arXiv:1909.08776v2 [cs.RO] 3 Mar 2020

Learning Multi-Robot Decentralized Macro-Action-Based Policies via a Centralized Q-Net

Yuchen Xiao, Joshua Hoffman, Tian Xia and Christopher Amato

***Abstract*— In many real-world multi-robot tasks, high-quality solutions often require a team of robots to perform asynchronous actions under decentralized control. Decentralized multi-agent reinforcement learning methods have difficulty learning decentralized policies because of the environment appearing to be non-stationary due to other agents also learning at the same time. In this paper, we address this challenge by proposing a macro-action-based decentralized multi-agent double deep recurrent Q-net (MacDec-MADDRQN) which trains each decentralized Q-net using a centralized Q-net for action selection. A generalized version of MacDec-MADDRQN with two separate training environments, called Parallel-MacDec-MADDRQN, is also presented to leverage either centralized or decentralized exploration. The advantages and the practical nature of our methods are demonstrated by achieving near-centralized results in simulation and having real robots accomplish a warehouse tool delivery task in an efficient way.**

I. INTRODUCTION

Multi-robot systems have become ubiquitous in our daily lives, such as drones for applications such as agricultural inspection, warehouse robots, and self-driving cars [1]–[3]. For example, consider a warehouse environment (Fig. 1a), where a Fetch robot [4] and two Turtlebots [5] are autonomously delivering tools in order to assist two humans with their assembly tasks. To be more efficient, the robots should be able to predict which tool the human workers will potentially need rather than always waiting for a human's request, while collaborating with the other robots to find the tool in advance and passing it to one of the Turtlebots (Fig. 1b) for delivery (Fig. 1c). Performing these high-quality coordination behaviors in large, stochastic and uncertain environments is challenging for the robots, because it requires the robots to operate asynchronously according to local information while reasoning about cooperation between teammates.

Although, several multi-agent deep reinforcement learning approaches have been proposed and have achieved high-quality performance [6]–[10], these methods assume synchronized primitive actions. Our very recent work [11] bridged this gap by proposing the first asynchronous macro-action-based multi-agent deep reinforcement learning frameworks. Macro-actions naturally represent temporally extended robot controllers that can be executed in an asynchronous manner [12], [13]. In that paper, we proposed approaches for both learning decentralized macro-action-value functions and centralized joint-macro-action-value functions. However, the decentralized method, using Decentralized Hysteretic DRQN with Double DQN (Dec-HDDRQN), performed poorly in large and complex domains. Nevertheless, decentralized execution is necessary for cases when there is limited or no communication between robots.

Khoury College of Computer Sciences, Northeastern University, Boston, MA 02115, USA {xiao.yuch, hoffman.j, xia.tia}@husky.neu.edu, c.amato@northeastern.edu

(a) (b) (c)

Fig. 1: Warehouse tool delivery task: (a) Three robots deliver tools to two humans; (b) Collaborative tool passing; (c) Correct tool delivered.

In this paper, we improve the learning of decentralized policies via two contributions: (a) A new macro-action-based decentralized multi-agent deep double-Q learning approach, called MacDec-MADDRQN, which adopts *centralized training with decentralized execution* by allowing each individual decentralized Q-net update to use a centralized Q-net; (b) MacDec-MADDRQN introduces a choice of ϵ-greedy exploration, either based on the centralized Q-net or the decentralized Q-nets. The best choice is often not clear without knowledge of domain properties. Therefore, a more general version, called Parallel-MacDec-MADDRQN, is proposed, in which, the centralized Q-net is trained purely based on the experiences generated by using centralized ϵ-greedy exploration in one environment, simultaneously, agents perform decentralized exploration in a separate environment, and each decentralized Q-net is then optimized using the decentralized data and the centralized Q-net.

We evaluate our methods in both simulation and in hardware. In simulation, our methods outperform the previous decentralized method by either converging to a much higher value or learning faster in both a benchmark domain and a Warehouse Tool Delivery domain with a single human involved. We also deploy the decentralized policies learned in simulation on real robots which shows high-quality cooperation to deliver the correct tools in an efficient way. To our knowledge, this is the first instance of running a set of decentralized macro-action-based policies that were trained via deep reinforcement learning on a team of real robots.

II. BACKGROUND

We first discuss macro-action-based Dec-POMDPs [12], [13] and deep Q-learning, and then provide an overview of our previous related approach [11].

A. MacDec-POMDPs

Decentralized fully cooperative multi-agent decision-making under uncertainty can be modeled as a decentralized POMDP (Dec-POMDP) [14]. Due to the assumption of synchronous actions that require the same amount of time for each agent, Dec-POMDPs are not applicable to multi-robot planning and learning scenarios in real-world. MacDec-POMDPs, formalized by introducing macro-actions into Dec-POMDPs, inherently allow asynchronous execution among robots with temporally extended macro-actions that can begin and end at different times for each agent.

Formally, a MacDec-POMDP is defined as a tuple $\langle I, S, A, \Omega, M, \zeta, O, T, Z, R \rangle$, where I is a finite set of agents; S is a finite set of environment states; $A \equiv \times_i A_i$ and $\Omega \equiv \times_i \Omega_i$ are the spaces of joint-primitive-action and joint-primitive-observation respectively; $M \equiv \times_i M_i$ is the joint set of each agent's finite macro-action space M_i; $\zeta \equiv \times_i \zeta_i$ is the set of joint macro-observations over agents' finite macro-observation space ζ_i. Given a macro-action-based policy, each agent i is allowed to asynchronously choose a macro-action $m_i = \langle \beta^m, I^m, \pi^m \rangle_i$ that depends on individual macro-action-observation histories, where $\beta^m : H_i^A \rightarrow [0, 1]$ is the stochastic termination condition and $I^m \subset H_i^M$ is the initiation set of the corresponding macro-action m_i, respectively depending on the primitive-action-observation history space H_i^A and macro-action-observation history space H_i^M of agent i; $\pi^m : H_i^A \rightarrow A_i$ denotes the low-level policy to achieve the macro-action m, and during the execution, each agent's primitive-observation $o_i \in \Omega_i$ is generated according to probability observation function $O_i(o_i, a_i, s) = \Pr(o_i \mid a_i, s)$, and a shared immediate reward $r(s, \vec{a})$, where $\vec{a} \in A \equiv \times_i A_i$, is issued according to the reward function $R : S \times A \rightarrow \mathbb{R}$. Importantly, considering the stochastic terminations and the asynchronous executions of macro-actions over agents, the transition function is defined as $T(s', \vec{\tau}, s, \vec{m}) = \Pr(s', \vec{\tau} \mid s, \vec{m})$, where $\vec{\tau}$ is the time-step at which *any agent* i completes its macro-action m_i, and also indicates the termination of the joint-macro-action $\vec{m}$; Successively, a new joint-macro-observation $\vec{z} \in \zeta \equiv \times_i \zeta_i$ is generated based on the macro-observation function $Z(\vec{z}, \vec{m}, s') = \Pr(\vec{z} \mid \vec{m}, s')$; Note that, each agent keeps updating its primitive observation every time-step, but only updates macro-observation when its current macro-action has terminated. The objective is to optimize the joint high-level policy $\Psi = \times_i \Psi_i$ such that the following expected discounted return from an initial state s_0 is maximized:

$$\Psi^* = \arg\max_{\Psi} \mathbb{E}\left[\sum_{t=0}^{h-1} \gamma^t r(s_{(t)}, \vec{a}_{(t)}) \mid s_{(0)}, \pi, \Psi\right] \quad (1)$$

B. Deep Recurrent Q-Network and Double DQN

Deep Q-learning is a state-of-the-art approach using a deep Q-network (DQN) parameterized by θ as an action-value approximator which is iteratively updated to minimize the loss: $\mathcal{L}(\theta) = \mathbb{E}_{<s,a,r,s'> \sim \mathcal{D}}\Big[\big(y - Q_\theta(s, a)\big)^2\Big]$, where $y = r + \gamma \arg\max_{a'} Q_{\theta^-}(s', a')$. Experience replay and a less frequently updated target Q-network, parameterized by θ^-, are employed for improving performance and stabilizing learning [15]. DQN with a recurrent layer (DRQN) has been widely adopted in partial observable domains to allow agent's actions to depend on abstractions of action-observation histories rather than states (or a single observation) [16]. Double DQN incorporates double Q-learning [17] into DQN to provide an unbiased target action-value estimation, $y = r + \gamma Q_{\theta^-}(s', \arg\max_{a'} Q_\theta(s', a'))$ [18], which leverages the above two Q-networks. In this paper, we mainly compare our decentralized learning approaches with Decentralized Hysteretic DRQN (which uses two learning rates for more robust updating against negative TD error) with Double DQN (Dec-HDDRQN) [6], and centralized learning via Double DRQN (Cen-DDRQN).

C. Learning Macro-Action-Based Deep Q-Nets

Although there has been several popular multi-agent deep reinforcement learning methods achieving impressive performance in cooperative as well as competitive domains [6]–[10], they all require primitive actions and synchronous action execution. There was no principled way to utilize these methods to learn macro-action-based policies, in which the challenges were how to properly update macro-action values and correctly maintain macro-action-observation trajectories.

To cope with the above challenges, in our previous work [11], we first proposed a decentralized macro-action-based learning method that is based on Dec-HDDRQN with a new buffer called Macro-Action Concurrent Experience Reply Trajectories (Mac-CERTs). This buffer contains the macro-action-observation experience represented as a tuple $\langle z_i, m_i, z_i', r_i^c \rangle$ for each agent i, where $r_i^c = \sum_{t=t_{m_i}}^{\tau} r_t$ is an accumulated reward for the macro-action m_i from its beginning time-step t_{m_i} to the termination step τ. In the training phase, each agent individually updates its own macro-action-value function $Q_{\theta_i}(h_i, m_i)$, using a concurrent mini-batch of sequential experiences sampled from Mac-CERTs, by minimizing the loss: $\mathcal{L}(\theta_i) = \mathbb{E}_{<z_i, m_i, z_i', r_i^c> \sim \mathcal{D}}\Big[\big(y_i - Q_{\theta_i}(h_i, m_i)\big)^2\Big]$, where $y_i = r_i^c + \gamma Q_{\theta_i^-}\big(h_i', \arg\max_{m_i'} Q_{\theta_i}(h_i', m_i')\big)$ and h_i denotes the macro-action-observation history of agent i. For cases when a centralized macro-action-based policy is possible, we also proposed a novel centralized replay buffer called Macro-Action Joint Experience Replay Trajectories (Mac-JERTs) [11]. At each execution step, this buffer collects a joint macro-action-observation experience represented as a tuple $\langle \vec{z}, \vec{m}, \vec{z}', \vec{r}^c \rangle$, where $\vec{r}^c = \sum_{t=t_{\vec{m}}}^{\vec{\tau}} r_t$ is a shared joint accumulated reward for the agents' joint macro-action $\vec{m}$ from its beginning time-step $t_{\vec{m}}$ to the ending time-step $\vec{\tau}$ when *any agent* terminates its macro-action. The centralized macro-action-value function $Q_\phi(\vec{h}, \vec{m})$ is then optimized by minimizing the loss: $\mathcal{L}(\phi) = \mathbb{E}_{<\vec{z}, \vec{m}, \vec{z}', \vec{r}^c> \sim \mathcal{D}}\Big[\big(y - Q_\phi(\vec{h}, \vec{m})\big)^2\Big]$, where $y = \vec{r}^c + \gamma Q_{\phi^-}\big(\vec{h}', \arg\max_{\vec{m}'} Q_\phi(\vec{h}', \vec{m}' \mid \vec{m}^{\text{undone}})\big)$. Here, $\vec{m}^{\text{undone}}$ is the joint-macro-action over the agents who have not completed their macro-actions in the sampled experience.

Note that, this conditional operation considers the agents' asynchronous macro-action execution status which is accessible from Mac-JERTs during training.

Building on our previous work, in this paper, we extend Double DQN to decentralized multi-agent macro-action-based policy learning under partial observability in the manner of *centralized training with decentralized execution*. In this new method, each agent is able to update its own Q-net by taking into account the effects of other agents' behaviors in the environment, naturally surmounting the non-stationary environment issue from each agent's perspective.

III. APPROACH

In multi-agent environments, decentralized learning causes non-stationarity from each agent's perspective as other agents policies change during learning. Learning a centralized joint-action-value function to guide each agent's decentralized policy updating has been being a very popular training manner to conquer the non-stationarity [7], [8]. VDN and QMIX also use centralized training by first training a centralized, but factored, Q-net that is decomposed into a decentralized Q-net for each agent for use in execution [9], [10]. In this section, we propose a new multi-agent Double DQN-based approach, called MacDec-MADDRQN, to learn decentralized macro-action-value functions that are trained with a centralized joint macro-action-value function.

A. Macro-Action-Based Decentralized Multi-Agent Double Deep Recurrent Q-Net (MacDec-MADDRQN)

Double DQN has been implemented in multi-agent domains for learning either centralized or decentralized policies [11], [19], [20]. However, in the decentralized learning case, each agent independently adopts double Q-learning purely based on its own local information. Learning only from local information often impedes agents from achieving high-quality cooperation.

In order to take advantage of centralized information for learning decentralized Q-networks, we train the centralized joint macro-action-value function Q_ϕ and each agent's decentralized macro-action-value function Q_{θ_i} simultaneously, and the target value for updating decentralized macro-action-value function Q_{θ_i} is then calculated by using the centralized Q_ϕ for macro-action selection and the decentralized target-net $Q_{\theta_i^-}$ for value estimation.

More concretely, consider a domain with N agents, and both the centralized Q-network Q_ϕ and decentralized Q-networks Q_{θ_i} for each agent i are represented as DRQNs [16]. The experience replay buffer $\boldsymbol{\mathcal{D}}$, a merged version of Mac-CERTs and Mac-JERTs, contains the tuples $\langle \mathbf{z}, \mathbf{m}, \mathbf{z}', \mathbf{r^c}, \vec{r}^c \rangle$, where $\mathbf{z} = \{z_0, ..., z_N\}$, $\mathbf{m} = \{m_0, ..., m_N\}$ and $\mathbf{r^c} = \{r_0^c, ..., r_N^c\}$. In each training iteration, agents sample a mini-batch of sequential experiences to first optimize the centralized joint macro-action-value function Q_ϕ in the way mentioned in Section II-C, and then update each decentralized macro-action-value function by minimizing the squared TD error:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{<\mathbf{z},\mathbf{m},\mathbf{z}',\mathbf{r^c},\vec{r}^c>\sim\boldsymbol{\mathcal{D}}}\Big[\big(y_i - Q_{\theta_i}(h_i, m_i)\big)^2\Big] \quad (2)$$

where,

$$y_i = r_i^c + \gamma Q_{\theta_i^-}\Big[h_i', \big[\arg\max_{\mathbf{m}'} Q_\phi(\mathbf{h}', \mathbf{m}')\big]_i\Big] \quad (3)$$

In Eq. 3, $\big[\arg\max_{\mathbf{m}'} Q_\phi(\mathbf{h}', \mathbf{m}')\big]_i$ implies selecting the joint macro-action with the highest value and then selecting the individual macro-action for agent i. In this updating rule, not only are double estimators $Q_{\theta_i^-}$ and Q_ϕ applied to counteract overestimation on target Q-values, but also a centralized heuristic on action selection is embedded. Now, from each agent's perspective, the target Q-value is calculated by assuming all agents will behave based on the centralized Q-net next step (Eq. 3), in which the provided global information by the centralized Q-net will help each agent to avoid getting trapped in local optima and also facilitates them to learn cooperation behaviors.

Additionally, similar to the idea of the conditional operation for training a centralized joint macro-action-value function discussed in Section II-C, in order to obtain a more accurate prediction by taking each agent's macro-action executing status into account, Eq. 3 can be rewritten as:

$$y_i = r_i^c + \gamma Q_{\theta_i^-}\Big[h_i', \big[\arg\max_{\mathbf{m}'} Q_\phi(\mathbf{h}', \mathbf{m}' \mid \mathbf{m^{undone}})\big]_i\Big] \quad (4)$$

B. ϵ-greedy Exploration Policy Selection

Exploration is also a difficult problem in multi-agent reinforcement learning. ϵ-greedy exploration has been widely used in many methods such as Q-learning to generate training data [21]. In DQN-based methods, as a hyper-parameter, ϵ often acts with a linear decay along with the training steps from 1.0 to a lower value to achieve the trade-off between exploration and exploitation. And, exploration can be done based on either the centralized or decentralized policies. Centralized exploration may help to choose cooperative actions more often that would have a low probability of being selected from decentralized policies, and decentralized exploration may provide more realistic data that is actually achievable by decentralized policies.

Therefore, in our approach, besides tuning ϵ, we introduce a hyper-selection for performing a ϵ-greedy behavior policy that can perform either centralized exploration based on Q_ϕ or decentralized exploration using each agent's Q_{θ_i}.

However, without having enough knowledge about the properties of a given domain in the very beginning, it is not clear which exploration choice is the best. To cope with this, we propose a more generalized version of MacDec-DDRQN, called *Parallel-MacDec-MADDRQN*, summarized in Algorithm 1. The core idea is to have two parallel environments with agents respectively performing centralized exploration (cen-ϵ-greedy) and decentralized exploration (dec-ϵ-greedy) in each. The centralized Q_ϕ is first trained purely using the centralized experiences, while each agent's decentralized Q_{θ_i} is then optimized using Eq. 4 with only decentralized experiences. The performance of this algorithm in the Warehouse domain is presented in Section IV-B

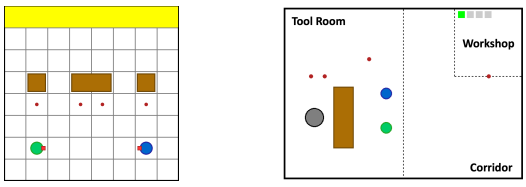
Algorithm 1 Parallel-MacDec-MADDRQN

Initialize centralized Q-Networks: Q_ϕ, Q_ϕ^-
Initialize decentralized Q-Networks for each agent i: Q_{θ_i}, $Q_{\theta_i}^-$
Initialize two parallel environments *cen-env*, *dec-env*
Initialize two step counters $t_{\text{cen-env}}$, $t_{\text{dec-env}}$
Initialize centralized buffer $\mathcal{D}_{\text{cen}} \leftarrow$ Mac-JERTs
Initialize decentralized buffer $\mathcal{D}_{\text{dec}} \leftarrow$ Mac-CERTs
Get initial joint-macro-observation $\vec{z}$ for agents in *cen-env*
Get initial macro-observation z_i for each agent i in *dec-env*
for *dec-env-episode* = 1 to M **do**
  Agents take joint-macro-action with cen-ϵ-greedy using Q_ϕ
  Store $\langle \vec{z}, \vec{m}, \vec{z}', \vec{r}^{\,c} \rangle$ in $\mathcal{D}_{\text{cen}}$
  $t_{\text{cen-env}} \leftarrow t_{\text{cen-env}} + 1$
  Each agent i takes macro-action with dec-ϵ-greedy using Q_{θ_i}
  Store $\langle z_i, m_i, z_i', r_i^c \rangle$ in $\mathcal{D}_{\text{dec}}$
  $t_{\text{dec-env}} \leftarrow t_{\text{dec-env}} + 1$
  if $t_{\text{dec-env}}$ mod $I_{\text{train}} == 0$ **then**
    Sample a mini-batch $\mathcal{B}_{\text{cen}}$ of sequential experiences $\langle \vec{z}, \vec{m}, \vec{z}', \vec{r}^{\,c} \rangle$ from $\mathcal{D}_{\text{cen}}$
    Perform a gradient decent step on $\big(y - Q_\phi(\vec{h}, \vec{m})\big)^2_{\mathcal{B}_{\text{cen}}}$, where $y = \vec{r}^{\,c} + \gamma Q_{\phi^-}\big(\vec{h}\,', \arg\max_{\vec{m}'} Q_\phi(\vec{h}\,', \vec{m}\,' \mid \vec{m}^{\text{undone}})\big)$.
    Sample a mini-batch $\mathcal{B}_{\text{dec}}$ of sequential experiences $\langle z_i, m_i, z_i', r_i^c \rangle$ from $\mathcal{D}_{\text{dec}}$ for each agent i
    Perform a gradient decent step on $\big(y_i - Q_{\theta_i}(h_i, m_i)\big)^2_{\mathcal{B}_{\text{dec}}}$, where $y_i = r_i^c + \gamma Q_{\theta_i^-}\Big[h_i', \big[\arg\max_{\mathbf{m}'} Q_\phi(\mathbf{h}', \mathbf{m}' \mid \mathbf{m}^{\mathbf{undone}})\big]_i\Big]$
  end if
  if $t_{\text{dec-env}}$ mod $I_{\text{TargetUpdate}} = 0$ **then**
    Update centralized target network $\phi^- \leftarrow \phi$
    Update each agent's decentralized target network $\theta_i^- \leftarrow \theta_i$
  end if
  if $t_{\text{cen-env}} =$ max-episode-length **or** terminal state **then**
    Reset *cen-env*
    Get initial joint-macro-observation $\vec{z}$ for agents in *cen-env*
  end if
  if $t_{\text{dec-env}} =$ max-episode-length **or** terminal state **then**
    Reset *dec-env*
    Get initial macro-observation z_i for each agent i in *dec-env*
  end if
end for

IV. SIMULATION EXPERIMENTS

In this section, we describe two macro-action-based multi-robot domains designed in our previous work [11], the Box Pushing (BP) domain and the Warehouse Tool Delivery (WTD) domain. We evaluate our approaches in these two domains while comparing with macro-action-based Dec-HDDRQN, fully centralized training via DDRQN (Cen-DDRQN), and some ablations we consider.

A. Domain Setup



(a) Box Pushing (b) Warehouse Tool Delivery

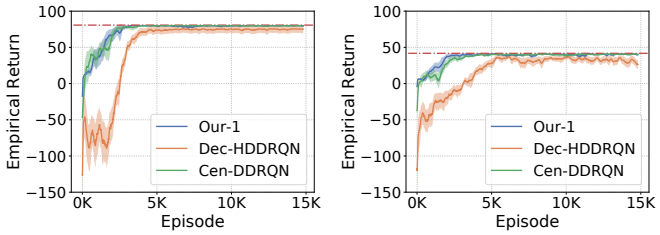
Fig. 2: Experimental environments in simulation

1) Box Pushing: (Fig. 2a). This domain has two mobile robots with the goal of cooperatively pushing a big box (middle brown square), which is only movable when two robots push it together, to the goal area (yellow bar at the top). The difficulties come from partial observability (each robot is only allowed to observe one cell in front) and two small boxes which attract the robots to learn the sub-optimal policy that is pushing one small box on its own. We provide two categories of macro-actions for each robot: (a) One step macro-actions, ***Turn-left, Turn-right*** and ***Stay***; (b) Long-term macro-actions, ***Move-to-small-box(i)*** and ***Move-to-big-box(i)***, navigate the robot to one of the red waypoints below the corresponding box and ends with the robot facing it; ***Push*** commands the robot to keep moving forward until arriving at the boundary of the grid world, touching the big box, or pushing a small box to the goal area. The macro-observation space for each robot consists of four different possible values associated with the cell in front of the robot: *empty*, *teammate*, *boundary*, *small box* and *big box*. Robots obtain $+100$ or $+10$ rewards respectively for pushing the big box or a small box to the goal area, and a -5 penalty is assigned to the team when any robot pushes the big box alone or hits the boundaries. Robots also get -0.1 reward per time-step. Note that each episode terminates either by reaching the horizon limitation or when any box pushed to the goal area.

2) Warehouse Tool Delivery: (Fig. 2b). In order to test if our approach is applicable to address real-world industrial problems, we developed a tool delivery task for a warehouse environment (5×7 continuous space), in which, one human works on an assembly task (4 steps in total and each step takes 18 units time) in the workshop. The human always starts from step one and needs a particular tool for each future step to continue. The objective of the three robots is to assist the human to finish the assembly task as soon as possible by collaboratively searching for the right tools in the proper order on the brown table and then passing them to one of the mobile robots (the green or blue) to accomplish the delivery in time. To make this problem more challenging, the info about the correct tool that the human needs for future step is not known to the robots, so it has to be learned during training. Also, the human is only allowed to possess one tool at a time from the mobile robots.

Each mobile robot has three macro-actions: ***Go-to-WS*** navigates the robot to the red waypoint at the workshop; ***Go-to-TR*** drives the robot to the upper right waypoint in the tool room; the duration of these two macro-actions depends on the robot's moving speed (0.6 in our case); ***Get-Tool*** navigates the robot to the pre-assigned waypoint beside the table and waits there until either obtaining one tool from the gray robot or 10 time-steps have passed. Also, there are four applicable macro-actions for the gray robot: ***Wait-M*** lasts 1 time-step; ***Search-Tool(i)*** takes 6 time-steps to find the tool i and place it in the staging area (lower left on the table where can hold at most two tools). Running this action when the staging area is fully occupied leads the robot to pause for 6 time-steps. ***Pass-to-M(i)*** lasts 4 time-steps to pass one of the tools from the staging area, in first-in-first-out order, to mobile robot i.

We allow each mobile robot to capture four different features in a macro-observation, including *location*, *the human's*



(a) Grid world 10×10 (b) Grid world 30×30

Fig. 3: Comparison of the average performance via three different learning approaches in BP domain.

current step (only accessible when in the workshop), *the tools being carried* by that robot, and *the number of tools* in the staging area (only observable when in the tool room). While, the gray robot can monitor *which mobile robot* is beside the table and *the number of tools* in the staging area.

The global rewards provide -1 each time-step to encourage the robots to deliver the object(s) in a timely manner without causing the human to pause; a penalty of -10 is given when the gray robot executes ***Pass-to-M(i)*** but no mobile robots are beside the table; a bonus of $+100$ is awarded to the entire team when the robots successfully deliver a correct tool to the human.

B. Results in the Box Pushing Domain

We first evaluate our method *MacDec-MADDRQN* (Our-1) with centralized ϵ-greedy exploration in Box Pushing domain, and compare its performance with Dec-HDDRQN and Cen-DDRQN. In all three methods, the decentralized Q-net consists of two MLP layers, one LSTM layer [22] and another two MLP layers, in which there are 32 neurons on each layer with Leaky-Relu as the activation function for MLP layers. The centralized Q-net has the same architecture but 64 neurons in the LSTM layer. The performance for two sizes of the domain is shown in Fig. 3, which is the mean of the episodic discounted returns ($\gamma = 0.98$) over 40 runs with standard error and smoothed by 20 neighbors. The optimal returns are shown as red dash-dot lines.

In both scenarios, the advantages of having the centralized Q_ϕ in the double-Q updating (Eq. 4) is seen by it achieving similar performance to Cen-DDRQN and converging to the optimal returns earlier than Dec-HDDRQN. Furthermore, in the bigger world space (Fig. 3b), our method even leads to slightly faster learning than the fully centralized approach. This is because centralized Q-learning deals with the joint macro-observation and joint macro-action space, which is much bigger than the decentralized spaces from each agent's perspective. Our method has the key benefit of utilizing centralized information, but learning over a smaller space.

C. Results in the Warehouse Tool Delivery Domain

We test our second proposed algorithm Parallel-MacDec-MADDRQN (Our-2) in this warehouse domain using the same evaluation procedure mentioned above. The results shown in Fig. 4 are generated by using the same neural network architecture as the one adopted in the BP domain but with 32 neurons in each MLP layer and 64 neurons in LSTM layer for both the centralized Q-net and each decentralized Q-net because of the bigger macro-action and macro-observation spaces.

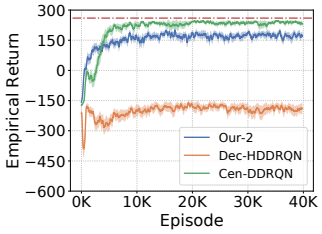


Fig. 4: Performance of three different learning methods in WTD.

The most challenging part in this domain is that robots need to reason about collaborations among teammates and which tool the human will need next. However, the gray robot, that plays the key role of finding the correct tool for delivery, does not have any knowledge about the human's need nor any direct observation of the human's status. Also, the mobile robots cannot observe each other. From the gray robot's perspective, the reward for its selection is very delayed, which depends on the mobile robots' choice and their moving speeds. For these reasons, each robot individually learning from local signals (in Dec-HDDRQN) leads to much lower performance but the centralized learner can achieve near-optimal results. Our approach achieves significant improvement while learning decentralized policies, but due to the limitation of local information, it inherently cannot perform as well as the centralized policy in such a complicated domain. Nevertheless, the near-optimal behaviors are still learned by our Parallel-MacDec-MADDRQN, which are presented in the real robot experiments (Section V).

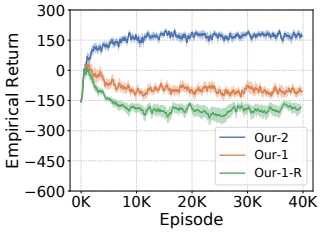


Fig. 5: Results of ablation experiments in WTD.

We also conducted ablation experiments in WTD in order to investigate 1) the necessity of separately training the centralized Q-net and decentralized Q-nets in two environments by comparing Parallel-MacDec-MADDRQN (Our-2) with MacDec-MADDRQN with centralized exploration (Our-1); 2) the significance of including centralized Q_ϕ in double-Q updating to optimize each decentralized Q_{θ_i} (Eq. 4) by performing Our-1 with regular deep double-Q learning (referred to Our-1-R). The results shown in Fig. 5 reveal that Our-2 outperforms other two ablations, which gives the affirmative answers to the above questions.

V. HARDWARE EXPERIMENTS

To verify that the learned decentralized policies in Parallel-MacDec-MADDRQN can effectively control a team of robots to achieve high-quality results in practice, we recreated the warehouse domain using three real robots: one Fetch robot [4] and two Turtlebots [5] (Fig. 6). A rectangle space with dimension $5.0\,\mathrm{m}$ by $7.0\,\mathrm{m}$ was taped to resemble the warehouse in the simulation (Section IV-A). All the pre-

defined waypoints and robots' initial positions were placed equal in ratio to the simulation. Also, the real-world human's task is to build a small table in the workshop, requiring three particular tools in the following order: a tape measure, a clamp and an electronic drill (from YCB object set [23]).

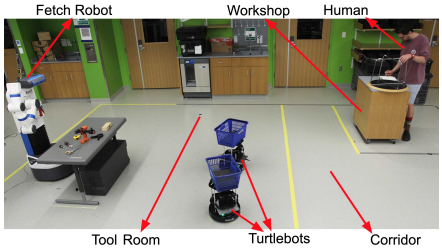
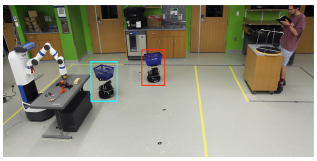


Fig. 6: Hardware experiment setup.

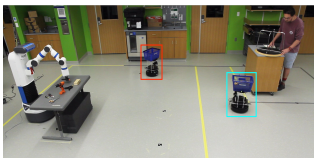
Each robot had its own decentralized macro-observation space designed over ROS [24] services that kept broadcasting the signals about Turtlebots' locations, human's state (only accessible to the Turtlebot when it is located in workshop area), the status of each Turtlebot's basket, and the number of objects in the staging area (only observable in the tool room). Fetch's manipulation macro-actions are achieved by first projecting the point cloud data captured by Fetch's head camera into an OpenRAVE [25] environment and performing motion planning using the OMPL [26] library. The Turtlebot's movement macro-actions are controlled via the ROS navigation stack.

Fig. 7 shows the sequential cooperative behaviors performed by the robots. Although there is no direct interaction between the Fetch and the human, the trained policy learned the correct tools that the human needed and commanded the Fetch to find them in the proper order. Furthermore, the Fetch behaved intelligently such that: (a) Fig. 7c-7e, after placing the clamp into the staging area followed by observing no Turtlebot beside the table, it continued to look for the third object instead of waiting for Turtlebot-0 (bounded in red) to come over; (b) Fig. 7e-7f, after finding the electronic drill, it first passed the clamp (the correct second object that the human needed) to Turtlebot-0 who arrived the table ahead of Turtlebot-1(bounded in blue). Meanwhile, Turtlebots were also clever in such a way that: (a) they delivered the three tools in turn, instead of letting one of them deliver all the tools or perform delivery only after having all the tools in the basket which actually would make the human wait; (b) they directly went to the human for delivery after obtaining a tool from the Fetch without any redundant movement, e.g. going to the tool room waypoint again.

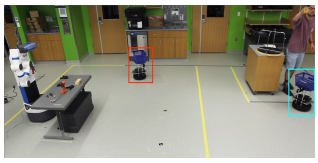


(a) Fetch searches and stages the tape measure as T-1 approaches the table.

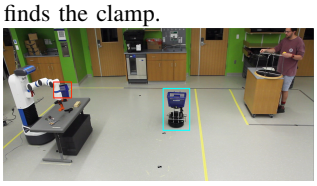
(b) Fetch sees T-1 arriving and passes it the tape measure, while T-0 reaches workshop and observes human's state.



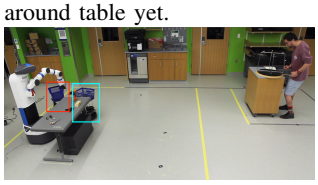
(c) T-1 observes the tape measure in its basket and moves to workshop, while T-0 goes back tool room and Fetch finds the clamp.



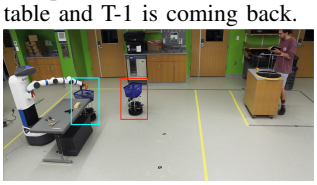
(d) T-1 deliveries the tape measure and T-0 runs to the table for the second tool, while Fetch notices no teammate around table yet.



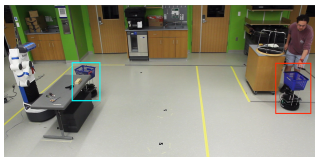
(e) Fetch grabs the electronic drill and stages it next to the clamp, while T-0 waits besides table and T-1 is coming back.



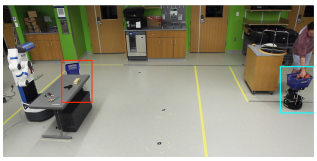
(f) Fetch observes T-0 has been ready there and passes clamp to it, in the mean time, T-1 arrives at the table.



(g) T-0 immediately goes to send the 2nd tool and Fetch passes the last tool to T-1.



(h) Human gets the clamp from T-0, and T-1 is going to deliver the electronic drill.



(i) The last tool is passed to the human by T-1 and the entire delivery task is completed.

Fig. 7: Behaviors of robots running the decentralized policies (learned via Parallel-MacDec-MADDRQN) in the warehouse domain, where Turtlebot-0 (T-0) is bounded in red and Turtlebot-1 (T-1) is bounded in blue.

VI. CONCLUSION

This paper introduces MacDec-MADDRQN and Parallel-MacDec-MADDRQN: two new macro-action-based multi-agent deep reinforcement learning methods with decentralized execution. These methods enable each agent's decentralized Q-net to be trained while capturing the effects of other agents' actions by using a centralized Q-net for decentralized policy updating. The results in the benchmark Box Pushing domain demonstrate the advantage of our methods where the decentralized training achieves equally good performance as the centralized one. Furthermore, the warehouse domain results confirm the benefits and the efficiency of our new double-Q updating rule. Importantly, a team of real robots running the decentralized policies learned via our method performed efficient and reasonable behaviors in the warehouse domain, which validates the usefulness of our macro-action-based deep RL frameworks in practice.

Acknowledgements. This research was funded by ONR grant N00014-17-1-2072, NSF award 1734497 and an Amazon Research Award (ARA).

REFERENCES

[1] X. Liu, S. Chen, S. Aditya, N. Sivakumar, S. Dcunha, C. Qu, C. Taylor, J. Das, and V. Kumar, "Robust fruit counting: Combining deep learning, tracking, and structure from motion," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018*, 12 2018, pp. 1045–1052.

[2] J. J. Enright and P. R. Wurman, "Optimization and coordinated autonomy in mobile fulfillment systems," in *Proceedings of the 9th AAAI Conference on Automated Action Planning for Autonomous Mobile Robots*, 2011, pp. 33–38.

[3] D. L. Rosenband, "Inside waymo's self-driving car: My favorite transistors," *2017 Symposium on VLSI Circuits*, pp. C20–C22, 2017.

[4] M. Wise, M. Ferguson, D. King, E. Diehr, and D. Dymesich, "Fetch & freight : Standard platforms for service robot applications," in *Workshop on Autonomous Mobile Service Robots, International Joint Conference on Artificial Intelligence*, 2016.

[5] A. Koubaa, M.-F. Sriti, Y. Javed, M. Alajlan, B. Qureshi, F. Ellouze, and A. Mahmoud, "Turtlebot at office: A service-oriented software architecture for personal assistant robots using ROS," *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pp. 270–276, 2016.

[6] S. Omidshafiei, J. Pazis, C. Amato, J. P. How, and J. Vian, "Deep decentralized multi-task multi-agent reinforcement learning under partial observability," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2017, pp. 2681–2690.

[7] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *AAAI 2018: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[8] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *Neural Information Processing Systems (NIPS)*, 2017.

[9] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. Foerster, and S. Whiteson, "QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning," in *ICML 2018: Proceedings of the Thirty-Fifth International Conference on Machine Learning*, 2018.

[10] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. F. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel, "Value-decomposition networks for cooperative multi-agent learning," *CoRR*, 2017.

[11] Y. Xiao, J. Hoffman, and C. Amato, "Macro-action-based deep multi-agent reinforcement learning," in *3nd Annual Conference on Robot Learning (CoRL)*, 2019.

[12] C. Amato, G. D. Konidaris, and L. P. Kaelbling, "Planning with macro-actions in decentralized POMDPs," in *Proceedings of the Conference on Autonomous Agents and Multiagent Systems*, 2014.

[13] C. Amato, G. Konidaris, L. P. Kaelbling, and J. P. How, "Modeling and planning with macro-actions in decentralized POMDPs," *Journal of Artificial Intelligence Research*, vol. 64, pp. 817–859, 2019.

[14] F. A. Oliehoek and C. Amato, *A Concise Introduction to Decentralized POMDPs*. Springer Publishing Company, Incorporated, 2016.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[16] M. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable mdps," in *AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents (AAAI-SDMIA15)*, 2015.

[17] H. V. Hasselt, "Double Q-learning," in *Advances in Neural Information Processing Systems 23*, 2010, pp. 2613–2621.

[18] H. v. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI'16, 2016, pp. 2094–2100.

[19] D. Simões, N. Lau, and L. P. Reis, "Multi-agent double deep Q-networks," in *Progress in Artificial Intelligence*. Springer International Publishing, 2017, pp. 123–134.

[20] Y. Zheng, Z. Meng, J. Hao, and Z. Zhang, "Weighted double deep multiagent reinforcement learning in stochastic cooperative environments," in *15th Pacific Rim International Conference on Artificial Intelligence*, 07 2018.

[21] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018.

[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[23] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Benchmarking in manipulation research: Using the Yale-CMU-Berkeley object and model set," *IEEE Robotics Automation Magazine*, vol. 22, no. 3, pp. 36–52, 2015.

[24] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[25] R. Diankov and J. Kuffner, "Openrave: A planning architecture for autonomous robotics," Carnegie Mellon University, Pittsburgh, PA, Tech. Rep., 2008.

[26] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012.