IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 13 May 2024, accepted 30 May 2024, date of publication 3 June 2024, date of current version 11 June 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3409076

RESEARCH ARTICLE

Multiagent Hierarchical Reinforcement Learning With Asynchronous Termination Applied to Robotic Pick and Place

XI LAN, YUANSONG QIAO, (Member, IEEE), AND BRIAN LEE
Software Research Institute, Technological University of the Shannon, Athlone, N37 HD68 Ireland

Corresponding author: Xi Lan (xi.lan@tus.ie)

This work was supported in part by the Confirm Centre for Smart Manufacturing funded by the Science Foundation Ireland (SFI) under Grant SFI/16/RC/3918, and in part by European Regional Development Fund.

ABSTRACT Recent breakthroughs in hierarchical multi-agent deep reinforcement learning (HMADRL) are propelling the development of sophisticated multi-robot systems, particularly in the realm of complex coordination tasks. These advancements hold significant potential for addressing the intricate challenges inherent in fast-evolving sectors such as intelligent manufacturing. In this study, we introduce an innovative simulator tailored for a multi-robot pick-and-place (PnP) operation, built upon the OpenAI Gym framework. Our aim is to demonstrate the efficacy of HMADRL algorithms for multi robot coordination in a manufacturing setting, concentrating on their influence on the gripping rate, a crucial indicator for gauging system performance and operational efficiency.

INDEX TERMS Multi-agent system, pick and place, multi-agent-hierarchical reinforcement learning, multi-robot system, asynchronous termination.

I. INTRODUCTION

Over the past decade, manufacturing systems have undergone a significant evolution, becoming more intelligent and automated. This transformation has been driven by advancements in hardware and software technologies, including the Industrial Internet of Things (IIoT), which has led to increased productivity and reduced operating costs. A notable trend in this evolution is the increasing use of robotics, particularly in response to challenges such as the need to adapt or re-purpose production lines to meet shifting market demands or the introduction of new product types. In this context, teams of robots, often referred to as multi-robot systems (MRS), are being more and more deployed. The primary challenge in an MRS lies in coordinating the actions of individual robots to optimize overall team performance [1]. Multi-robot systems, a subset of Multi-Agent Systems (MAS), consist of multiple autonomous agents that interact with one another and their environment to achieve shared or individual objectives [2]. These systems often integrate with Reinforcement Learning (RL) to form multi-agent reinforcement learning (MARL) frameworks, or multi-agent deep reinforcement learning (MADRL) when employing deep RL (DRL), such as the implementation of Deep Q Networks (DQN) highlighted by Mnih et al. [3].

One prominent application of multi-robot teams in modern manufacturing is in Pick and Place (PnP) tasks, characterized by simple, repetitive, and monotonous actions. The PnP process involves two sub-processes: sensing (visual perception and classification of parts) and gripping (efficient separation of parts) [4]. Dimensioning a PnP system appropriately to optimize gripping performance when multiple robots are involved presents a significant challenge. Commonly, field experiences are used for sizing, but this may not be optimal for some PnP scenario's where robots act concurrently with possibly different pick action time lengths for each robot that could lead to uneven workload distribution among robots.

Hierarchical reinforcement learning (HRL) [5] has been proposed as an approach to dealing with the problem of learning concurrent actions for multi agent systems [6]. HRL introduces a hierarchical structure in the learning process, allowing agents to learn at different abstraction

The associate editor coordinating the review of this manuscript and approving it for publication was Mohammad AlShabi.

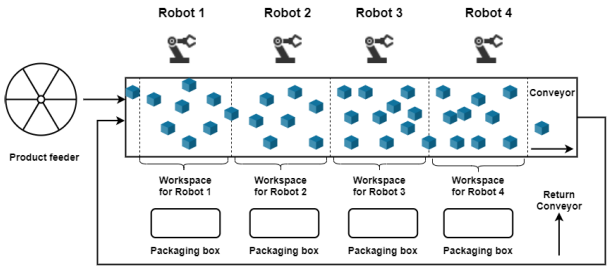

FIGURE 1. Top view of multi-robot system based on [8].

levels. This methodology is effective in complex tasks with long-term dependencies and is instrumental in making decisions across multiple temporal scales [7]. Integrating these concepts, researchers are exploring the use of multi-agent deep reinforcement learning and hierarchical reinforcement learning in multi-robot systems. This integration promises to significantly enhance the autonomy, efficiency, and decision-making capabilities of robotic teams.

This paper presents a novel method that employs hierarchical multi-agent deep reinforcement learning to tackle the coordination challenges in multi-robot PnP systems. This initial work seeks to assess the viability of this approach. To the best of our knowledge, this study is a first attempt at applying HMADRL for multi-robot PnP coordination. The structure of the paper is as follows: Section II offers an overview of PnP operations and discusses the optimization techniques that have been utilized up to this point. Section III descibe related work. Section IV describes concurrent multi agents systems including hierarchical RL. Section V describes the methodology implemented in our study including the design of our bespoke PnP simulator. In Section VI, we elaborate on the experiments conducted and their corresponding results. Finally, Section VII concludes the paper with a summary of our findings and an outline of prospective future research directions.

II. PICK AND PLACE BACKGROUND

A. PICK AND PLACE DESCRIPTION

An overview of a generalised multi-robot pick and place system is given in Fig.1. It is composed of multiple identical robots, a part feed device, a moving conveyor, multiple packaging boxes, and a return conveyor. Each robot has its own non-overlapping workspace on the conveyor and a packaging box on the other side of the conveyor. Each robot picks up parts moving through its workspace and places them into packaging boxes as optimally as possible. A part feeder feeds parts onto one end of the conveyor belt following some probability distribution [8]. Parts move along the conveyor belt with a fixed flow while the parts remaining at the other end typically return back to the conveyor via a return conveyor.

B. SIZING OF INDUSTRY APPLICATION

In most PnP tasks on a moving conveyor, there is no workflow optimization and the sizing of the PnP system is done empirically. The flow of parts can be controlled, and the system parameters tuned to achieve the maximum throughput for the conveyor system. The parameters of interest include:

- part flow rate f_p i.e. number of parts fed onto conveyor per time unit, [8], [9]. High part flow can improve the productivity but will have more parts left on the conveyor. On the contrary low part flow will reduce the productivity.
- the average time for a robot to perform a pick and place task, t_p. t_p is a characteristic of the robot and is not tunable. Metrics derived from t_p include:
  - the number of pick-and-placed parts per unit-time i.e. so-called *part cadence PPas* or *throughput*
  - the *gripping rate, G*, i.e. the number of parts picked by the robot over the total number that enter the workspace and
  - *efficiency*, i.e. the throughput over the maximum throughput
- the velocity v_b of the belt
- the distance d between two consecutive parts on the belt.

In most cases the items are equally spaced (distance d and v_b / d matches the minimum robot pick-and-place time). The sizing of such a conveyor system is then done empirically according to the following decision sequence [9],:

1) A particular input part flow rate f_p is chosen.
2) The part cadence *PPas* is estimated
3) The minimum number of robot is calculated: $N_{min} = f_p / PPas$.
4) A margin of safety is included by adding one or more robots to N_{min}

In the above system, the set of parts to be picked by a particular robot can be considered as a queue of jobs waiting to be ''served'' and a simple first-in-first-out (FIFO) rule is adopted as scheduling discipline to select the next job (i.e. part) from the queue i.e. pick the next part.

However, there are cases where the arrival of parts onto the belt is not fixed and is characterised by a stochastic process [8], [10]. In these cases, the distance between the incoming parts varies greatly. This means that the robot picking time is no longer constant within its workspace and the serving time of the parts in the queue (i.e., the time needed to perform the pick-and-place ''service'' they require) varies with their moving position. In these situations the FIFO rule may perform very inefficiently, thus requiring different scheduling strategies for different robots.

The challenge for these dynamic serving time PnP systems then becomes how to choose the appropriate set of scheduling disciplines, in the face of random part distribution on the conveyor, to optimise the overall system performance. This is inherently a multi-robot coordination challenge.

III. RELATED WORK

Mattone et al. [10] focused on optimizing gripping strategies for single robot PnP systems. For multi-robot systems, however, more intricate algorithms and scheduling rules are

necessary. Huang et al. [8] suggested the integration of part-dispatching rules with a greedy randomized adaptive search procedure (GRASP) and a Monte Carlo strategy for coordinating multiple PnP robots. Bozma and Kalalıoğlu [11] developed a multi-robot coordination approach for PnP tasks on a conveyor belt using non-cooperative game theory, where each robot's action was informed by local observations, including the actions of adjacent robots. Daoud et al. [12] examined the effectiveness of three metaheuristics—ant colony optimization, particle swarm optimization, and genetic algorithms—in maximizing part picking with minimal execution times.

In recent developments, researchers have started applying MADRL to PnP systems. Wang et al. [13] introduced a novel algorithm, MRCDRL, utilizing end-to-end MADRL for addressing obstacle avoidance challenges among multiple robots. Gomes et al. [14] conducted a case study using DRL to improve the efficiency of a collaborative robot (cobot) in PnP tasks within human-robot interaction scenarios. Liu et al. [15] investigated the application of DRL for training single robot manipulators in simulations and successfully transferring these learned policies to real-world tasks, showcasing the potential of sim-to-real transfer in complex manipulative tasks. Kim et al. [16] introduced a multi-agent approach based on Deep Q-Networks (DQNs) designed to adapt and make improved decisions within dynamic environments. These DQN agents represent various components of the manufacturing process responsible to access and prioritise jobs. They are continuously learning and hence improve their ability to make scheduling decision, which in turn leads to improvements in task allocation and mass customisation. Zhou et al. [17] proposed a smart factory model featuring a range of components and introduced a decentralized job scheduling approach using a multi-agent actor-critic method. Each machine in this setup is paired with an actor-critic agent, further enhanced by target policy and target critic networks. These agents communicate to observe each other's states, allowing them to effectively navigate the dynamic and non-stationary environment of the factory.

In the context of HRL, a crucial contribution was made by Sutton and Barto [18], who developed a significant framework for temporal abstraction in reinforcement learning, effectively bridging the gap between MDPs and semi-MDPs. This framework has been fundamental in hierarchical approaches, especially for breaking down complex robotic tasks into smaller, more manageable sub-tasks, thus enhancing learning efficiency and performance in multi-robot PnP systems. Further extending, Dietterich [19] explored the hierarchical decomposition of reinforcement learning tasks, providing a robust methodology for structuring problems in a hierarchical fashion. This approach significantly aids in managing the complexity inherent in multi-agent environments. Luo et al. [20] introduced a hierarchical reinforcement learning (HRL) approach for production scheduling, using a two-tiered system to minimize tardiness and maximize machine utilization. A high-level DDQN agent sets global goals, while a low-level DDQN handles dispatching rules. However, this added complexity can impede the learning process's convergence. Also in their further paper [21] they developed a hierarchical multi-agent proximal policy optimization (HMAPPO) approach for dynamic scheduling problems with no-wait constraints. This method uses three types of agents: a controller, a job selector, and a machine assigner. The controller sets temporary goals, while the job and machine agents quickly adapt to real-time demands. Despite its hierarchical structure facilitating layered learning, the system isn't fully decentralized, and optimality of the combined policies isn't guaranteed. Despite these significant advancements in the field, there remains a gap in research specifically focused on the optimization of gripping rates in PnP systems. Our research aims to address this gap by applying and extending these foundational concepts of HRL to the specific challenge of optimizing gripping rates in multi-robot PnP systems.

IV. CONCURRENT MULTI AGENT SYSTEMS

A. PARTIALLY OBSERVABLE MDP

In multi-agent reinforcement learning (MARL), the dynamics of decision-making are substantially more intricate than in single-agent scenarios. This complexity arises due to the influence of both the environment and the strategies of other agents on the optimal policy for any given agent. Agents in such environments must either collaborate or compete to effectively navigate and resolve task-specific challenges. Theoretical frameworks like game theory are instrumental in understanding these strategic interactions among agents. Consequently, the conventional single-agent MDP model is expanded into a Markov game to accommodate multi-agent interactions.

In a collaborative multi-agent environment, it is typically essential for each agent to possess knowledge about the other agents to inform its decision-making process. However, many existing methods overlook a crucial reality—agents may not always have unrestricted access to this information. In the context of distributed multi-agent scenarios, the world is generally only partially observable to each agent. To address this partial observability, Partially Observable Markov Decision Processes (POMDPs) extend the concept of MDPs. In a POMDP, agents form policies as mappings from a set of beliefs about the state to actions, based on a history of observations to estimate the state probabilities. This partial observability poses heightened challenges in multi-agent settings, as agents may lack information about other agents' states or the entire environment [22]. The POMDP framework can be extended to allow for multiple distributed agents to base their decisions on their local observations. This model is called decentralized POMDP (Dec-POMDP). This model integrates joint actions and observations, where each agent operates based on its individual perceptions and actions without real-time communication with others.

In Dec-POMDP, agents independently make decisions based on partial information, which is crucial in scenarios with limited or no communication. This approach allows agents to navigate complex environments where collective state transitions are determined by the confluence of individual agents' actions [22]. Thus, POMDP and Dec-POMDP provide robust frameworks for modeling and solving intricate and uncertain scenarios characteristic of multi-agent systems.

One fundamental distinction in MARL algorithms is whether they adopt a centralized or decentralized training approach. In centralized approaches, a central controller or critic observes the complete state of the environment and the actions of all agents. This centralized entity provides a global perspective, enabling efficient coordination among agents. The new approach of centralised training decentralised execution (CTDE) [23] is widely using in MARL POMDP, agents undergo training with access to sharing information, but during online execution, they operate in a decentralized manner with agents' local observation [24].

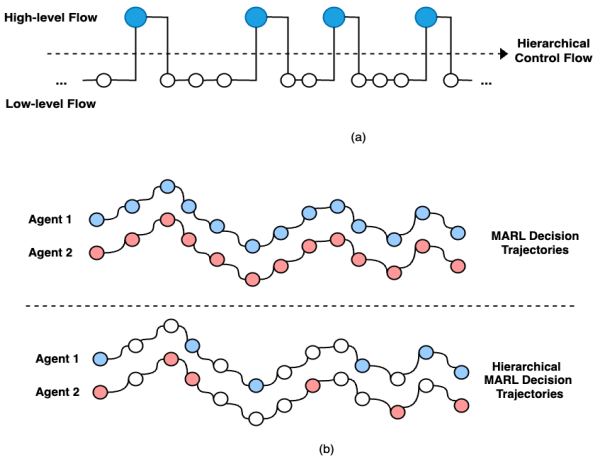

FIGURE 2. (a) Hierarchical control flow. (b) Decision trajectories for MARL and hierarchical MARL. For (a) and hierarchical MARL in (b), solid and hollow circles denote intrinsic goals and primitive actions respectively. (following [27]).

B. TEMPORAL ABSTRACTION AND HIERARCHICAL MARL

A Multi-robot PnP with concurrent actions modelled as a MARL/MADRL system is an example of an AI system that requires learning, planning, and representing knowledge at multiple levels of *temporal abstraction*. Temporal abstraction is instrumental in managing tasks that span a range of time scales, as it simplifies complexities by enabling agents to overlook minute details that are not directly relevant to the primary task [25].

Hierarchical RL can be used to address temporal abstraction by decomposing a temporally extended reinforcement learning task into a hierarchy of subproblems or subtasks such that each subtask is a higher-level action that persists for a longer timescale compared to a lower-level action and the agent operates and learns simultaneously at multiple levels of abstraction. Each higher-level policy learns to perform the task by choosing optimal subtasks as the higher-level actions.

As per literature describing multi agent temporal abstraction we model a two level hierarchical learning task as a Markov game. Here, each agent establishes intrinsic, multi-step goals at the higher level, followed by executing a series of primitive actions at the lower level to fulfill these goals - see Figure 2 (a). It is assumed that the intrinsic goals are temporal abstraction of the simple tasks or skills that can be achieved or accomplished independently. This approach not only eases the complexity inherent in the original problem but also preserves the multiagent dynamics.As a result, agents are equipped to learn cooperative and coordinated behaviors at the higher strategic level.

Intrinsic goals are a key factor to achieving effective hierarchical coordination. Intrinsic motivation - as opposed to *extrinsic motivation* from the environment - involves providing the agent with additional rewards or goals that are not explicitly defined by the external task but emerge from within the agent itself. The idea is to encourage the agent to explore its environment and learn more effectively. By incorporating intrinsic motivation, the agent may exhibit behaviors such as curiosity, novelty-seeking, or skill development that go beyond the immediate external task. The goal of incorporating intrinsic motivation is to enhance the agent's learning process, promote exploration, and enable the agent to discover more robust and adaptive policies. It can be particularly useful in scenarios where the external reward signals may be sparse or delayed, as intrinsic motivation can help the agent to learn more efficiently in the absence of immediate external feedback.

Based on the above multi agent temporal abstraction model, we consider hierarchical MARL as illustrated in Figure 2 (b). The higher level of the hierarchy can be modeled as a Semi-Markov game, similar to the Multiagent Semi-MDP (MSMDP), since intrinsic goals may last for multiple time steps. SMDP's were pioneered by [26] as an extension to the basic RL framework for representing temporal abstraction i.e. the actions in SMDPs take variable amounts of time and temporal abstraction is captured as the (hierarchical) interplay between SMDPs and MDPs.

A Multi-agent Semi-Markov Decision Process (MSMDP) is characterized by six elements: (D, S, A, P, R, T) and is defined in the following manner:

1) Agent Set D: This represents a finite group of n agents, where each agent, denoted as i in D.
2) Joint Action Space A: The combined actions of all agents are represented in A, which is the product of individual action sets A_i for each agent i, where i ranges from 1 to n.
3) State Set S: S is the set of all possible states in the system.
4) Reward Function R: This is a function that maps states in S to real numbers, representing the rewards.

5) Transition Probability Function P: Defined as P: $S \times IN \times S \times A \rightarrow [0, 1]$, where IN is the set of natural numbers, this function determines the probability of transitioning from one state to another, given a particular joint action and time steps. For instance, $P(s', N|s,\vec{a})$ represents the probability that the joint action $\vec{a}$ will transition the system from state s to state s' in N time steps.
6) Termination Scheme T: Since joint actions consist of temporally extended individual actions that may not conclude simultaneously, the multi-step transition probability P is influenced by the manner in which decision epochs are defined, which in turn depends on the termination scheme T.

Returning to Figure 2 (b), formally, each agent i receives observation o_t^i and chooses a goal $g_t^i \in G^i$, where G^i denotes the set of all possible intrinsic goals. A new goal $g_{t+\tau}^i$ is selected until the current goal g_t^i is achieved or terminated after τ steps of low-level executions. The next state $s_{t+\tau}$ is determined by the multi-step transition probability function P. The objective of agent i's high-level policy π^i is to maximize the cumulative extrinsic reward.

Whereas we model the higher level as SMDPs, we model the lower level of hierarchy as MDPs. Agent i receives the intrinsic observation ϕ_t^i which depends on observation o_t^i and current goal g_t^i, and chooses an action $a_t^i \in A_{g_t^i}^i$, where $A_{g_t^i}^i (\subseteq A^i)$ is a set of all available primitive actions under the current goal. Given an intrinsic goal, agent i learns a low-level policy π_g^i via optimizing the cumulative intrinsic reward.

The above discussion is detailed further in the next section which describes how HRL agent can be realised using Deep Q Networks. Following [28] we can consider the two layers of a HRL agent to consist of a *controller* (lower level) and *meta-controller* (upper level) - see Figure 3

The *meta-controller*, receive the current state s_t, selects a goal g_t from a predefined goal space $\mathcal{G}$. The *controller* is then responsible for determining an action a_t, taking into account both the state s_t and the goal g_t. The goal g_t is maintained across a sequence of time steps, until it is fulfilled or until the end state is reached. The role of the internal critic is to assess goal attainment and allocate an intrinsic reward $r_t(g)$ to the *controller*. The objective of *controller* is to maximize the sum of intrinsic rewards, formulated as:

$$R(g) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}(g).$$

Conversely, the *meta-controller* is tasked with maximizing the sum of extrinsic rewards, $F_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} f_{t'}$, where f_t symbolizes the extrinsic rewards emanating from the environment.

Finally we can consider an example scenario to illustrate these ideas. This is the Multi Agent Trash Collection (MATC) [5], [27]. In this scenario a several agents pick trash cans over extended area (often modelled as a grid world) and deposit the trash to a central bin. For example consider a case with two

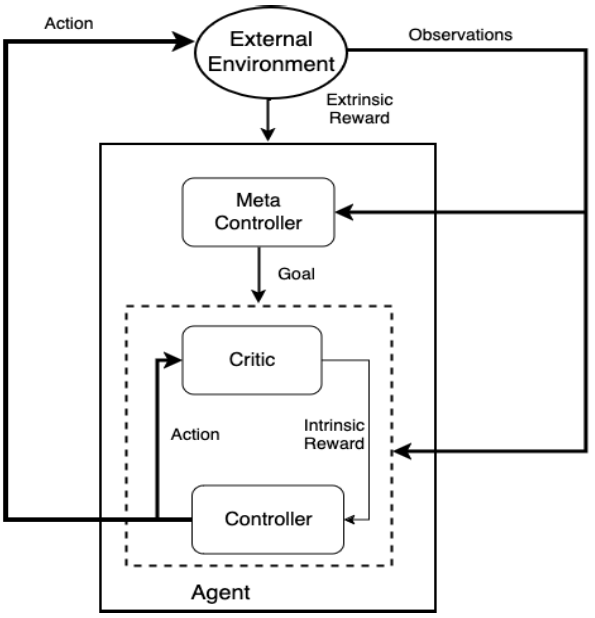

FIGURE 3. HDQN agent.

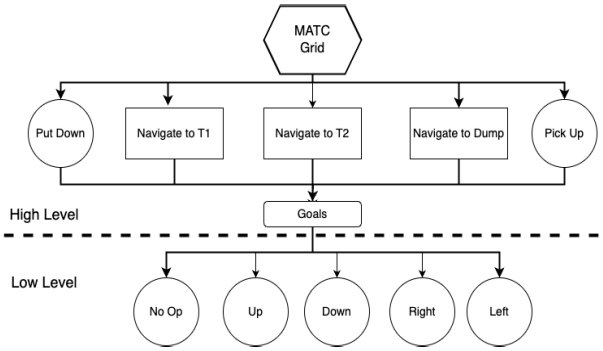

FIGURE 4. Temporal abstraction for the MATC scenario.

agents A1, A2 and trash cans T1, T2 and central bin *Dump*. Agents need to learn three skills here. First, how to do each subtask, such as navigate to trash cans T1 or T2 or Dump, and when to perform Pick or Put action. Second, the order to carry out the subtasks, for example go to T1 and collect trash before heading to Dump. Finally, how to coordinate with each other, i.e., agent A1 can pick up trash from T1 whereas agent A2 can service T2. The strength of the HRL methods (when extended to multi-agent domains) is that they can serve as a base for efficiently learning all these three types of skills. In these methods, the overall task is decomposed into a collection of primitive actions and temporally extended (non-primitive) subtasks that are important for solving the problem. A task graph representation for this scenario can be derived from from [27] - see Figure 4

Each agent has 7 actions, including navigation actions (i.e., up, down, left, right and no-op) and operation actions (i.e., pick-up and put-down). The task is abstracted into two one-step operation goals (i.e., pick-up, put-down) and several

navigation goals that are built over navigation actions. Once a multi-step action is chosen on the high level by the controller the destination of the object referenced by the action is given as a goal to the lower level. The agent learns a low level policy to move to the destination with the minimal amount of moves. A binary intrinsic reward function for low-level policy learning, which gives 1 for reaching the goal and - 0.01 otherwise. At the upper level agent receive a reward related to the speed of completion of the trash pick up. Note that the upper level goals include two primitive actions also. The reason for this is that the problem is constructed such that the lower level policy learns over navigation actions which the pick-up and put-down actions are not. The no-op action is used for special cases such as when the destination is reached and when pick and put are chosen as the high level goals.

C. TERMINATION SCHEMES

Joint actions in a HRL Dec-POMDP consist of temporally extended individual actions that may not conclude simultaneously and which may exhibit a range of *termination schemes*. Three such schemes, namely *t_any, t_all,* and *t_continue* were introduced in [6] for temporally extended joint actions - see Figure 5.

In the *t_any* termination scheme, the subsequent decision epoch is determined by the first action within the ongoing joint action reaching completion. The actions that have not yet terminated are interrupted, and the next decision epoch ensues.

In the *t_all* termination scheme, the next decision epoch occurs when all actions within the ongoing joint action have concluded. After completing an action, an agent waits (takes the idle action) until all other agents finish their current actions, leading to a synchronized decision epoch where all agents jointly choose the next action.

The *t_continue* termination scheme shares similarities with *t_any*, as the next decision epoch is triggered by the termination of the first action within the current joint action. However, in *t_continue*, agents whose activities have not terminated are not interrupted. Only those agents whose actions have concluded select new actions, creating a scenario where only a subset of agents make decisions at each decision epoch. After an agent completes an action, the next decision epoch occurs exclusively for that agent, and it selects its next action based on the ongoing actions performed by other agents.

We can categorise these approaches as either *synchronous* or *asynchronous* [5]. In synchronous strategies- such as *t_any*, *t_all*- every agent selects their actions simultaneously at each decision point as showing in Figure 5. However, a drawback of synchronous approaches is the potential for less optimal policies. This happens because agents need to either delay or prematurely conclude their lower-level tasks to comply with the synchronization demands [5]. Such issues could intensify as the number of agents increases.

On the other hand, asynchronous strategies - such as *t_continue*- are defined by only some agents making action decisions at each decision epoch as as showing in Figure 5. This setup eliminates the need for a central system to synchronize agents, enabling a decentralized approach to decision-making. This flexibility allows agents to develop more adaptable policies, though it may also present challenges in achieving effective coordination among them.

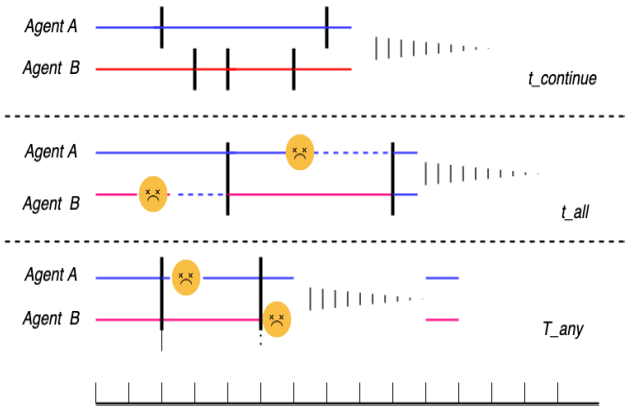

FIGURE 5. Task termination schemes.

V. METHODOLOGY

A. MADRL APPROACH FOR PnP

We consider the multirobot PnP gripping process as a hierarchical RL multi-agent system with asynchronous termination where the agents (robots) cooperate to optimise the throughput of products through the system. The PnP system acts as a decentralised POMDP where each agent has only partial observability of the whole system and takes local actions based on this information.

Moreover, we leverage the approach proposed by Bozma and Kalalıoğlu [11] to model the multi robot system PnP conveyor. This formulation is depicted in Figure 6, where the workspace of each robot is segmented into several arbitrarily smaller sub-workspaces where each such sub-workspace holds one part placed on the belt. Taken to its limit the conveyor belt can be imagined as a grid of point locations much like a computer vision image. We use this *'conveyor belt as image'* paradigm as the basis for the model design and, following Mnih et al. [3], use DQN for the algorithm development. When building a DQN based HRL, both controller and meta-controller are usually implemented as (separate) DQN components [27], [28], with a centralised training, decentralised execution approach used for the MARL case.

To achieve temporal abstraction we use *part-dispatching rules/scheduling disciplines* (e.g., first-in-first-out (FIFO) and shortest distance (SD)) as actions at the **higher (meta-controller) layer** of the hierarchical DQN multi agent system.

The use of part dispatching rules for coordination in a multi robot pick and place system has been investigated by a number of researchers [8], [10], [29], [30]. Different

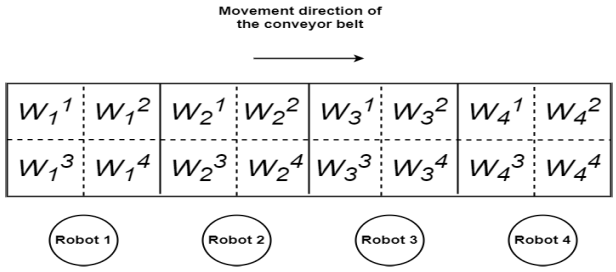

FIGURE 6. Workspace for four robots.

part-dispatching rules can be used for the different robots such that each robot picks up as many products moving through its workspace as possible. The challenge for these dynamic serving time PnP systems then becomes how to choose the appropriate set of scheduling disciplines, in the face of random product distribution on the conveyor, so as to optimise the overall system performance. The scheduling disciplines considered for our study include the following options:

1) First In First Out (FIFO): This discipline involves picking up the part that first entered the robot's workspace on the conveyor.
2) Last In First Out (LIFO): Under this discipline, the robot picks up the part that most recently entered its workspace on the conveyor.
3) Service In Random Order (SIRO): This approach allows the robot to randomly select any part within its workspace on the conveyor for pick up.
4) Shortest Distance (SD): This method prioritizes picking up the part that is closest to the upper edge of the conveyor, minimizing travel distance.
5) Longest Distance (LD): Contrary to SD, this discipline focuses on picking up the part that is nearest to the lower edge of the conveyor.

Each of these disciplines offers a unique approach to task execution in the context of our multi-agent system.

At the **lower (controller) layer** of the HDQN three primitive actions suggest themselves i.e.

1) Pick i.e. robot picks a a part from its workspace
2) Place i.e. a robot places a part in the receiving bin
3) No-oP i.e. the robot takes no action - this action is used to emulate the passing of time as the robot arm moves either to pick or place the part. No-oP is a common primitive action in hierarchical RL [5], [27]

The task decomposition illustrating the interaction between the higher and lower layers is shown in Figure 7. The upper layer actions are selected by the meta controller policy while the lower layer actions are chosen the controller policy. As noted previously both levels of controller are normally implemented as DQN models and that is indeed the case for the meta-controller here. However the lower level of our agent is considerably simplified by the fact that the pick and place operations are in fact intrinsically part of the upper

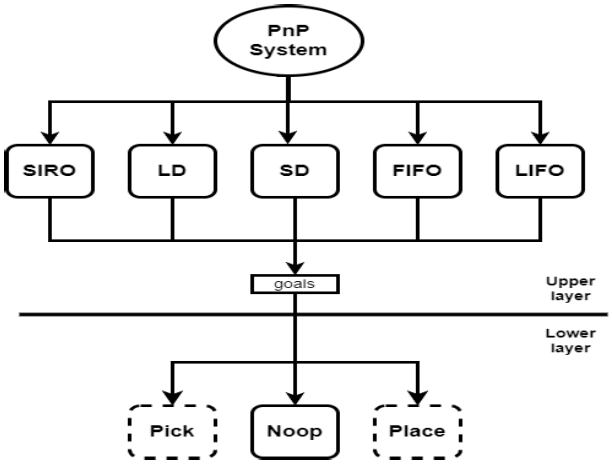

FIGURE 7. PnP temporal abstraction.

level actions within our simulator as will be discussed in the next section. While they could be abstracted out and cast as stand-alone primitive actions it would be an artificial construct and would offer no theoretical nor implementation advantage - this fact is indicated by the dashed line defining the actions. In consequence we are left with just a single low level primitive action -NooP -which negates the need for action selection. The policy is deterministic with just one action for every state.

A more detailed description of the agent architecture is given in section VI.

B. PnP SIMULATION

1) SIMULATOR OPERATION

Simulation is a critical tool in the realm of robotics algorithm development, offering a platform to assess the efficiency, safety, and robustness of new algorithms, as highlighted by Staranowicz and Mariottini [31]. A variety of both commercial and open-source robotic simulators, such as Gazebo [32] and Vrep [33], are available, providing valuable support for the development of diverse robotic scenarios, including pick-and-place operations. These simulators, often integrated with the Robot Operating System (ROS) [34], allow for algorithms crafted in the simulation environment to be directly applied to real-world robots. Additionally, these simulators typically offer programming interfaces for seamless integration with algorithm development environments.

However, despite their versatility and utility, these simulators do not completely align with our specific requirements. Their primary focus is on visual perception and the control of robotic arm movements for pick-and-place tasks. Our research, in contrast, centers on higher-level coordination tasks that are better addressed through discrete event simulation. This approach enables us to abstractly model the simulation, avoiding the extensive development and testing demands associated with programming robotic arm behaviors as required in systems like Gazebo or Vrep.

SimPy [35], is a discrete event simulation framework in Python, previously utilized in Deep RL simulations within healthcare systems. While SimPy excels in modeling concurrent systems, it lacks the capacity to represent data-rich environments such as those found in pick-and-place settings. Consequently, we have developed our own Python-based discrete event simulator tailored for the pick-and-place conveyor belt. This custom simulator allows us to effectively develop and test our mutli agent RL approach in a more controlled and relevant environment.

In the simulator the conveyor belt is represented as a grid structure. The x-axis of this grid signifies the length of the belt, while the y-axis represents its width. Each cell within this grid stands for a possible position where a part might be located on the belt. The presence of a part in a cell is denoted by a one (1), and its absence is indicated by a zero (0).

Our experimental setup is based on that described by Huang et al [8] which was, in turn, based on their discussions with engineers at a robotic company. It consists of a multi-robot system featuring four identical robotic arms and their corresponding packaging boxes, alongside a continuously moving conveyor belt. Following Huang we choose a maximum arm speed of 7.0 meters per second. The conveyor speed can be varied but is normally at a consistent speed of 0.8 meters per second (as Huang). While different distributions may be used for the purpose [36] parts are fed onto the belt following a normal distribution, as with Huang. The part flow rate can be varied and ranges between 8 to 16 parts per second as with Huang.

The conveyor belt is represented as a grid measuring 64 cells in length and 8 cells in width. Each robot is restricted to picking up parts solely from its own workspace, and it has full access to every part within this area. After all robots complete their designated action A_t at a given time step t, the simulator provides a feedback reward R for that step, which is relayed back to the agent.

Figure 8 illustrates the architecture of the simulator, which operates with two primary control threads: the Agent and the Dispatcher. The Robot Handler, a crucial component, functions within the main thread of the Agent. Additionally, the simulator features shared data structures, namely the Task Queue and the Conveyor, which facilitate coordinated operations between these components.

The Dispatcher in the simulator serves two key roles: firstly, it manages the addition and removal of parts on the belt, using a Gaussian distribution for placement; secondly, it processes tasks from the job queue for the pick and place operations. The conveyor belt is modeled as a double-ended queue where parts are added and removed, effectively simulating the movement of items along the belt. The simulation's clock frequency, which influences the conveyor's operational pace, is set based on the selected speed of the belt.

The Robot Handler, conforming to the OpenAI Gym interface, acts as a virtual controller for the robots. It is

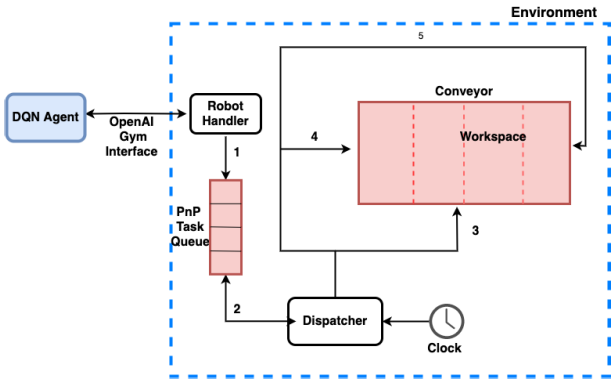

FIGURE 8. Simulator structure.

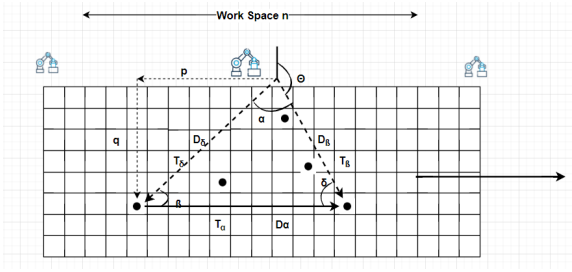

FIGURE 9. Scheduling discipline with relative velocity.

responsible for managing the scheduling disciplines and/or primitive actions for each robot. During each timestep of an episode, the Robot Handler determines - according to the scheduling discipline action- the most suitable part for each robot to pick, subsequently placing a corresponding job in the Task Queue with a calculated delay, allowing for the robot arm to intercept the part as it moves along the belt.

In parallel, the Dispatcher manages the task timeouts for picking operations from the Task Queue. This involves removing the part from the Conveyor and queuing a new job with the delay for the robot arm's return journey to the bin. Upon timeout of this job, the Dispatcher 'places' the part in the bin, indicated by an increment in the bin counter. After all placement jobs are completed, the Dispatcher signals the Robot Handler, thereby transferring control back to the DQN Agent.

2) SCHEDULING DISCIPLINE IMPLEMENTATION

Within the conveyor belt simulation, the movement of the belt is synchronized with the robots' pick-and-place actions. This necessitates precise calculation of the part's future position, where the robot arm is expected to intercept and pick it up. Given the known variables - the velocity of the belt, the initial position of the part, and the speed at which the robot arm moves - determining the future location of the part is achievable through the application of relative velocity principles [37].

Each robot in our simulation is positioned at the central point of its designated workspace. The process for calculating the future position of a part relative to each robot, as shown in Figure 9, involves the following steps:

1) Given the known horizontal distance p (distance from the robot to the part's initial location) and vertical distance q (distance between the robot and part vertically), we can compute the diagonal distance D_δ.
2) Using q and D_β, the angle β is determined based on the Sine Rule.
3) With the angle β, the speed of the robot arm, and the speed of the belt known, we apply the concept of relative velocity: sinβ / robot speed = sinα / belt speed. This allows us to determine α and subsequently calculate the remaining angles: α, β, δ, θ.
4) Utilizing relative velocity principles and the known angles along with $D\delta$, we calculate $D\alpha$ (the horizontal displacement from the part's original to future location) and T_δ (the time it takes for the part to reach its future position).
5) $D\delta$ is represented as the number of grid cells between the part's initial and future locations. If the future position falls outside the robot's workspace, a null value is returned. $T\delta$, expressed in seconds, is then converted into the number of clock ticks necessary for the part to reach the future position at the current belt speed. This information, along with other relevant details, is then added to the Task Queue as previously described.

VI. EXPERIMENTS

Figure 10 provides a overview of our approach. The top section of the figure shows an environment that complies with the OpenAI Gym standards, featuring a simulated hierarchical multi-robot system for pick-and-place tasks. Below this is the multi agent controllers which are implemented as simple deterministic NooP action selections. On the bottom is shown the meta-controller DQN.

Our PnP system is defined as a Dec-POMDP with the following characteristics:

1) Agent Set D: Comprises a collection of agents, denoted as $1, \ldots, n$.
2) The state space S includes the 'image' of the conveyor belt, represented as a two-dimensional tensor. This aligns with our chosen representation of the conveyor belt.
3) Joint Action Space A: Defined as $a_1x \ldots xa_n$, where each a_i corresponds to a specific scheduling discipline.
4) Joint Observation Space O: Constituted as $o_1x \ldots xo_n$, where each o_i is the two-dimensional tensor that depicts what agent d_i can see on the conveyor belt.
5) Joint Time Steps T: Defined as $t_1x \ldots xt_n$, where each t_i is the number of steps each robot spends from pick to place.
6) Reward Structure: The reward (R) aims to maximize the number of parts picked in the least time, varying with the type of task:
   a) *extrinsic reward*: Each agent receives a reward that is directly proportional to their efficiency in picking parts. For agent i who picks up n_i parts within t_i time, the reward R_i is computed as 1,

$$R_i = \begin{cases} n_i/t_i, & n_i > 0 \\ 0, & n_i = 0 \end{cases} \tag{1}$$

   b) *intrinsic reward*: The intrinsic reward is calculated by the number of occurrences of the NooP action i.e. for agent i with k_i occurrences of NooP, the intrinsic reward IR_i will be:

$$IR_i = k_i * -0.01 \tag{2}$$

The architecture of the DQN in our study includes three convolutional layers, followed by a fully-connected hidden layer, and concludes with a fully-connected output layer. The initial input layer processes a two-dimensional vector representing the length and width of the conveyor. The fully-connected hidden layer is composed of 512 rectified linear units, while the output layer is a fully-connected linear layer that outputs the action values for each possible combination of disciplines. For the optimization process, we employ the RMSprop optimizer, as ori.

As mentioned in the introduction, we selected HDQN as the principal algorithm for training the agents. Following a series of trial and error, we established that each training episode would consist of 5000 steps, with the overall training extending across 1200 episodes. This study maintains three constant parameters for the robots and the conveyor system. Drawing from the real robot arm movement speed detailed by Huang et al. [8], we established a baseline speed of 3m/s for our robot arms. Additionally, the conveyor belt speed was set at 0.8m/s, and the rate of part replacement was fixed at 10.

A. HDQN BASELINE PERFORMANCE

Training was conducted using a GPU server, outfitted with an Intel Xeon 5120 CPU featuring 56 cores and a clock speed of 2.20GHz, complemented by 250GB of RAM and four Nvidia Tesla V100-SMX2 GPUs, each with 32GB of memory. The software stack included Python 3.8 and PyTorch 2.0.1, which provided the necessary computational frameworks for our models. Code implementation was carried out in PyCharm Community Edition 2023. On average, the training process spanned approximately 24 hours, with nearly 60% of this duration dedicated to simulating the delays encountered in robot pick and place operations within the environment.

To evaluate the performance of the multi-agent systems using different methodologies, we analyzed the gripping rate G_t. This rate is calculated as the ratio of the total number of items picked N_p by all robots in a given episode to the total number of parts presented on the conveyor belt N_t in the same

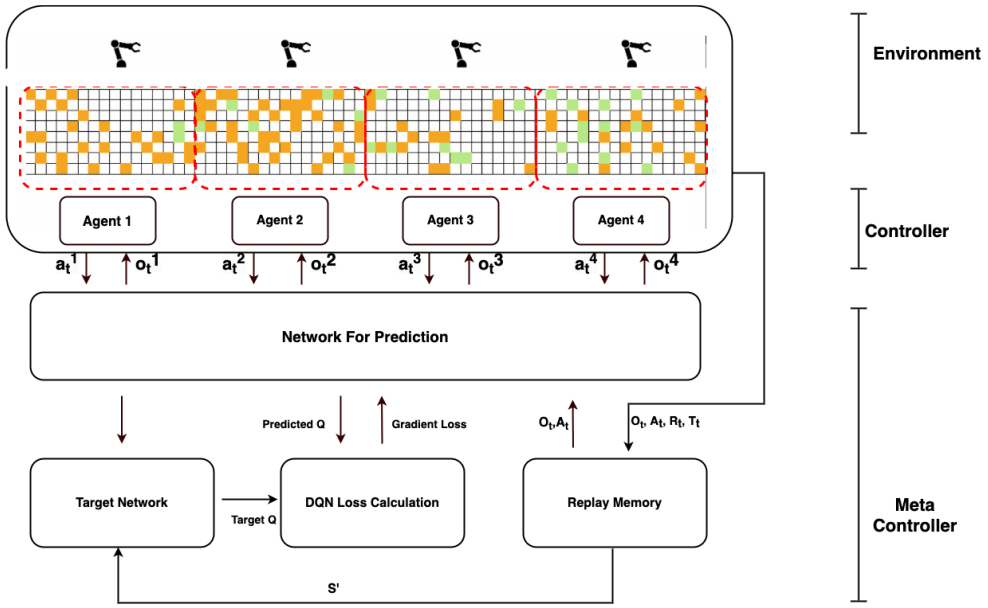

FIGURE 10. Hierarchical DQN multi agent approach.

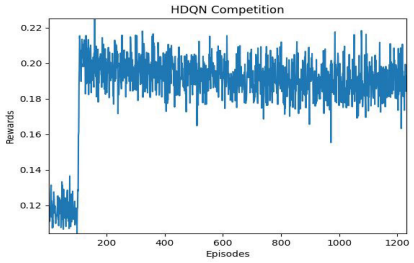

FIGURE 11. Average reward over 1200 episodes for PnP Task.

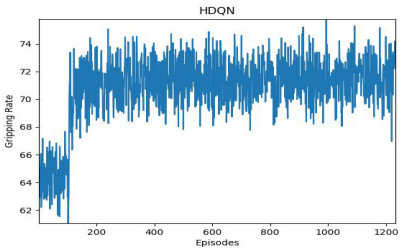

FIGURE 12. System gripping rate over 1200 episodes for PnP Task.

episode as

$$G_t = \frac{N_p}{N_t} \times 100 \tag{3}$$

ginally utilized in the seminal DQN study [3], to effectively minimize loss.

Regarding memory and training parameters, the replay memory is configured to hold 10,000 instances. The network update frequency for the target model is set at a standard rate of every 100 episodes. Additionally, we set a discount factor γ of 0.9 and learning rate α of 0.01, which serves to encourage the agents to identify the most advantageous action before the termination of each episode. The training of the agents commenced following the completion of 100 episodes. As illustrated in Figure 11, the data clearly shows that the average rewards per step for all agents stabilize at approximately 0.18. Further, as shown in Figure 12, it is observed that once the agents reach a stable state, HDQN consistently attains an average gripping rate of 72%.

Figure 13 displays the distribution of action occurrences across a span of roughly 6 million steps, excluding the NooP

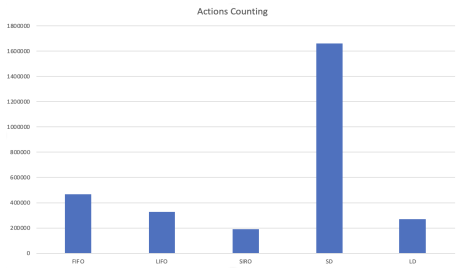

FIGURE 13. Actions counting distribution.

action. The data clearly indicates that the SD action is the most frequently choice among all robots, followed closely by FIFO as the second most common strategy.

B. COMPARISON OF GRIPPING RATES ACROSS DIFFERENT ARM SPEEDS

The influence of arm speed on pick-and-place efficiency is substantial. Echoing the findings of Bozma and Kalalıoğlu [11], we incorporated robots with diverse speed capabilities in our study, classifying them into slow, medium,

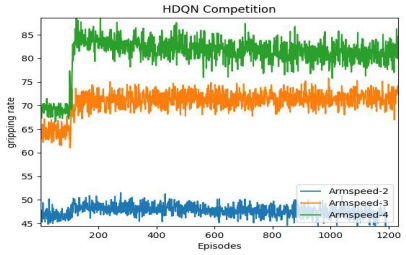

FIGURE 14. Gripping rate over 1200 episodes for Speed 2,3,4.

and fast categories with respective speeds of 2, 3, and 4m/s. To evaluate the gripping rate under these varied conditions, we utilized the HDQN algorithm. The data, as depicted in Figure 14, reveal a marked disparity in gripping rates across the different robot speed categories. The robot with the slow speed registers an average gripping rate near 47%, in contrast, the medium and fast-speed robots demonstrate higher gripping rates of approximately 72% and 82%, respectively. These results highlight the critical role of arm speed in optimizing pick-and-place task performance.

C. COMPARISON OF GRIPPING RATES ACROSS DIFFERENT PARAMETERS

The performance of the pick-and-place system was methodically examined through a comprehensive series of simulations, each differentiated by distinct parameters:

1) Robot speeds were classified into three tiers: speed 2 (slow), speed 4 (medium), and speed 6 (fast).
2) Product flow rates were tested at three levels—10, 12, and 15—determining the frequency of product availability.

In total, this established 27 scenarios for evaluation. The efficiency of each scenario was measured by the gripping rate, serving as the primary metric of performance. Figure 15 illustrates scenarios where the conveyor belt speed is set at 0.6m/s. Given the relatively slow movement of the belt, scenarios featuring robots with higher speed settings consistently achieve nearly 100% gripping rates across all part flow conditions. Conversely, robots operating at slow and medium speeds exhibit a decrease in gripping rate as the part flow rate increases. Figure 16 presents outcomes for a conveyor belt speed of 0.8 meters per second, considered a medium speed. In scenarios incorporating fast robots, the gripping rates remain impressively high, exceeding 90% across varying part flow rates. However, for robots designated as slow or medium in speed, there is an observable decrease in gripping efficiency—approximately 10%—each time the part flow rate increases a level. Figure 17, where the conveyor operates at a comparatively fast 1 meter per second, we observe that fast robots maintain a high gripping rate above 90% when the part flow is set at 10 and 12. Yet, when faced with a part flow of 15, their efficiency dips to 75%. Robots classified as slow or of medium speed also demonstrate a reduction in gripping rates by approximately 10% with each increment in part flow levels.

D. COMPARISON OF GRIPPING RATES ACROSS DIFFERENT SCENARIOS

Our simulator is capable of adapting to a wide array of RL configurations. It was initially focused on scenarios where a single agent controlled multiple robots, providing a window into the nuances of individual agent efficacy within multi-robot frameworks. Subsequently, we extended the simulator's capabilities to accommodate multi-agent synchronous reinforcement learning, thereby enabling a deeper investigation into the cooperative dynamics that emerge when multiple agents interact within a shared space.

Our methodology for evaluating the effectiveness of different configurations relied heavily on the gripping rate metric. This choice allowed us to accurately measure efficiency across various PnP tasks. For our analyses, we selected data that with optimal performance within each tested scenario. This included employing a cooperative environment for both the single-agent multi-robot system and the multi-agent synchronous (*t_all* termination scheme) framework, while a competitive environment was used to assess the multi-agent asynchronous (*t_continuous* termination scheme) model, with the setup parameters including a robot arm speed of 3, a belt speed of 0.8 meters per second, and a part flow rate of 10. As depicted in Figure 18, showing intriguing patterns of performance across the tested models. Notably, the gripping rate comparison indicates that while the single-agent and multi-agent synchronous systems exhibit similar efficiency levels, the single-agent model slightly edges out its multi-agent counterpart. We attribute this minor performance disparity to the complete observation capabilities of the single agent, in contrast to the partial observations inherent in the multi-agent setup.

A standout finding from our investigation is the distinct performance leap demonstrated by the HDQN in the asynchronous multi-agent environment. The HDQN model showcases a gripping rate approximately 10% higher than that achieved by the multi-agent synchronous system. This pronounced improvement underscores the HDQN's adeptness at navigating the intricacies of asynchronous operations, highlighting its potent capability for optimizing collaborative tasks in complex, multi-agent pick-and-place environments. We attribute this efficiency to the HDQN's use of a continuous termination scheme *t_continue*, which effectively removes idle times compared to the other two scenarioes as *t_all* scheme where robots wait for all tasks to conclude. This approach contributes to a swifter completion of tasks.

These findings not only validate the effectiveness of hierarchical learning strategies in enhancing task performance but also emphasize the critical role of observation scope and agent coordination in achieving optimal outcomes in robotic pick-and-place operations. Through this exploration of different RL configurations, our work sheds light on

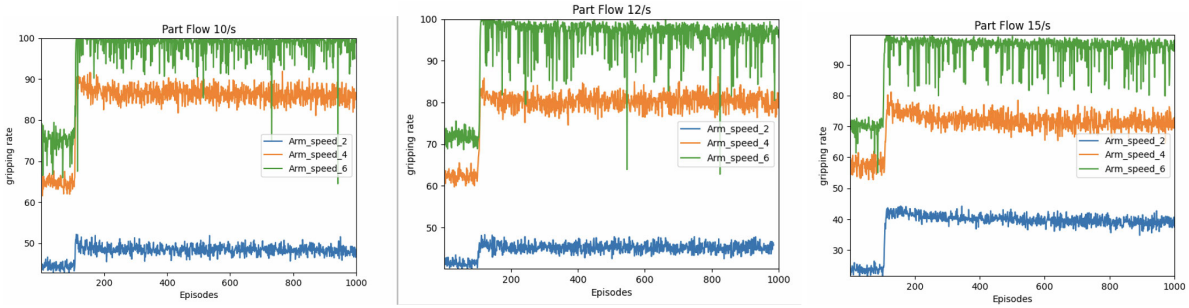

FIGURE 15. Belt speed 0.6m/s.

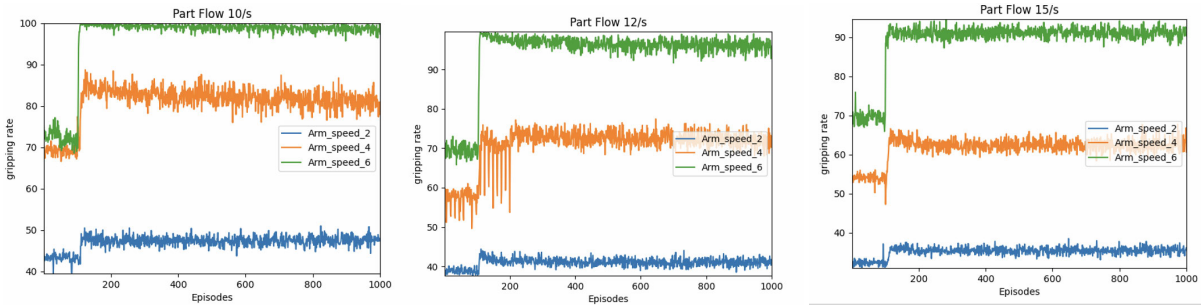

FIGURE 16. Belt speed 0.8m/s.

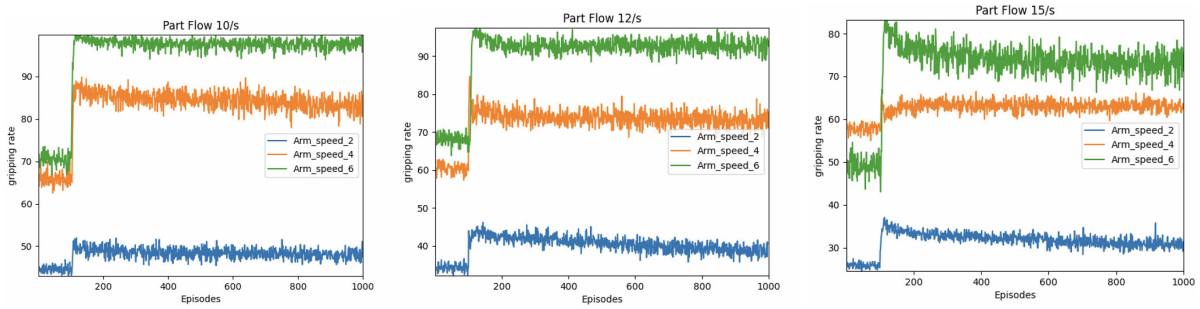

FIGURE 17. Belt speed 1.0m/s.

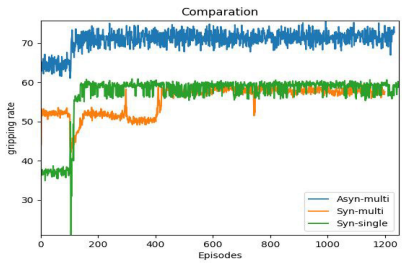

FIGURE 18. Gripping rate over 1200 episodes for single,multi,hierarchical.

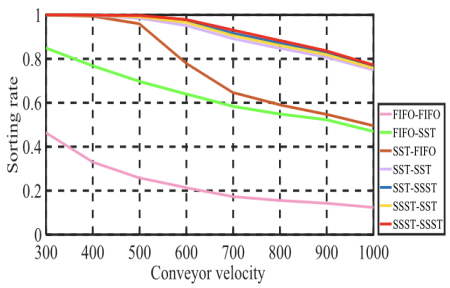

FIGURE 19. Gripping rate/sorting rate for different scheduling discipline and belt speed combinations - from [38].

the intricate interplay between agent autonomy, cooperation, and system architecture, offering valuable perspectives for future advancements in multi-agent reinforcement learning applications.

E. COMPARISON WITH OTHER RESULTS

As we believe this to be the first work to address the use of HRL for multi robot PnP, there is no other HRL work

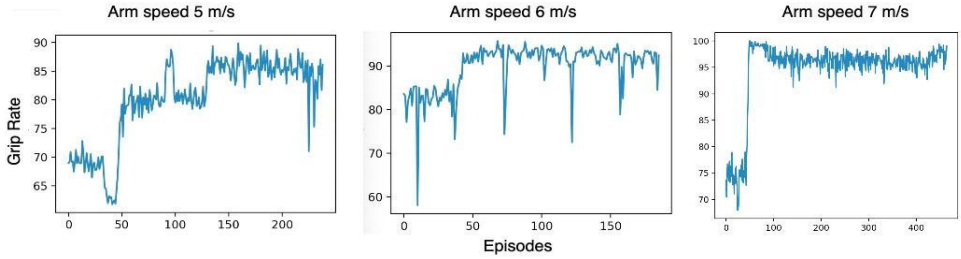

FIGURE 20. Gripping rate comparison with Huang et al. [8].

with which we can compare our results. There are however a number of comparable works. Foremost amongst those is Huang et al. [8] whihc has been described earlier. This work guided our own experimentation with respect to the conveyor belt sizing e.g. robot arm and belt speeds, work area size etc.

Yu et al. [38] also compare the use of scheduling disciplines for a two robot pick and place system. They use the FIFO and Shortest Sorting Time (SST) algorithms as well as a custom defined Second Shortest Sorting Time (SSST) algorithm and define the coordination problem as the optimisation of the choice of three scheduling disciplines across two robots so that the total number of candidate multi-robot coordination choices is nine (= 3*3). They then analyse the performance of different combinations of scheduling discipline at different belt speed and with a mean distance of 30 cm between each part on the belt- see Figure 19. No figures are give for arm speed, nor placement rate.

Zhou and Jiang [39] develop a PNP sequence planning algorithm for a two robots conveyor belt system that is optimal in a finite field of view. The computational efficiency is sufficiently fast for the actual picking process and the overall picking rateand maximum picking throughput are 10-20 % better than existing commonly used methods such as FIFO and SPT. Their simulation assumes a 1 meter/second belt speed. However no figures are given for gripping rate performance making direct comparison with our work difficult.

We compare results for arm speeds 6 and 7 ms/s with the optimal result reported by Huang across their various experiments - see Figure 20 - showing gripping rate for arm speeds 5,6 and 7 m/s, and Table 1 for a comparison of configurations and results. See also sorting rate reported by Chu - Figure 19. These results demonstrate performance of our system in terms of gripping rate is at least on a par with the state of the art as exemplified by Huang and Chu.

We believe that, combined with the gripping rate performance, HRL's hierarchical structure, efficient exploration-exploitation trade-offs, and flexibility in task decomposition offers greater advantages for coordinating multiple robots in conveyor belt pick-and-place tasks than the other approaches. By leveraging hierarchical learning, robots can effectively manage complexity, adapt to variations, and optimize performance in dynamic and uncertain environments, surpassing the capabilities of traditional greedy algorithms like GRASP.

TABLE 1. Comparison.

Name	Arm Speed	Gripping Rate	Belt Speed	Part Flow
Huang	Max 7.8	99.4 %	0.8m/s	15.8
Us	6	90 %(Avg.)	0.8m/s	15
Us	7	97.5 %(Avg.)	0.8m/s	15
Chu	-	90 %(Avg.)	0.8m/s	-

VII. CONCLUSION

In this work, we have introduced a novel approach to addressing the challenges of coordination in multi-robot pick-and-place systems through the application of hierarchical multi-agent deep reinforcement learning. We described the architecture of our pick and place simulator. This simulator was purpose-built to facilitate reinforcement learning applications in multi-robot PnP scenarios, ensuring compatibility and relevance to our HMADRL methodology. The design considerations taken into account reflect the need for high-fidelity simulations that accurately capture the challenges inherent in real-world PnP tasks.

In our explanation of the methodology, we have provided a thorough description of the HMADRL algorithms employed, with a focus on their hierarchical structure and the resulting advantages. We clarified the design decisions, underscoring their alignment with the imperatives of multi-agent coordination and the pursuit of enhanced learning efficacy.

We detailed the findings derived from implementing the HDQN algorithm, compare the performance metrics across a spectrum of robotic arm speeds. Additionally, we illustrated the predominance of specific actions among the four robots, as evidenced by the action frequency table. Moreover, we showed that the performance of our multi robot PnP system is comparable performance wise to the state of the art.

However, moving beyond laboratory settings and simulations into real-world manufacturing environments poses several challenges for the scalability of the HMADRL approach. Scalability concerns primarily revolve around the integration of HMADRL with existing industrial systems

including computer vision systems needs to generate the environment representation and the resulting computational complexity as the number of robots and tasks scales up. Practical deployment also requires robust error handling and recovery protocols to manage the inevitable uncertainties and operational anomalies in real-world environments. Adapting our HMADRL approach to include self-diagnostic and adaptive learning capabilities could significantly enhance its practical utility, making it more resilient and adaptive to the dynamic conditions of a manufacturing floor. In conclusion, while our HMADRL approach offers promising results in simulated environments, substantial work remains to adapt and scale these algorithms for widespread real-world application. These advancements will be critical in transitioning from theoretical models to practical, deployable systems that can genuinely transform industrial manufacturing operations.

REFERENCES

[1] A. Farinelli, N. Boscolo, E. Zanotto, and E. Pagello, "Advanced approaches for multi-robot coordination in logistic scenarios," *Robot. Auto. Syst.*, vol. 90, pp. 34–44, Apr. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092188901630447X

[2] Y. Rizk, M. Awad, and E. W. Tunstel, "Cooperative heterogeneous multi-robot systems: A survey," *ACM Comput. Surv.*, vol. 52, no. 2, pp. 1–31, Apr. 2019, doi: 10.1145/3303848.

[3] V. Mnih, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015. [Online]. Available: http://www.nature.com/articles/nature14236

[4] R. Mattone, L. Adduci, and A. Wolf, "Online scheduling algorithms for improving performance of pick-and-place operations on a moving conveyor belt," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 1998, pp. 2099–2105.

[5] M. Ghavamzadeh and S. Mahadevan, "Hierarchical multiagent reinforcement learning," *Auton. Agents Multi-Agent Syst.*, 2004.

[6] K. Rohanimanesh and S. Mahadevan, "Learning to take concurrent actions," in *Advances in Neural Information Processing Systems*, S. Becker, S. Thrun, and K. Obermayer, Eds. Cambridge, MA, USA: MIT Press, 2002.

[7] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dyn. Syst.*, vol. 13, no. 1, pp. 41–77, Jan. 2003. [Online]. Available: https://api.semanticscholar.org/CorpusID:386824

[8] Y. Huang, R. Chiba, T. Arai, T. Ueyama, and J. Ota, "Robust multi-robot coordination in pick-and-place tasks based on part-dispatching rules," *Robot. Auto. Syst.*, vol. 64, pp. 70–83, Feb. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889014002395

[9] G. Humbert, M. T. Pham, X. Brun, M. Guillemot, and D. Noterman, "Comparative analysis of pick & place strategies for a multi-robot application," in *Proc. IEEE 20th Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2015, pp. 1–8.

[10] R. Mattone, M. Divona, and A. Wolf, "Sorting of items on a moving conveyor belt. Part 2: Performance evaluation and optimization of pick-and-place operations," *Robot. Comput.-Integr. Manuf.*, vol. 16, nos. 2–3, pp. 81–90, Apr. 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0736584599000411

[11] H. I. Bozma and M. E. Kalalıoğlu, "Multirobot coordination in pick-and-place tasks on a moving conveyor," *Robot. Computer-Integrated Manuf.*, vol. 28, no. 4, pp. 530–538, Aug. 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0736584511001396

[12] S. Daoud, H. Chehade, F. Yalaoui, and L. Amodeo, "Efficient metaheuristics for pick and place robotic systems optimization," *J. Intell. Manuf.*, vol. 25, no. 1, pp. 27–41, Feb. 2014.

[13] D. Wang, H. Deng, and Z. Pan, "MRCDRL: Multi-robot coordination with deep reinforcement learning," *Neurocomputing*, vol. 406, pp. 68–76, Sep. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231220305932

[14] N. M. Gomes, F. N. Martins, J. Lima, and H. Wörtche, "Reinforcement learning for collaborative robots pick-and-place applications: A case study," *Automation*, vol. 3, no. 1, pp. 223–241, Mar. 2022. [Online]. Available: https://www.mdpi.com/2673-4052/3/1/11

[15] W. Liu, H. Niu, W. Pan, G. Herrmann, and J. Carrasco, "Sim-and-real reinforcement learning for manipulation: A consensus-based approach," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2023, pp. 3911–3917.

[16] Y. G. Kim, S. Lee, J. Son, H. Bae, and B. D. Chung, "Multi-agent system and reinforcement learning approach for distributed intelligence in a flexible smart manufacturing system," *J. Manuf. Syst.*, vol. 57, pp. 440–450, Oct. 2020.

[17] T. Zhou, D. Tang, H. Zhu, and Z. Zhang, "Multi-agent reinforcement learning for online scheduling in smart factories," *Robot. Comput.-Integr. Manuf.*, vol. 72, Dec. 2021, Art. no. 102202. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0736584521000855

[18] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction* (Adaptive Computation and Machine Learning). Cambridge, MA, USA: MIT Press, 1998.

[19] T. G. Dietterich, "Hierarchical reinforcement learning with the MAXQ value function decomposition," *J. Artif. Intell. Res.*, vol. 13, pp. 227–303, Nov. 1999.

[20] S. Luo, L. Zhang, and Y. Fan, "Dynamic multi-objective scheduling for flexible job shop by deep reinforcement learning," *Comput. Ind. Eng.*, vol. 159, Sep. 2021, Art. no. 107489. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360835221003934

[21] S. Luo, L. Zhang, and Y. Fan, "Real-time scheduling for dynamic partial-no-wait multiobjective flexible job shop by deep reinforcement learning," *IEEE Trans. Autom. Sci. Eng.*, vol. 19, no. 4, pp. 3020–3038, Oct. 2022.

[22] F. A. Oliehoek and C. Amato, *A Concise Introduction to Decentralized POMDPs* (SpringerBriefs in Intelligent Systems). Cham, Switzerland: Springer, 2016. [Online]. Available: http://link.springer.com/10.1007/978-3-319-28929-8

[23] L. Kraemer and B. Banerjee, "Multi-agent reinforcement learning as a rehearsal for decentralized planning," *Neurocomputing*, vol. 190, pp. 82–94, May 2016. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0925231216000783

[24] X. Lyu, Y. Xiao, B. Daley, and C. Amato, "Contrasting centralized and decentralized critics in multi-agent reinforcement learning," 2021, *arXiv:2102.04402*.

[25] R. E. Fikes, P. E. Hart, and N. J. Nilsson, "Learning and executing generalized robot plans," *Artif. Intell.*, vol. 3, pp. 251–288, Jan. 1972. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0004370272900513

[26] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artif. Intell.*, vol. 112, nos. 1–2, pp. 181–211, Aug. 1999. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370299000521

[27] H. Tang, J. Hao, T. Lv, Y. Chen, Z. Zhang, H. Jia, C. Ren, Y. Zheng, Z. Meng, C. Fan, and L. Wang, "Hierarchical deep multiagent reinforcement learning with temporal abstraction," 2018, *arXiv:1809.09332*.

[28] T. D. Kulkarni, K. R. Narasimhan, A. Saeedi, and J. B. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," 2016, *arXiv:1604.06057*.

[29] X. Lan, Y. Qiao, and B. Lee, "Coordination of a multi robot system for pick and place using reinforcement learning," in *Proc. 2nd Int. Conf. Comput. Autom. (CompAuto)*, Aug. 2022, pp. 87–92.

[30] X. Lan, Y. Qiao, and B. Lee, "Towards pick and place multi robot coordination using multi-agent deep reinforcement learning," in *Proc. 7th Int. Conf. Autom., Robot. Appl. (ICARA)*, Feb. 2021, pp. 85–89.

[31] A. Staranowicz and G. L. Mariottini, "A survey and comparison of commercial and open-source robotic simulator software," in *Proc. 4th Int. Conf. Pervasive Technol. Rel. Assistive Environ.* New York, NY, USA: Association for Computing Machinery, May 2011, pp. 1–8, doi: 10.1145/2141622.2141689.

[32] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2004, pp. 2149–2154.

[33] E. Rohmer, S. P. N. Singh, and M. Freese, "V-REP: A versatile and scalable robot simulation framework," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Nov. 2013, pp. 1321–1326.

[34] M. Quigley, "ROS: An open-source robot operating system," in *Proc. ICRA*, 2009.

[35] M. Allen and T. Monks, "Integrating deep reinforcement learning networks with health system simulations," 2020, *arXiv:2008.07434*.
[36] G. C. Causey, R. D. Quinn, and M. S. Branicky, "Testing and analysis of a flexible feeding system," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 1999, pp. 2564–2571.
[37] NumberSkill Math and Chemistry Tuition. (Sep. 2012). *Relative Velocity Example 4—Interception*. [Online]. Available: https://www.youtube.com/watch?v=Nl54obZhgtc
[38] C. Yu, X.-J. Liu, F. Qiao, and F. Xie, "Multi-robot coordination for high-speedpick-and-place tasks," in *Proc. IEEE Int. Conf. Robot. Biomimetics (ROBIO)*, Dec. 2017, pp. 1743–1750.
[39] M. Zhou and R. Jiang, "Optimal strategy for pick-and-place system with two robots," *J. Phys., Conf. Ser.*, vol. 2216, no. 1, Mar. 2022, Art. no. 012022, doi: 10.1088/1742-6596/2216/1/012022.

XI LAN received the M.S. degree from the Technological University of the Shannon, Athlone, Ireland, where she is currently pursuing the Ph.D. degree in computer science. Her research interests include the use of deep reinforcement learning for multi-robotic systems in industrial environments and their use in cybersecurity situational awareness.

YUANSONG QIAO (Member, IEEE) received the Ph.D. degree in computer applied technology from the Institute of Software, Chinese Academy of Sciences, Beijing, China, in 2008. He is currently a Senior Research Fellow with the Software Research Institute (SRI), Technological University of the Shannon, Ireland. He is also a Science Foundation Ireland (SFI) Funded Investigator with the SFI CONFIRM Smart Manufacturing Centre. His research interests include future internet architecture, blockchain systems, robotic control and coordination, and edge intelligence and computing. He is a member of the IEEE (Communications, Computer, and Robotics and Automation Societies; and Blockchain Community) and ACM (SIGCOMM and SIGMM).

BRIAN LEE received the B.E. degree in electronic engineering from University College Dublin, Ireland, in 1980, the M.S. degree in computer science from the University of Limerick, Ireland, in 1991, and the Ph.D. degree in computer science from the Trinity College Dublin, in 2004.

From 1980 to 2004, he was a Software and System Engineer with Ericsson Ltd., in the operation and maintenance of fixed and mobile telephone systems across a range of functional domains, technologies, and roles in both Europe and USA. He was the Research Manager of the Ericsson Network Management Research Group, Ireland, from 2004 to 2009, researching performance and service management in mobile networks. He joined Athlone Institute of Technology, in 2009, as the Director of the Software Research Institute, where his research interests include adaptable and autonomous networks and cybersecurity situational awareness. He has extensive experience in leading research activities in both industry and academia, including coordination of Horizon 2020 and Horizon Europe Research Actions.