Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

Constraint-Based Multi-Agent Reinforcement Learning for Collaborative Tasks

Xiumin Shang*[1] | Tengyu Xu[2] | Ioannis Karamouzas[3] | Marcelo Kallmann[1,4]

[1]Electrical Engineering and Computer Science, University of California Merced, California, United States.
[2]Meta Platform Inc.
[3]School of Computing, Clemson University, South Carolina, United States.
[4] Marcelo Kallmann is now a Principal Scientist at Amazon Robotics - this work was initiated before joining Amazon and is not related to Amazon work.

Correspondence
*Xiumin Shang, University of California Merced, California, United States. Email: xshang@ucmerced.edu

Summary

In order to be successfully executed, collaborative tasks performed by two agents often require a cooperative strategy to be learned. In this work, we propose a constraint-based multi-agent reinforcement learning approach called Constrained Multi-agent Soft Actor Critic (C-MSAC) to train control policies for simulated agents performing collaborative multi-phase tasks. Given a task with n phases, the first $n-1$ phases are treated as constraints for the final task phase objective, which is addressed with a centralized training and decentralized execution approach. We highlight our framework on a tray balancing task including two phases: tray lifting and cooperative tray control for target following. We evaluate our proposed approach and compare it against its unconstrained variant (MSAC). The performed comparisons show that C-MSAC leads to higher success rates, more robust control policies, and better generalization performance.

KEYWORDS:
multi-agent, reinforcement learning, tray balance task, collaborative tasks

1 | INTRODUCTION

In recent years deep Reinforcement Learning (RL) approaches have shown great potential in training control policies used in gaming and robotics. In particular, a family of actor critic algorithms has been developed for tackling tasks with complex continuous action spaces which have become widely used in the field of deep RL. Extensive research and development have

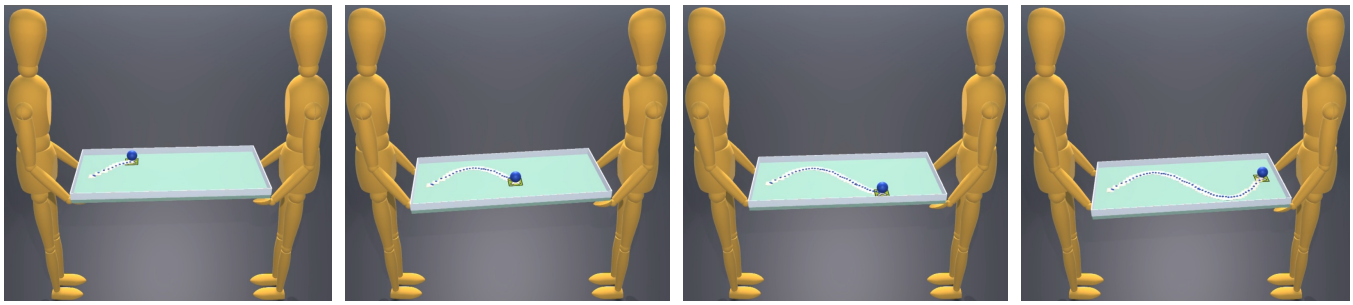


Figure 1 Agents collaboratively balancing a tray for a ball to follow a moving target.

been dedicated to such methods, which include Deep Deterministic Policy Gradient (DDPG)[1], Proximal Policy Optimization (PPO)[2], and Soft Actor Critic (SAC)[3]. In this paper we address the use of SAC for multi-phase collaborative tasks.

Many applications such as gaming and robotics often depend on multiple agents cooperating together to solve a complex task. Some of these tasks are sequential in nature and have distinct phases that are governed by different objectives as the task progresses. While it is generally easy for humans to discover cooperative interaction strategies, training agents to handle such tasks is a fairly complex problem, mostly because of two fundamental challenges. The first is related to building a framework where the agents can interact in a collaborative manner, and the second pertains to the training of the agents allowing them to take appropriate actions while intelligently cooperating with each other in order to achieve a common task objective.

With respect to the first challenge, there is a rich collection of available frameworks for experimenting with single-agent reinforcement learning algorithms, such as OpenAI Gym[4], RoboGym[5], ML-Agents[6], and Atari games[7]. These frameworks have been used for controlling both simple objects and complex human-like characters or robots, and as well for learning policies in an end-to-end manner. There are also frameworks developed for specific purposes. Examples include controlling an agent to play Go at a world-class level[8], controlling a robotic arm to grasp or manipulate objects[9], and simulating physical characters to perform realistic human level skills[10]. However, building a framework for multi-agent problems is significantly more challenging because the behavior of one agent is affected by the other agent, requiring synchronizing movements in the same dynamic environment and designing a reward function that incentivizes coordination or competition according to possibly different objectives.

With respect to the second challenge, single-agent RL approaches have already been extended to the multi-agent domain for solving tasks that need multi-agent interactions, such as for two-player competitive sports[11] and cooperative agent communication[1,12,13]. In such domains, the centralized training and decentralized execution (CTDE) paradigm, and its integration with DDPG[14], called MADDPG[1], has been widely used. In this paradigm, agents are trained using centralized information but execute separate policies in a decentralized manner. We adopt a similar architecture in our work, but rely on the SAC learning method[3] because of its superior ability to solve tasks in continuous action spaces over other actor critic learning methods.

While CTDE can learn both cooperative and competitive strategies, directly using it to solve a complex task that can be decomposed into multiple phases is not straightforward due to the complexity of designing a proper multi-objective reward function. A common solution is to use hierarchical RL controllers to solve each sub-objective separately with lower-level controllers, and then train an upper-level controller to provide intermediate goals for the lower-level ones[15,16]. However, implementing multi-level controllers introduces complexity to the architecture and requires additional hyperparameter tuning.

To address these issues, we adopt the safe RL concept from Xu et al.[17]. The original idea behind safe RL is to define task constraints for the purpose of safety and stability. However, constraints can also be leveraged to model the different sub-objectives of a task which can be decomposed into sequential sub-tasks or phases. In this paper we propose to treat all sub-objectives except the final one as constraints inside a multi-agent learning framework, and optimize the final objective only if none of the constraints' objectives are violated for each agent.

We define a tray balancing task (Figure 1) in order to train and evaluate the proposed training approach. To solve it, two agents need to cooperate for controlling the position and orientation of a tray. They need to control it precisely so that a ball rolling on the surface of the tray can follow a pre-defined target trajectory. We define two phases: the objective of the first phase is for the agents to lift the tray appropriately, and the objective of the second phase is to control the tray in order to make the ball follow its target trajectory. We study the performance of our model by analyzing the objective rewards for different trajectories. We also evaluate the model on trajectories that are unseen during training in order to study the generalization capability of our method, and we analyze our model robustness by evaluating its performance in the presence of disturbance forces.

Overall, our contributions to this work can be summarized as follows:

- we develop a multi-agent framework able to physically simulate collaborative tasks in a shared environment,
- we propose a constraint-based algorithm to learn policies for multi-agent cooperative tasks with sequential objectives, and
- we evaluate the trained policies with tray balance tasks in order to study their performance, generalization, and robustness.

2 | RELATED WORK

2.1 | Multi-Agent Environment

Building multi-agent environments is very challenging since the reward design has to not only consider task objectives but also interaction strategies between agents. Works on multi-agent learning have addressed control and communication strategies between 2D agents [1], and also emergent strategies during learning [18,19]. There is however a lack of frameworks for experimentation with human-like behaviors achieved from joint-level control. Recently, one work has created such an environment for training human-like characters to compete with each other in sports games like fencing and boxing [11]. While these environments are task-focused, our environment expands the possibilities for accomplishing cooperative tasks that include human upper-body and arms movement.

2.2 | Multi-Agent Reinforcement Learning

The multi-agent learning problem is challenging because of difficulties with agent scalability, non-stationarity of the environment, and non-unique learning objectives. A straightforward approach to solving multi-agent control problems is to train a joint action policy for all agents using a single-agent RL algorithm, or directly extend single-agent RL algorithms where each agent learns independently by considering other agents as part of the environment, such as by independent Q-learning [20]. However, the above solutions suffer from scalability and stability issues. Lowe et al. [1] proposed a parameter sharing approach to tackle this problem, called centralized training and decentralized execution approach (CTDE). This is extended from the actor-critic framework, where each agent uses a centralized critic to access all agents' observations and action parameters so that it can learn an approximate model of the other agents' online policies within a stationary environment. The learned policy only uses local information so it can be used by each agent without further communication between agents. Thereafter, additional algorithms have been developed based on the CTDE framework in order to address different aspects of typical multi-agent systems, for example, credit assignment with integration of the independent actor-critic method [21], scalability by adding mean field Q learning and actor-critic learning [22], stability with SAC learning [23], and incorporation of attention with SAC learning [12] in order to enable faster and more stable learning [24].

Our work also adopts the concept of improving the stability in multi-agent learning [23] by integrating SAC [3] into the CTDE framework. However, in comparison to other recent approaches [23,25,26], we study the applicability of CTDE with SAC in a complex human coordination task that is decomposed into multiple phases. As evaluation (see Section 6) we use CTDE with SAC as a baseline to evaluate our proposed constraint-based learning model.

2.3 | Multi-Objective Learning

Multi-objective learning involves learning tasks that have two or more objectives to optimize. It has the lifelong learning properties [27], which means an agent can be trained on a sequence of relatively easy tasks to gain experience and develop a more informative measure of reward, which can then be leveraged when performing harder tasks. Complex tasks that need to be broken down into sub-tasks based on their sub-objectives are quite common. When designing the objectives of a given task, the sub-objectives can be defined as separate objectives without connection but sharing the same action space, like a robot arm picking and placing objects in different boxes [28], agents working together to push different objects into different locations [29]. Sub-objectives can also be defined as sequential objectives, such that in order to finish the final objective, the previous objectives have to be completed first, which is aligned with our task design. Example tasks solved in this manner are: an agent moving to a target location while moving objects or obstacles along the moving path [30], a biped character walking and needing to maintain natural gait motion with a walking target [31], and a four-legged robot walking to a target location [15]. While most of existing works focus on optimizing multi-objective learning for a single agent, in this paper, we extend the approach to a multi-agent multi-task setup, where two virtual agents need to control the force applied on each side of a tray in order to solve a cooperative tray balancing problem.

2.4 | Reinforcement Learning with Constraints

Learning while considering constraints is a popular approach used in safe reinforcement learning [32], where the focus is on preventing the agent from taking actions or entering states that are too risky, since ensuring safety is always critical when the

learned strategies need to be deployed to real world systems. Different ways of specifying constraints have been proposed by previous researchers, including using constraints on the expected return or cumulative costs[33,17], and defining regions of the state space that will result in agent failure. In our work, we focus on addressing a new formulation for a cooperative task that involves two sequential objectives, and we propose to define our constraint as a threshold on the expected return from the first objective. The learning process will advance to optimize the final objective only when the constraint has been satisfied.

3 | OVERVIEW

We are interested in solving multi-agent collaborative tasks in a physically-simulated environment, with focus on a tray balancing task. In this section we provide details of our task design. The overall framework used to train and evaluate our method is illustrated in Figure 2.

3.1 | Environment and Tasks

The environment is simulated inside the physics simulator of the Unity game engine, and it includes two virtual agents, a ball, a moving target, and a tray with four anchor points at each corner. We consider two trajectories for the tray balancing task: an ellipse trajectory and an S-curve trajectory as shown in figure 3. The details of these trajectories are described in Section 6.1. In order to complete the task, the two agents start with a standing initial pose along the two sides of the tray, and then outstretch their arms to reach the two anchor points on their side of the tray. The arms of the humanoids are controlled by inverse kinematics (IK)[34], the end effectors are their hands' position and rotation, and the two anchor points' position and rotation are their desired targets. After reaching the anchor points, we require the agents to perform the tray balancing task in a sequential manner in 2 phases: in phase 1, they need to lift up the tray to a fixed height while maintaining the balance of the tray, and in phase 2, they need to manipulate the tray to guide the ball to follow the target on a moving trajectory. The task process inside the physics simulator in Figure 2 illustrates this process.

3.2 | Framework

Our framework has three main components, the physics simulator, the MARL (Multi-Agent Reinforcement Learning) controller, and the IK (Inverse Kinematic) controller. The physics simulator is used to simulate different task environments, the MARL controller is the Python API that runs our multi-agent learning approaches during both the training and execution processes, and the IK controller is used to control the humanoid arms during both processes. At each learning step, the MARL controller receives the states and rewards information and sends out the actions to the physics simulator via the communication channels provided by the Unity ML-Agents plugin[35]. In the meantime, the MARL controller collects the anchor points' position and rotation information and sends them to the IK controller as its desired control target for the agents' arm movement.

4 | APPROACH

4.1 | Problem Formulation

We formulate our problem as a two-agent cooperative game, modeled as a partially observable Markov Decision Process (MDP); our approach can be easily extended to more agents. At each time step $t \in [0, T]$, the MDP tuple is defined as $\{\mathcal{O}, \mathcal{S}_i^t, \mathcal{A}_i^t, \mathcal{S}_i^{t+1}, \mathcal{R}_i^t, \mathcal{T}\}$, where T is the max steps of each episode, $i \in \{0, 1\}$ is the index of the agent. Agent i observes partial state $s_i^t \in \mathcal{S}_i^t$ from environment $\mathcal{O}$, takes action $a_i^t \in \mathcal{A}_i^t$ that leads to a new state according to the dynamics model $\mathcal{T}$, and receives a scalar reward signal from its reward function $r_i^t = \mathcal{R}(s_i^t, a_i^t)$, and $r_i^t \in [0, 1]$. For each agent i, the goal is to find an optimal policy $\pi_{\theta_i}(a|s)$ that maximizes its expected accumulated reward J_i, where θ_i denotes the parameters of the policy.

4.2 | Multi-Agent Soft Actor Critic Learning (MSAC)

In our multi-agent setup, we adopt the CTDE framework incorporated with SAC algorithm[3] because its extra entropy term increases the policy's exploration ability and robustness, and we name it Multi-agent Soft Actor Critic algorithm (MSAC). During

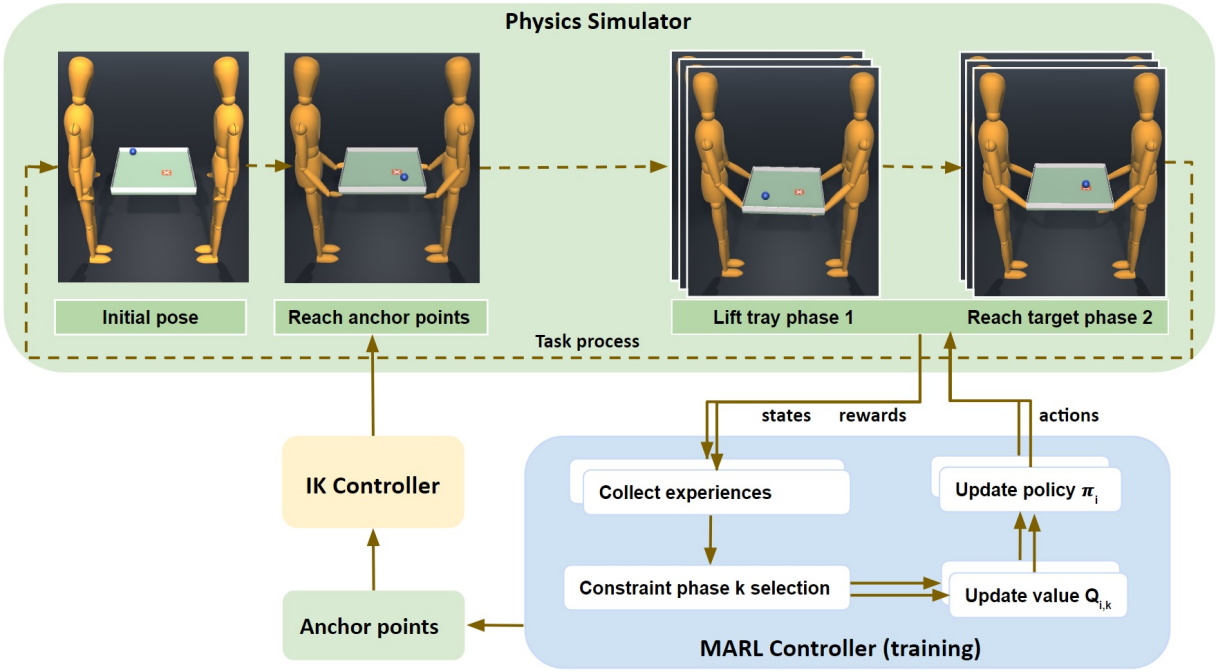


Figure 2 The overview of our environment and framework.

training, we jointly conduct policy evaluation and policy iteration, where for each agent i we concurrently learn a stochastic policy π_{θ_i} and two Q-functions with parameters $\phi_{i,j}$, j is identifier of Q function, in the range of $[1, 2]$.

During policy iteration, we use Eq 1 to optimize, in the direction of maximizing the accumulated reward, the policy function $J(\theta_i)$:

$$\nabla_{\theta_i} J(\theta_i) = \nabla_{\theta_i} \frac{1}{|\mathcal{B}_i|} \sum_{\mathcal{B}_i} \{\min_{j=1,2} Q_{\phi_{i,j}}(\mathbf{s}^t, \mathbf{a}^t) - \alpha \log \pi_{\theta_i}(a_i^t | s_i^t)\}. \tag{1}$$

The policy function for each agent is parameterized as θ_i, $\mathcal{B}_i$ denotes a batch of experiences sampled from the replay buffer, $\log \pi_{\theta_i}(a_i^t | s_i^t)$ is the entropy term to increase the variation of action space, and α is a learned variable indicating the contribution of an entropy regularization term. $Q_{\phi_{i,j}}(\mathbf{s}^t, \mathbf{a}^t)$ is calculated from the states $\mathbf{s}$ and actions $\mathbf{a}$ of all agents, as a centralized evaluation operation.

During policy evaluation, we optimize the loss function $\mathcal{L}(\phi_{i,j})$ using Eq 2 to evaluate the learned policies by minimizing the difference of the value function $Q_{\phi_{i,j}}(\mathbf{s}^t, \mathbf{a}^t)$ and its target value y_i for each agent:

$$\nabla_{\phi_{i,j}} \mathcal{L}(\phi_{i,j}) = \nabla_{\phi_{i,j}} \frac{1}{|\mathcal{B}_i|} \sum_{\mathcal{B}_i} (Q_{\phi_{i,j}}(\mathbf{s}^t, \mathbf{a}^t) - y_i)^2. \tag{2}$$

The target value y_i is calculated with a separate target value network $Q_{target}(s^{t+1}, a^{t+1})$ in order to maintain stability when updating policy network, as shown in Eq 3:

$$y_i = r_i + \gamma \{\min_{j=1,2} Q_{\phi_{i,j,target}}(\mathbf{s}^{t+1}, \mathbf{a}^{t+1}) - \alpha \log \pi_{\theta_i}(a_i^{t+1} | s_i^{t+1})\}. \tag{3}$$

Both value functions $Q_{\phi_{i,j}}$ and their target value functions $Q_{\phi_{i,j,target}}$ share the same network structure, and they are parameterized as ϕ_i and $\phi_{i,target}$ separately.

4.3 | MSAC with Constraints

Constraints used in safe RL problems can be categorized into primal and primal-dual approaches. Primal approaches focus on the design of objective functions and do not need Lagrange multipliers as dual variables during the optimization process, thus simplifying the implementation process and reducing training time [32]. Due to its proven global convergence, we adopt the primal approach [17] in our framework and call it Constrained Multi-agent Soft Actor Critic (C-MSAC).

We assume the task can be divided into n sequential phases; we consider objectives in the first $n-1$ phases as constraints while the n^{th} phase objective determines the final objective. The idea is to optimize the final objective function with reward accumulated from the n^{th} phase while guaranteeing that all constrained objectives from previous phases can be satisfied. Our multi-agent problem with constraints can thus be formalized as the following optimization problem:

$$\max_{\theta_i} J_{i,n}(\theta_i), s.t. J_{i,k}(\theta_i) >= d_{i,k}, k < n, \tag{4}$$

where $J_{i,k}$ denotes the total expected return of the k-th constraint phase, and $d_{i,k}$ is a threshold that is calculated from a Monte Carlo estimate return with step reward $r_0 = 0.9$, and discount parameter $\gamma = 0.99$ for the k-th constraint:

$$d_{i,k} = \sum_{t=0}^{T-1} \gamma^t r_0. \tag{5}$$

Hence, when all constraint phases reach their respective thresholds $d_{i,k}$, we find the optimal policy $\pi_{\theta,i}$ for each agent. For each phase k, we learn a separate value function $Q_{i,k}$, that is, we learn n Q functions for each agent, and during each optimization step, one Q function will be optimized based on Eq 7. In this section, we use $Q_{i,k}$ as the shortened version of $Q_{\phi_{i,j,k}}$ for simplicity.

At phase k, the expected return $J_{i,k}$ is calculated as:

$$J_{i,k}(\theta_i) = \frac{1}{|m|} \sum_{s_i^0, a_i^0 \sim \xi, \pi_{\theta_i}}^{m} \rho_{i,k} Q_{i,k}(s_i^0, a_i^0). \tag{6}$$

The term $Q_{i,k}(s_i^0, a_i^0)$ computes the accumulated return of a rollout starting from (s_i^0, a_i^0) while following policy π_{θ_i}. State s_i^0 denotes an initial state that is sampled uniformly from an initial distribution ξ and a_i^0 is the corresponding action based on the latest policy π_{θ_i}. $\rho_{i,k}$ is a weight ratio for each rollout, where a value of 1 is used in our experiments. We average $m = 40$ rollouts to calculate $J_{i,k}$.

To combine the constraints with policy iteration and policy evaluation, we integrate the constraint phases into the MARL process. At each optimization step, we first choose a phase k to optimize by selecting a Q function using Eq 7:

$$select(Q) = \begin{cases} Q_{i,k} \text{ if } \exists k < n : J_{i,k}(\theta_i) < d_{i,k}, \\ Q_{i,n} \text{ else } k = n. \end{cases} \tag{7}$$

To perform policy iteration, we then replace $Q_{\phi_{i,j}}$ in Eq 1 with the chosen value function $Q_{i,k}$. For policy evaluation, we replace $Q_{\phi_{i,j}}$ in Eq 2 with the value function $Q_{i,k}$ of the chosen constrained phase, leading to the following critic update:

$$\nabla_\phi \mathcal{L}(\phi_{i,k}) = \nabla_\phi \frac{1}{|\mathcal{B}_i|} \sum_{\mathcal{B}_i} (select(Q) - y_{i,k})^2, \tag{8}$$

where $y_{i,k}$ is the target value calculated in the same way as in Eq 3, but specifically for the ith agent at phase k. If multiple constrained phases are not satisfied, we can choose any phase to maximize its expected return.

4.4 | Algorithm

Algorithm 1 summarizes the procedure of our proposed approach C-MSAC during the training process. Lines 4-9 show the process of collecting samples from multiple environments for both agents in order to enrich the training dataset and speed up the training. Lines 10-17 perform the learning process in that we iteratively select a phase k to optimize until all phases have been saturated with their own thresholds based on Eq 4.

5 | TRAINING

5.1 | State and Action Representation

In our environment, the state information includes the tray, the ball, the moving target, and the two anchor points for each agent. Let p, q, v, qv denote position, rotation, velocity, and angular velocity respectively. The state is represented as $s_i = [s^B, s^T, s^t, s^{Bt}, s^{Tf}]$, where $s^B = [p_B, q_B, v_B, qv_B]$ represents the state of the ball, $s^T = [p_T, q_T, v_T, qv_T]$ represents the state of the tray, s^t is the position of the moving target, s^{Bt} contains the euclidean and quaternion distance between the ball and

Algorithm 1 Constrained Multi-agent Soft Actor Critic Algorithm (C-MSAC)

Initialize policy network θ_i, value estimation networks $\phi_{1,i}$, $\phi_{2,i}$ with random weights for each agent;
Initialize replay buffer $\mathcal{D}$ as empty dictionary ;
for each step **do**
  for each environment m **do**
    Sample action a^t from policy $\pi(a^t_{m,i} \Big| s^t_{m,i})$;
    Proceed one step in the environment;
    Observe reward $r^t_{m,i,k}$ and next state $s^{t+1}_{m,i}$ from environment;
    Concatenate all data into tuple (s^t_m, a^t_m, $r^t_{m,k}$, s^{t+1}_m) and send to the replay buffer $\mathcal{D}$;
  end for
  if step > batch_size **then**
    Sample a group s^0_i from ξ_i;
    Calculate total return $J_{i,k}$ with Eq 6 for all phases k ;
    Sample a batch data from $\mathcal{D}$;
    Select a phase k to optimize with Eq 7 ;
    Evaluate policy with Eq 8 ;
    Optimize policy with Eq 1
  end if
end for
Output optimal policy θ_i for agent i.

the moving target that the ball needs to follow for, and s^{Tf} includes the euclidean distance between the tray and a fixed target position that is slightly higher than the tray's initial position.

The action a^k_i in our task is defined as the force applied at each holding point on the tray in x, y, z direction. It is calculated by multiplying a fixed scalar to the normalized [-1,1] control signals learned from the RL policy. We use 100 as the scaling constant.

5.2 | Reward Design

The tray balance task includes two phases: lifting up the tray and reaching the target, we need to design reward functions for each phase based on its own objective, the details are described below.

5.2.1 | Lifting the Tray

In this phase, the agent's objective is to lift up the tray to the fixed target position p_0 while maintaining its balance relative to the identity quaternion q_0. To satisfy this objective, the reward function is designed as:

$$r_{lift} = 0.8 * r_{dist} + 0.2 * r_{ang}.$$

Here r_{dist} is the Euclidean distance between the tray and the fixed target position, to encourage the tray to move closer to the fixed target position, and it is calculated:

$$r_{dist} = \exp[-5||p_T - p_0||^2].$$

r_{ang} is the quaternion orientation difference between the tray and the identity quaternion, to encourage the tray to balance itself, it is calculated:

$$r_{ang} = \exp[-20(1 - ||q_T \ominus q_0||^2)].$$

5.2.2 | Reaching the Target

In this phase, the agent's objective is to adjust the tray position and orientation to control the ball to follow a moving target on the tray. To encourage the ball to stay on the moving target, we use the Euclidean distance between the ball and the moving target, as represented in the reward function:

$$r_{target} = \exp[-25||p_B - s^t||^2].$$

The target's moving path is controlled by curve equations. We use two different curves: an ellipse (where the major and minor axes are randomized at the beginning of each training episode) and an S-curve which is defined by a cubic equation.

The agents have to collaborate to finish both phases' objectives, leading to a Nash equilibrium[36], meaning that each agent is taking their best policy to respond to the other agent, and its gains will be undermined if a different policy is taken. Considering the equilibrium, the reward we use for each agent in each phase is the shared team average reward, and calculated as the average of both agents in that phase[37]: $r = \frac{1}{N}\sum_{i \in N}(r_i)$.

5.3 | Early Termination

Early termination[10,38] is a common technique used in reinforcement learning, to help RL agents stop learning bad behaviors and favor the collected samples more efficiently, thus achieving significantly faster learning. In our task, we introduced three early termination conditions to terminate the current episode during training:

- first, when the ball touches the edge of the tray, because it can easily get stuck in the corner of the tray;
- second, when the ball or tray drops on the ground, as it will never get recovered from this state;
- third, when the height of any anchor point goes below a threshold (0.85m in our experiments), as the agent's hands will not be able to reach them with IK solution).

5.4 | Training Details

5.4.1 | Network Architecture

For each agent, we use the same network structure as SAC[3], but add $(n-1)$ value networks and target value networks to represent the constraint objectives of each phase. All networks consist of three fully connected layers with 256 hidden units in the first two layers, and relu is used as the activation function. The last layer outputs the mean and log value of the policy. The learning rate is 0.0003, while the gamma is 0.99.

5.4.2 | Multi-Environment Training

One of the learning thresholds for an RL approach is sample efficiency. To speed up the training we implement our environment as a multi-environment setup allowing us to collect 4 times training data per frame rate.

6 | EXPERIMENTS AND RESULTS

6.1 | Target Trajectories

In our environment, the xoz and $y-up$ coordinate system is being used. We train both models on two types of parametric trajectories, as shown in Figure 3:

1) *Randomized ellipse*: described with the ellipsoid equation:

$$f(t_i) = (a\cos(\theta_1(t_i))\cos(\theta_2(t_i)),\ b\sin(\theta_1(t_i)),\ b\cos(\theta_1(t_i))\sin(\theta_2(t_i)),$$

where θ_1 is the longitude angle change of meridian plane xoy in radian, θ_2 is the latitude angle change of the equatorial plane xoz in radian. a, b are the major axis and minor axis in the range of $[0.1, 0.3]$ for the ellipse shape randomization. The time step term t_i is the ratio of the current training step and total steps for each episode.

2) S-curve: described as a cubic Bèzier curve with control points P_0, P_1, P_2, and P_3.

$$f(t_i) = (1-t_i)^3 P_0 + 3(1-t_i)^2 t P_1 + 3(1-t_i)t^2 P_2 + t_i^3 P3.$$

In our task, we fixed points P_0 and P_3, and adjusted P_1, P_2 to get the desired S-shaped curve.

For the evaluation of unseen trajectories, we use two additional types of curves: square and triangle, as shown in Figure 4, they are generated by connecting three(triangle) and four (square) fixed positions relative to the position of the tray.

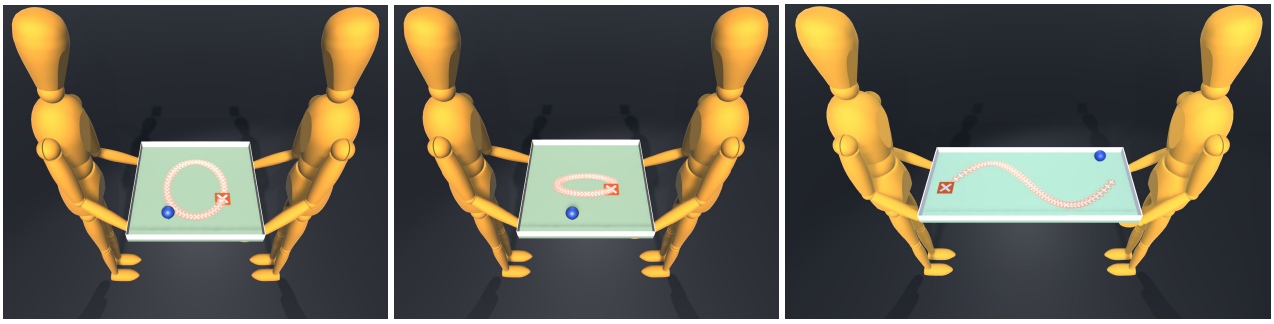


Figure 3 Left and center: ellipse examples, Right: S-curve example.

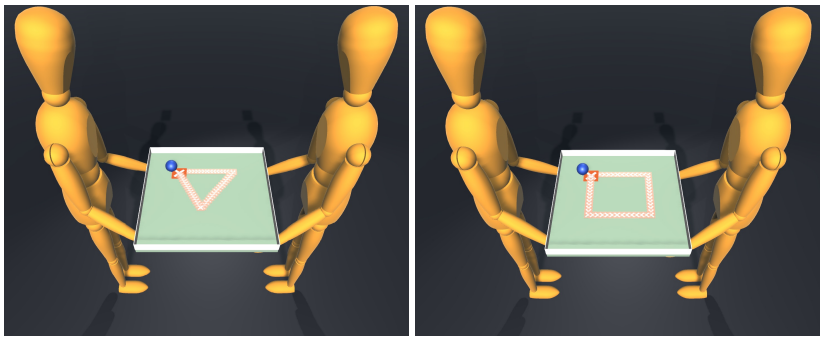


Figure 4 Generalization to unseen trajectories. Left: triangle. Right: square.

6.2 | Evaluation Metrics

We compare our proposed approach of multi-agent soft actor critic with constraints (C-MSAC) to MSAC approach. For MSAC we used a linear combination of rewards from each phase to calculate the total reward for each agent:

$$r_i = 0.85 r_{lift} + 0.16 r_{target} + (-0.1) r_{time}$$

where the optimization process is calculated according to Section 4.1. For C-MSAC each phase is optimized separately based on the reward accumulated from this phase as discussed in Section 4.3.

We compare both models on three criteria:

- **Mean target reward and on-target performance**: where we evaluate the models on the same family of trajectories that were used to train the models.
- **Generalization ability**: where we evaluate the models on unseen trajectories.
- **Robustness**: where we introduce extra disturbances to the environment and analyze the ability of the model to restore balance.

6.3 | Results

6.3.1 | Mean Target Reward

We train both MSAC and C-MSAC for 45,000 episodes each on the randomized ellipse trajectory and the S-curve trajectory tasks. After every 2,000 episodes of training, we run validation on 100 episodes to measure the target reward (r_{target}) on the objective phase. Figure 6 shows the mean and standard deviation of the normalized reward for MSAC and C-MSAC on the ellipse and S-curve trajectories.

For the ellipse trajectory, the C-MSAC model starts off slightly lower than the MSAC model in terms of target reward. This makes sense since the C-MSAC model is initially focusing on the tray lifting constraint objective. Eventually, the C-MSAC model

surpasses MSAC yielding a significantly higher target reward. Furthermore, C-MSAC is much more stable, exhibiting similar performance across different evaluation trials as compared to MSAC's performance that is characterized by large variance.

For the S-curve trajectory, both models have a similar target reward in the initial stages of training. However, in the later stages, the C-MSAC model surpasses the MSAC model yielding a policy very close to generating the maximum possible normalized reward (1.0).

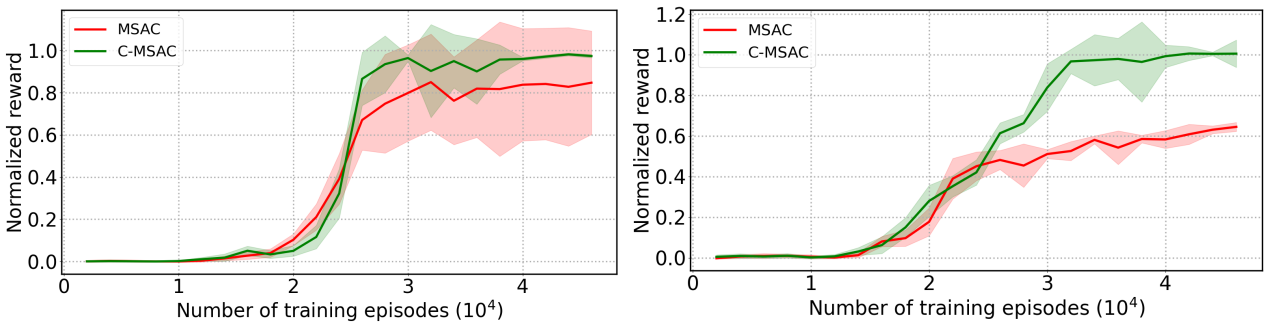


Figure 5 Normalized reward on test episodes for the target objective phase (phase 2). Left: ellipse trajectory. Right: S-curve trajectory.

6.3.2 | Phase Analysis

We also measured the target reward for the tray lifting constraint phase (phase 1) for both models on the 100 test episodes. Figure 6 shows the mean and standard deviation of the normalized constraint reward r_{lift} for both models on the ellipse and S-curve trajectories. The trend here is fairly similar to that for the objective phase reward. In the initial stages of training, the constraint phase reward is almost the same for the MSAC and the C-MSAC models. As the training progresses, the constraint phase reward for the C-MSAC model increases more rapidly and exhibits smaller variance compared to the MSAC model. Towards the middle of the training process, as the C-MSAC model starts exceeding the threshold value for the constraint phase, the corresponding reward starts to saturate.

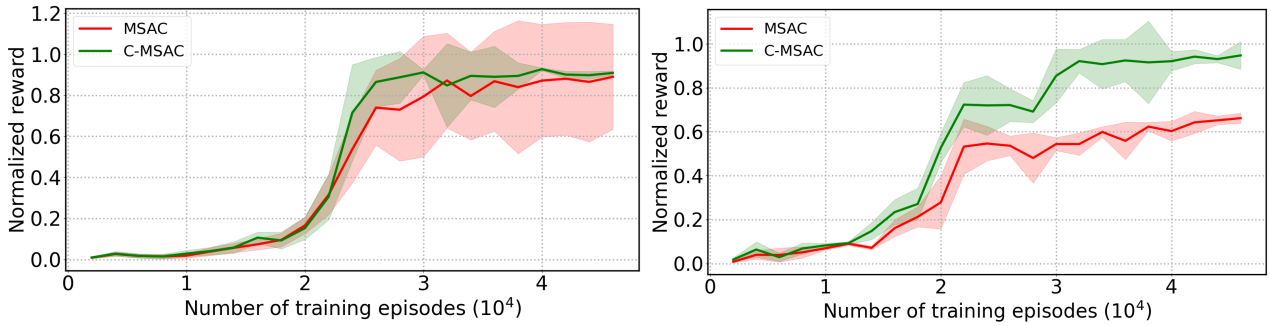


Figure 6 Normalized reward on test episodes for the constraint phase (phase 1). Left: ellipse trajectory. Right: S-curve trajectory.

6.3.3 | On-target Performance

In addition to the target reward, we also measured the on-target performance for both the MSAC and C-MSAC models for the ellipse and the S-curve trajectories. We define the ball to be on-target for a given step if the target reward (r_{target}) is higher than 0.95, and then measure the percentage of steps for which the ball stays on target for a given episode. To quantify the distribution

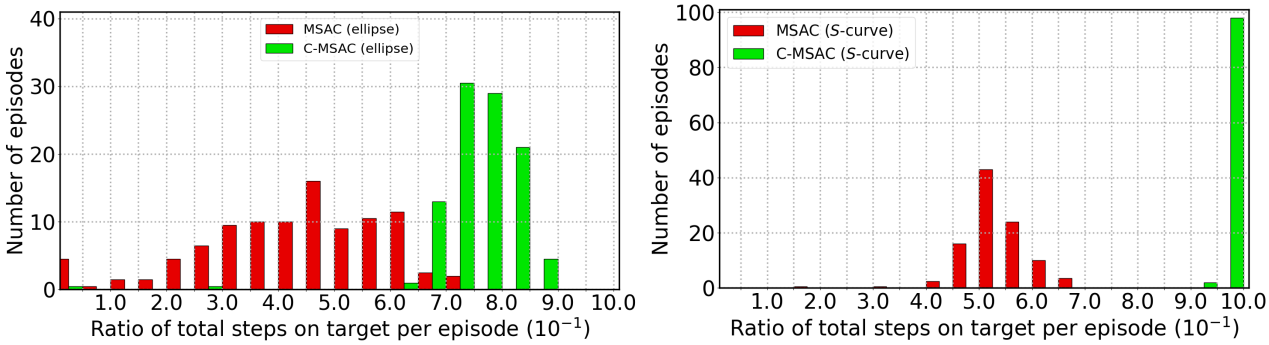


Figure 7 Histogram of on-target steps ratio. Left: ellipse trajectory. Right: S-curve trajectory.

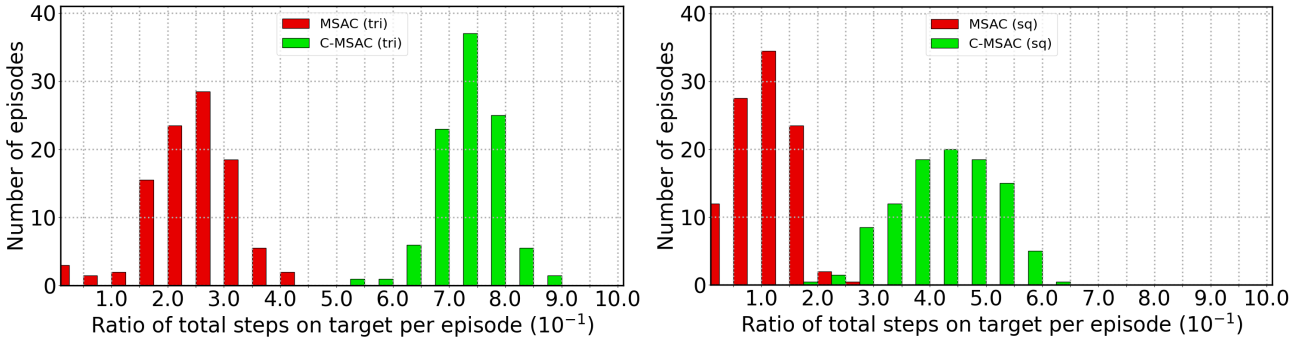


Figure 8 Histogram of on-target steps ratio. Left: triangle trajectory. Right: square trajectory.

of the on-target percentage, we draw a histogram of the number of episodes against the percentage of on-target steps for a given episode. Figure 7 shows the histograms for the ellipse and the S-curve trajectories, respectively.

It is clear from the figure that the average on-target time for the C-MSAC model is much higher than that for the MSAC model for both the ellipse and S-curve trajectories. This shows that the C-MSAC model is able to learn the fine-grained controls necessary to keep the ball on target most of the time and follow the curve smoothly. Our supplemental video also demonstrates this capability visually.

6.3.4 | Generalization Ability

After every 2,000 steps of training, we also evaluated the MSAC and C-MSAC models (trained on randomized ellipse trajectories) on two types of unseen trajectories: square and triangle. We observe that both MSAC and C-MSAC models are able to generalize to the unseen trajectories; however, the C-MSAC model yields a better average reward (Figures 9) and on-target performance (Figure 8) in comparison to the MSAC model. This shows that the C-MSAC model also exhibits better ability to generalize in comparison to the MSAC model.

One interesting finding here is that the performance on the square trajectory is slightly worse than that on the triangular trajectory for both models. Upon observation, we noted that this is likely due to the presence of an extra corner on the square compared to the triangle. Both models handle smooth curves and straight lines relatively well, but sometimes have difficulty handling sharp corners. The extra vertex on the square increases the chance of the ball moving far away from the target. We suspect that the difficulty in handling sharp corners might be because of the fact that our training data only includes smooth curves. It might be interesting to include trajectories with corners in the training data in future experiments.

6.3.5 | Robustness

We have studied the robustness of the two models against unexpected disturbances. For some early exploration in this direction, we hit the tray with additional balls at different angles and positions on the tray in order to disrupt the task. From our observations,

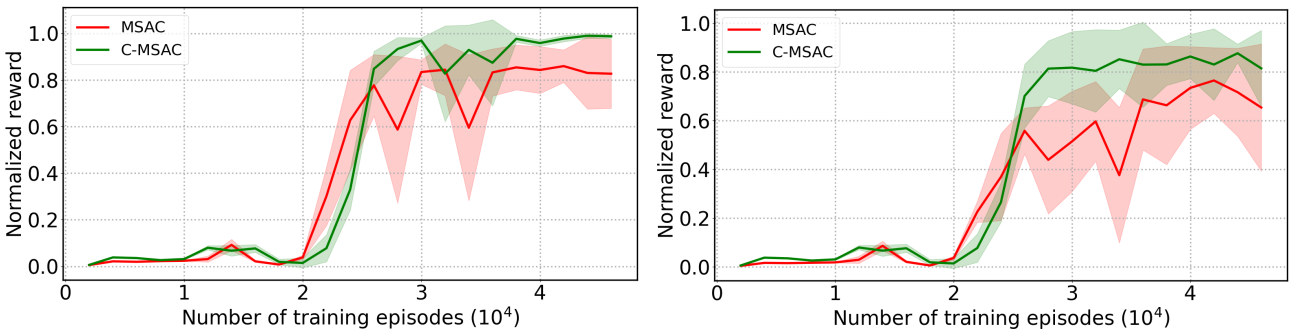


Figure 9 Normalized target reward on test episodes. Left: triangle trajectory. Right: square trajectory.

the C-MSAC model exhibited a superior capability to recover from such unexpected disturbances. The accompanying video to this paper shows the capability of C-MSAC to recover from such disruptions.

To formalize our observations, we set up an experiment to statistically analyze and compare the robustness of the C-MSAC and MSAC models. In this experiment, we introduce a disturbance force to the agent actions every 10 steps during the test episodes. We sample this force from a Gaussian distribution, scale it up with a magnitude factor, and add it to the agent actions. We evaluate both models on 100 test episodes and obtain the mean and standard deviation of the target reward for different values of the disturbance force magnitude.

Figure 10 shows the results for the MSAC and C-MSAC models on the ellipse and S-curve trajectories. As the force magnitude increases, the target reward for both models reduces and eventually approaches zero. However, the mean reward for the C-MSAC model is consistently higher than that for the MSAC model. The C-MSAC model reward also has a smaller standard deviation compared to the MSAC model, indicating higher stability and consistency across different test episodes.

We note here that we did not apply any disturbance to the environment during training time. We hypothesize that the robustness for these models arises from the entropy term used in the SAC algorithm which encourages exploration.

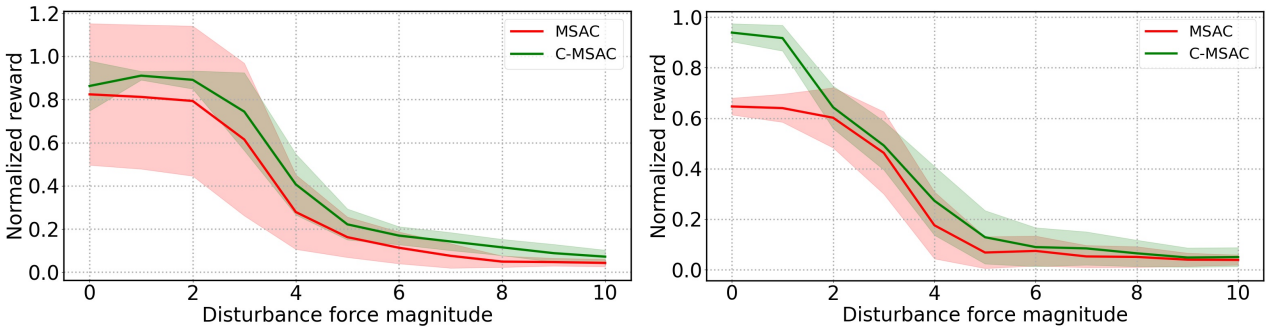


Figure 10 Normalized target reward for different disturbance force magnitudes. Left: ellipse trajectory. Right: S-curve trajectory.

7 | CONCLUSIONS

In this paper, we propose a constraint-based multi-agent reinforcement learning approach to solve multi-phase collaborative tasks. We present a framework to simulate tray balancing and target following tasks with different trajectories. We evaluate the proposed constraint-based (C-MSAC) model on this task and compare it against a baseline that does not employ constraints (MSAC). Our results show that the proposed model is able to exhibit better on-target performance, better generalization ability, and improved robustness in comparison to the baseline.

Two main limitations can be observed in our present work. While the proposed constraint-based framework is very general and applicable to complex multi-phase tasks, in this work we have only explored a task of two phases: a tray lifting constraint

phase and a tray balancing target phase. Additionally, the arm movements in our environment are not controlled by an RL policy but rather calculated by Inverse Kinematics (IK) based on the anchor points transformation. This sometimes results in some glitchy movements, for example when the agents suddenly move the tray to avoid the ball from going out of bounds, resulting in a sudden pose change due to the arm having to follow the anchor points.

As future work we plan to extend this work by using physically simulated humanoids with joint-level RL-based control for the arms instead of using IK. This will also provide us the opportunity to fully leverage the power of the constraint-based framework by introducing additional phases to the task such as an initial constraint for the arms to reach the tray anchor points. In addition, while currently we focus on collaborative tasks, our framework can also be generalized to competitive tasks, opening interesting avenues for future work.

References

1. Lowe R, Wu YI, Tamar A, Harb J, Pieter Abbeel O, Mordatch I. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems* 2017; 30.

2. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* 2017.

3. Haarnoja T, Zhou A, Abbeel P, Levine S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: PMLR. ; 2018: 1861–1870.

4. Brockman G, Cheung V, Pettersson L, et al. Openai gym. *arXiv preprint arXiv:1606.01540* 2016.

5. OpenAI . Robogym. https://github.com/openai/robogym; 2020.

6. Juliani A, Berges VP, Teng E, et al. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627* 2018.

7. Bellemare MG, Naddaf Y, Veness J, Bowling M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 2013; 47: 253–279.

8. Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning. *nature* 2015; 518(7540): 529–533.

9. Andrychowicz OM, Baker B, Chociej M, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research* 2020; 39(1): 3–20.

10. Peng XB, Abbeel P, Levine S, Panne v. dM. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)* 2018; 37(4): 1–14.

11. Won J, Gopinath D, Hodgins J. Control strategies for physically simulated characters performing two-player competitive sports. *ACM Transactions on Graphics (TOG)* 2021; 40(4): 1–11.

12. Iqbal S, Sha F. Actor-attention-critic for multi-agent reinforcement learning. In: PMLR. ; 2019: 2961–2970.

13. Yu C, Velu A, Vinitsky E, et al. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in Neural Information Processing Systems* 2022; 35: 24611–24624.

14. Lillicrap TP, Hunt JJ, Pritzel A, et al. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* 2015.

15. Nachum O, Ahn M, Ponte H, Gu S, Kumar V. Multi-agent manipulation via locomotion using hierarchical sim2real. *arXiv preprint arXiv:1908.05224* 2019.

16. Florensa C, Duan Y, Abbeel P. Stochastic neural networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1704.03012* 2017.

17. Xu T, Liang Y, Lan G. Crpo: A new approach for safe reinforcement learning with convergence guarantee. In: PMLR. ; 2021: 11480–11491.

18. Kurach K, Raichuk A, Stańczyk P, et al. Google research football: A novel reinforcement learning environment. *arXiv preprint arXiv:1907.11180* 2019.

19. Baker B, Kanitscheider I, Markov T, et al. Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528* 2019.

20. Tampuu A, Matiisen T, Kodelja D, et al. Multiagent cooperation and competition with deep reinforcement learning. *PloS one* 2017; 12(4): e0172395.

21. Foerster J, Farquhar G, Afouras T, Nardelli N, Whiteson S. Counterfactual multi-agent policy gradients. In: . 32. ; 2018.

22. Yang Y, Luo R, Li M, Zhou M, Zhang W, Wang J. Mean field multi-agent reinforcement learning. In: PMLR. ; 2018: 5571–5580.

23. Lu H, Gu C, Luo F, Ding W, Zheng S, Shen Y. Optimization of task offloading strategy for mobile edge computing based on multi-agent deep reinforcement learning. *IEEE Access* 2020; 8: 202573–202584.

24. Rashid T, Samvelyan M, Schroeder C, Farquhar G, Foerster J, Whiteson S. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In: PMLR. ; 2018: 4295–4304.

25. Cheng Z, Liwang M, Chen N, Huang L, Du X, Guizani M. Deep reinforcement learning-based joint task and energy offloading in UAV-aided 6G intelligent edge networks. *Computer Communications* 2022; 192: 234–244.

26. Wu X, Li X, Li J, Ching P, Leung VC, Poor HV. Caching transient content for IoT sensing: Multi-agent soft actor-critic. *IEEE Transactions on Communications* 2021; 69(9): 5886–5901.

27. Chen Z, Liu B. Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 2018; 12(3): 1–207.

28. Hundt A, Killeen B, Greene N, et al. "Good robot!": Efficient reinforcement learning for multi-step visual tasks with SIM to real transfer. *IEEE Robotics and Automation Letters* 2020; 5(4): 6724–6731.

29. Yang HY, Wong SK. Agent-based cooperative animation for box-manipulation using reinforcement learning. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 2019; 2(1): 1–18.

30. Nachum O, Gu SS, Lee H, Levine S. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems* 2018; 31.

31. Peng XB, Berseth G, Yin K, Van De Panne M. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)* 2017; 36(4): 1–13.

32. Garcıa J, Fernández F. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research* 2015; 16(1): 1437–1480.

33. Achiam J, Held D, Tamar A, Abbeel P. Constrained policy optimization. In: PMLR. ; 2017: 22–31.

34. Kucuk S, Bingul Z. *Robot kinematics: Forward and inverse kinematics*. INTECH Open Access Publisher London, UK . 2006.

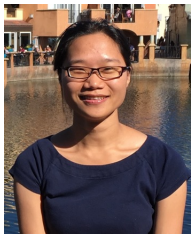
35. Juliani A, Berges VP, Teng E, et al. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627* 2020.

36. Osborne MJ, Rubinstein A. *A course in game theory*. MIT press . 1994.

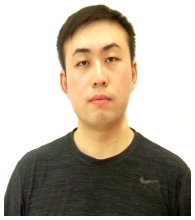
37. Vrancx P, Verbeeck K, Nowé A. Decentralized learning in markov games. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 2008; 38(4): 976–981.

38. Babadi A, Naderi K, Hämäläinen P. Self-imitation learning of locomotion movements through termination curriculum. In: 2019 (pp. 1–7).

AUTHOR BIOGRAPHY



Xiumin Shang is currently a Ph.D. candidate in Electrical Engineering and Computer Science at the University of California Merced. She received her M.Eng. in Mechanical Engineering and B.Eng. in Computer Engineering from Harbin Engineering University in China. Her research interests include topics of multi-agent learning, reinforcement learning, computer animation, virtual reality, and their applications in various fields such as car racing games, smart grid optimization, and currently focusing on employing machine learning techniques in agent animation that can be used in virtual reality environments.



Tengyu Xu is currently a Research Scientist at Meta Platform Inc. He received his Ph.D. degree from the Department of Electrical and Computer Engineering, The Ohio State University at August 2022. Dr. Xu's research interests are centered around reinforcement learning and optimization. His research works have been published at conferences such as NeurIPS, ICML and ICLR. Particularly, his fundamental theoretical works in Q-learning and actor-critic-type algorithms have been widely followed by researchers in the reinforcement learning theory community.

Ioannis Karamouzas is an Associate Professor in the School of Computing at Clemson University. Prior to joining Clemson, he was a postdoctoral associate at the University of Minnesota and received his Ph.D. from Utrecht University. His research focuses on developing motion planning algorithms for autonomous agents in both physical and virtual worlds. His work has been integrated into commercial applications including computer games and pedestrian simulation suites. He is a recipient of an NSF CAREER award and is currently serving on the editorial board of IEEE Robotics and Automation Letters.

Marcelo Kallmann is currently a Principal Scientist at Amazon Robotics. Previously he was an Amazon Scholar and a Professor, Founding Faculty, and EECS graduate chair at the University of California, Merced. Before UC Merced he was research faculty at the University of Southern California (USC) and a research scientist at the USC Institute for Creative Technologies (ICT). He received his Ph.D. degree from the École Polytechnique Fédérale de Lausanne (EPFL), Switzerland. He has over a hundred technical publications in computer graphics, computer animation and robotics. His academic research has received funding from NSF, ARO, and CITRIS, and his work on triangulations for path planning runs inside The Sims 4, often cited as the best selling PC game of 2014 and 2015.