias between the replay buffer and the policy, we only keep a set of the latest trajectories. The overall algorithm is summarized in Algorithm 2.

V. Evaluation

We implement our neural network policy using PyTorch and the differentiable simulator in C++ as a custom network layer. One computational bottleneck is the divergence-free projection, for which we pre-factorize the lefthand side matrix on the CPU before training, and then solve linear systems on GPU. All experiments are performed on a desktop machine with an i7-10750H CPU and Nvidia GTX-1080 GPU. All the forward-backward gradient propagations are performed on GPU, except for the differentiable simulator on CPU. We evaluate our method on two tasks, partially labeled navigation and evacuation. We compare our method with three model-free RL baselines (PPO [46], SAC [47], and DDPG [48]) to train our policy using the same reward function. All the training uses maze-like environments with randomized obstacle shapes, and we use parameters $r = 1$ unit, $l_0 = 10$ unit, $\Delta = 0.1$s, $\alpha_{\text{lr}} = 3 \times 10^{-5}$, and $\gamma = 0.98$. We terminate a trajectory after 128 timesteps. Thanks to our stable agent simulator allowing a large timestep size $\Delta t = 1$, we find short trajectories is sufficient for agents to reach their goals, which significantly boosts training speed. We use 128 timesteps for evacuation task and 256 for others. We use a minimal replay buffer size, i.e., $\mathcal{D}$ only stores the latest sampled trajectory, and we train a single epoch after sampling one trajectory.

Method	Global	#LCNN	#GCNN	#GNN	#MLP
[2]	No	N(LiDAR)	0	No	N
[1]	Yes	N(LiDAR)	1	No	N
[35]	Yes	N(Image)	0	Yes	N
Ours	Yes	0	1	No	1

TABLE II: We summarize the number of network inferences during each timestep. The second column indicates whether the network gets access to the global map information, which can be acquired either through a GCNN [1] or a GNN [35]. The local features can be represented as a small image or the LiDAR data, as indicated in the third column.

Algorithm 2: Model-Based Policy Learning

Input: $\mathcal{S}, \alpha_{lr}, \theta$
Output: Optimized θ
$\mathcal{D} \leftarrow \varnothing$
while Stopping criterion not met **do**
  for Each sampling iteration **do**
    Sample a trajectory $\tau \sim f, \pi_i, x_i^0 \sim \mathcal{S}$
    $\mathcal{D} \leftarrow \mathcal{D} \bigcup \tau$
  Delete old trajectories from $\mathcal{D}$
  for Each learning iteration **do**
    Sample a batch $\mathcal{B} \subset \mathcal{D}$
    Update θ according to Equation (10)
Return θ

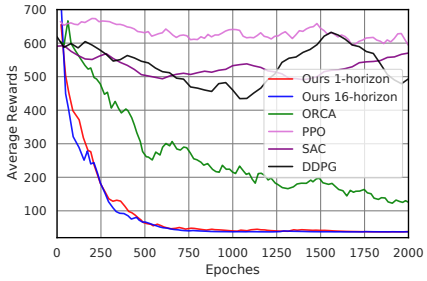

Fig. 3: On the navigation task, we evaluate the convergence history of our method with $H = 1$ and $H = 16$, the model-free baselines, and a variant of our method using ORCA to provide gradient information. The performance is measured by the average ultimate agent-to-region distance, over a fixed subset of unseen, testing problems.

The partially labeled navigation task is illustrated in Figure 7a and b. We assume there are $M = 3$ groups of agents, each of which has a distinct goal position. All the groups use the same control policy π. Each policy takes three images: 1) the agent positions of the current group, 2) the agent positions of the $M-1$ other groups, and 3) the target configuration. We use a single CNN to digest all three images. For this task, we use the reward function:

$$R(x_i^{t+\Delta t}) = \text{dist}(x_i^t) - \text{dist}(x_i^{t+\Delta t}), \tag{11}$$

where the function dist measures the distance from the agent to the target. The initial distribution $\mathcal{S}$ of problem settings involves the randomized agent and target positions, as well as a number of agents in each group. Our random sampler ensures that initial agent positions are collision-free and target positions are at least $2r$ apart. In Figure 3, we compare the performance of our method with $H = 1$ and $H = 16$, the model-free baselines, and a variant of our method using ORCA to provide gradient information. model-free methods fail by having the agents stuck among large obstacles, while learning with ORCA converges slowly. Our method with either a short- or long-horizon achieves similarly ideal performance. In

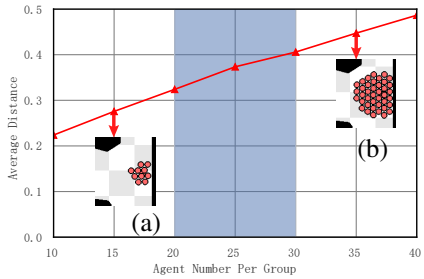

Fig. 4: We plot the average ultimate agent-to-target distance, against the number of agents in each group. All the target positions are randomized and unseen. The blue region is the agent number used during training. Even if we use a lower or higher number of agents, our policy maintains similar performance. Note the agent-to-target distance increases with the number of agents because the agents cluster around the target position due to the collision-free constraints as illustrated in (a) and (b).

Figure 4, we evaluate the learned navigation policy on unseen, randomized target positions and agent sizes. Our method generalizes to both higher and lower numbers of agents than the training setting, achieving similar policy performance.

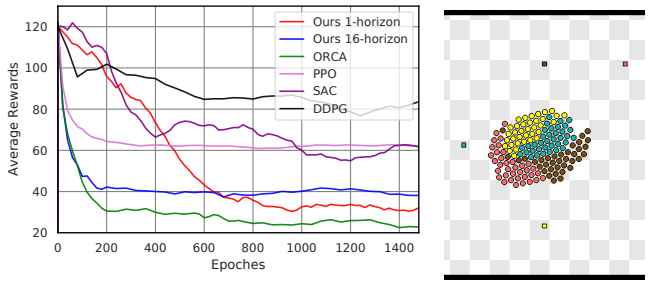

Fig. 5: Comparison as Figure 3 in a congested scenario (right).

To further highlight the benefits of our method. We setup a navigation task involving 4 groups of agents moving towards each other to reach the other side of a workspace with no obstacles. In this case, there can be many collisions among agents leading to congestion. As illustrated in Figure 5, our method trained with $H = 32$ converges to a much better policy than one trained using $H = 1$. We observe that, without obstacles, model-free methods also converge to a good policy, still inferior to our method with $H = 32$ though.

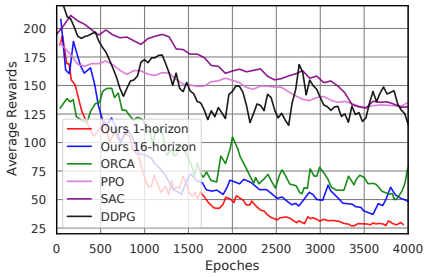

Fig. 6: Comparison as Figure 3 but on the evacuation task.

Our second evacuation task is illustrated in Figure 7c and d. We assume there are M exits in the area and the agents' goal is to reach one of the exit as fast as possible. Our reward signal is formulated as:

$$R(x_i^{t+\Delta t}) = \min_{j=1,\cdots,M} \text{dist}_j(x_i^t) - \min_{j=1,\cdots,M} \text{dist}_j(x_i^{t+\Delta t}), \tag{12}$$

where dist_j measures the distance between the agent and the jth exit. As compared with our first task, the evacuation task is more challenging by involving the exit assignment problem. In this case, our policy only takes two images: 1) the agent positions and 2) the target configuration. All other settings are the same as the partially labeled navigation task. We again compare all the baselines in Figure 6, while our method with $H = 1$ and $H = 16$ achieves a similarly ideal performance.

Task	DDPG	SAC	PPO	ORCA	Ours($H = 1$)	Ours($H = 32$)
Navigation($M = 2$)	4.206	5.518	5.223	0.775	0.488	**0.390**
Navigation($M = 3$)	4.169	4.934	5.683	1.013	0.476	**0.389**
Navigation($M = 4$)	4.681	5.222	5.510	1.812	0.500	**0.400**
Evacuation($M = 2$)	3.861	3.374	3.991	0.932	**0.631**	1.286
Evacuation($M = 3$)	1.637	1.941	2.881	0.973	**0.721**	0.820
Evacuation($M = 4$)	1.228	1.330	2.597	0.925	**0.714**	0.907

TABLE III: The average agent-to-target distance on a set of unseen testing problems, achieved by difference methods on the two tasks.

We find our navigation policy can also be generalized to different numbers of agent groups. In the first block of Table III, we train on $M = 3$ but test on $M = \{2, 3, 4\}$. The results show that our method with $H = 32$ consistently performs the best. On the other hand, the more challenging evacuation task does not allow generalization on M, so we train 3 different policies for $M = \{2, 3, 4\}$ as summarized in the second block of Table III. For the navigation task, we find our method with $H = 1$ consistently performs the best. This is because long-horizon reasoning is particularly useful for resolving congested scenarios. For the evacuation task, agents tend to find the closest exit to avoid collisions so that short-horizon training would provide more stable gradient information.

VI. Conclusion & Limitation

We present an end-to-end differentiable learning algorithm for multi-agent navigation tasks. Our method combines a novel

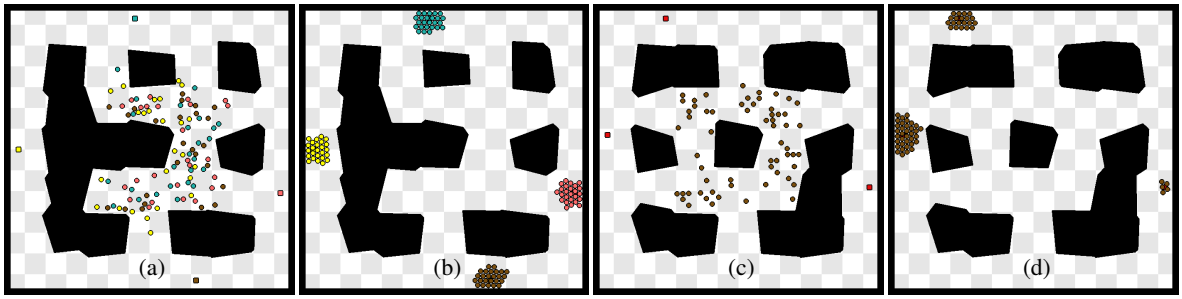

Fig. 7: (a): The initial configuration of a partially labeled navigation task with $M = 4$ groups of agents (yellow, green, brown, and red), with the final configuration shown in (b). (cd): The initial and final configurations of the evacuation task with three exits in red.

differentiable navigation simulator, a kernel-based compact policy parameterization, and a model-based learning algorithm utilizing the gradient information. On the two learning tasks, partially labeled navigation and evacuation, we show that our method outperforms the model-free RL algorithm by more than one order of magnitude. The major limitation of our method lies in the requirement of a differentiable reward function. For more complex tasks, such differentiable reward can be acquired via inverse reinforcement learning such as [49].

REFERENCES

[1] T. Fan, P. Long, W. Liu, and J. Pan, "Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios," *The International Journal of Robotics Research*, vol. 39, no. 7, pp. 856–892, 2020.

[2] Q. Tan, T. Fan, J. Pan, and D. Manocha, "Deepmnavigate: Deep reinforced multi-robot navigation unifying local & global collision avoidance," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 6952–6959.

[3] P. Xu and I. Karamouzas, "Human-inspired multi-agent navigation using knowledge distillation," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2021, pp. 8105–8112.

[4] F. Martinez-Gil, M. Lozano, and F. Fernández, "Multi-agent reinforcement learning for simulating pedestrian navigation," in *International Workshop on Adaptive and Learning Agents*, Springer, 2011, pp. 54–69.

[5] Y. A. Hasan, A. Garg, S. Sugaya, and L. Tapia, "Defensive escort teams for navigation in crowds via multi-agent deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5645–5652, 2020.

[6] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012, pp. 4906–4913.

[7] Y. Li, J. Wu, J.-Y. Zhu, J. B. Tenenbaum, A. Torralba, and R. Tedrake, "Propagation networks for model-based control under partial observation," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 1205–1211.

[8] Y. Li, H. He, J. Wu, D. Katabi, and A. Torralba, "Learning compositional koopman operators for model-based control," in *International Conference on Learning Representations*, 2020.

[9] J. Liang, M. Lin, and V. Koltun, "Differentiable cloth simulation for inverse problems," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[10] Y.-L. Qiao, J. Liang, V. Koltun, and M. C. Lin, "Efficient differentiable simulation of articulated bodies," in *International Conference on Machine Learning*, PMLR, 2021, pp. 8661–8671.

[11] T. Takahashi, J. Liang, Y.-L. Qiao, and M. C. Lin, "Differentiable fluids with solid coupling for learning and control," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 6138–6146.

[12] J. v. d. Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics research*, Springer, 2011, pp. 3–19.

[13] I. Karamouzas, N. Sohre, R. Narain, and S. J. Guy, "Implicit crowds: Optimization integrator for robust crowd simulation," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, pp. 1–13, 2017.

[14] R. L. Hughes, "The flow of human crowds," *Annual review of fluid mechanics*, vol. 35, no. 1, pp. 169–182, 2003.

[15] S. Chenney, "Flow Tiles," in *Symposium on Computer Animation*, R. Boulic and D. K. Pai, Eds., The Eurographics Association, 2004.

[16] S. Weißmann, U. Pinkall, and P. Schröder, "Smoke rings from smoke," *ACM Trans. Graph.*, vol. 33, no. 4, Jul. 2014.

[17] P. W. Koh and P. Liang, "Understanding black-box predictions via influence functions," in *International conference on machine learning*, PMLR, 2017, pp. 1885–1894.

[18] B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viegas, *et al.*, "Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav)," in *International conference on machine learning*, PMLR, 2018, pp. 2668–2677.

[19] J. Yu, "Intractability of optimal multirobot path planning on planar graphs," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 33–40, 2015.

[20] 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.

[21] J. Alonso-Mora, A. Breitenmoser, P. Beardsley, and R. Siegwart, "Reciprocal collision avoidance for multiple car-like robots," in *2012 IEEE International Conference on Robotics and Automation*, IEEE, 2012, pp. 360–366.

[22] D. Bareiss and J. van den Berg, "Generalized reciprocal collision avoidance," *The International Journal of Robotics Research*, vol. 34, no. 12, pp. 1501–1514, 2015.

[23] J. S. Grover, C. Liu, and K. Sycara, "Deadlock analysis and resolution for multi-robot systems," in *International Workshop on the Algorithmic Foundations of Robotics*, Springer, 2020, pp. 294–312.

[24] S. Dergachev and K. Yakovlev, "Distributed multi-agent navigation based on reciprocal collision avoidance and locally confined multi-agent path finding," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2021, pp. 1489–1494.

[25] T. Weerakoon, K. Ishii, and A. A. F. Nassiraei, "An artificial potential field based mobile robot navigation method to prevent from deadlock," *Journal of Artificial Intelligence and Soft Computing Research*, vol. 5, no. 3, pp. 189–203, 2015.

[26] J. E. Godoy, I. Karamouzas, S. J. Guy, and M. Gini, "Implicit coordination in crowded multi-agent navigation," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[27] A. Treuille, S. Cooper, and Z. Popović, "Continuum crowds," *ACM Transactions on Graphics (TOG)*, vol. 25, no. 3, pp. 1160–1168, 2006.

[28] L. Bottou, "Stochastic gradient descent tricks," in *Neural networks: Tricks of the trade*, Springer, 2012, pp. 421–436.

[29] T. Du, K. Wu, A. Spielberg, W. Matusik, B. Zhu, and E. Sifakis, "Functional optimization of fluidic devices with differentiable stokes flow," *ACM Trans. Graph.*, vol. 39, no. 6, 2020.

[30] Y. Li *et al.*, "Fluidic topology optimization with an anisotropic mixture model," *ACM Trans. Graph.*, 2022.

[31] Y. Hu *et al.*, "Chainqueen: A real-time differentiable physical simulator for soft robotics," in *2019 International conference on robotics and automation (ICRA)*, IEEE, 2019, pp. 6265–6271.

[32] Y. Li, T. Du, K. Wu, J. Xu, and W. Matusik, "Diffcloth: Differentiable cloth simulation with dry frictional contact," *ACM Transactions on Graphics (TOG)*, 2022.

[33] T. Du *et al.*, "Diffpd: Differentiable projective dynamics," *ACM Trans. Graph.*, vol. 41, no. 2, 2021.

[34] X. Ji, H. Li, Z. Pan, X. Gao, and C. Tu, "Decentralized, unlabeled multi-agent navigation in obstacle-rich environments using graph neural networks," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2021, pp. 8936–8943.

[35] Q. Li, F. Gama, A. Ribeiro, and A. Prorok, "Graph neural networks for decentralized multi-robot path planning," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 11 785–11 792.

[36] A. Paszke *et al.*, "Automatic differentiation in pytorch," 2017.

[37] B. Amos and J. Z. Kolter, "Optnet: Differentiable optimization as a layer in neural networks," in *International Conference on Machine Learning*, PMLR, 2017, pp. 136–145.

[38] A. Best, S. Narang, and D. Manocha, "Real-time reciprocal collision avoidance with elliptical agents," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 298–305.

[39] Y. Ma, D. Manocha, and W. Wang, "Efficient reciprocal collision avoidance between heterogeneous agents using ctmat," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '18, Stockholm, Sweden: International Foundation for Autonomous Agents and Multiagent Systems, 2018, 1044–1052.

[40] K. Guo, D. Wang, T. Fan, and J. Pan, "Vr-orca: Variable responsibility optimal reciprocal collision avoidance," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4520–4527, 2021.

[41] D. Harmon, E. Vouga, B. Smith, R. Tamstorf, and E. Grinspun, "Asynchronous contact mechanics," in *ACM SIGGRAPH 2009 papers*, 2009, pp. 1–12.

[42] A. Forsgren, P. E. Gill, and M. H. Wright, "Interior methods for nonlinear optimization," *SIAM review*, vol. 44, no. 4, pp. 525–597, 2002.

[43] A. Treuille, A. McNamara, Z. Popović, and J. Stam, "Keyframe control of smoke simulations," in *ACM SIGGRAPH 2003 Papers*, 2003, pp. 716–723.

[44] K. Polthier and E. Preuß, "Identifying vector field singularities using a discrete hodge decomposition," in *Visualization and mathematics III*, Springer, 2003, pp. 113–134.

[45] R. Fedkiw, J. Stam, and H. W. Jensen, "Visual simulation of smoke," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '01, New York, NY, USA: Association for Computing Machinery, 2001, 15–22.

[46] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[47] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, PMLR, 2018, pp. 1861–1870.

[48] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[49] J. Ho and S. Ermon, "Generative adversarial imitation learning," *Advances in neural information processing systems*, vol. 29, 2016.