2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Daejeon Convention Center
October 9-14, 2016, Daejeon, Korea

D_{++}: Structural Credit Assignment in Tightly Coupled Multiagent Domains

Aida Rahmattalabi, Jen Jen Chung, Mitchell Colby, Kagan Tumer

Abstract— **Autonomous multi-robot teams can be used in complex coordinated exploration tasks to improve exploration performance in terms of both speed and effectiveness. However, use of multi-robot systems presents additional challenges. Specifically, in domains where the robots' actions are tightly coupled, coordinating multiple robots to achieve cooperative behavior at the group level is difficult. In this paper, we demonstrate that reward shaping can greatly benefit learning in multi-robot exploration tasks. We propose a novel reward framework based on the idea of *counterfactuals* to tackle the coordination problem in tightly coupled domains. We show that the proposed algorithm provides superior performance (166% performance improvement and a quadruple convergence speed up) compared to policies learned using either the global reward or the difference reward [1].**

I. Introduction

Autonomous multi-robot teams can be used in complex exploration tasks for improved information gathering performance over single robot systems, in terms of both speed and effectiveness. However, multi-robot coordination is a complex control problem, especially in tightly-coupled tasks where there is a greater mutual dependence of the robots on each other's performance [2]. In such cases, the performance of a multi-robot system is highly sensitive to the level of coordination among the individual robots' policies.

In many multi-robot problems, such as space exploration, environmental monitoring, and search and rescue tasks, only a high level description of the mission is at hand, and agents only have access to local sensor information to decide their actions. Communication, which is a means to improve the performance of cooperative agents, is also often expensive or limited. Effective team performance becomes even more challenging when robots' actions are tightly coupled, requiring agents to extensively coordinate their actions in order fulfill their mission. In such cases, coordinated policies can be difficult to define a priori and distributed policy learning is often employed to optimize team strategies.

Distributed policy learning has been demonstrated to produce effective team performance for loosely coupled tasks [3]–[6]. In particular, reward-shaping techniques [7]–[9] have been used to address the structural credit assignment problem for implicit coordination solutions where inter-robot communication is unavailable. In this regard, the difference evaluation function [1], [10] is a shaped reward that makes use of *counterfactuals* to query the direct effect of an individual's contribution to the team performance. This reward signal can be computed from locally available information [11], and it is both *aligned* to the system evaluation and *sensitive* to the individual robot's action, providing an effective training signal when the system reward function is smooth.

However, missions that require tightly coupled actions introduce reward functions that are inherently non-smooth and often involve step changes that represent the specific multi-robot coupling requirements of the tasks. This is one of the major problems that challenges many existing reward-shaping techniques since they are not capable of providing an adequate evaluation signal for learning the joint policies necessary for achieving such tightly coupled tasks.

In this work, we propose the notion of *stepping stone* actions as actions that are potentially useful in terms of the system objective but are not rewarded because other agents in the team have not yet found their proper actions. And we focus on rewarding such actions to improve coordination. To this end, we extend the idea in the difference reward and explore the manipulation of *counterfactuals* to provide agents with a stronger feedback signal on potentially good joint actions. D_{++}, our proposed reward function, computes the effect of introducing multiple identical agents to the system and executes a targeted sweep over the number of available agents, providing a rapid estimate of the joint action required to achieve a task. D_{++} is then used to compute the evaluation signal in a cooperative coevolutionary algorithm for training the neural network control policies of each agent.

We demonstrate our proposed reward framework on an environmental monitoring task that requires robots to perform multiple simultaneous observations of points of interest (POIs) to successfully gather information. We show that with D_{++}, the team of robots is able to greatly improve policy learning rates over existing shaped rewards in terms of the overall team information gathering performance.

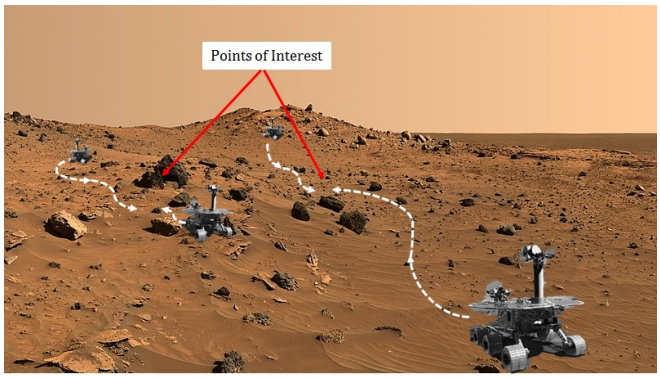
II. Problem Formulation

The problem that we investigate in this paper is multi-robot coordination in a tightly coupled environmental monitoring task. It consists of a set of homogeneous rovers on a two dimensional plane that must observe a set of POIs within a given time window [1], [5]. Prior knowledge of POIs, such as their locations, the number of POIs and the utility of observing any particular POI is not known and robots must

This work was supported by NASA grant NNX14AI10G and the US Department of Energy - National Energy Technology Laboratory under contract no. DE-FE0012302.

Aida Rahmattalabi, Jen Jen Chung, Kagan Tumer are with the Autonomous Agents and Distributed Intelligence Lab, School of Mechanical, Industrial and Manufacturing Engineering, Oregon State University, OR, 97330, USA {rahmatta, jenjen.chung, kagan.tumer}@oregonstate.edu

Mitchell Colby is with Scientific Systems Company, Inc. colby.mk@gmail.com



http://www.teciber.com/wp-content/uploads/2015/03/marte.jpg

Fig. 1: Rover domain representation with sample robot paths. Multiple simultaneous observations (in this case two) must be made of a particular POI to have any value to the system. Starting from different locations, well-coordinated robots navigate to the POIs and make observations.

learn to coordinate such that the team's utility function is maximized.

The key difficulty of this domain is that multiple simultaneous observations of a POI are required otherwise no reward is received by the team for those observations. Figure 1 illustrates the basic problem setup. A POI is considered observed only if $m > 0$ robots observe that POI from within a specified observation radius, which is not known to the robot. This problem formulation ensures that team coordination is essential to the completion of the task.

To formalize the problem, we first focus on the simple case where $m = 2$ observations are required in order to observe a POI. In this case, if more than two robots observe a POI, only the observations of the closest two robots are considered and their observation distances are averaged in the computation of the global system evaluation G, which is formulated as:

$$G(\mathbf{z}) = \sum_i \sum_j \sum_k \frac{V_i N^1_{i,j} N^2_{i,k}}{\frac{1}{2}(\delta_{i,j} + \delta_{i,k})}, \tag{1}$$

where $\mathbf{z}$ is the joint state-action of the team, V_i is the value of observing the i-th POI, and $\delta_{i,j}$ is the distance between the j-th robot, and the i-th POI. The variables $N^1_{i,j}$ and $N^2_{i,k}$ indicate whether robots j, k were within the observation distance δ_0 and were the closest two robots to the i-th POI. That is,

$$N^1_{i,j} = \begin{cases} 1, & \text{if } \delta_{i,j} < \delta_0 \text{ and } \delta_{i,j} < \delta_{i,l},\ \forall l \neq j, \\ 0, & \text{otherwise,} \end{cases} \tag{2}$$

$$N^2_{i,k} = \begin{cases} 1, & \text{if } \delta_{i,k} < \delta_0 \text{ and } \delta_{i,k} < \delta_{i,l},\ \forall l \neq j,k, \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

The overall team objective is to maximize the global system evaluation (1).

Sensor detections of POIs and other robots are discretized into four quadrants, (*north-east, north-west, south-west, south-east*), with respect to the robot body frame (see Fig. 2). The detections within each quadrant are used to compute the state input vector of the learned neural network controller. More specifically, the state variable representing

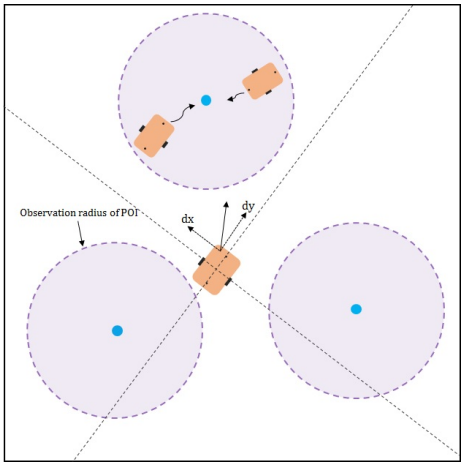


Fig. 2: Diagram of the rover domain. The world is broken up into four quadrants relative to the robots position and orientation. POIs and fellow robots that are observed in each quadrant are summed resulting in 8 state input variables. At each timestep, the robot's neural network controller yields two continuous outputs $[d_x, d_y]$, which determine the robot's motion in the next timestep. Each POI has an observation radius such that only robots within that radius are able to observe that POI.

robot detections in quadrant q for robot j is defined as:

$$s_{ROB_{j,q}} = \sum_{j' \in J_q} \frac{1}{L_{j'}}, \tag{4}$$

where J_q is the set of observed and estimated robots in quadrant q, and $L_{j'}$ is the relative distance from robot j to robot j'. The state variable representing POI detections in quadrant q of robot j is defined as:

$$s_{POI_{j,q}} = \sum_{i \in I_q} \frac{V_i}{L_i}, \tag{5}$$

where I_q is the set of observed POIs in quadrant q, and L_i is the relative distance between robot j and POI i. These state variables give an approximate representation of the world, reducing the location and number of robots and POIs in each quadrant to a representative value.

Each member of the team of robots executes a policy described by a neural network. Known as universal approximators [12], neural networks have the ability to model continuous state-action control policies given only a coarse representation of the system state [10]. To train a neural network, the weights are adjusted in such a way that given the current state as an input, the network returns an action that maximizes a particular utility function. The performance of the learned policy is strongly dependent on the utility function used to evaluate policies.

In this work, we use one-hidden-layer feed-forward neural network controllers and we train these neuro-controllers with cooperative coevolutionary algorithms (CCEA) where we use the D_{++} evaluation function as the fitness function, and we compare the results against those trained with the difference (D) and global (G) evaluation functions.

III. Background

A. Fitness Function Shaping

In many multi-robot coordination domains there is a difference between maximizing the system-level fitness value and maximizing a single agent's fitness value. This can be referred to as reward shaping and is aimed at enhancing learning by providing agents with a reward signal that is more sensitive to their own actions while still reflecting the improvement in the system's performance.

B. Difference Evaluation Function

The difference evaluation function [1] is a shaped reward signal that provides agent-specific evaluation by removing a large amount of the noise created by the actions of other agents in the system [13]. It is defined as:

$$D_{\mathbf{i}} = G\left(\mathbf{z}\right) - G\left(\mathbf{z}_{-i} \cup \mathbf{c}_i\right), \tag{6}$$

where $\mathbf{z}$ is the joint state, $G(\mathbf{z})$ is the global system performance, $\mathbf{z}_{-i}$ are all the state-actions on which agent i has no effect and $\mathbf{c}_i$ is the *counterfactual* term, which is a fixed vector used to replace the effects of agent i. Thus, $G(\mathbf{z}_{-i} \cup \mathbf{c}_i)$ is the evaluation of a theoretical system without the contribution of agent i. Any action taken by agent i to increase D_i simultaneously increases G. This property is termed *alignment*, and is a key property of any shaped reward. Further, the second term in (6) removes portions of $G(\mathbf{z})$ that are not related to agent i, resulting in an improved signal-to-noise ratio. In other terms, agent i's impact on D_i is much higher than its relative impact on $G\left(\mathbf{z}\right)$ [14]. This property is termed *sensitivity*. Both [10], [15] have previously shown that the alignment and sensitivity properties of the difference evaluation function results in agent-specific feedback, which leads to superior learning performance.

IV. Approach

A. D_{++}: An Extension to the Difference Reward in Tightly Coupled Multiagent Domains

One of the challenges in distributed learning is that with limited or no prior knowledge of the task and limited communication capabilities, agents must randomly search the state-action space until they receive a reward signal to evaluate their policies. It is particularly more challenging in tightly coupled domains since the task cannot be accomplished unless tight coordination among the agents is established and maintained. Considering the huge joint state-action space of a multiagent system, the probability of agents simultaneously executing the right actions is very low. Thus, agents receive no feedback on their policies in large portions of the joint space. In such domains, it is critical to provide rewards for *stepping stone* actions, actions that would lead to good outcomes if joined by other agents. To illustrate, in the case of the exploration problem discussed in this paper, consider a scenario where three robots are required to simultaneously observe a POI. The robots start from random locations and should synchronize their policies to reach the POIs simultaneously. However, it is unlikely that randomly moving robots are able to learn this task within a reasonable amount of time since only after all three robots successfully observe a POI do they receive a reward. In the absence of system feedback they are not capable of evaluating any potential improvement in their policies.

It is clear that the key problem is to be able to distinguish between a case where two robots are within the observable region of a POI and they only need one more robot to complete their subteam, and a case where all three of them are wandering without purpose in the environment.

Any reward function should enable agents to evaluate their actions in the absence of their teammates. This can lead to a huge reduction in the search space and an increase in the probability of success. Building upon the above argument, we extend the idea of *counterfactual* in the difference reward and we propose a reward function, referred to as D_{++}, to address this problem. The *counterfactual* that we propose computes the effect of introducing multiple identical agents to the system and compares it to the current state of the system. The mathematical formulation of D_{++} is,

$$D^n_{++}\left(i\right) = \frac{G\left(\mathbf{z}_{+(\cup_{\mathbf{i}=\mathbf{1},\ldots,\mathbf{n}})\mathbf{i}}\right) - G\left(\mathbf{z}\right)}{n}, \tag{7}$$

In the above equation, $\mathbf{z}_{+(\cup_{\mathbf{i}=\mathbf{1},\ldots,\mathbf{n}})\mathbf{i}}$ indicates all the states on which agent i has added n *counterfactual* agents. This allows the primary agent to investigate the effect of hypothetically introducing n other agents. Also, in the formulation of D^n_{++}, if the value of n is set to -1, D^n_{++} will yield the definition of D as described in equation (6), noting that the *counterfactual* term $\mathbf{c}_i$ is an arbitrary vector which can indeed be set to $\vec{0}$. Also the division by n is to normalize with respect to the number of *counterfactual* agents.

As in the difference reward, agents optimize the global system objective by calculating each agents' contribution to the system performance. This is calculated based on the difference an agent makes by executing a certain action. This idea is further extended in D_{++}, which evaluates how the introduction of multiple identical agents can benefit the system. It is noteworthy that in tightly coupled tasks, D does not provide any feedback unless tight coordination among the agents is established and maintained. On the other hand, D_{++} fails to provide this gradient information where a sufficient number of agents have coordinated. So it is reasonable to leverage both D and D_{++} to the benefit of the system. This leads us to the final definition of our reward framework which is based on the idea of both removing and adding agents, described by D and D_{++}, respectively.

At this stage, the reward shaping will be combined with a search problem over different numbers of *counterfactual* agent additions. In order to guide the search and improve the computation, we make a basic assumption that in any state of the agent, the entire multiagent system should be able to accomplish the task (if at all possible). For example, in the present problem domain, if a robot is in the observable region of a POI, by adding the entire team as *counterfactual* agents, the system can make an observation of that POI.

This simple assumption allows us to evaluate whether there exists a solution before we begin incrementing *counterfactual* agents in the D_{++} algorithm. Also, it gives an upper bound for the number of allowable *counterfactual* agents. This assumption, however, is only valid if the problem is well-posed. Algorithm 1 summarizes our approach.

Algorithm 1 D_{++}

```
calculate D_{++}^{-1} using (7)
calculate D_{++}^{totalAgents-1} using (7)
if D_{++}^{totalAgents-1} ≤ D^{-1} then
    Return D_{++}^{-1}
else
    n ← 0
    repeat
        n = n + 1
        Calculate D_{++}^{n} using (7)
        if D_{++}^{n} > D_{++}^{n-1} then
            Return D_{++}^{n}
    until (n ≤ totalAgents − 1)
Return D_{++}^{-1}
```

Algorithm 1 begins with the calculation of $D = D_{++}^{-1}$ and $D_{++}^{totalAgents-1}$. Note that the $totalAgents-1$ plus the agent itself make up the entire team of agents. If $D_{++}^{totalAgents-1}$ is less or equal to D, it means that adding agents does not benefit the agent. In other terms, it does not receive a higher reward from the system if there were more agents helping it in performing the task. In our exploration domain, this corresponds to either a case where an agent is outside the observable region of all the POIs and thus it is not possible to observe any of the POIs, or a case where there are a sufficient number of agents already within the observable range. In the second case, D can simply be used to reward each agent based on its contribution. It can be seen that by making this assumption, we can greatly reduce the computation by eliminating the need to loop over all the values of n before realizing all of this computation is in vain.

Given the above definition of the D_{++} algorithm, we will dedicate the following two subsections to describe the learning framework adopted in this work and and how D_{++} is incorporated in the learning procedure.

B. Evolutionary Algorithms

Evolutionary Algorithms (EAs) are a class of stochastic search algorithms that often outperform classical optimization algorithms [16]. Typically, EAs contain three basic mechanisms: solution generation, mutation, and selection. Starting with an initial set of candidate solutions (the population), these mechanisms are used to generate new solutions and retain existing solutions that show improvement based on a system utility. Simple EAs can be applied to a variety of single-agent learning tasks, however, when dealing with large cooperative multi-robot problems, modifications are required for an effective performance. One such modification is co-evolution, where multiple populations evolve simultaneously in order to develop policies for interacting agents.

C. Cooperative Coevolutionary Algorithms

The standard Cooperative Coevolutionary Algorithm (CCEA) [17] has been used in this study as a base algorithm for learning control policies [13]. CCEAs are an extension of EAs for multiagent systems and have been shown to perform well in cooperative multiagent domains [18]. The standard CCEA is detailed in Algorithm 2. In CCEA, N co-evolving populations of neural networks (neuro-controllers) are utilized to form teams comprised of N agents. One member of each population is extracted for each agent to form a team which then operates in the problem domain. At each generation, k mutated networks are generated in each population by mutating the parent networks. Then, $2k$ teams of agents are formed and simulated. The performance of each simulated team is subsequently evaluated using a fitness function, $F(z)$, and assigned to every agent in the team. In this paper, we propose D_{++} as the fitness function, and we compare it to using G and D. Finally, k networks from each population are selected based on greedy selection to proceed to the next generation. This process is repeated for a set number of generations.

Algorithm 2 Standard CCEA

```
Initialize N populations of k neural networks
for Generation do
    for Population do
        produce k successor solutions
        mutate successor solutions
    for i = 1 → 2k do
        randomly select one agent from each population
    without replacing it into population pool
        add agents to team T_i
        simulate T_i in domain
        assign fitness to each agent in T_i using F(z)
    for Population do
        select k solutions using ε-greedy selection
```

D. Computational Complexity

In computing the difference reward, two calculations of the global evaluation function are made by each agent ($G(\mathbf{z})$ and $G(\mathbf{z}_{-i} \cup \mathbf{c}_i)$) while an agent learning with the global reward only calls $G(\mathbf{z})$ once for each policy evaluation. Here, the assumption is that each agent makes the reward computations locally and no feedback is broadcast to the system. For D_{++}, however, this number is increased since agents must continue adding *counterfactual* agents until they reach a value greater than D, and each iteration requires additional calls to $G(\mathbf{z})$. This adds to the computational time of the learning algorithm. Table I compares the computational complexity for evaluating a single policy as the number of calls to $G(\mathbf{z})$. In Table I, c_i is the minimum number of *counterfactual* agents needed to be added by agent i. This term is bounded between

TABLE I: COMPARISON OF NUMBER OF CALLS TO G FOR THREE DIFFERENT REWARD FUNCTIONS

Evaluation function	G	D	D_{++}
Calls to G	1	2	$c_i + 3$

$[0, \textit{total number of agents} - 1]$ since we assume the entire team of agents should be able to accomplish the task. In cases where a sufficient number of agents are present to complete a coupled task, c_i is equal to 0 (i.e. D provides a sufficient learning signal). In cases where adding any number of agents cannot increase the system performance, c_i is also 0. This is because, when calculating the D_{++} reward, first $(\textit{total number of agents} - 1)$ agents are added and if the returning reward is not greater than D, it can be understood that iterating over fewer *counterfactual* agents will similarly not yield a better reward. Lastly, if not enough agents are within the observable radius of a POI, c_i takes a value between $[1, \textit{required number of observations} - 1]$. We will revisit this problem in the results section where we show that the average value of c_i decreases for the team of agents throughout learning, making it increasingly more efficient to use D_{++} as the learning signal.

V. EXPERIMENT SETUP

To investigate the performance of D_{++} learners, different experiments are presented and compared against agents trained using the difference evaluation function or global evaluation as learning signal. Generally, at every timestep, each robot uses a policy encoded by the neural network controller to determine two controls in the (x, y) directions. Each controller is an 8-input, 9-hidden unit, 2-output fully connected feed-forward neural network. The weights are adjusted via the (CCEA) algorithm, explained in Algorithm 2. The eight inputs are the state variables as defined in (4) and (5). The neural network output provides two values that lie in $[0, 1]$ and each of these are mapped to one of the two controls. Each agent has a population of 15 policies, initialized using a normal distribution $\mathcal{N}(0, 1)$.

In the following experiments, the survey region is 30×30 units in size, containing 12 POIs and 10 agents. Each POI requires a fixed number of simultaneous observations within its observation range which is defined as r_{POI} = 4.0. In the first experiment, each POI requires 3 simultaneous observations. In the second experiment, in order to analyze how the learning algorithm performs with tighter agent coupling, the number of required observations for each POI was doubled. The maximum allowable robot movement is defined as $d_{max} = 1$ unit/timestep. In this problem, we assume full observability for the robot sensors.

VI. RESULTS

A. 12 Robots, 10 POIs, 3 Required Simultaneous Observations

Averaged learning results associated with all three reward functions, G, D, D_{++} with error bars reporting the standard error in the mean are given in Fig. 3. These results show that agents learning with the D_{++} evaluation signal outperform agents using global evaluations or the difference evaluation function alone. In fact, D_{++} produces policies that perform up to 100% better that those of D and at a quadruple speedup in the learning rate.

Figure 4 shows the paths of the team of robots after being trained with the D_{++} reward function. The dots in magenta are the POIs, and they vary in size based on their value to the system. The POIs are spread across the exploration region. Robots are initially located in the center and should spread and explore the area of interest. As seen in Fig. 4, robots have successfully formed teams of three robots and have coordinated to observe multiple POIs in their proximity. The significance of these results is that team formation has been performed in an implicit manner, requiring no communication among agents or any prior information about other agents' intended behavior. This approach is particularly useful in cases where communication is limited or expensive since it can provide more reliable and robust results.

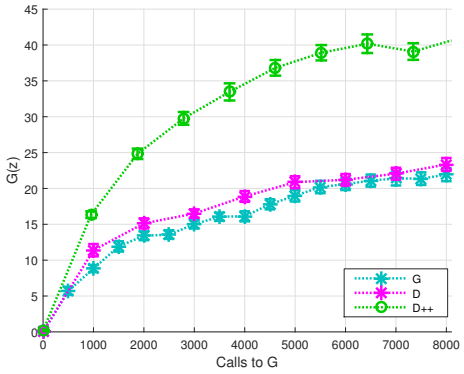


Fig. 3: Observation performance of policies trained on G, D, D_{++} for 12 robots, 10 POIs each requiring 3 simultaneous observations.

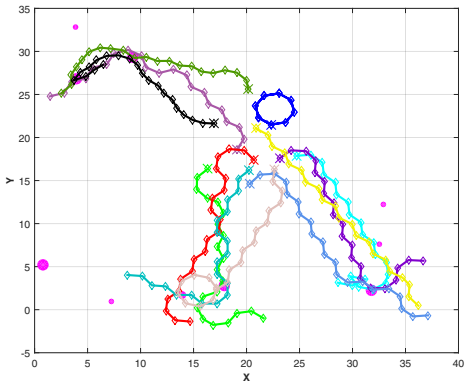


Fig. 4: Robots paths executed by policies learned using the D_{++} reward function. POIs are represented as pink circles; larger circles indicate that the POI has a higher observation value.

B. 12 Robots, 10 POIs, 6 Required Simultaneous Observations

To investigate the ability of the proposed algorithm in dealing with tighter agent couplings, we doubled the number of required simultaneous observations. Figure 5 indicates

that as the coupling of the system increases, both G and D lose applicability since learning algorithms using either of these reward functions relies heavily on multiple agents simultaneously selecting the right actions, which is highly unlikely in tightly coupled domains. However, D_{++} significantly promotes coordination by rewarding *stepping stone* robot policies.

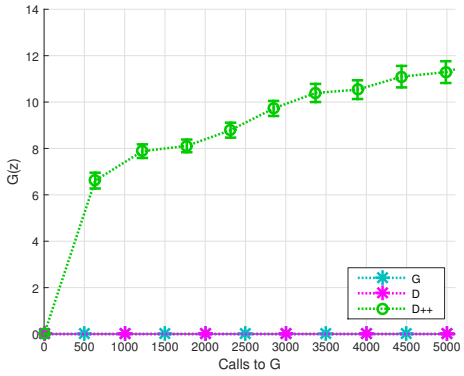


Fig. 5: Observation performance of policies trained on G, D, D_{++} for 12 robots, 10 POIs each requiring 6 simultaneous observations.

C. Computation Time Analysis

Recall from Section IV-D that the D_{++} calculation requires additional calls to G. However, as seen, agents trained using D_{++} learn to coordinate relatively quickly compared to the results from using either G or D. This results in a decrease in the number of required *counterfactual* agents and also a decrease in the computation effort needed. Figure 6 supports the above argument by indicating the average required number of simulated *counterfactual* agents in D_{++} calculations during training. There is a sharp decrease in the calculations required in D_{++} within the first 500 generations.

VII. Conclusion

In this paper, we developed control policies for a team of robots in a tightly coupled environmental monitoring task using D_{++} as the learning signal. D_{++} builds on the idea of *counterfactuals* and promotes coordination by rewarding policies that are the *stepping stones* to accomplishing the system objective. Notably, as the coupling of the system increases, both G and D fail since they strictly rely on multiple agents simultaneously selecting the right actions, which is highly unlikely in tightly coupled domains. We showed that policies trained on D_{++} result in superior performance compared to those trained on G and D in terms of both convergence speed and performance.

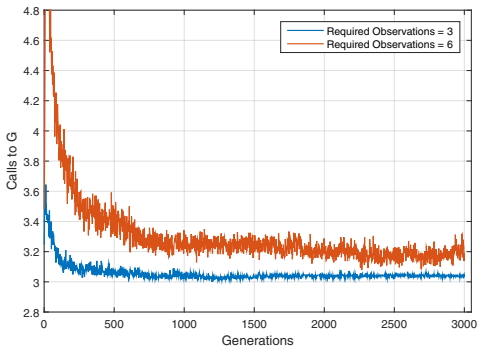


Fig. 6: Average calls to G across progressive learning generations in the D_{++} calculation for two cases where POIs require 3 and 6 simultaneous observations, respectively. Averages over 50 trial runs are plotted along with shaded 95% confidence intervals.

References

[1] A. Agogino and K. Tumer, "Efficient evaluation functions for multi-rover systems," in *Genetic and Evolutionary Computation–GECCO 2004*. Springer, 2004, pp. 1–11.

[2] B. P. Gerkey and M. J. Matarić, "Pusher-watcher: An approach to fault-tolerant tightly-coupled robot coordination," in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, vol. 1. IEEE, 2002, pp. 464–469.

[3] P. Stone and M. Veloso, "Multiagent systems: A survey from a machine learning perspective," *Autonomous Robots*, vol. 8, no. 3, pp. 345–383, 2000.

[4] M. Bowling and M. Veloso, "Simultaneous adversarial multi-robot learning," in *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, vol. 3, 2003, pp. 699–704.

[5] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," *Autonomous Agents and Multi-Agent Systems*, vol. 11, no. 3, pp. 387–434, 2005.

[6] K. Tuyls, P. J. T. Hoen, and B. Vanschoenwinkel, "An evolutionary dynamical analysis of multi-agent learning in iterated games," *Autonomous Agents and Multi-Agent Systems*, vol. 12, no. 1, pp. 115–153, 2006.

[7] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *International Conference on Machine Learning*, 1999, pp. 278–287.

[8] S. Devlin and D. Kudenko, "Theoretical considerations of potential-based reward shaping for multi-agent systems," in *The 10th International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2011, pp. 225–232.

[9] ——, "Dynamic potential-based reward shaping," in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2012, pp. 433–440.

[10] A. Agogino and K. Tumer, "Efficient evaluation functions for evolving coordination," *Evolutionary Computation*, vol. 16, no. 2, pp. 257–288, 2008.

[11] M. Colby, J. J. Chung, and K. Tumer, "Implicit adaptive multi-robot coordination in dynamic environments," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 5168–5173.

[12] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[13] M. Colby and K. Tumer, "Shaping fitness functions for coevolving cooperative multiagent systems," in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*. International Foundation for Autonomous Agents and Multiagent Systems, 2012, pp. 425–432.

[14] D. H. Wolpert and K. Tumer, "Optimal payoff functions for members of collectives," *Advances in Complex Systems*, vol. 4, no. 02n03, pp. 265–279, 2001.

[15] M. Colby and K. Tumer, "Fitness function shaping in multiagent cooperative coevolutionary algorithms," *Autonomous Agents and Multi-Agent Systems*, pp. 1–28, 2015.

[16] D. B. Fogel, "An introduction to simulated evolutionary optimization," *Neural Networks, IEEE Transactions on*, vol. 5, no. 1, pp. 3–14, 1994.

[17] M. A. Potter and K. A. De Jong, "A cooperative coevolutionary approach to function optimization," in *Parallel problem solving from naturePPSN III*. Springer, 1994, pp. 249–257.

[18] S. G. Ficici, O. Melnik, and J. B. Pollack, "A game-theoretic and dynamical-systems analysis of selection methods in coevolution," *Evolutionary Computation, IEEE Transactions on*, vol. 9, no. 6, pp. 580–602, 2005.