Learning multi-robot coordination from demonstrations

Xuan Wang*
Electrical and Computer Engineering
George Mason University
Fairfax, VA 22030
xwang64@gmu.edu

Wanxin Jin
GRASP Laboratory
University of Pennsylvania
Philadelphia, PA, 19104
jinwx@seas.upenn.edu

Abstract

This paper develops a Distributed Differentiable Dynamic Game (DDDG) framework, which enables learning multi-robot coordination from demonstrations. We represent multi-robot coordination as a dynamic game, where the behavior of a robot is dictated by its own dynamics and objective that also depends on others’ behavior. The coordination thus can be adapted by tuning the objective and dynamics of each robot. The proposed DDDG enables each robot to automatically tune its individual dynamics and objectives in a distributed manner by minimizing the mismatch between its trajectory and demonstrations. This process requires a new distributed design of the forward-pass, where all robots collaboratively seek Nash equilibrium behavior, and a backward-pass, where gradients are propagated via the communication graph. We test the DDDG in simulation with a team of quadrotors given different task configurations. The results demonstrate the capability of DDDG for learning multi-robot coordination from demonstrations.

1 Introduction

The control and coordination of large-scale multi-robot systems has long been a challenging problem due to their combinatorial nature and the need for robots to make sequential and interactive decisions [1]. In recent years, the significant advances in machine learning have brought new solutions to these problem [2]. Perhaps the most representative approach is multi-agent reinforcement learning (MARL), which extends the traditional RL by incorporating graph and game formulations [3, 4]. Despite a wide class of successful application MARL has achieved, including two or multi player games [5, 6], multi-robot control [7], autonomous driving [8], etc., we also note the challenge of MARL methods: (i) the difficulty of designing rewards (and other individual robot parameters) for satisfactory coordination performance, and (ii) the inefficiency and lack of scalability of finding coordination policies for large-scale systems with real-world communication constraints [3, 9].

Motivated by those challenges, this paper aims to directly learn how to coordinate from desired coordinating demonstrations. We characterize the multi-robot coordination as a result of a dynamic game. Each robot, subject to its dynamics constraints, seek to optimize an objective function, which also depends on the actions of its neighboring robots. Thus, the final coordination of multiple robots is implicitly determined by the specification of the objective and dynamics of each robot. We will develop a Distributed Differentiable Dynamic Game (DDDG) framework to learn the objective function and the dynamics constraint of each robot from multi-robot coordinating demonstrations. The proposed DDDG framework features a distributed learning process, which maintains a communication topology in both forward and backward-passes.

* Preprint. Under review.
1.1 Literature review

**Multi-Agent Reinforcement Learning.** We start by reviewing MARL, which is a counterpart of the proposed DDDG framework for achieving multi-robot coordination. Inherited from single-system reinforcement learning, the goal of MARL is to search for the optimal policy of each robot to maximize the accumulated reward across the overall network [10, 11]. Under this setup, the reward of each robot must be carefully designed in a way that proper behavioral coordination among all connected robots in the system can be attained [12]. For sophisticated tasks, designing rewards in a multi-robot system is nontrivial because the resulting behavior of all robots are mutually coupled. Failure on properly assigning the reward function can significantly degrade the whole system coordination preference [13]. On the other hand, to update policy, MARL typically assumes all robots in the system must be able to observe the full state of the overall system [3]. Such a requirement is undesirable in engineering practice when large numbers of robots and limited budget of communication are involved. To enable policy training under partial observation, another formulation called extensive-form games [14, 15] was proposed. The implementation of this approach requires full historical trajectories of robots, and thus could increase the training difficulty of the policies [16]. There also exist off-policy RL algorithms, which rely on greedy actions that may not be optimal for the long run [17]. It must be pointed out although we identified some potential challenges for implementing MARL, we are fully aware its significant advances in recent years, such as multi-robot control [7], autonomous driving [8], etc. Since the proposed DDDG is able to automate the design of rewards in a multi-robot coordinate systems, it thus can be considered as a complementary approach to general MARL, which can then be used to derive the optimal robot coordination.

**Inverse Optimal Control.** Another category of methods relevant to the proposed framework is inverse optimal control (IOC) [18, 19], in some contests also known as inverse reinforcement learning (IRL) [20]. IOC/IRL focuses on learning the objective (cost or reward) function of a decision-making system from optimal behavior of expert. Various criteria have been proposed to update the objective function, including maximum margin [21], maximum entropy [22] and minimal optimality violation [23]. However, existing results in these fields are insufficient due to two major reasons. First, in multi-robot settings, neighboring robots will have coupled terms in their objective functions. Since the forward and inverse optimal control are designed for only monolithic (centralized) systems [18, 20–24], they do not take into account the interaction among robots’ collective behaviors under communication topology constraint. Second, existing IOC algorithms typically rely on strict mathematical assumptions, such as given knowledge of robot dynamics models, linearizable weights on selected feature spaces [21–24]. However, these assumptions are unrealistic for heterogeneous multi-robot systems with real world applications. Most recently, a Pontryagin differentiable programming approach [25] has been proposed, which relaxes these assumptions based on the idea of differentiating the optimality condition of a robot’s trajectory.

**Non-cooperative Dynamic Game and its Inverse Problem.** Non-cooperative dynamic game generalizes optimal control to a multi-robot setup [26]. Its nature is a strategic interaction, where the robots’ have coupled payoff functions, and each robot tries to predict the others’ actions in order to optimize its own payoff. The set of robots’ choices, when no one can improve its payoff by changing its strategy, constitutes the solution to the game called Nash equilibrium. Common approaches to obtaining a Nash equilibrium include: computing the equilibrium directly from optimality conditions [27], or designing an algorithm whose dynamics asymptotically converge to the desired Nash equilibrium [28]. Analogous to optimal control and IOC, non-cooperative dynamic game also has an inverse problem, i.e., given robots collective trajectories satisfying a Nash equilibrium, one inversely learns what game the robots aim to solve [29, 30]. The proposed framework is inspired by both the forward and inverse problems of the non-cooperative dynamic game. Nevertheless, the state of the art in these fields is insufficient to serve our demands due to multiple notable differences. On one hand, existing approaches for solving dynamic games assume the robots’ dynamics are given a priori [31–33]. This assumption is not applicable to heterogeneous robots whose dynamics are subject to uncertainties or undetermined parameters. On the other hand, existing results on inverse dynamic game are built on a centralized structure [29, 30, 34, 35]. This requirement causes scalability issues for the large-scale multi-robot system, where transmitting and processing all data at a central location burdens the communication, storage, and computation capabilities of the system.
2.1 A parameterized dynamic game model for distributed multi-robot coordination

We formulate a parameterized dynamic game model from a bottom-up perspective. Consider a system of $m$ robots. Suppose each robot solves its own optimal control problem $P_i(\theta_i)$, which is parameterized by a vector $\theta_i \in \mathbb{R}^{r_i}$ as follows:\footnote{For the simplicity of notation, we only keep $\theta_i$ as the argument of $J_i(\cdot)$ and $f_i(\cdot)$, by omitting all $x$ and $u$.}

\[
\begin{align*}
\min & \quad J_i(\theta_i) = \sum_{t=0}^{T-1} c_i^t(x_i^t, u_i^t, x_{N_i}^t, \theta_i) + h_i(x_i^T, x_{N_i}^T, \theta_i), \\
\text{s.t.} & \quad x_{i}^{t+1}(\theta_i) = f_i(x_i^t, u_i^t, \theta_i) \quad \text{with given } x_i^0.
\end{align*}
\]

Here, $t \in \{0, 1, \cdots, T - 1\}$ is the time step with $T$ being the time horizon. $f_i(\cdot) \in \mathbb{R}^{n_i}$ is the dynamics of robot $i$ with $x_i^t \in \mathbb{R}^{n_i}$ and $u_i^t \in \mathbb{R}^{m_i}$ being the robot’s state and control input at each time step. $J_i(\cdot) \in \mathbb{R}$ is the control objective function with $c_i^t(\cdot) \in \mathbb{R}$ and $h_i(\cdot) \in \mathbb{R}$ denoting the running and final costs, respectively. We assume both $f_i(\cdot)$ and $J_i(\cdot)$ are twice differentiable. To characterize the fact that connected robots have interactive behaviors, we set their objective functions to be coupled, i.e., $J_i(\cdot)$ depends not only on the state/input of robot $i$, but also on that of its neighbors, denoted by $x_{N_i}^t = \{x_j^t | j \in N_i\}$, with $N_i$ being the neighbor set of robot $i$. The neighborhoods of robots define the communication topology of the whole system. Such communication topology can be formulated by the edge set of a graph $\mathcal{G}$ whose vertices are associated with the robots. For the notation convenience, let $x_i = \{x_i^0, \cdots, x_i^T\}$ and $u_i = \{u_i^0, \cdots, u_i^{T-1}\}$ denote the trajectories of system states, and control inputs for robot $i$, respectively. Further define $\xi_i = \{x_i, u_i\}$, which represents the full trajectory of robot $i$.

Given $P_i(\theta_i)$, we use its solution (minimizer) to characterize the behavior of robot $i$. Followed by this, the collective behavior of the overall multi-robot systems is characterized by the composition of $P_i(\theta_i)$ in all robots, which forms a dynamic game $P(\Theta)$ parameterized by $\Theta = \col \{\theta_1, \cdots, \theta_m\} \in \mathbb{R}^r$ with $r = \sum_{i=1}^{m} r_i$. Since the objective functions of $P_i(\theta_i)$’s are mutually coupled, each robot must take into account the actions of its neighbors before determining its own behavior. Such strategic interaction is characterized by the solution of the game $P(\Theta)$, which is a special set of state-input-trajectories $\xi_i^*(\Theta) = \{x_i^*(\Theta), u_i^*(\Theta)\}$ for all $i \in \{1, \cdots, m\}$, satisfying the condition called Nash equilibrium

\[
J_i(\xi_i^*(\Theta), \xi_{N_i}^*(\Theta), \theta_i) \leq J_i(\xi_i, \xi_{N_i}^*(\Theta), \theta_i) \quad \text{for any } \xi_i \in \Xi_i(\theta_i). \quad \text{(N.E.)}
\]

Here, $\Xi_i(\theta_i)$ is the set of all feasible trajectories of robot $i$ satisfying its initial condition and system dynamics. $\Xi_i(\theta_i)$ is a function of $\theta_i$ because the dynamics $f_i(\cdot)$ is parameterized by $\theta_i$. In (N.E.), we slightly abuse the notation of $J_i(\cdot)$ by writing its arguments as $\xi_i$, $\xi_{N_i}$, and $\theta_i$. Thus, $P(\Theta)$ is a tunable...
model whose (N.E.) characterizes the coordination of the multi-robot system. By finding proper Θ, this model has the capability of generating various types of distributed multi-robot coordination schemes, as we explain next.

**Remark 2.1.** (Parameterized dynamic game for distributed multi-robot coordination.) The parameterized dynamic game \( P(\Theta) \) has the following favorable properties.

(i) **Versatile coordination:** Due to the coupled objective function, even if a single robot adjusts the parameters of its local optimal control problem, the overall Nash equilibrium will change accordingly. This ensures the versatility of the model to characterize complex multi-robot coordination schemes.

(ii) **Distributed coordination:** \( P(\Theta) \) is a non-cooperative game as it does not rely on coalitions. However, by specifying mutually beneficial objective structures, it can lead to fully distributed coordination strategies that lead all robots to accomplish a global task.

(iii) **Learning-based coordination:** Since both \( f_i \) and \( J_i \) are parameterized by \( \theta_i \), learning the parameterized dynamics allows the robots to identify their unknown dynamical features then precisely control their motions; learning the parameterized coupled objective functions allows the robots to determine their coupled interest then achieve adaptive distributed coordination as a team. To respect heterogeneity, we allow different robots to have different objective functions and system dynamics.

\[ \square \]

### 2.2 Problem formulation

Built on the dynamic game model \( P(\Theta) \) with \( \Theta = \text{col} \{ \theta_1, \cdots, \theta_m \} \) being tunable, our goal in this paper is to learn a proper \( \Theta^* \) in the whole system, i.e., the \( \theta_i^* \) for each robot, from a set of coordinating demonstrations \( \{ \xi_1^d, \xi_2^d, \cdots, \xi_m^d \} \) of robots, \( d \in \{1, \cdots, D\} \) that are desirable for accomplishing a global task. To technically formulate our problem of interest, we make the following assumptions:

**Assumption 1.** The following statement holds. For any \( \Theta \in \mathbb{R}^r \), the Nash equilibrium of \( P(\Theta) \) exists and is unique.

**Remark 2.2.** (Justification of Assumption 1.) In this paper, we will iteratively learn \( \Theta \) from an initial guess to the desired \( \Theta^* \), thus, the first statement in Assumption 1 ensures the existence and uniqueness of (N.E.) through out this process. As sufficient conditions [36], the existence can be guaranteed by the convexity of \( J_i \) on \( x_i, x_N, \) and \( u_i \); the uniqueness can then be guaranteed by the strictly convexity of such function.

Building on Assumption (1), our goal is to learn the desired parameter \( \Theta^* \) such that the generated trajectories by \( P(\Theta^*) \) try to mimic the given demonstrations. We will do this in a distributed fashion by introducing the following loss function at each robot,

\[
L_i(\xi_i^*(\Theta), \xi_i^d) = \sum_{d=1}^{D} ||\xi_i^*(\Theta) - \xi_i^d||_2^2 \]

which characterizes the inconsistency between the samples \( \xi_i^d \) and the optimal trajectory \( \xi_i^*(\Theta) \) reconstructed from the Nash equilibrium of \( P(\Theta) \) by using a same initial condition as \( \xi_i^d \). Note that apart from using the least square formulation for the loss function, other types of loss function can be found in the following references based on maximum margin [21], maximum entropy [22] and optimality violation [23]. In this paper, we consider the robot’s trajectories at each time instant to be equally important, therefore use the Euclidean distance to characterize the mismatch. With this loss function, we formulate the following problem of interest.

**Problem 1.** Suppose Assumption 1 holds. Given data samples \( \{ \xi_1^d, \xi_2^d, \cdots, \xi_m^d \} \), \( d \in \{1, \cdots, D\} \). We let each robot \( i \) locally knows \( P_i(\theta_i) \), \( L_i(\cdot) \), and \( \xi_i^d \). We seek to develop a fully distributed algorithm that learns the parameter \( \Theta^* = \text{col} \{ \theta_1^*, \cdots, \theta_m^* \} \) by minimizing the loss function in all robots, i.e.,

\[
\Theta = \text{col} \{ \theta_1, \cdots, \theta_m \} \quad \min \sum_{i=1}^{m} L_i(\xi_i^*(\Theta), \xi_i^d). \]

### 3 Distributed Differentiable Dynamic Game Framework

In this section, we introduce a distributed framework, named **Distributed Differentiable Dynamic Game**, to solve Problem 1. A visualization of this framework is provided in Fig. 1.
An intuitive idea for solving Problem 1 is to update \( \theta \) based on its full gradient with respect to equation (2), i.e., \( \sum_{i=1}^{m} \frac{dL_i}{d\theta_i} \). However, this approach suffers from scalability issues because for each robot, the gradients need to be evaluated for all \( L_i(\xi^*_i(\theta), \xi^*_j) \), \( \ell \in \{1, \ldots, m\} \). In this paper, we introduce the local gradient descent in (3), which has better tractability but may require stricter step-sizes choices to ensure stability. Actually, the idea of local gradient is implicit in many RL algorithms. A typical example is the well-known actor-critic method, which individually applies gradient descent to update the actor model and the critic model [38]. Another justification of (3) can be found in [39, 40], which presents an exponential decay property characterizing how the distance between two agents affects their dependency in a multi-agent system. Its interpretation under our formulation means \( \frac{dL_i}{d\theta_i} \) decays exponentially as the distance between \( i \) and \( j \) in \( G \) grows large. Finally, recall that the global and local loss functions defined in (1) and (2) are both non-negative. Thus, they share the same minimizer at 0, when the demonstrations and the generated trajectories are perfectly matched. In the presence of measurement noise that the trajectories do not perfectly match, the loss function in each robot seeks

\[
\theta^{k+1} = \theta^k - \eta^k \frac{dL_i(\xi^*_i(\theta), \xi^*_j)}{d\theta_i} \bigg|_{\theta^k}
\]

where

\[
\frac{dL_i(\xi^*_i(\theta), \xi^*_j)}{d\theta_i} \bigg|_{\theta^k} = \frac{\partial L_i(\xi^*_i(\theta), \xi^*_j)}{\partial \xi^*_i(\theta)} \cdot \frac{\partial \xi^*_i(\theta)}{\partial \theta_i} \bigg|_{\theta^k},
\]

\( \eta^k \) is the learning rate, and the chain rule arises naturally from the definition of \( L_i(\cdot) \). By iteratively executing update (3), Algorithm 1 summarizes the Distributed Differentiable Dynamic Game framework for solving Problem 1. It is called differentiable because the second term of the chain rule (4) differentiates the optimal trajectory corresponding to the learning parameter \( \theta \). The mechanism of such differentiation will be presented in Subsection 3.3. In line with Fig. 1, the step 4 of Algorithm 1 is the forward-pass of the figure, and steps 5-6 are the backward-pass. To implement this algorithm, the computationally challenging parts are the forward-pass in step 4: solving the Nash equilibrium of the dynamic game, and the forward-pass in step 6: evaluating the change of parameter \( \theta \) to its optimal trajectory \( \frac{d\xi^*_i(\theta)}{d\theta} \). Later, we will develop distributed solvers to address both passes. For termination criteria (cf. step 3), we write it in terms of a threshold for all robots. This can be easily achieved by running a search on an indicator and the depth of the search equals to the diameter of the graph \( G \). Step 3 can also be replaced by a distributed termination criterion [37] with propagate the indicator with nominal communication overheads.

**Remark 3.1.** *(Full gradient v.s. local gradient):* An intuitive idea for solving Problem 1 is to update each \( \theta \), based on its full gradient with respect to equation (2), i.e., \( \sum_{i=1}^{m} \frac{dL_i}{d\theta_i} \). However, this approach suffers from scalability issues because for each robot, the gradients need to be evaluated for all \( L_i(\xi^*_i(\theta), \xi^*_j) \), \( \ell \in \{1, \ldots, m\} \). In this paper, we introduce the local gradient descent in (3), which has better tractability but may require stricter step-sizes choices to ensure stability. Actually, the idea of local gradient is implicit in many RL algorithms. A typical example is the well-known actor-critic method, which individually applies gradient descent to update the actor model and the critic model [38]. Another justification of (3) can be found in [39, 40], which presents an exponential decay property characterizing how the distance between two agents affects their dependency in a multi-agent system. Its interpretation under our formulation means \( \frac{dL_i}{d\theta_i} \) decays exponentially as the distance between \( i \) and \( j \) in \( G \) grows large. Finally, recall that the global and local loss functions defined in (1) and (2) are both non-negative. Thus, they share the same minimizer at 0, when the demonstrations and the generated trajectories are perfectly matched. In the presence of measurement noise that the trajectories do not perfectly match, the loss function in each robot seeks
Algorithm 1: Distributed Differentiable Dynamic Game (DDDG) Algorithm.

1. Input $\xi^i_t, d \in \{1, \ldots, D\}$.
2. Initialize $k = 0, \theta^k = 0$ as a random parameter for $P_i(\theta_i)$.
3. while $\max(\theta^{k+1}_i - \theta^k_i) \geq \epsilon_\theta$ for $i \in \{1, \ldots, m\}$ do
   // Termination criteria.
   4. Solve the game $P(\Theta^k)$ based on Algorithm 2 with current parameter $\Theta^k$ to obtain its Nash equilibrium $\xi^*_i(\Theta^k)$ for robot $i$; // The forward-pass in Fig. 1.
   5. Compute $\frac{\partial L_i}{\partial \xi^*_i(\Theta)} \big|_{\xi^*_i(\Theta^k)}$ based on $\xi^*_i(\Theta^k)$ and the definition of $L_i(\xi^*_i(\Theta), \xi^*_i)$;
      // backward-pass (a).
   6. Compute $\frac{\partial \xi^*_i}{\partial \Theta}$ based on a distributed solver described in Algorithm 3;
      // backward-pass (b).
   7. $\theta^{k+1}_i = \theta^k_i - \eta^k \left( \frac{\partial c^i_0(\Theta)}{\partial \xi^*_i(\Theta)} \big|_{\xi^*_i(\Theta^k)}, \frac{\partial c^i_1(\Theta)}{\partial \Theta} \big|_{\Theta^k} \right)$; // Parameter update.
   8. $k = k + 1$
9. end
10. Output $\theta^*_i = \theta^k_i$.

3.2 Distributed Shooting-based Nash-Equilibrium Solver

A key process in Algorithm 1 is to solve the Nash equilibrium of the dynamic game with current parameter $\Theta^k$ (cf. step 4 of Algorithm 1, the forward-pass of Fig. 1). To facilitate this, we develop a distributed shooting-based Nash-equilibrium seeking approach in Algorithm 2.

To understand the propose algorithm, note that the Nash equilibrium of the non-cooperative game is a collection of the optimal trajectories of the robots’ local optimal control problem. Thus, for any $\Theta$, the trajectory $\xi^*_i(\Theta) = \{x^i_{0:T}(\Theta), u^i_{0:T-1}(\Theta)\}$ can be characterized by a distributed discrete-time Pontryagin’s Minimum Principle (PMP) [41] condition, which reads:

robot dynamics: $x^i_{t+1}(\Theta) = \frac{\partial H^i_1}{\partial \lambda^i_{t+1}(\Theta)} = f_i(x^i_t(\Theta), u^i_t(\Theta), \theta_i)$ \hspace{1cm} (5a)

system input: $0 = \frac{\partial H^i_1}{\partial u^i_t(\Theta)} = \frac{\partial c^i_0}{\partial u^i_t(\Theta)} + \frac{\partial f^i_0}{\partial u^i_t(\Theta)} \lambda^i_{t+1}(\Theta)$ \hspace{1cm} (5b)

costate equation: $\lambda^i_{t+1}(\Theta) = \frac{\partial H^i_1}{\partial x^i_{t+1}(\Theta)} = \frac{\partial c^i_1}{\partial x^i_{t+1}(\Theta)} + \frac{\partial f^i_1}{\partial x^i_{t+1}(\Theta)} \lambda^i_{t+1}(\Theta)$ \hspace{1cm} (5c)

boundary condition: $\lambda^i_0(\Theta) = \frac{\partial h_i}{\partial x^i_{T+1}(\Theta)}$, $x^i_0(\Theta) = x^i_{T+1}$ \hspace{1cm} (5d)

Here,

$$H^i_1(x^i_t, u^i_t, x^i_{N}, \theta_i, \lambda^i_{t+1}) = c^i_1(x^i_t, u^i_t, x^i_{N}, \theta_i) + f_i(x^i_t, u^i_t, \theta_i)^T \lambda^i_{t+1}$$ \hspace{1cm} (6)

is the Hamiltonian equation for robot $i$ at each time step. For notation simplicity, we use $\frac{\partial G}{\partial \sigma^*_t}$ to denote the derivative of function $G(\sigma)$ with respect to $\sigma$ evaluated at $\sigma^*_t$.

Lemma 3.2. (Nash-equilibrium seeking based on distributed PMP condition): The distributed discrete-time PMP condition (5) is necessary for (N.E.). Furthermore, if the Hamiltonian’s Hessian matrix corresponding to $u^i_t$ is positive definite for all time steps, the condition is sufficient and the equilibrium of Algorithm 2 solve the (N.E.) of the dynamic game.
A proof of this lemma is given in appendix. Equation (5) is essentially a special use of Lagrange multipliers [42] which validates the optimality conditions for state-input pairs of the robot in a time-iterative fashion from 0 : T. With this optimality condition (5), our algorithm 2 consider only the control trajectory \( u^*_i = \{u^t_i=0, u^1_i, \cdots, u^{(T-1)}_i \} \) as the variable (with \( \tau \) being the time-step) and bring it into (5-a-c,d) to compute (shoot) the system state, co-state trajectories. Then we use equation (5-b) to evaluate the correction term \( \Delta u^*_i \) to update \( u^*_i \). Note that the shooting process happens interactively among all robots, as they have to acquire their neighbors state \( x^j_N \), in step 6 to evaluate coupled costs \( c^j_i \) and \( h_i \). Fig. 2 validates the effectiveness of Algorithm 2, as we use it to solve a dynamic game among 4 robots for formation initialization. In this figure, the vertical axis characterizes the mismatch between the results of Algorithm 2 and the exact Nash equilibrium of the game. We tested three trials. It can be observed that the proposed shooting-based algorithm converge to the exact Nash equilibrium, which validates the effectiveness of the algorithm. More details about the simulation setup will be provides in Sec. 4.

![Figure 2: Employing Algorithm 2 to solve the Nash equilibrium.](image)

**Algorithm 2:** Distributed Shooting-based Nash-Equilibrium seeking for Dynamic Games.

1. **Input** \( \theta_k^i \) and \( u^*_i(\Theta^{k-1}) \), if \( k \geq 1 \).
2. **Initialize** \( \tau = 0 \). If \( k \geq 1 \), set \( u^t_i=0 \leftarrow u^*_i(\Theta^{k-1}) \); if \( k = 0 \), set \( u^t_i=0 \) as a random trajectory of robot \( i \)’s control input.
3. **repeat**
   4. \( \{u^t_i=0, u^1_i, \cdots, u^{(T-1)}_i \} \leftarrow u^*_i \); // Unpack control input sequence.
   5. Starting from the initial state \( x_i^{t=0} \), iterate the robot dynamics equation (5a) to compute \( x_i^{t+1} \) for all \( t = 0, \cdots, (T-1) \), based on \( x^t_i, u^t_i, \theta^k_i \); // Compute state trajectory.
   6. Communicate within each robot’s neighbor set \( N_i \), and acquire their system states \( x^j_N \) for all \( t = 1, \cdots, T \).
   7. Compute the final co-state \( \lambda^T_i \) based on \( x^T_i \) and \( \theta^k_i \) following the boundary condition (5d).
   8. Then reversely iterate co-state equation (5c) to compute \( \lambda^t_i \) for all \( t = T, \cdots, 1 \), based on \( x^t_i, x^j_N, \lambda^{t+1}_i, \theta^k_i \); // Compute co-state trajectory.
   9. Compute \( \Delta u^t_i = \frac{\partial c^j_i}{\partial u^t_i} + \frac{\partial f^t_i}{\partial u^t_i} \lambda^{t+1}_i \) for all \( t = 0, \cdots, (T-1) \); // Evaluate the optimality condition (5b).
10. Define \( \Delta u^*_i = \{\Delta u^t_i=0, \Delta u^t_i=0, \cdots, \Delta u^{(T-1)}_i \} \)
11. \( u^{t+1}_i = u^t_i - \gamma^t \Delta u^*_i \); // Update control input sequence.
12. \( \tau \leftarrow \tau + 1 \)
13. **until** \( \| \Delta u^*_i \| \leq \epsilon_u, \forall i \in \{1, \cdots, m\} \).
14. **Output** \( \xi^*_i(\Theta^k) = \{u^*_i(\Theta^k), x^*_i(\Theta^k)\} \).

Apart from Algorithm 2, there exist other distributed Nash-equilibrium seeking algorithms for non-cooperative games [43–46]. In this paper, the specialties of dynamic games arise from the dynamics constraint of each robot, which is imposed on robot trajectories (state-input pairs). This fact precludes the use of method in [43] which does not consider constraints. In addition to this, the results in [44–46] employ the following projected flow to handle constraints, which guarantees the
With Algorithm 2 to facilitate the forward-pass of Fig. 1, another key component of Algorithm 1 is to introduce \( \sigma \) \( x \) into a new equation set that describes the derivatives of these trajectories corresponding to the \( x \) \( \sigma \) and to automatically satisfy the robots’ dynamics constraints. Furthermore, the update (7) has a large variable dimension as \( \xi \) includes system states \( x_i \), inputs \( u_i \), and a latent co-states \( \lambda_i \), adding extra difficulties on its implementation. These challenges necessitate the development of Algorithm 2, where we use a shooting mechanism (c.f. steps 5-7) to reduce the variable dimension and to automatically satisfy the robots’ dynamics constraints.

3.3 Distributed Gradient Solver.

With Algorithm 2 to facilitate the forward-pass of Fig. 1, another key component of Algorithm 1 is computing the gradient of \( \theta \) following the chain rule (4) (cf. steps 5-6 of Algorithm 1, the backward-pass of Fig. 1). In (4), \( \frac{\partial \mathcal{L}_i}{\partial \theta} \bigg|_{\xi^i(\theta^i)} \) is readily accessible because the function \( \mathcal{L}_i(\xi^i(\Theta), \xi^i) \) is explicitly defined, and the evaluation points are known based on Algorithm 2. The major obstacle arises from the second half of the chain rule, where \( \frac{\partial \xi^i(\Theta)}{\partial \theta} \) characterizes the change of the robot’s optimal trajectory corresponding to the change to its local parameter. Given a general optimal control system, since its solution trajectory \( \xi^i(\Theta) \) has no analytical form, one possible way to compute \( \frac{\partial \xi^i(\Theta)}{\partial \theta} \) is by numerical approximation [47]. However, the feasibility of this approach is thwarted by the large number of robots and the complexity of their trajectories considered in this paper. Motivated by these, we next present a new method to compute \( \frac{\partial \mathcal{C}(\Theta)}{\partial \theta} \) in Algorithm 3, whose idea is based on differentiating the Pontryagin’s Minimum Principle (5) along the parameter \( \Theta \) [25]. This converts the original PMP equations describing the optimality condition of the trajectories of the system into a new equation set that describes the derivatives of these trajectories corresponding to the parameter \( \Theta \). The differential PMP reads:

\[
X_i^{t+1} = M_i^x X_i^t + N_i^x U_i^t + C_i^x
\]

(8a)

\[
0 = M_i^u X_i^t + N_i^u U_i^t + \sum_{j \in N_i} Q_{u,i,j}^t X_j^t + S_{u,i} \Lambda_i^{t+1} + C_i^u
\]

(8b)

\[
\Lambda_i^t = M_i^\Lambda X_i^t + N_i^\Lambda U_i^t + \sum_{j \in N_i} Q_{\sigma,i,j}^t X_j^t + S_{\sigma,i} \Lambda_i^{t+1} + C_i^\Lambda
\]

(8c)

\[
\Lambda_i^T = M_i^T X_i^T + C_i^T, \quad X_i^0 = 0
\]

(8d)

where for the left-hand side, we use the following definitions to simplify the notations:

\[
X_i^t = \frac{\partial x^t(\Theta)}{\partial \Theta} \in \mathbb{R}^{n_i \times r}, \quad U_i^t = \frac{\partial u^t(\Theta)}{\partial \Theta} \in \mathbb{R}^{m_i \times r}, \quad \Lambda_i^t = \frac{\partial \lambda_i^{t+1}(\Theta)}{\partial \Theta} \in \mathbb{R}^{n_i \times r}
\]

(9)

By comparing the right-hand sides of (5) and (8), note that the arguments of the Hamiltonian equation include \( x_i, u_i, x_{N_i}, \lambda_i \), whose values depend on the choice of parameter \( \Theta \). For simplicity, we introduce \( \sigma_i \in \{ x_i, u_i, \lambda_i \} \) as a common representation for these variables, then it follows:

\[
\frac{\partial \left( \frac{\partial H_i}{\partial \sigma_i} \right)}{\partial \Theta} = M_{i,\sigma_i} \frac{\partial x_i^t(\Theta)}{\partial \Theta} + N_{i,\sigma_i} \frac{\partial u_i^t(\Theta)}{\partial \Theta} + \sum_{j \in N_i} Q_{i,\sigma_i,j}^t \frac{\partial x_j^t(\Theta)}{\partial \Theta} + S_{i,\sigma_i}^t \frac{\partial \lambda_i^{t+1}(\Theta)}{\partial \Theta} + C_i^t
\]

(10)

where the matrices are defined as

\[
M_{i,\sigma_i} = \frac{\partial^2 H_i}{\partial \sigma_i \partial x_i^t}, \quad N_{i,\sigma_i} = \frac{\partial^2 H_i}{\partial \sigma_i \partial u_i^t}, \quad Q_{i,\sigma_i,j}^t = \frac{\partial^2 H_i}{\partial \sigma_i \partial x_j^t}, \quad S_{i,\sigma_i}^t = \frac{\partial^2 H_i}{\partial \sigma_i \partial \lambda_i^{t+1}}, \quad C_i^t = \frac{\partial^2 h_i}{\partial x_i^t \partial \Theta}, \quad M_i^T = \frac{\partial^2 h_i}{\partial x_i^T \partial \Theta}, \quad C_i^T = \frac{\partial^2 h_i}{\partial x_i^T \partial \Theta}
\]

(11)

satisfactory of robot’s system dynamics,

\[
\xi_i^{t+1} = \mathcal{P}_{\Xi_i(\theta_i^t)} \left( \xi_i^t - \gamma I \frac{\partial \mathcal{J}_i}{\partial \xi_i} \right)
\]

(7)

where \( \mathcal{P}_{\Xi_i(\theta_i^t)} \) is the Euclidean projection to the set \( \Xi_i(\theta_i^t) \) of all system feasible trajectories. However, for non-linear dynamics, implementing \( \Xi_i(\theta_i^t) \) is computationally difficult. Furthermore, the update (7) has a large variable dimension as \( \xi_i \) includes system states \( x_i \), inputs \( u_i \), and a latent co-states \( \lambda_i \), adding extra difficulties on its implementation. These challenges necessitate the development of Algorithm 2, where we use a shooting mechanism (c.f. steps 5-7) to reduce the variable dimension and to automatically satisfy the robots’ dynamics constraints.
and we use $\frac{\partial^2 H_i^T}{\partial x_i^T \partial x_i^T}$ to denote the second-order derivative of $H_i^T(\cdot)$ evaluated at $\{\sigma_i^T(\Theta), x_i^T(\Theta)\}$.

Note that all $M, N, Q, S, C$ matrices in (8) are readability computable from (11), based on the facts that the function $H_i^T(\cdot)$ is explicitly defined, the evaluation points $\{x_i^T(\Theta), u_i^T(\Theta)\}$ are obtained from Algorithm 2 as $\xi_i^T(\Theta)$, and $\Lambda_i^T(\Theta)$ can be obtained from equations (5b-c). Since (8) is a linear equation set, solving it gives us the required gradients $\frac{\partial E_i^T(\Theta)}{\partial q_i^k}$ in (4). Next, we present how this equations can be solved in a distributed manner.

A distributed gradient solver

We start by assembling (8) into a compact form for all time instances $t = 0 : T$,

\[
\begin{align*}
\mathcal{M}_i \mathbf{X}_i^{0:T} + \mathcal{N}_i \mathbf{U}_i^{0:T-1} + \sum_{j \in \mathcal{N}_i} (\mathcal{Q}_{i,j} \mathbf{X}_j^{1:T}) + \mathcal{S}_i \Lambda_i^{0:T} + \mathcal{C}_i &= 0 \quad (12)
\end{align*}
\]

where $\mathcal{M}_i, \mathcal{N}_i, \mathcal{Q}_{i,j}, \mathcal{S}_i$, and $\mathcal{C}_i$ are assembled from the corresponding matrices in (8). For example, the matrix $\mathcal{M}_i$ is composed of $M_{A,i}, M_{u,i}, M_{x,i}$ for $t \in \{1, \cdots, T-1\}$ and $M_{R,i}^T$. To continue, define

\[
\mathbf{Y}_i = \begin{bmatrix} \mathbf{X}_i^{0:T} \\ \mathbf{U}_i^{0:T-1} \\ \Lambda_i^{0:T} \end{bmatrix}, \quad \mathbf{A}_{i,i} = \begin{bmatrix} \mathcal{M}_i & \mathcal{N}_i & \mathcal{S}_i \end{bmatrix}, \quad \mathbf{A}_{i,j} = \begin{bmatrix} \mathcal{Q}_{i,j} & 0 & 0 \end{bmatrix} \quad \text{for } j \in \mathcal{N}_i. \quad (13)
\]

Then equation (12) can be further rewritten as,

\[
\mathbf{A}_{i,i} \mathbf{Y}_i + \sum_{j \in \mathcal{N}_i} (\mathbf{A}_{i,j} \mathbf{Y}_j) + \mathbf{C}_i = 0. \quad (14)
\]

where $\mathbf{Y}_i$ is the local unknown of robot $i$, $\mathbf{A}_{i,i}$ and $\mathbf{A}_{i,j}$ are known matrices, and $\mathbf{Y}_j, j \in \mathcal{N}_i$ is the coupled unknown from $i$’s neighbors. Since each robot in the network possesses an equation in the form of (14), to compute a set of $\mathbf{Y}_i, i \in \{1, \cdots, m\}$ satisfying all these equation, we essentially need to solve the following compact equation set

\[
\sum_{i=1}^{m} (\mathbf{\Psi}_i \mathbf{Y}_i + \mathbf{\hat{C}}_i) = 0 \quad (15)
\]

where

\[
\mathbf{\Psi}_i = \begin{bmatrix} \mathbf{A}_{1,i}^T & \mathbf{A}_{2,i}^T & \cdots & \mathbf{A}_{m,i}^T \end{bmatrix}^T, \quad \mathbf{\hat{C}}_i = \begin{bmatrix} 0 & \cdots & \mathbf{C}_{i}^T & \cdots & 0 \end{bmatrix}^T. \quad (16)
\]

In $\mathbf{\Psi}_i$, the matrices $\mathbf{A}_{\ell,i}, \ell \in \{1, \cdots, m\}$ follows the definition in (13) if $i \in \mathcal{N}_\ell$ and $\mathbf{A}_{\ell,i} = \mathbf{0}$ otherwise. In $\mathbf{\hat{C}}_i$, the matrix $\mathbf{C}_{i}^T$ is located at the $i$th block. From the definition of (16), by stacking the matrices $\mathbf{A}_{\ell,i}$ and $\mathbf{C}_{i}^T$, each row block of (15) is essentially associated with one (14) for $i \in \{1, \cdots, m\}$. Further note that network $\mathcal{G}$ is undirected, i.e., $i \in \mathcal{N}_\ell$ yields $\ell \in \mathcal{N}_i$, thus, robot $i$ has access to $\mathbf{\Psi}_i$ based on its local communication with its neighbors. Now, suppose each robot $i$ knows $\mathbf{\Psi}_i$ and $\mathbf{\hat{C}}_i$, we introduce Algorithm 3 for the agents to efficiently solve its $\mathbf{Y}_i$.

Algorithm 3 is fully distributed, in the sense that the computation of each robot only rely on its own state and the states of its neighbors. Fig. 3 shows the effectiveness of Algorithm 3, as we use it to solve the differential PMP condition associated with 4 UAVs for formation initialization. In this figure, the vertical axis characterizes the mismatch between the results of Algorithm 3 and the exact solution to the equation set (15). We tested three trials. It can be observed that the proposed distributed gradient solver converge exponentially to the exact solution, which validates the effectiveness of the algorithm. More details about the simulation setup will be provides in Sec. 4. Such convergence can be theoretically characterized by the following result.

![Figure 3: Employing Algorithm 3 to solve the differential PMP condition.](image)
Algorithm 3: Distributed Differential Pontryagin’s Minimum Principle (PMP) solver.

1. **Input** $\xi^i_*(\Theta^k), \theta^k$.
2. **Compute** $X_{i}^{0:T}(\Theta^k)$ using equations (8b-c) with $\xi^i_*(\Theta^k) = \{x_{i}^{0:T}(\Theta^k), u_{i}^{0:T-1}(\Theta^k)\}$.
3. **Compute** matrices $\mathbf{M}_i, \mathbf{N}_i, \mathbf{S}_i, \mathbf{O}_{i,j}$, $j \in \mathcal{N}_i$ for (12) based on $\xi^i_*(\Theta^k), X_{i}^{0:T}(\Theta^k), \theta^k$ and the equations in (11).
4. **Compose** matrices $A_{\ell,i}$ and $A_{i,j}$, $j \in \mathcal{N}_i$ according to (13).
5. **Acquire** matrices $A_{\ell,i}$, $\ell \in \mathcal{N}_i$ from each neighbor $\ell$ of robot $i$. Assign $A_{\ell,i} = 0$ for $\ell \notin \mathcal{N}_i$.
6. **Compose** matrices $\Psi_i, \hat{C}_i$ based on definition (16).
7. **Initialize** $\tau = 0$, $\alpha \in \mathbb{R}_+$, and $Y_{i}^{\tau} = 0$, $Z_{i}^{\tau} = 0$ as random matrices with proper sizes.
8. **while** $\max(|v^\tau_i| \geq \varepsilon_k)$ for $i \in \{1, \ldots, m\}$ **do**
   9. $v^\tau_i = \Psi_i Y_{i}^\tau - C_i - \sum_{\ell \in \mathcal{N}_i} (Z_{i}^\tau - Z_{i,\ell}^\tau)$.
10. $Y_{i}^{\tau+1} = Y_{i}^\tau - \alpha \Psi_i^\tau v^\tau$;
11. $Z_{i}^{\tau+1} = Z_{i}^\tau + \alpha v^\tau$;
9. **end**
12. **Obtain** $X_{i}^{0:T}, U_{i}^{0:T-1}$ from $Y_{i}^\tau$ based on definition (13).
13. **Output** $\frac{\partial \xi^i_*(\Theta)}{\partial \theta^k} = \{X_{i}^{0:T}, U_{i}^{0:T-1}\}$

**Lemma 3.3.** (Validity of Algorithm 3): Suppose the network $\mathbb{G}$ is undetected and connected, suppose equation set (15) has at least one solution, by Algorithm 3, if the step-size $\alpha$ is sufficiently small, the state $Y_{i}^\tau$ of robot $i$ will converge asymptotically to a state $Y_{i}^*$, where the set of $\{Y_{i}^*, i = 1, \ldots, m\}$ forms a solution to (15).

**Proof.** The establishments of Algorithm 3 is based on one of our previous work for solving coupled linear constraints using distributed network flows [48]. The following update

$$
Y_{i}^{\tau+1} = Y_{i}^\tau - \alpha \Psi_i^\tau v^\tau \quad (17a)
$$

$$
Z_{i}^{\tau+1} = Z_{i}^\tau + \alpha v^\tau \quad (17b)
$$

with $v_i^\tau = \Psi_i Y_{i}^\tau - \hat{C}_i - \sum_{\ell \in \mathcal{N}_i} (Z_{i}^\tau - Z_{i,\ell}^\tau)$, is a first order discretization of the algorithm in [49]. With a proper choice of $\alpha$, the convergence of update (17) is exponential and can be theoretically certified. Building on this result, here, we only need to verify that the equilibrium of (17) solves problem (15). Specifically, the equilibrium of (17) implies $v_i^* = \Psi_i Y_{i}^* - \hat{C}_i - \sum_{\ell \in \mathcal{N}_i} (Z_{i}^* - Z_{i,\ell}^*) = 0$ for all $i \in \{1, \ldots, m\}$. It follows that

$$
\sum_{i=1}^{m} v_i^* = \sum_{i=1}^{m} \left( \Psi_i Y_{i}^* + \hat{C}_i - \sum_{\ell \in \mathcal{N}_i} (Z_{i}^* - Z_{i,\ell}^*) \right) = 0 \quad (18)
$$

Since the network is undirected, one has

$$
\sum_{i=1}^{m} \sum_{j \in \mathcal{N}_i} Z_{j}^* = \sum_{i=1}^{m} \sum_{j \in \mathcal{N}_i} |\mathcal{N}_j| |Z_{j}^*| = \sum_{i=1}^{m} |\mathcal{N}_i| Z_{i}^* = \sum_{i=1}^{m} \sum_{j \in \mathcal{N}_i} Z_{i}^* \quad (19)
$$

Equations (19) and (18) yield (15). This completes the proof. □

**Remark 3.4.** (Synergistic implementation of Algorithms 1, 2, 3): As shown in Fig. 1, the presented distributed learning-based coordination framework is equipped with a two-layer updating scheme: the Algorithm 1 as the outer layer to update the parameter set; the Algorithms 2 and 3 as the inner layer to compute the necessary quantities. Thus, each iteration of Algorithm 1 needs to wait until Algorithms 2 and 3 converge. It is worth mentioning that in Figs. 2-3, we use random initial states to verify Algorithms 2-3, and they take relatively large numbers of iterations to converge. However, in the practical implementation of these algorithms, one can use the result of previous iteration in
Algorithm 2-3, to warm-start the current iteration. Then, with desired smoothness properties on the system dynamics and objective functions, these algorithms require significantly less number of iterations to converge.

4 Experiments

This sections presents simulation experiments to validate the Distributed Differentiable Dynamic Game in Algorithm 1.

We use planar quadrotor models to characterize our robots, which can be considered as the abstraction of 3-D quadrotors that are restricted to move in a plane, or ground vehicles driven by horizontally installed propellers. Due to the symmetric nature of the vehicle, the number of propellers are reduced from 4 to 2. Building on the model shown in Fig. 4, we consider 4 vehicles as a team and study their learning-based coordination in terms of formation motion. Each vehicle has non-linear dynamics with 6 dimension system states, i.e.,

\[ x = \text{col} \{ p, v, \phi, \omega \} = \text{col} \{ p_x, p_y, v_x, v_y, \phi, \omega \} \in \mathbb{R}^6 \]

which are corresponding to horizontal/vertical displacement, horizontal/vertical velocity, angle and angular velocity, respectively. For the robot in Fig. 4, its non-linear motion dynamics follow the standard manipulator equations

\begin{align*}
    m_x \ddot{p}_x &= -(u_1 + u_2) \sin \phi \\
    m_y \ddot{p}_y &= (u_1 + u_2) \cos \phi \\
    I \ddot{\phi} &= l(u_1 - u_2)
\end{align*}

which takes the propeller thrusts \((u_1, u_2)\) as system inputs. \(m_x\) is the mass, \(I\) is the moment of inertia, \(l\) is the wing length. For objective functions, depending on the scenario and robot index, each robot takes specific combinations of the candidate objective functions in Table 1.

Table 1: Candidate objective functions

| Running cost                        | Running cost functions                                                                 |
|--------------------------------------|----------------------------------------------------------------------------------------|
| Energy cost                          | \(c_{U,i}^t = \|u_i\|^2\)                                                             |
| Formation maintenance (position)     | \(c_{P,i}^t = \sum_{j \in N_i} \|p_i^t - p_j^t - s_{ij}\|^2\)                         |
| Formation maintenance (distance)     | \(c_{D,i}^t = \sum_{j \in N_i} \|p_i^t - p_j^t\| - d_{ij}\|^2\)                        |
| Formation maintenance (velocity)     | \(c_{V,i}^t = \sum_{j \in N_i} \|v_i^t - v_j^t\|^2\)                                |
| Formation maintenance (tactical)     | \(c_{T,i}^t = \sum_{j \in N_i} \|(x_i^t - x_j^t)^T \cdot F\|\|^2\)                    |
| Collision avoidance                  | \(c_{C,i}^t = \sum_{j \in N_i} \frac{1}{\|p_i^t - p_j^t\|^2}\)                       |
| Risk avoidance                       | \(c_{R,i}^t = \frac{1}{\min_{r \in \Omega_{risk}} \|p_i^t - r\|^2}\)                |

| Terminal cost                        | Terminal cost functions                                                                 |
|--------------------------------------|----------------------------------------------------------------------------------------|
| Target position                      | \(h_{P,i} = \|p_i^T - p_i^*\|^2\)                                                    |
| Target velocity                      | \(h_{V,i} = \|v_i^T - v_i^*\|^2\)                                                    |

In Table 1, for each robot, the energy cost is defined by the thrust generated by its propellers. For formation maintenance (position), \(s_{ij} \in \mathbb{R}^2\) is a vector characterizing the relative position of robots \(i, j\) for formation purposes. For formation maintenance (distance), \(d_{ij} \in \mathbb{R}\) is a scalar characterizing the relative distance between robots \(i, j\) for formation purposes, note that \(d_{ij}\) does not encode direction.
information and requires a closed triangle loop among three robots to define a rigid formation. For formation maintenance (velocity), we guarantee all robots are moving towards a same directing with a same speed. For formation maintenance (tactical), our goal is to ensure the line connecting two robots \( i \) and \( j \) are perpendicular to a vector \( F \in \mathbb{R}^2 \). For collision avoidance, we use a reciprocal function to repel robots from colliding with each other. For risk avoidance, the set \( \Omega_{\text{risk}} \subset \mathbb{R}^2 \) defines risky areas, and the penalty increases as the robot moves close to the risky area. For terminal costs, \( p_i^T \) and \( v_i^T \) represent the desired final position/velocity for robot \( i \).

Corresponding to the optimal control formulation in \( P_i(\theta_i) \), we use (20) to characterize the motion dynamics of robots and specific combinations of the candidate objective functions in Table 1 to characterize the objective function. By assigning different weights and parameters to these functions, the robots can achieve collective behaviors for all three scenarios demonstrated in Fig. 5. This validates our statement in remark 2.1, which addresses the versatility of the proposed learning-based coordination framework. To be more specific,

- **Scenario (a)**, the robots start from different initial positions with 0 speed, their goal is to initialize, within the time horizon, a desired (linear-like) formation towards the \( Y' \) axis, with velocity 2. During this process, they have to avoid collision and risky areas and walls. Such coordination is achieved by their local objective functions, where the running cost is defined as a combination of \( c_{U,i}, c_{P,i}, c_{V,i} \), and the terminal cost is defined as a combination of \( h_{P,i}, h_{V,i} \). The underlying graph of this scenario is a loop, each pair of connected robots use relative position \( s_{ij} \) to enforce a desired formation. To validate our learning algorithm, we parameterize the model by assuming the mass \( m_s \) of the robots are unknown. For the objective function, we assume the desired relative positions \( s_{ij} \) are unknown, the risk avoidance weight \( w_{R,i} \) is unknown. Thus, \( \theta_i = \{m_s,i, s_{ij}, w_i\} \).

- **Scenario (b)** is similar to (a), but the robots aim to initialize a diamond formation. Instead of using relative position, the formation in this scenario is maintained by the distance cost \( c_{D,i}^t \) among connected robots. Thus, robots’ running costs are defined as combinations of \( c_{U,i}, c_{D,i}^t, c_{V,i} \), and for the underlying graph, the robots are pair-wisely connected among the indices \( \{1, 2, 4\}, \{2, 3, 4\} \), which form two triangle loops to ensure the rigidity of the formation. Building on this, we parameterize the model by assuming the mass \( m_s \) of the robots are unknown. For the objective function, we assume the desired distances \( d_{ij} \) are unknown, the risk avoidance weight \( w_i \) is unknown. Thus, \( \theta_i = \{m_s,i, d_{ij}, w_i\} \).

- **Scenario (c)**, the robots are initialized with a diamond formation obtained in scenario (b). Robots 1, 2, 3 are aware of a potential target from in direction of \( F = [-1, \sqrt{3}]^\top \), they want to form a new formation offering them positional advantage against that target. This is enabled by the tactical formation maintenance cost \( c_{T,i}^t \), which enforces the positions of the three robots on a line that is perpendicular to \( F \). The robots’ running costs are defined as combinations of \( c_{U,i}, c_{D,i}^t, c_{V,i}, c_{T,i}^t, c_{C,i}^t, c_{R,i}^t \). For the underlying graph, robots 1, 2 are mutually connected, robots \( \{2, 3, 4\} \) are pair-wisely connected. To parameterize the model, the mass \( m_s \) of the robots are assumed to be unknown. To parameterize the objective function, desired distances \( d_{ij} \) are unknown, the target direction \( F \), the risk avoidance weight \( w_i \) is unknown. Thus, \( \theta_i = \{m_s,i, d_{ij}, F\} \).
We test the learning-based coordination algorithm in the above three scenarios. The time horizon is $T = 30$. We first bring the true values $\theta^*_i$ to $P_i(\theta^*_i)$ and run Algorithm 2 with random initial states for three trials to create training data sets. We then consider $\theta^*_i$ are unknown and employ Algorithm 1 to iteratively learn the parameters in a distributed manner, such that $\theta^k_i \rightarrow \theta^*_i$ for $\forall i \in \{1, 2, 3, 4\}$. The simulation results are presented in Fig. 6. It can be observed that in all scenarios, the total loss of the system described in problem 2 converge to zero. This validates the effectiveness of Algorithm 1. It is worth noting that the proposed algorithm employs a two-layer updating mechanism, requiring Algorithms 2-3 run in the inner-loop. To show the convergence of these algorithm, for the first iteration of Algorithm 1, the two inner-loop algorithms use randomly initialize their variables. The result for the Nash-equilibrium seeking process (c.f. forward-pass of Fig. 1) is presented in Fig. 2; the result for the gradient solving process (c.f. backward-pass of Fig. 1) is presented in Fig. 3. Each iteration of the two algorithms warm-starts with the initial guess being the result from the previous iteration. The required convergence time can be significantly reduced.

5 Conclusion and Future Work

We have considered learning multi-robot coordination from demonstrations. We have adopted a parameterized dynamic game as our learning model, which is able to generate versatile multi-robot coordinating schemes in a fully distributed manner. We learn such dynamic game from demonstrations by proposing a Distributed Differentiable Dynamics Game (DDDG) framework that iteratively tunes the parameters of the dynamics and objective functions in the dynamic game in a fully distributed manner. The implementation of the DDDG algorithm requires computing the Nash equilibrium of a dynamic game and the backpropagating the gradients of robots’ trajectories, which has led us to develop a distributed shooting-based Nash-equilibrium solver and a distributed gradient solver, respectively. We have shown the effectiveness of the proposed DDDG framework through analysis and simulated experiments. For future works, we plan to adapt the DDDG framework into an online MPC setting. Besides, since our current algorithm relies on given demonstrations, we are also interested in integrating the DDDG with a reinforcement learning framework, to learn best coordination through robots’ interaction with environments.

References

[1] Y. Rizk, M. Awad, and E. W. Tunstel, “Cooperative heterogeneous multi-robot systems: A survey,” ACM Computing Surveys (CSUR), vol. 52, no. 2, pp. 1–31, 2019.

[2] J. Verbraeken, M. Wolting, J. Katzky, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, “A survey on distributed machine learning,” ACM Computing Surveys (CSUR), vol. 53, no. 2, pp. 1–33, 2020.

[3] K. Zhang, Z. Yang, and T. Başar, “Multi-agent reinforcement learning: A selective overview of theories and algorithms,” Handbook of Reinforcement Learning and Control, pp. 321–384, 2021.

[4] L. Busoniu, R. Babuska, and B. De Schutter, “A comprehensive survey of multiagent reinforcement learning,” IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 38, no. 2, pp. 156–172, 2008.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot et al., “Mastering the game of go with deep neural networks and tree search,” nature, vol. 529, no. 7587, pp. 484–489, 2016.

[6] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse et al., “Dota 2 with large scale deep reinforcement learning,” arXiv preprint arXiv:1912.06680, 2019.
[7] Z. Liang, J. Cao, S. Jiang, D. Saxena, J. Chen, and H. Xu, “From multi-agent to multi-robot: A scalable training and evaluation platform for multi-robot reinforcement learning,” arXiv preprint arXiv:2206.09590, 2022.

[8] S. Shalev-Shwartz, S. Shammah, and A. Shashua, “Safe, multi-agent, reinforcement learning for autonomous driving,” arXiv preprint arXiv:1610.03295, 2016.

[9] P. Hernandez-Leal, B. Kartal, and M. E. Taylor, “A survey and critique of multiagent deep reinforcement learning,” Autonomous Agents and Multi-Agent Systems, vol. 33, no. 6, pp. 750–797, 2019.

[10] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press, 2018.

[11] C. J. Watkins and P. Dayan, “Q-learning,” Machine learning, vol. 8, no. 3, pp. 279–292, 1992.

[12] J. Eschmann, “Reward function design in reinforcement learning,” Reinforcement Learning Algorithms: Analysis and Applications, pp. 25–33, 2021.

[13] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, “Concrete problems in ai safety,” arXiv preprint arXiv:1606.06565, 2016.

[14] M. J. Osborne and A. Rubinstein, A course in game theory. MIT press, 1994.

[15] Y. Shoham and K. Leyton-Brown, Multiagent systems: Algorithmic, game-theoretic, and logical foundations. Cambridge University Press, 2008.

[16] S. Srinivasan, M. Lanctot, V. Zambaldi, J. Pérolat, K. Tuyls, R. Munos, and M. Bowling, “Actor-critic policy optimization in partially observable multiagent environments,” Advances in neural information processing systems, vol. 31, 2018.

[17] R. Munos, T. Stepleton, A. Harutyunyan, and M. Bellemare, “Safe and efficient off-policy reinforcement learning,” Advances in neural information processing systems, vol. 29, 2016.

[18] S. Levine and V. Koltun, “Continuous inverse optimal control with locally optimal examples,” in Proceedings of the 29th International Conference on International Conference on Machine Learning, 2012, pp. 475–482.

[19] W. Jin, D. Kulić, S. Mou, and S. Hirche, “Inverse optimal control from incomplete trajectory observations,” The International Journal of Robotics Research, vol. 40, no. 6-7, pp. 848–865, 2021.

[20] A. Y. Ng and S. J. Russell, “Algorithms for inverse reinforcement learning.” in Icml, vol. 1, 2000, p. 2.

[21] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, “Maximum margin planning,” in Proceedings of the 23rd international conference on Machine learning, 2006, pp. 729–736.

[22] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, “Maximum entropy inverse reinforcement learning.” in Aaai, vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438.

[23] P. Englert, N. A. Vien, and M. Toussaint, “Inverse kkt: Learning cost functions of manipulation tasks from demonstrations,” The International Journal of Robotics Research, vol. 36, no. 13-14, pp. 1474–1488, 2017.

[24] A. A. Y. Ng, “Apprenticeship learning via inverse reinforcement learning,” in Proceedings of the twenty-first international conference on Machine learning, 2004, p. 1.

[25] W. Jin, Z. Wang, Z. Yang, and S. Mou, “Pontryagin differentiable programming: An end-to-end learning and control framework,” Advances in Neural Information Processing Systems, vol. 33, pp. 7979–7992, 2020.

[26] T. Basar and G. Zaccour, Handbook of dynamic game theory. Springer, 2018.

[27] A. Bressan, “Noncooperative differential games. a tutorial,” Department of Mathematics, Penn State University, p. 81, 2010.

[28] A. Bressan and W. Shen, “Small bv solutions of hyperbolic noncooperative differential games,” SIAM journal on control and optimization, vol. 43, no. 1, pp. 194–215, 2004.

[29] T. L. Molloy, J. J. Ford, and T. Perez, “Inverse noncooperative differential games,” in 2017 IEEE 56th Annual Conference on Decision and Control (CDC). IEEE, 2017, pp. 5602–5608.

[30] X. Lin, S. C. Adams, and P. A. Beling, “Multi-agent inverse reinforcement learning for certain general-sum stochastic games,” Journal of Artificial Intelligence Research, vol. 66, pp. 473–502, 2019.
[31] A. Haurie, J. B. Krawczyk, and G. Zaccour, *Games and dynamic games*. World Scientific Publishing Company, 2012, vol. 1.

[32] F. Köpf, J. Inga, S. Rothfuß, M. Flad, and S. Hohmann, “Inverse reinforcement learning for identification in linear-quadratic dynamic games,” *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 14 902–14 908, 2017.

[33] W. Jin and S. Mou, “Distributed inverse optimal control,” *Automatica*, vol. 129, p. 109658, 2021.

[34] J. Jairo Inga Charaja, *Inverse dynamic game methods for identification of cooperative system behavior*. KIT Scientific Publishing, 2021.

[35] K. Cao and L. Xie, “Game-theoretic inverse reinforcement learning: A differential pontryagin’s maximum principle approach,” *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[36] T. Başar and G. J. Olsder, *Dynamic noncooperative game theory*. SIAM, 1998.

[37] N. Francez, “Distributed termination,” *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 2, no. 1, pp. 42–55, 1980.

[38] L. Zheng, T. Fiez, Z. Alumbaugh, B. Chasnov, and L. J. Ratliff, “Stackelberg actor-critic: Game-theoretic reinforcement learning algorithms,” in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, no. 8, 2022, pp. 9217–9224.

[39] D. Gamarnik, “Correlation decay method for decision, optimization, and inference in large-scale networks,” in *Theory Driven by Influential Applications*. INFORMS, 2013, pp. 108–121.

[40] G. Qu, A. Wierman, and N. Li, “Scalable reinforcement learning of localized policies for multi-agent networked systems;” in *Learning for Dynamics and Control*. PMLR, 2020, pp. 256–266.

[41] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[42] K. Park, C. A. Felippa, and U. Gumaste, “A localized version of the method of lagrange multipliers and its applications,” *Computational Mechanics*, vol. 24, no. 6, pp. 476–490, 2000.

[43] M. Ye and G. Hu, “Distributed nash equilibrium seeking by a consensus based approach,” *IEEE Transactions on Automatic Control*, vol. 62, no. 9, pp. 4811–4818, 2017.

[44] F. Salehisadaghiani and L. Pavel, “Distributed nash equilibrium seeking: A gossip-based algorithm,” *Automatica*, vol. 72, pp. 209–216, 2016.

[45] K. Lu, G. Jing, and L. Wang, “Distributed algorithms for searching generalized nash equilibrium of noncooperative games;” *IEEE transactions on cybernetics*, vol. 49, no. 6, pp. 2362–2371, 2018.

[46] Y. Zou, B. Huang, Z. Meng, and W. Ren, “Continuous-time distributed nash equilibrium seeking algorithms for non-cooperative constrained games,” *Automatica*, vol. 127, p. 109535, 2021.

[47] J. P. Perdew, K. Burke, and M. Ernzerhof, “Generalized gradient approximation made simple,” *Physical review letters*, vol. 77, no. 18, p. 3865, 1996.

[48] X. Wang, S. Mou, and B. D. Anderson, “Scalable, distributed algorithms for solving linear equations via double-layered networks,” *IEEE Transactions on Automatic Control*, vol. 65, no. 3, pp. 1132–1143, 2019.

[49] X. Wang and S. Mou, “A distributed algorithm for achieving the conservation principle,” in *2018 Annual American Control Conference (ACC)*, 2018, pp. 5863–5867.