# Interaction-Aware Multi-Agent Reinforcement Learning for Mobile Agents with Individual Goals

Anahita Mohseni-Kabir[1], David Isele[2], and Kikuo Fujimura[2]

*Abstract*— **In a multi-agent setting, the optimal policy of a single agent is largely dependent on the behavior of other agents. We investigate the problem of multi-agent reinforcement learning, focusing on decentralized learning in non-stationary domains for mobile robot navigation. We identify a cause for the difficulty in training non-stationary policies: mutual adaptation to sub-optimal behaviors, and we use this to motivate a curriculum-based strategy for learning interactive policies. The curriculum has two stages. First, the agent leverages policy gradient algorithms to learn a policy that is capable of achieving multiple goals. Second, the agent learns a modifier policy to learn how to interact with other agents in a multi-agent setting. We evaluated our approach on both an autonomous driving lane-change domain and a robot navigation domain.**

## I. Introduction

Single agent reinforcement learning (RL) algorithms have made significant progress in game playing [1] and robotics [2], however, single agent learning algorithms in multi-agent settings are prone to learn stereotyped behaviors that over-fit to the training environment [3], [4]. There are several reasons why multi-agent environments are more difficult: 1) interacting with an unknown agent requires having either multiple responses to a given situation or a more nuanced ability to perceive differences. The former breaks the Markov assumption, the latter rules out simpler solutions which are likely to be found first. 2) Intentions and goals of other agents are not known and must be inferred. This also can break the Markov assumption. 3) Agents are co-evolving, and their policies are non-stationary during training. In this work we investigate a property associated with non-stationary policies: partially successful policies for one agent can get repeated multiple times or 'burned in', causing the other agents to adapt to that specific behavior. This sets off a chain of mutual adaptation that encourages agents to only visit suboptimal regions of the state space.

When training independent agents [5], the competing learning processes of multiple agents is sufficiently difficult that either the agents fail to learn, or the agents learn one-after-the-other, resulting in stereotyped policies that are sensitive to the behavior of other agents. Recent approaches to multi-agent reinforcement learning have relaxed the independence of an agent by exploiting centralized training and assuming the other agents' actions are known [6], [7]. While these techniques are more successful, we show that they still exhibit one-after-the-other learning, producing highly-dependent policies.

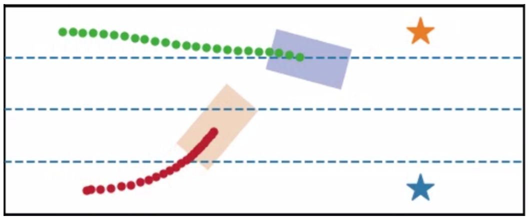


Fig. 1: Double lane-change problem. The bottom car and the top car are crossing one another to go to the star on the top right and bottom right of the environment respectively.

We propose the use of an independent curriculum-based training procedure for learning policies that avoids mutual adaptation without using either centralized training or knowledge of the other agents actions. We start from the observation that when agents learn at different rates (as often happens under random initialization), one agent learns a policy *around* the other agents' policies. Since the policies being accommodated are suboptimal during the early stages of learning, the agents drive each other into poor local optima. One solution to pacing the learning of multiple agents is self-play [8], but self-play requires symmetric agents. As an alternative, we consider learning via a curriculum [9]. The use of curriculum learning lets us pace the learning of each agent, while allowing us to handle the case of agents with different goals.

We structure our curriculum by first learning an optimal single agent and then explicitly learning the modifications required to interact with other agents. This approach leverages the intuition that when other agents are not present in the environment, the agent should behave like a single agent trying to reach a goal. We address the interaction-aware decision making problem in the second stage of the curriculum. In the second stage, we introduce an architecture that uses the learned single-agent policy, and adjusts it with a learned interactive multi-agent policy.

We consider two robotic applications where the agents policies must be mutually consistent in order to achieve the intended goals: a lane-change scenario (Fig. 1) and a mobile navigation scenario. The agents do not have access to the goals or intentions of other agents and are learning different policies simultaneously. We show that not only is our curriculum-based approach better able to learn the desired behaviors, but that the learned policies also generalize against agents that were not included in the training process.

[1] Anahita Mohseni-Kabir is with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA anahitam@andrew.cmu.edu. This work has been conducted while she was at Honda Research Institute USA.

[2] David Isele and Kikuo Fujimura are with Honda Research Institute USA {disele, kfujimura}@honda-ri.com

## II. Related Work

Multiple works have focused on reinforcement learning methods for multi-agent domains in fully cooperative, fully competitive, and mixed environments [10]. Foerster et *al.* [6] propose a multi-agent actor-critic method to address the challenges of multi-agent credit assignment. Different from our work where each agent pursues its individual goal, their approach is appropriate for problems with a single shared task. Lowe et *al.* [7] propose an actor-critic approach, called MADDPG, that augments the agent's critic with action policies of other agents and is able to successfully learn the coordination policy between multiple agents. Unlike our approach these methods use centralized training. He et *al.* [11] presents new models based on deep Q-network for decision-making in adversarial games which jointly learns a policy and different strategies of opponents. Our approach is demonstrated in the non-stationary case where the agents are co-evolving together while the opponents in their approach have a set of fixed strategies.

In social dilemma research, most works focus on one-shot or repeated tasks [12], [13] and ignore that in real world scenarios the behaviors are temporally extended. Among the most relevant RL approaches in this area, is work on the sequential prisoner's dilemma (SPD) [14] which leverages deep Q-networks to study effects of environmental parameters on the agents degree of cooperation. In contrast to this work that focuses on the impact of the games' parameters on the agents' behavior, we provide a multi-agent learning approach for cooperative problems with individual goals. Another relevant work [15] proposes a deep RL approach for mutual cooperation in SPD games. Their approach adaptively selects its policy with the proper cooperation degree based on the detected cooperation degree of an opponent. Different from their approach, our approach is not specific to two player games. In addition, since we focus on mobile navigation problems our approach learns how to react to the continuous policies of other agents, not just their cooperation degree.

Significant amount of work has focused on motion planning for autonomous vehicles [16] where the problem of intention prediction or trajectory estimation of other agents has been studied. Among these, relevant work has focused on intention-aware POMDP planning for autonomous vehicles [17]. These methods leverage machine learning methods to learn models of other agents as also surveyed in [18]. In contrast to these approaches, we focus on RL algorithms for interaction-aware agents where the agents are co-evolving together and their motion models are dependent on others policies. Another work presents an approach for computing optimal trajectories for multiple robots in a distributed fashion [19]. In this work, the interactions between the robots are specified as a set of distance constraints, and each robot iteratively improves its graph search by using the constraints and other robots' trajectories. Differently, we focus on domains, such as the autonomous driving domain, where the agents do not have access to the other agents' trajectories. Our formulation is not limited to problems where the interaction is represented as a set of distance constraints, and the agents learn to interact with one another through trial-and-error.

## III. Approach

We focus on multi-goal multi-agent settings, where each agent cooperates with other agents in order to accomplish its individual goal. We leverage the intuition that in many settings, like autonomous driving, the interaction between multiple agents is limited to certain parts of the state space where conflict of interest is present, otherwise the agent behaves according to its single-agent policy. *I.e.*, the agent starts with its own single-agent policy and adapts it to account for the multiple agents that appear in the environment. We propose a two stages approach to learn multiple interactive policies for multiple agents. In the first stage of learning, the agents learn a single agent policy to accomplish their individual goals. We then freeze the single agent model and combine it with the multi-agent model. The multi-agent model is trained in the multi-agent setting to learn a modifier policy that accounts for the presence of the other agents.

### *A. Single Agent Module*

We model an agent with individual goal as a partially observable Markov Decision Process (MDP) [20] with goals [21]. The partially observable MDP is defined as a tuple $< \mathcal{S}, \mathcal{O}, \mathcal{A}, P, R, G, \mathcal{G}, \gamma >$ in which $\mathcal{S}$ represents the possible states of the world, $\mathcal{O}$ represents the agent's possible observations, $\mathcal{A}$ is a set of actions, $P : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ determines the distribution over next states, G is the agent's goal, $\mathcal{G}$ is the goal distribution, $R : \mathcal{S} \times \mathcal{G} \times \mathcal{A} \rightarrow \mathbb{R}$ is an immediate reward function, and $\gamma \in [0, 1]$ is the discount factor. In this work, since the noise between $\mathcal{O}$ and $\mathcal{S}$ is small, we use Markov Decision Process (MDP) approaches to solve the partially observable MDP. The solution to an MDP is a policy $\pi_\theta : \mathcal{O} \times \mathcal{G} \times \mathcal{A} \rightarrow [0, 1]$ where $\theta$ is the parameters of the policy. For continuous actions, $\pi_\theta$ is assumed to be Gaussian, and in our work the mean is represented by a neural network with parameters $\theta$. The robot seeks to find a policy $\pi_\theta$ that maximizes the expected future discounted reward $R = \sum_{t=0}^{T} \gamma^t r_t$. In the following paragraphs, we add subscript "self" or "s" to our notation to show the agent's own properties and "single" or "sng" to highlight the single agent scenario.

We use a decentralized actor-critic policy gradient algorithm to learn the single agent policies. The single agent model gets observation $o_s$ and goal $g_s$ as inputs, and outputs an action $a_s$. Each agent learns a policy $\pi_{sng}(a_s|o_s, g_s)$, according to its individual goal-specific reward function $R_{sng}(s_s, a_s, g_s)$ in the absence of other agents. The decentralized actor-critic policy gradient algorithm maximizes $J_{sng}(\theta) = \mathbb{E}_{o_s \sim p_s^\pi, a_s \sim \pi_{sng}, g_s \sim \mathcal{G}} [R_{sng}]$ by ascending the following gradient:

$$\nabla_\theta J(\theta) = \mathop{\mathbb{E}}_{\tau \sim p_\theta(\tau), g_s \sim \mathcal{G}} [\nabla_\theta \log \pi_{sng}(a_s|o_s, g_s) \\ A^\pi(o_s, a_s, g_s)]$$

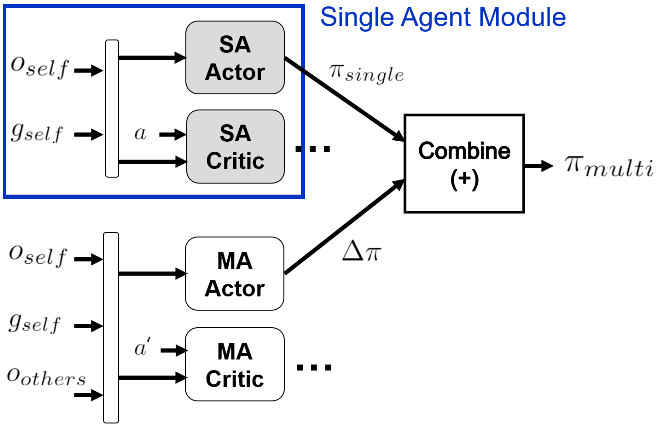


Fig. 2: Multi-agent (MA) module. The single agent actor-critic models (gray boxes) are frozen during training in the multi-agent setting. The output of SA and MA actor models are given as inputs $a$ and $a'$ to the critic models respectively.

Where $p_s^\pi$ is the state distribution, and $A^\pi$ is the advantage function [22]. We use $\tau \sim p_\theta(\tau)$ to refer to $o_s \sim p_s^\pi, a_s \sim \pi_{sng}$. For simplicity, $\theta$ is removed. The gray colored boxes in Fig. 2 show the actor-critic model for the single agent.

### B. Multiple Agents Module

We assume that each agent has a noisy estimate of the other agents' states, but does not have access to the other agents' actions. We model the multi-agent decision problem as a partially observable Markov Game [23], modified to accommodate mixed goals, and defined as the tuple $< N, \mathcal{S}, \{O_i\}_{i \in N}, \{A_i\}_{i \in N}, \{R_i\}_{i \in N}, \{G_i\}_{i \in N}, P, \mathcal{G}, \gamma >$ with N agents. The possible configurations of the agents is specified by $\mathcal{S}$. Each agent $i$ gets an observation $o_i \in O_i$ which includes both the agent's noisy observation of its own state $o_s$ and a noisy observation of other agents $o_o$. Each agent has its own set of actions $A_i$, a goal $G_i \sim \mathcal{G}$, and a reward function $R_i : \mathcal{S} \times \mathcal{G} \times A_i \rightarrow \mathbb{R}$. The markov game includes a transition function $P : \mathcal{S} \times A_1 \times ... \times A_N \rightarrow \mathcal{S}$ which determines the distribution over next states. The solution to the markov game is a policy for each agent $i$ $\pi_{\theta_i} : O_i \times \mathcal{G} \times A_i \rightarrow [0, 1]$ where $\theta_i$ is the parameters of the policy. For continuous actions problems, $\pi_{\theta_i}$ is assumed to be a Gaussian where the mean is modeled by neural networks. Each agent seeks to find a policy $\pi_{\theta_i}$ that maximizes its own expected future discounted reward $R_i = \sum_{t=0}^{T} \gamma^t r_{it}$. We remove $i$ for simplicity and add subscript "self" or "s" to our notation to show the agent's own properties. We add "multi" or "mlt" to highlight the multi-agent scenario, and subscript "others" or "o" refers to other agents' properties.

We modify each agent's $R_{sng}$ to account for the presence of other agents in the environment. Each agent is rewarded based on its individual objective, and punished if it gets into conflicts (*e.g.*, collisions in mobile agent scenarios). The new reward function for each agent is as follows where $C$ is a positive constant that penalizes the agent for conflicts, and $\mathbb{1}_{conflict}(s_s, s_o)$ determines if conflict is present:

$$\text{R}_{mlt}(s_s, a_s, g_s, s_o) = R_{sng}(s_s, a_s, g_s) - C \times \mathbb{1}_{conflict}(s_s, s_o)$$

We use a decentralized actor-critic policy gradient algorithm to learn the multi-agent policies. Each agent learns an actor-critic model that accounts for the multiple agents in the environment. The model gets observation $o_s$, goal $g_s$, and $o_o$ as inputs, and outputs an action $a_s$. Each agent learns a policy $\pi_{mlt}(a_s|o_s, g_s, o_o)$, according to its multi-agent goal-specific reward function $R_{mlt}(s_s, a_s, g_s, o_o)$ in the presence of other agents. The decentralized actor-critic policy gradient algorithm maximizes $J_{mlt}(\theta) = \mathbb{E}_{o_s \sim p_s^\pi, a_s \sim \pi_{mlt}, g_s \sim \mathcal{G}, o_o \sim p_o^\pi} [R_{mlt}]$ by ascending the gradient:

$$\nabla_\theta J(\theta) = \mathop{\mathbb{E}}_{\tau \sim p_\theta(\tau), g_s \sim \mathcal{G}} [\nabla_\theta \log \pi_{mlt}(a_s|o_s, g_s, o_o) \\ A^\pi(o_s, a_s, g_s, o_o)]$$

Where $p_o^\pi$ is the other agents' state distribution. We use $\tau \sim p_\theta(\tau)$ to refer to $o_s \sim p_s^\pi, a_s \sim \pi_{mlt}, o_o \sim p_o^\pi$.

Fig. 2 shows our architecture for both reaching the individual goal and cooperative behavior. The multi-agent module includes the single agent (SA) module from the previous stage of the curriculum. Each agent leverages its learned single agent actor-critic models that achieve its individual goal, freezes them and combines them with multi-agent actor-critic models that address the cooperative behavior. Compared to the single agent models, the multi-agent models have access to an estimation of the other agents' state $o_o$. For simplicity, Fig. 2 only shows the actor models, the critic models have the same structure. In this work, we used a summation to combine the single agent and multi-agent models since we found it sufficient in our experiments.

## IV. EXPERIMENTS

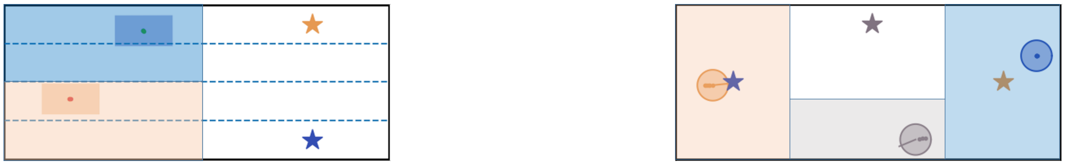
In this section, first we discuss our network architecture. We then delve into the details of the environments and experimental setup, and then discuss our results.

### A. Algorithm and Network Architecture

We use the Trust Region Policy Optimization (TRPO) [22] algorithm to learn the actor-critic models. We use the same architecture and parameters as the OpenAI baseline implementation [24]. We use ReLU as the activation function instead of tanh in the original implementation. The actor-critic models each have 2 hidden layers with 128 neurons. We call our approach Interaction-Aware TRPO or IATRPO.

### B. Simulation Environments

We tested our proposed approach on the following two environments. Both environments are designed such that interaction is required for successfully achieving the individual goals. The environments are shown in Fig. 3.



(a) Lane-change environment. The bottom and top agents start from the bottom left and top left quarters respectively.

(b) Robot navigation environment. Each agent starts from a random position in their corresponding side of the course.

Fig. 3: Environments. Agents are crossing one another to go to the stars on their opposite side of the course.

*1) Lane-change:* This environment consists of two cars that are crossing to go to their goal destination (matching color stars). At each episode a pair of non-adjacent goals are selected and are kept constant throughout the episode. The agents' positions are selected randomly in the top left and bottom left quarters of the course. The agents start with a 0 velocity, 0 angular velocity, and 0 heading angle. We call this environment "C2". We created an easier version of this environment where the goals are fixed to the 1st and 3rd lanes, and the agents' position has randomness only in the $+x$ direction. We call this environment "C2-fixed".

The agent's state includes its $x$ and $y$ position, velocity, angular velocity, heading angle, if it is broken due to collision with other agents or the environment, and if it has reached its goal. The observation noise for $s_s$ is in $[-0.01, 0.01]$. The cars have acceleration and angular acceleration as their actions $a_s$ with uniform noise in $[-0.1, 0.1]$. The car can reach a minimum and maximum velocity of $-1$ and $1$ respectively, and a minimum and maximum angular velocity of $-1$ and $1$ respectively. The multi-agent module for each agent has access to other agent's x and y positions, velocity, heading, angular velocity with a uniform noise in $[-0.1, 0.1]$. The cars use a bicycle kinematics model [25].

The reward function for the single agent scenario and the multi-agents scenario are as follows with a reward scale 3. Function $d$ computes the euclidean distance between the center of the car and agent's goal. Function $collision(s_s, env)$ or $collision(s_s, s_o)$ specify if the agent is in collision with the environment or the other agents respectively.

$$R_{sng}(s_s, a_s, g_s) = \begin{cases} -1, & \text{if } collision(s_s, env) \\ 1, & \text{if } d(s_s, g_s) < 0.4 \\ \frac{d(s_s, g_s)}{1000}, & \text{otherwise} \end{cases}$$

$$R_{mlt}(s_s, a_s, g_s, s_o) = R_{sng}(s_s, a_s, g_s) + \begin{cases} -1, & \text{if } collision(s_s, s_o) \\ 0, & \text{otherwise} \end{cases}$$

*2) Multi-Robot Navigation:* This environment consists of two or three mobile robots that are crossing one another to go to their goal destination (matching color stars). Three fixed goals are located at the top, left and right sides of the course. The agents' positions are selected randomly in the left (goal in right region of the environment), right (goal in left region), and bottom (goal in top region) of the course to assure that the agents pass one another to go to their goal position. The environment with 2 agents has the same setting, but the bottom (gray) agent is not present. We call the environment with 2 agents and 3 agents "R2" and "R3" respectively. The robots have the same state space and parameters as the cars in the lane changing environment. Each robot uses the unicycle kinematics model [26]. The reward function is as before.

## C. Results

We provide quantitative and qualitative results on the performance of our approach. As our baseline, we compare against MADDPG [7]. We leveraged the main contribution of the MADDPG approach and implemented the multi-agent version of the TRPO [22] algorithm (MATRPO) where each agent is provided with the action values of the other agents. TRPO was used in place of DDPG because it consistently outperform DDPG in all our experiments. TRPO was also used to train the IATRPO for all stages of the curriculum.

## D. Qualitative Results

Figures 4 and 5 illustrate the agents learned policies on C2 and R3 environments. In both scenarios, all the agents must cross paths to reach their goal destinations (matching color stars). We show the single agent policies (output of the single agent module) with dashed lines, and the multi-agent policies (output of the multi-agent module) with solid lines. We ran the learned multi-agent models and observed that the agents are successfully able to learn how to interact. We refer to the agents based on their colors or their start positions.

In the multi-agent policy in Fig. 4, the bottom agent (orange or abbreviated as O) slows down for the top agent (blue or B) to pass first and then it goes to its goals. The top agent (B) also modifies the path that it takes to not get into collision with the bottom agent. In the multi-agent policy in Fig. 5, the bottom agent (gray or G) learns to go first with maximum speed, the right agent (B) slows down and modifies its path for the bottom agent (G) to pass first. The left agent (O) modifies its speed to prevent a collision with the other agents. Notice that the single agent policies differ from the multi-agent policies both in the speed and the path that the agents take. If all the agents had executed their single agent policies, they would have collided with one another. Please refer to the video accompanying the paper to see examples of successful and failed executions.

## E. Quantitative Results

We evaluate the IATRPO approach against the MATRPO approach and report the final results in three evaluations:

**Success Rate:** Table I shows the success rate of our approach against the MATRPO approach. We ran both approaches on

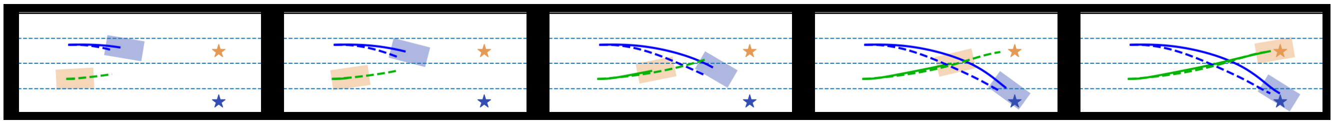


Fig. 4: IATRPO's final policy on the lane-change environment.

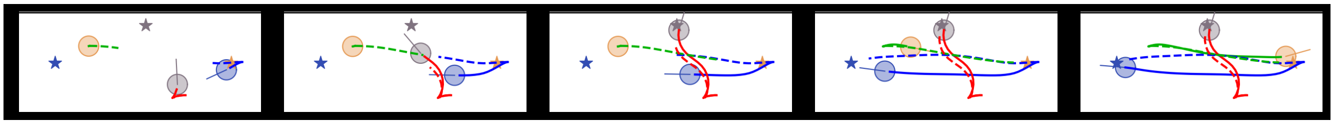


Fig. 5: IATRPO's final policy on the robot navigation environment.

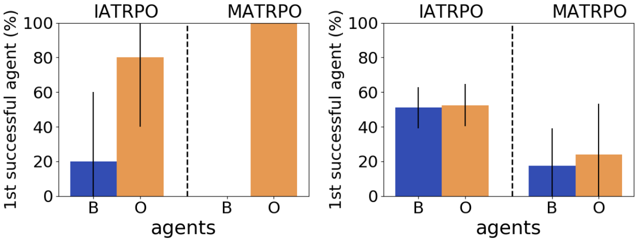
TABLE I: Success of the algorithms on the 4 environments.

| Environment | MATRPO success rate (%) | IATRPO success rate (%) |
|---|---|---|
| C2-fixed | $97.88 \pm 0.31$ | $99.4 \pm 0.33$ |
| C2 | $0 \pm 0$ | $94.3 \pm 2.99$ |
| R2 | $36.92 \pm 45.22$ | $90.96 \pm 1.43$ |
| R3 | $0 \pm 0$ | $88.02 \pm 4.11$ |

the 4 environments with 5 random seeds, and we report the mean and standard deviation of the results here. An episode is successful if all agents are within a distance of $0.4$ from their goals $d(s_{self}, g_{self}) < 0.4$. To compute the success rate, the final learned policies were run on $1000$ random episodes. Both MATRPO and IATRPO approaches give a high accuracy on the C2-fixed environment, but the success rate of the IATRPO algorithm is higher. On the C2 environment, which has a greater amount of randomness in the start position and has random goals, the MATRPO algorithm is not able to learn successful policies for all the agents, but the IATRPO algorithm has $94.3 \pm 2.99\%$ success rate. MATRPO learns a successful policy for one agent, but the second agent is stuck in a local optima, having only learned to not collide with the environment or the successful agent. Most of the failure cases in IATRPO happens around the boundaries of the environments or when the agents are too close to one another. We believe this is because of the noise associated with both the agent's action and the observations of others.

The performance of IATRPO is much higher than MATRPO on the R2 environment. In half of the random runs MATRPO did not learn a successful policy for both agents. However, IATRPO has a success rate of around $90\%$. MATRPO completely failed to learn a successful policy on the R3 environment, but IATRPO achieves a success rate of $88.02 \pm 4.11\%$.

**Level of Interaction:** To measure how interactive the policies learned by IATRPO and MATRPO are, we look at the extent of stereotyping. We ran both approaches on C2-fixed and R2 environments where MATRPO was able to learn successful policies. We performed 5 training runs with random seeds and tested the final learned policies on a $1000$ random episodes. We estimate how interactive the policies are by finding which agent reached its goal first (interactiveness metric). For each agent, we compute the mean and standard deviation of the interactiveness metric on the 5 runs. In each algorithm run if one agent always waits for the other agent to go first and compromises, the agents are considered non-interactive. In a two agents scenario, the ideal case is if both agents have a mean of around $50\%$ with a low standard deviation. Fig. 6 shows the results of the two algorithms on the C2-fixed and R2 environments. In each run on the C2-fixed where we applied the MATRPO algorithm, one agent always waited for the other agent to go first. However, IATRPO was able to learn more interactive policies than the MATRPO policies where both the agents sometimes compromised. Fig. 7 provides more evidence of why, in the MATRPO training, one agent always compromises.



(a) Results on C2-fixed.

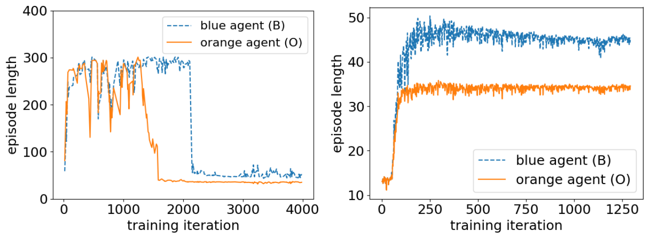
(b) Results on R2.

Fig. 6: Shows which agent achieved its goal first.

Fig. 7 shows the mean episode length of both the algorithms in one of the training runs on the C2-fixed environment. When the mean episode length becomes constant, the agent has converged to a successful policy. In the MATRPO training, the bottom agent (O) converges to the successful policy, and after about 800 training iterations the top agent (B) adapts its policy to the orange agent's policy and converges as well. However, with IATRPO the two agents learn a successful policy around the same time.

We applied both the algorithms on the R2 environment (Fig. 6) and noticed that in IATRPO the two agents have a better balance where an agent arrives first about $50\%$ of the times. The agents are less balanced when using MATRPO, and have higher variance than the IATRPO approach.

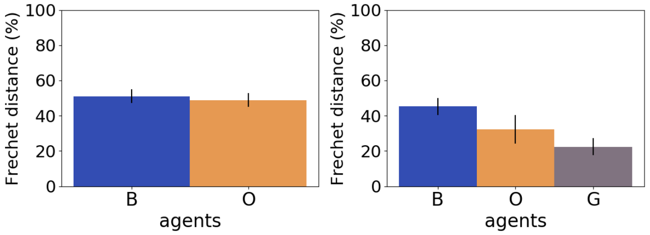
We also investigate the influence of our two stages approach on the interactiveness of the agents. We measured the distance between the single agent trajectories and the multi-agent trajectories to measure how much each agent modified its trajectory to account for the other agents in the IATRPO algorithm. We use the Fréchet distance for this comparison. As before, we use the final learned policy, run it on $1000$ random episodes and compute the distance between



(a) MATRPO training.

(b) IATRPO training.

Fig. 7: Mean episode length in one experiment for training multi-agent policies on the C2-fixed. IATRPO uses a curriculum so the number of iterations is not comparable.



(a) C2 environment.

(b) R3 environment.

Fig. 8: Fréchet distance between the single agent trajectories and the multi-agent trajectories in IATRPO algorithm.

the single agent module's trajectory and the multi-agent module's trajectory. For each agent, we average the computed distance and use that to compute the overall compromise $(\%)$ that each agent makes compared to other agents. Fig. 8 shows the results on the C2 and R3 environments. Although the top agent (B) gets the first place $72.43 \pm 37.19\%$ of the times, the changes in the distance is almost equal for the two agents in the C2 environment.

In the R3 environment, the bottom agent (G) always arrives first at the goal, but the distance between its single agent trajectory and multi-agent trajectory is about $20\%$. This implies that the bottom agent is also trying to adapt its policy to the other agents' policies. The right agent and the left agent get the second place $82.59 \pm 22.84\%$ and $18.46 \pm 22.64\%$ respectively. The overall impact of the right agent (B) on distance is $45\%$ compared to the left agent (O) $32\%$. This implies that both the agents change their single agent policies, the right agent mostly changes the path that it takes, and the left agent mostly changes its speed to account for the other agents.

**Generalizability:** We conducted experiments where we look at the performance of agents not trained together. This is a scenario known in the literature to cause agents to fail due to dependent policies [3], [4]. We used the 5 random training runs and generated 20 pairs (or triples) of agents where the agents in each pair are trained separately using a different seed. Table II shows the success rate of MATRPO and IATRPO on the four environments. We used the same approach as above to compute the success rate on a 1000 random episodes. The performance of the MATRPO algorithm is not affected much in C2-fixed experiments, but it has drastically decreased in R2 experiments. We believe the reason is that in C2-fixed experiments with MATRPO, the bottom agent (O) learns to always go first regardless of what the top agent (B) is doing. Even when the bottom agent is tested against other top agents (Bs), both the agents show the same behavior thus the performance of the algorithm does not get affected. However, in the R2 experiments with MATRPO, the agents show a more interactive behavior than C2-fixed, but have higher variance, thus when we test the agents which were not trained together against each other the performance drastically decreases.

TABLE II: Success rate of the algorithms on the 4 environments when tested on agents that were not trained together.

| Environment | MATRPO success rate $(\%)$ | IATRPO success rate $(\%)$ |
|---|---|---|
| C2-fixed | $97.83 \pm 0.72$ | $98.71 \pm 1.29$ |
| C2 | NA | $69.22 \pm 26.44$ |
| R2 | $2.18 \pm 8.56$ | $77.05 \pm 9.46$ |
| R3 | NA | $68.87 \pm 17.44$ |

The success rate of IATRPO also decreases when we test it on the 20 pairs (or triples). The performance degrades less on the easier environments such as C2-fixed and on the environments where the agents learned a more interactive policy with low variance such as R2. The success rate for the C2 and R3 degrades more than R2 since the agents learned a less interactive behavior with high variance.

## V. CONCLUSIONS AND DISCUSSION

We focus on multi-agent settings where each agent learns a policy to simultaneously achieve its individual goal and cooperate with other agents. We provide a curriculum learning approach and an architecture that learn how to adapt single agent policies to the multi-agent setting. We showed the performance of our approach on two robotics problems where learning multi-agent interactive policies are essential for the agents in order to achieve their goals.

In future work, we plan to address the following issues: 1) our formulation is generalizable to domains with inhomogeneous agents since we make no assumptions regarding homogeneity, our future work involves testing the approach on such domains. 2) If a model is learned in an environment with $N$ agents, we can apply the same model on an environment with $<= N$ agents where we assume the non-existent agent is at a position far away from the others. However, a limitation of our work is that a new model should be learned if we increase the number of agents. We believe this can be handled by only considering robots within a limited sensor range (the $N$ nearest neighbors). 3) To remove the assumption of always having an estimate of the other agents' state information in real robotic systems, each robot can maintain (learn) an approximation of the true state information of the other robots [27]. 4) Finally, to address difficulties in transferring policies that are learned in simulation to the real-world [28], we plan to test the ensemble-of-policies method proposed in [29] to enable the agents to learn policies that are more robust to the changes in the policies of others.

# References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, 2015.

[2] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *IJRR*, 2013.

[3] M. Raghu, A. Irpan, J. Andreas, R. Kleinberg, Q. V. Le, and J. Kleinberg, "Can deep reinforcement learning solve erdos-selfridge-spencer games?" *arXiv preprint arXiv:1711.02301*, 2017.

[4] M. Lanctot, V. Zambaldi, A. Gruslys, *et al.*, "A unified game-theoretic approach to multiagent reinforcement learning," in *NIPS*, 2017.

[5] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *ICML*, 1993.

[6] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," *arXiv preprint arXiv:1705.08926*, 2017.

[7] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *NIPS*, 2017.

[8] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. V. D. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, 2016.

[9] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *ICML*, 2009.

[10] L. Busoniu, R. Babuska, and B. D. Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE SMC-Part C*, 2008.

[11] H. He, J. Boyd-Graber, K. Kwok, and H. D. III, "Opponent modeling in deep reinforcement learning," in *ICML*, 2016.

[12] P. Mathieu and J. Delahaye, "New winning strategies for the iterated prisoner's dilemma," in *AAMAS*, 2015.

[13] J. Hao and H. Leung, "Introducing decision entrustment mechanism into repeated bilateral agent interactions to achieve social optimality," *AAMAS*, 2015.

[14] J. Z. Leibo, V. Zambaldi, M. Lanctot, J. Marecki, and T. Graepel, "Multi-agent reinforcement learning in sequential social dilemmas," in *AAMAS*, 2017.

[15] W. Wang, J. Hao, Y. Wang, and M. Taylor, "Towards cooperation in sequential prisoner's dilemmas: a deep multiagent reinforcement learning approach," *arXiv preprint arXiv:1803.00162*, 2018.

[16] B. Paden, M. Čáp, S. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IV*, 2016.

[17] H. Bai, S. Cai, N. Ye, D. Hsu, and W. Lee, "Intention-aware online pomdp planning for autonomous driving in a crowd," in *ICRA*, 2015.

[18] S. V. Albrecht and P. Stone, "Autonomous agents modelling other agents: A comprehensive survey and open problems," *Artificial Intelligence*, 2018.

[19] S. Bhattacharya, M. Likhachev, and V. Kumar, "Multi-agent path planning with multiple tasks and distance constraints," in *ICRA*, 2010.

[20] C. C. W. III and D. J. White, "Markov decision processes," *EJOR*, 1989.

[21] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *ICML*, 2015.

[22] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *ICML*, 2015.

[23] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine Learning Proceedings*, 1994.

[24] P. Dhariwal, C. Hesse, O. Klimov, *et al.*, "Openai baselines," https://github.com/openai/baselines, 2017.

[25] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design." in *IV*, 2015.

[26] G. Indiveri, "Kinematic time-invariant control of a 2d nonholonomic vehicle," in *CDC*, 1999.

[27] T. D. Barfoot, *State estimation for robotics*. Cambridge University Press, 2017.

[28] P. Christiano, Z. Shah, I. Mordatch, J. Schneider, T. Blackwell, J. Tobin, P. Abbeel, and W. Zaremba, "Transfer from simulation to real world through learning deep inverse dynamics model," *arXiv preprint arXiv:1610.03518*, 2016.

[29] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch, "Emergent complexity via multi-agent competition," *arXiv preprint arXiv:1710.03748*, 2017.