Together We Rise: Optimizing Real-Time Multi-Robot Task Allocation using Coordinated Heterogeneous Plays

AAAI Track

Aritra Pal
TCS Research
Mumbai, India
p.aritra@tcs.com

Anandsingh Chauhan
TCS Research
Mumbai, India
anandsingh.chauhan@tcs.com

Mayank Baranwal
TCS Research
Mumbai, India
baranwal.mayank@tcs.com

ABSTRACT

Efficient task allocation among multiple robots is crucial for optimizing productivity in modern warehouses, particularly in response to the increasing demands of online order fulfillment. This paper addresses the real-time multi-robot task allocation (MRTA) problem in dynamic warehouse environments, where tasks emerge with specified start and end locations. The objective is to minimize both the total travel distance of robots and delays in task completion, while also considering practical constraints such as battery management and collision avoidance. We introduce `MRTAgent`, a dual-agent Reinforcement Learning (RL) framework inspired by self-play, designed to optimize task assignments and robot selection to ensure timely task execution. For safe navigation, a modified linear quadratic controller (LQR) approach is employed. To the best of our knowledge, `MRTAgent` is the first framework to address all critical aspects of practical MRTA problems while supporting continuous robot movements.

KEYWORDS

Multi-Robot Task Allocation; Self-Play; Reinforcement Learning; Linear Quadratic Controller

ACM Reference Format:
Aritra Pal, Anandsingh Chauhan, and Mayank Baranwal. 2025. Together We Rise: Optimizing Real-Time Multi-Robot Task Allocation using Coordinated Heterogeneous Plays: AAAI Track. In *Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Detroit, Michigan, USA, May 19 – 23, 2025*, IFAAMAS, 9 pages.

1 INTRODUCTION

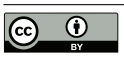
Cooperative multi-robot systems are increasingly being utilized across various domains, including transportation and logistics [7], search and rescue operations [25], environmental monitoring [19], precision agriculture [6], construction [27], and warehouse automation [10, 13]. These systems, characterized by the collaboration of multiple robots to achieve shared objectives, offer substantial benefits such as enhanced scalability & efficiency, and greater fault tolerance, making them indispensable in dynamic environments.

Intricacies of Warehouse Management: Automating warehouse operations with multi-robot systems presents a unique set of challenges, stemming from the complexities of spatial layouts, diverse task demands, varying robot capabilities, and the critical need for safe robotic navigation [4, 31]. These challenges can be broadly divided into three key objectives: (a) *Task Allocation*, (b) *Real-Time Robot Assignment*, and (c) *Path Planning*. While these objectives are interrelated, each introduces distinct sub-problems that must be resolved to achieve optimal warehouse performance. In a dynamic warehouse setting, real-time task generation is vital because tasks cannot be fully anticipated in advance. This unpredictability complicates the planning process, necessitating the prioritization of tasks based on their arrival times, required completion deadlines, and the need to minimize the total travel distances of robots while balancing the demands of ongoing tasks.

Moreover, the immediate and continuous allocation of robots to tasks demands seamless coordination, regardless of whether the robots are currently available or occupied. This emphasizes the need for real-time synchronization across multiple objectives, all while adhering to various physical and operational constraints. For example, effective path planning requires the creation of collision-free routes that navigate around static obstacles and account for the movements of other robots. Furthermore, it is crucial to consider the physical dynamics of the robots, such as acceleration, deceleration, and maneuverability-along with other practical considerations, such as their state-of-charge (SOC). These factors, often overlooked in existing warehouse management literature, are essential to the planning process.

The intricate interdependence of these challenges highlights the need for a sophisticated and adaptable multi-robot framework. Such a framework must systematically address the complexities of task allocation, real-time robot assignment, and path planning within the constantly changing environment of an automated warehouse. Neglecting the interconnected nature of these tasks often leads to sub-optimal performance and inefficiencies, undermining the overall effectiveness of warehouse operations.

Statement of Contributions: In this study, we present a self-play inspired novel framework, `MRTAgent`, which employs a bi-level RL strategy inspired to tackle the challenges of multi-task selection and multi-robot allocation. `MRTAgent` is designed to handle real-time task selection, dynamically allocate tasks to robots, and ensure safe navigation, all while accounting for critical constraints such as robot dynamics, charging needs, and specific task requirements. `MRTAgent` consists of three key components: (a) Task selection agent (Planner) to prioritize a task queued in the task buffer, (b) Robot selection agent (Executor) to allocate a robot to the recommended task, and (c) Navigator to plan collision-free trajectories of robots while adhering to physical and SOC constraints. The framework



Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Y. Vorobeychik, S. Das, A. Nowé (eds.), May 19 – 23, 2025, Detroit, Michigan, USA. © 2025 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

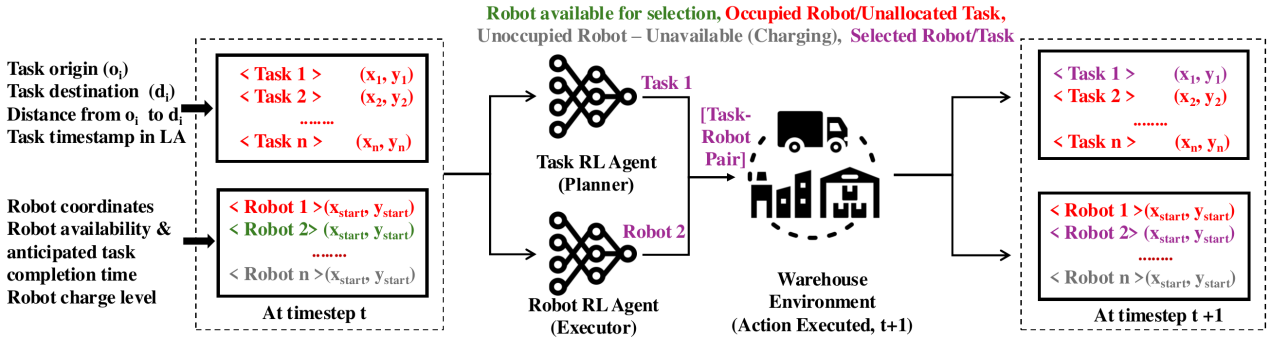


Figure 1: Self-play inspired bi-level reinforcement learning agent for assigning robots to tasks within a dynamic environment

is highlighted in Figure 1. The primary contributions of our work can be summarized as follows:

Coordinating RL agents for task and robot selections: Existing approaches for MRTA, such as in [2], primarily emphasize task selection with the assumption that the chosen task will always be assigned to an available (unoccupied) robot. Although this assumption simplifies the problem, it is highly sub-optimal. For instance, a robot might become available at a location far from the selected task's origin, while another robot in close proximity to the task origin may soon become available. We alleviate this by introducing heterogeneous RL agents, one each for task selection and robot allocation. These RL agents are trained concurrently in our framework in a self-play manner, gradually developing the coordination between the two.

Collision-free, physics-constrained multi-robot navigation: Practical robots do not operate in a grid world environment, allowing for sudden grid changes in any of the feasible directions. However, robots have some underlying dynamics and navigate continuously. Instead of using collision-free navigation algorithms applicable to grid worlds, we model these robots as double integrator systems and use linear quadratic regulators (LQRs) with artificial potential field (APF) for collision-free multi-robot navigation.

An end-to-end framework for MRTA: To the best of our knowledge, our approach is the first to consider multiple echelons of MRTA, requiring decision-making at each stage. We address every aspect of MRTA, including robot allocation even when robots are occupied, accounting for their SOC, underlying dynamics, and ensuring collision-free navigation under actuation constraints. We further validate our trained framework on datasets with distributional shifts, varying numbers of robots and tasks.

2 RELATED WORKS

Efficient MRTA and path planning are crucial for optimizing order fulfillment, resource management, and obstacle-free navigation in industrial environments such as automated warehouses and manufacturing plants. These processes ultimately enhance overall productivity, which has made MRTA a focal point of research over the past two decades. Research efforts in this area range from heuristic-driven approaches to contemporary learning-based methods [14]. Early work by [9] provided a comparative analysis of state-of-the-art (SOTA) multi-robot coordination strategies within specific domain contexts. Current MRTA research primarily focuses on two key elements: (a) model-driven optimization, as demonstrated by [30], and (b) communication-efficient decentralized algorithms, as seen in [2, 5].

The problem can also be framed as a multi-agent pickup and delivery (MAPD) challenge, which has been studied through both distributed and centralized approaches [17, 18, 26, 32]. However, most existing research in this area has concentrated on offline MAPD, whereas our approach emphasizes learning-based methods for online task allocation. This focus is driven by the need for reliable solutions in dynamic environments, where continually solving optimization problems can be computationally intensive.

Recent advancements in reinforcement learning (RL) for solving complex dynamic challenges have led to a trend toward learning-based approaches for managing warehouse complexities [1, 2, 33]. These learning strategies address various aspects of end-to-end warehouse management. For instance, [33] proposed a Q-learning framework to generate collision-free, secure paths for multi-robot systems. Conversely, the RL frameworks in [1, 2] focus on optimal task selection but neglect task-to-robot assignment, assuming constant robot availability post-selection. Additionally, these works leverage A^* coupled with optimal reciprocal collision avoidance [3] for collision-free navigation at the low-level path planning stage.

However, as previously discussed, most SOTA learning-based warehouse management approaches, including those mentioned, overlook a key benefit of RL: the ability to integrate multiple levels of warehouse management, and consideration of robots' constraints during the training phase of the RL agent. For instance, the learned policy in [1, 2] is limited in its applicability to realistic warehouse scenarios due to the neglect of constraints related to robot availability and SOC. Moreover, these approaches often aim for a seamless sequential flow of tasks without considering their generation times. Similarly, [22] utilizes a cooperative multi-agent RL framework under the assumption that robots never collide. Much of the prior work also neglects the complexities of robot dynamics in MRTA for warehouse settings, often simplifying robots to point objects or solving problems in basic square grid environments, without

accounting for robot acceleration, deceleration, and collision risks during path planning [21].

In our study, we address these gaps by incorporating task arrival times to ensure timely task execution while considering practical factors such as robot availability, SOC, robot dynamics, and collision-free path generation. Our framework also demonstrates robust performance under distribution shifts and with variable-sized fleets. Our MRTAgent addresses these limitations by integrating robot dynamics into the navigation planning process using a linear quadratic regulator (LQR)-based navigation path algorithm within the RL agent framework. This ensures that path planning is both effective and collision-free. Additionally, we design a reward structure that balances prompt task allocation with the shortest possible execution duration for allocated tasks. This approach ensures competitive runtime during the deployment phase, facilitating real-time task selection and robot allocation, and collision-free navigation considering physical dynamics through proposed MRTAgent.

3 PRELIMINARIES

3.1 States, Actions and Rewards

The problem of MRTA can be modeled as a Markov Decision Process (MDP) [24]. An MDP is denoted by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}_{\mathcal{A}}, r \rangle$, where $\mathcal{S}$ and $\mathcal{A}$ represent the finite sets of states and actions, respectively. For any $s, s' \in S$, the transition probability from state s to state s' under the action $a \in A$ is denoted by $p_a(s, s') \in P_A$. Finally, the step reward associated with each state-action pair (s, a) is depicted by $r(s, a)$. Below, we summarize the set of all possible states, actions, and rewards in the context of the MRTAgent validated within warehouse environment settings.

States: At each time step, the warehouse environment is characterized by a comprehensive state that includes detailed information about both tasks and robots. Incoming tasks are immediately stored in a buffer, forming a limited-size look-ahead (LA) queue in a First-In-First-Out(FIFO) manner. The Tasks RL agent, referred to as the *Planner*, is trained to optimally select tasks from this queue based on the current state of the environment. Simultaneously, the Robot RL agent, referred to as the *Executor*, responsible for executing tasks, selects most suitable robots to complete them.

The states of the *Planner* and *Executor* consist of the features related to tasks in the LA (denoted by $\mathcal{P}$) as well as the set of robots ($\mathcal{R}$). Each agent's state, denoted as $s_t \in \mathcal{S}$ at time-step t, encompasses the following components: (a) origin coordinates of tasks $\{o_i\}$, (b) destination coordinates of tasks $\{d_i\}$, (c) euclidean distance information between task origin and destination $\{k_i\}$, (d) timestamp of task appearance in the LA queue $\{l_i\}$, (e) robot coordinates $\{p_j\}$, (f) robot availability and the anticipated time for ongoing task completion $\{r_j\}$, (g) robot charge percentage $\{c_j\}$. Features (a)-(d) are task-specific and collectively have a dimensionality of 6 for each task. Conversely, features (e)-(g) are linked to robot-specific attributes (dim. = 4 for each robot).

Actions: The MRTAgent consists self-play inspired bi-level RL agent; planner and executor. The goal of the *planner* is to enhance task assignment, thereby minimizing total operational time. The planner's actions involve systematically selecting tasks from the queue while the executor focuses on robot allocation, with the objective of optimally assigning robots to the selected tasks to reduce overall operational expenses, and task execution delays. Thus, the actions executed in the environment consist of the selected task and its corresponding robot pair.

Rewards: The step reward attributed to a task-action pair encompasses two distinct components. The initial component is calculated based on the time it takes for the robot to travel from its current position to the task's starting point, termed *travel time to origin* (TRTO). The second component is the time gap between task arrival in the LA and the robot's initiation of execution, denoted as the *total time gap for the task* (TTGT). Let (x_{o_i}, y_{o_i}) and (x_{d_i}, y_{d_i}) represent the origin and destination coordinates of the i^{th}-indexed task. Similarly, (x_{r_j}, y_{r_j}) indicates the current position of robot j, which can vary based on whether the robot is idle, performing tasks, or at a charging location for recharging if needed. Additionally, we introduce t_{stamp_i} and t_{exec_i} to denote the time when task i appears in the task allocation (referred to as LA) and the time at which its execution begins, respectively. In each decision-making step, we use the variable allotT to signify the index of the selected task which then gets assigned to a robot selR. Furthermore, we utilize the function $d[(x_a, y_a), (x_b, y_b)]$ to calculate the distance between two points (x_a, y_a) and (x_b, y_b) within our system. The step reward for training the PPO agent is as follows:

$$\begin{aligned} R_{\text{step}} \quad = \quad & -d[(x_{r_{\text{selR}}}, y_{r_{\text{selR}}}), (x_{o_{\text{allotT}}}, y_{o_{\text{allotT}}})] \\ & -\alpha * (t_{\text{exec}_{\text{allotT}}} - t_{\text{stamp}_{\text{allotT}}}) \end{aligned} \tag{1}$$

The coefficient α represents positive constant. The first term in (1) corresponds to TRTO, while the second term is associated with TTGT. This reflects the principle that tasks should not remain unattended for too long.

4 OUR APPROACH

In this section, we introduce MRTAgent, a novel self-play-inspired bi-level RL framework (see Figure 2) designed to optimize MRTA for various industrial tasks. The MRTAgent is validated in warehouse environments scenario and comprises two RL agents: (a) Planner, and (b) Executor. Tasks are generated in real-time and initially populate a main task buffer. The planner has access to a small LA queue of tasks. When a task from LA queue is assigned to a robot for execution, a new task from the buffer is moved into the LA queue, making it available for selection by the Planner. If no new tasks are available in the buffer to replenish the LA queue, task with the longest duration in the LA queue is duplicated to maintain a consistent queue length, thereby increasing its likelihood of being selected by the Planner in subsequent steps. In the exceptional scenario where both the LA queue and the task buffer are empty, MRTAgent waits for a task to appear in the environment. The action selection process operates across three hierarchical levels:

Task Selection: At each instant t, the Planner selects a task from the LA queue for assignment to one of the robots, guided by the current state of the environment.

Robot Allocation: Upon task selection, the Executor identifies the most suitable robot for task execution. This decision considers the positions of all robots after completing their current assignments, availability and SOC. Notably, the Executor does not wait for robots to become available before making robot allocations, as the state

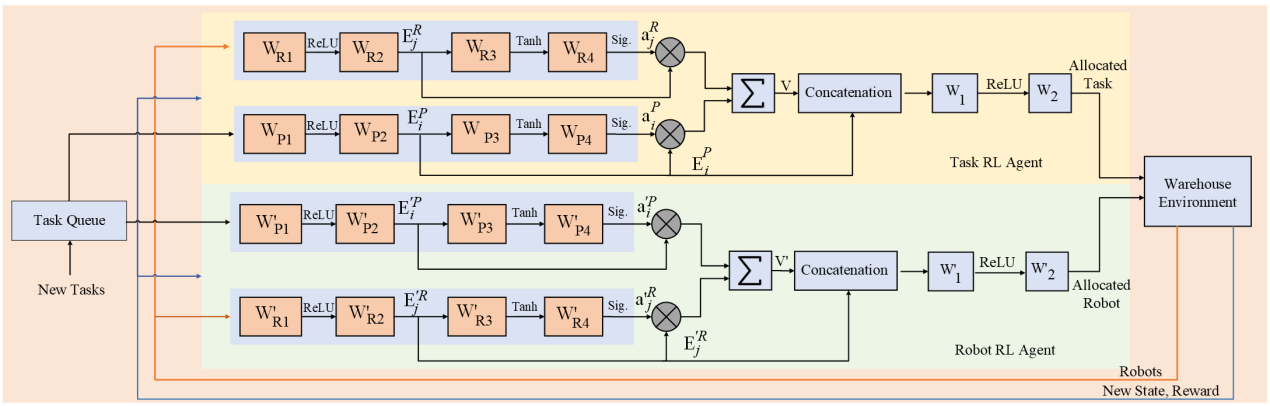


Figure 2: Self-Play Motivated Bi-level RL Agent Architecture with trainable weights denoted by $\{W_k, W'_k\}$

information includes time markers indicating when robots will be free.

Navigation: To guide the robots' movement, a linear quadratic regulator (LQR) based controller, augmented with potential functions for coordinated collision avoidance, considering all its dynamics is employed. This algorithm directs the robot from its current position to the task's starting point and subsequently to the destination, ensuring collision avoidance with moving obstacles.

4.1 MRTAgent framework

Both Planner and Executor agents of the `MRTAgent` utilize Proximal Policy Optimization (PPO) algorithm [28], with their architectures depicted in Figure 2. Both the agents are trained using a self-play inspired strategy, where one agent is actively trained while the other operates in evaluation mode, alternating every 40 episodes. This concurrent training framework ensures coordination between the two agents, facilitating efficient convergence and optimization of both the task selection and robot allocation processes.

4.2 Neural Network Architecture and Training

Planner and executor both employs a similar novel PPO-based framework to facilitate online task selection and robot allocation. This model architecture inspired from [1] is distinctly structured into three core segments (see Figure 2). The first segment involves the procedure of feature extraction, particularly focusing on attributes related to robots and tasks. The embedding for robot attributes is created using a sequence of four linear layers of dimensions [4, 16, 16, 1]. Concurrently, embeddings related to task attributes are generated using a similar sequence of four linear layers of dimensions [6, 16, 16, 1].

Let F_j^R and F_i^P represent the feature vectors for the robot $j \in \mathcal{R}$ and the ith-task for the planner and executor, then the associated embeddings are defined as:

$$\begin{aligned} E_i^P &= W_{P2} * \text{ReLU}(W_{P1} * F_i^P) \\ E_j'^R &= W'_{R2} * \text{ReLU}(W'_{R1} * F_j^R) \end{aligned}$$

The second module transforms the extracted embeddings by concatenating the embeddings of robots and tasks for both planner and executor. Subsequently, the concatenated feature is channelled through a linear layer consisting 48 input neurons and 8 output neurons with ReLU activation.

$$\begin{aligned} a_i^P &= \text{Sigmoid}(W_{P4} * \text{Tanh}(W_{P3} * E_i^P)) \\ a_j'^R &= \text{Sigmoid}(W'_{R4} * \text{Tanh}(W'_{R3} * E_j'^R)) \\ \pi^{\text{planner}} &= \left(\sum E_j^R * a_j^R, \sum E_i^P * a_i^P, E_i^P\right) \\ \pi^{\text{executor}} &= \left(\sum E_j'^R * a_j'^R, \sum E_i'^P * a_i'^P, E_j'^R\right) \end{aligned}$$

Final layer comprising a linear layer, this element operates with 8 input neurons and 1 output neuron.

$$\begin{aligned} Task^{Allocated} &= \text{Categorical}\left(W_2 * \text{ReLU}(W_1 * \pi^{\text{planner}})\right) \\ Robot^{Allocated} &= \text{Categorical}\left(W'_2 * \text{ReLU}(W'_1 * \pi^{\text{executor}})\right) \end{aligned}$$

4.3 Multi-Robot Navigation Algorithm: LQR with Artificial Potential Field

Linear Quadratic Regulator (LQR) is an optimal control strategy used to determine the control inputs that minimize a quadratic cost function over time for a linear system. It is widely used in control engineering to stabilize systems and optimize performance by balancing different aspects of system behavior, like minimizing energy use, overshoot, or settling time [16]. When modeling robot dynamics, a common simplification is to treat the robot as a double-integrator system. This model is particularly relevant for systems where the primary concern is controlling the position and velocity of the robot [29]. Consequently, state of robot i is typically represented by its position $p_i(t) = (x_i(t), y_i(t))$ and velocity $v_i(t)$, with the equations of motion represented as:

$$\dot{p}_i(t) = v_i(t), \quad \dot{v}_i(t) = u_i(t).$$

Here $u_i(t) = (u_{i_x}(t), u_{i_y}(t))$ is the control input, which directly influences the acceleration. Here, complete state of the i^{th}-robot is represented as: $z_i(t) = (p_i(t), v_i(t))$. Applying LQR to this model involves designing a controller that minimizes deviations from a desired trajectory while controlling the velocity and ensuring smooth acceleration.

Algorithm 1: MRTAgent

Initialize Planner & Executor policy parameters θ_{P0}, θ_{E0} and value function parameters ϕ_{P0}, ϕ_{E0} respectively.
for $itr = 0, 1, 2, \cdots K_1$ **do**
  for *episodes* $k = 0, 1, 2, \cdots K_2$ **do**
    Step t=0
    State $s_t := \{(o_i, d_i, k_i, l_i)_{\forall i \in \mathcal{P}}, (p_j, r_j, c_j)_{\forall j \in \mathcal{R}}\}$
    Each robot $j \in \mathcal{R}$ is executing a task (o_j, d_j)
    while *True* **do**
      Update $p_j \; \forall j \in \mathcal{R}$ using LQR with APF
      if $p_j == d_j$ **then**
        if $c_j <$ *threshold for some* $j \in \mathcal{R}$ **then**
          Robot j navigates to nearest available charging dock for recharging.
        end if
        if *itr is even* **then**
          run Planner policy $\pi_{Pk} = \pi(\theta_{Pk})$ in the environment to select a task $i \in \mathcal{P}$.
          Executor policy takes state s_t as input and allots a robot $j \in \mathcal{R}$.
        else
          Planner policy takes state s_t as input and selects a task $i \in \mathcal{P}$.
          run Executor policy $\pi_{Ek} = \pi(\theta_{Ek})$ to allocate a robot $j \in \mathcal{R}$.
        end if
      end if
      $t \leftarrow t + 1$
    end while
    if *itr is even* **then**
      Collect set of trajectories $\mathcal{D}_{Pk}$
      Compute rewards-to-go $\hat{R_{Pt}}$
      Compute advantage estimates $\hat{A_{Pt}}$ based on the current value function $V_{\phi_{Pk}}$
      Update policy by maximizing PPO-Clip obj.:

$$\theta_{Pk+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_{Pk}|T} \sum_{\tau \in \mathcal{D}_{Pk}} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{Pk}}(a_t|s_t)} A^{\pi_{\theta_{Pk}}}(s_t, a_t), g(\epsilon, A^{\pi_{\theta_{Pk}}}(s_t, a_t))\right)$$

      via stochastic gradient ascent with Adam;
      Fit value function by regression on MSE:

$$\phi_{Pk+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_{Pk}|T} \sum_{\tau \in \mathcal{D}_{Pk}} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R_{Pt}}\right)^2$$

      via gradient descent
    else
      Update θ_{Ek} & ϕ_{Ek} as above
    end if
  end for
end for

To extend LQR for multi-robot path planning and collision avoidance, the state and control matrices are expanded to accommodate multiple robots. Each robot is controlled using an individual LQR controller within this collective framework, allowing for the independent regulation of position and velocity while minimizing a global cost function that balances state deviations and control effort. For collision avoidance, an artificial potential field (APF) method is employed [23]. This approach introduces repulsive forces between robots when they come into close proximity, ensuring safe inter-robot distances. These repulsive forces are integrated into the control law, enabling the robots to avoid collisions while continuing to track their desired trajectories. This combined LQR-APF approach provides an effective solution for coordinating multiple robots in dynamic environments. More formally, we consider a system with N robots, each modeled as a double-integrator in 2D space. The dynamics of the overall system is given by:

$$\dot{\mathbf{z}}(t) = A_{\text{multi}}\mathbf{z}(t) + B_{\text{multi}}\mathbf{u}(t), \quad \text{with}$$

$$\mathbf{z}(t) = \begin{bmatrix} p_1(t) \\ v_1(t) \\ \vdots \\ p_N(t) \\ v_N(t) \end{bmatrix}, \mathbf{u}(t) = \begin{bmatrix} u_1(t) \\ \vdots \\ u_N(t) \end{bmatrix}$$

$$A_{\text{multi}} = I_N \otimes \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, B_{\text{multi}} = I_N \otimes \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix},$$

where I_N is the identity matrix of size N, and $\otimes$ denotes the Kronecker product. The objective of LQR is to minimize the following quadratic cost function:

$$J = \int_0^\infty (\mathbf{z}(t)^\intercal Q \mathbf{z}(t) + \mathbf{u}(t)^\intercal R \mathbf{u}(t))dt,$$

where Q and R are weighting matrices that penalize deviations from the desired state and control effort, respectively. The optimal control input that minimizes the cost function is given by $\mathbf{u}(t) = -K(\mathbf{z}(t) - \mathbf{z}_{\text{des}})$, where K is the LQR gain matrix, computed as $K = R^{-1}B_{\text{multi}}^\intercal P$, and $\mathbf{z}_{\text{des}}$ represents the target positions and velocities. Here P is the solution to the continuous-time algebraic Riccati equation (CARE):

$$A_{\text{multi}}^\intercal P + PA_{\text{multi}} - PB_{\text{multi}}R^{-1}B_{\text{multi}}^\intercal P + Q = 0.$$

In addition, we employ APF method that introduces a potential field around each robot that creates repulsive forces to avoid collisions. The repulsive potential between robots i and j is given by:

$$U_{\text{rep},ij}(z_i, z_j) = \begin{cases} \frac{1}{2}k_{\text{rep}}\left(\frac{1}{d_{ij}} - \frac{1}{d_{\min}}\right)^2 & \text{if} \quad d_{ij} < d_{\min} \\ 0 & \text{if} \quad d_{ij} \geq d_{\min} \end{cases}$$

where k_{rep} is a positive constant, $d_{ij} = \|p_i - p_j\|$ is the Euclidean distance between robots i and j, and $d_{\min}$ is the minimum allowable distance between robots. The final control input for each robot, incorporating both LQR control and APF-based collision avoidance, is given by:

$$u_i(t) = -K_i(z_i(t) - z_{\text{des},i}) - \nabla_{p_i}U_i.$$

This ensures that the robots not only follow their intended paths but also avoid collisions by adjusting their trajectories dynamically in response to nearby robots.

5 EXPERIMENTS

5.1 Datasets and Experimental Setup

The MRTAgent framework has been validated within warehouse environment settings, operating under several key parameters to ensure efficient functioning. Due to the absence of publicly accessible real-world datasets for similar problem scenarios, synthetic data has been employed to evaluate our MRTAgent framework. Each episode spans τ time units, and the synthetic datasets used for both training and evaluation are structured as square-shaped 2D continuous spaces ranging from $[0, 64] \times [0, 64]$, with task origins, destinations, and robot locations confined within this range.
In our experiments, task concentration varies to reflect specific periods of the day when the majority of tasks accumulate. Accordingly, datasets are generated using Gaussian distributions with varying means and standard deviations.The task's starting and ending coordinates, along with the task generation times, are provided to the planner through a limited-size LA. This MRTAgent framework is evaluated under two configurations:

- Normally distributed task arrival times
- Uniformly distributed task arrival times.

For each configuration, datasets containing 500 tasks per episode are generated according to the corresponding distributions. For instance, in the case of normally distributed tasks, the task generation times adhere to $\mathcal{N}(600, 50)$, where 600 represents the mean task generation time. For training, we considered a fleet of 10 robots, with the LA window length fixed at 5. The robots' charging threshold is set at 30%, meaning robots with a SOC below 30% must dock for recharging before resuming task execution. The steady charging rate is calibrated to be 16× the discharging rate.

The planner and executor agents are implemented using the PyTorch library in Python 3.8, with an Adam optimizer [15], a discount factor of 0.99, a lambda value of 0.95, a learning rate of 0.0003, an entropy coefficient of 0.001, a value function coefficient of 0.0002, and a batch size of 32. The policy networks for both the planner and executor are trained using the cross-entropy loss function, while the value networks are fine-tuned using the mean squared error loss metric.

5.2 Baselines

To the best of our knowledge, no existing work in the literature concurrently addresses multiple aspects of MRTA simultaneously. In light of the absence of established approaches, we propose two suitable baselines.
Brute-force optimal (BFO) : In this approach, all task-robot pairs (within the LA) undergo an exhaustive brute-force evaluation of time duration required for task execution by the robots, determined using standard Euclidean distance. The algorithm then selects the robot-action pair that minimizes this time duration. While brute-force optimal approach represents a locally optimal solution, the exhaustive evaluation significantly amplifies the run-time, posing practical challenges. Despite, this baseline is frequently adopted in the literature as a reference point for evaluating decoupled task allocation and navigation methodologies [2, 20].
FIFO : The FIFO baseline employs a dual-tiered decision framework. The initial allocation involves selecting the task that entered the LA queue first, aiming to reduce pending tasks within the queue and allocating the robot that can complete it earliest to minimize TTGT [11, 12]. Due to its simplicity, the FIFO approach requires the least execution time among all the considered approaches.

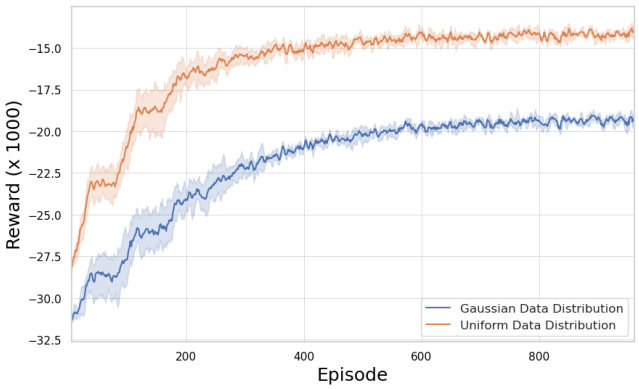


Figure 3: Learning curves of the MRTAgent

5.3 Training Details

The MRTAgent is trained through a self-play approach. Initially, the planner undergoes training for 40 episodes, while the executor remains in evaluation mode. After every 40 episodes, the roles of the planner and executor are reversed: the planner is switched to evaluation mode, and the executor is trained. This cycle is repeated 24 times, leading to a total of 960 training episodes. Each episode consists of 505 tasks, with 10 robots in the environment and a LA length of 5. Both the Planner and Executor agents are trained using the PPO algorithm [28]. The model is implemented using the PyTorch library in Python 3.8. Key hyperparameters include the Adam optimizer [15], a discount factor of 0.99, a lambda value of 0.95, a learning rate of 0.0003, an entropy coefficient of 0.001, a value function coefficient of 0.0002, and a batch size of 32. The policy network for both agents is trained with the cross-entropy loss function, while the value network is optimized using mean squared error (MSE) loss.

5.4 Experiments and Results

We now present a thorough comparative analysis of our proposed learning-based framework, MRTAgent, against baseline methods, namely the BFO approach and the FIFO strategy. Our experiments are designed to include scenarios that incorporate charging considerations, thereby addressing realistic operational challenges. The results consistently demonstrate the superiority of MRTAgent over the baseline methods.

Figure 3 presents the average training curve for the bi-level RL agents. This average is derived from four independent runs with different random seeds. The RL model is trained over 960 episodes, each comprising 505 tasks with 10 robots in the environment, and a look-ahead (LA) length of 5. The training process utilizes the PPO-based RL algorithm within the PFRL framework [8]. To simulate real-world task profiles, the training procedure employs task lists

Table 1: Cost ($\times 10^3$) evaluation on test dataset with 5 tasks in LA, 10 robots & 505 episodic tasks (the lower the better)

Test distibution	Uniform distribution data			Gaussian distribution data		
	MRTAgent	BFO	FIFO	MRTAgent	BFO	FIFO
Similar	**13.49 ± 0.44**	16.03	16.69	**18.23 ± 0.47**	18.60	19.08
	14.85 ± 0.15	17.72	18.35	**18.66 ± 0.06**	19.06	19.43
	13.72 ± 0.16	15.75	16.49	**19.10 ± 0.01**	19.77	20.41
	14.83 ± 0.26	16.97	17.21	**18.40 ± 0.01**	19.83	20.67
	14.47 ± 0.18	15.95	16.40	**18.67 ± 0.09**	19.04	19.73
Totally Different	**18.89 ± 0.12**	19.06	19.43	**15.84 ± 0.56**	17.72	18.35
	19.50 ± 0.67	20.15	20.56	**14.84 ± 0.44**	17.25	17.83
	18.88 ± 0.13	19.04	19.73	**13.98 ± 0.43**	15.75	16.49
	18.77 ± 0.24	19.58	19.65	**14.37 ± 0.38**	16.92	17.31
	18.53 ± 0.42	20.43	21.03	**14.82 ± 0.38**	15.95	16.40

Table 2: Cost ($\times 10^3$) comparison on Gaussian dist. dataset with 5 tasks in LA, 10 robots & 505 tasks/episode

Avg TRTO			Avg TTGT		
MRTAgent	BFO	FIFO	MRTAgent	BFO	FIFO
8.44	8.48	8.63	**11.38**	11.68	11.93
8.03	8.59	8.93	**11.09**	11.18	11.48
7.59	8.62	9.11	**10.79**	11.20	11.56
7.70	8.08	8.30	**10.90**	10.96	11.43
8.20	8.49	9.22	**10.70**	10.81	11.25

Table 3: Cost ($\times 10^3$) evaluation on Gaussian dist. dataset with 5 tasks in LA, 30 robots & 505 episodic tasks

	MRTAgent	BFO	FIFO
Gaussian dist. data	**8.41**	8.64	8.81
	8.93	9.27	9.63
	8.19	8.34	9.05
	7.49	7.63	8.53
	8.19	8.43	9.61

Table 4: Cost ($\times 10^3$) evaluation on Gaussian dist. dataset with 5 tasks in LA, 25 robots & 505 episodic tasks

	MRTAgent	BFO	FIFO
Gaussian dist. data	**9.25**	9.56	9.76
	10.05	10.37	10.44
	9.34	9.57	9.92
	8.95	9.42	9.69
	8.70	9.13	9.32

generated randomly from a Gaussian distribution, alongside uniformly sampled task lists. The reward plots indicate that MRTAgent achieves stable convergence towards an optimal policy. The improvements in rewards from their initial values suggest effective task selection and robot allocation, leading to reduced task waiting times within the LA window. It is worth noting that the RL agent is periodically trained with different random task lists, which prevents it from simply memorizing the performance on a specific task list. Instead, the agent learns to adapt to various scenarios, which explains the minor fluctuations in rewards across episodes as the agent refines its policy.

Table 5: Cost ($\times 10^3$) evaluation on Gaussian dist. dataset with 5 tasks in LA, 10 robots & 2005 episodic tasks

	MRTAgent	BFO	FIFO
Gaussian dist. data	**223.44**	237.2	279.38
	215.81	229.54	274.59
	218.44	230.08	277.88
	217.31	232.45	276.56
	214.67	227.45	274.31

After the training phase, the model's performance is evaluated on different test datasets. For the model trained on task lists following a specific Gaussian distribution, the evaluation is performed on two distinct datasets:

- Instances that are similar to the training data with the same mean and variance specifically with $\mathcal{N}(600, 50)$ distributed data,
- A dataset generated entirely from a uniform distribution specifically with $\mathcal{U}(0, 1000)$ distributed data.

This evaluation assesses MRTAgent's ability to handle distributional shifts. Conversely, the model trained on uniformly generated task lists is evaluated on:

- A dataset sampled from the same uniform distribution specifically with $\mathcal{U}(0, 1000)$ distributed data,
- A dataset generated from a Gaussian distribution pecifically with $\mathcal{N}(600, 50)$ distributed data,

offering insights into its performance under distributional shifts.

Table 1 provides a detailed comparison of the test results. The total number of tasks in each episode, the number of robots, and the LA queue length remain consistent with the training setup 505 tasks,

10 robots, and an LA length of 5. The results clearly demonstrate MRTAgent's consistent outperformance over the brute-force optimal and FIFO-based methods across almost all test scenarios. The MRTAgent framework outperforms the baselines in all instances, and for a completely different dataset, it performs significantly better compared to the baselines.

To further analyze MRTAgent's superiority over other baselines, we examine the individual reward components of all approaches, as shown in Table 2. The total cost (reward) consists of two primary components: (a) TRTO, which aims to minimize robot travel distance, and (b) TTGT, which seeks to reduce task execution delays. As shown in Table 2, MRTAgent achieves value, in the TRTO component lesser than the brute-force optimal approach signifying its ability to minimize travel distance for the tasks in the look-ahead queue, as well as in the TTGT component because the brute-force optimal approach does not prioritize minimizing delays for tasks already in the look-ahead. On the other hand, the FIFO approach, despite assigning tasks sequentially, does not effectively reduce the TTGT component. This is because the task endpoints may be far from the starting locations of subsequent tasks in the look-ahead, potentially causing delays in task execution. As a result, the TRTO component is typically larger than in the brute-force.

Variable number of robots: To assess MRTAgent's generalizability, we run experiments with varying numbers of robots during inference. Initially, the executor is trained with a fleet of 10 robots and 5 tasks in the LA, and the results are presented in Table 1. We then retrain the executor for a fleet of 30 robots, while keeping the planner in evaluation mode, to observe MRTAgent's performance and scalability. The results for the 30-robot scenarios are shown in Table 3. As expected, performing the same tasks with more robots incurs lower costs, as the TRTO and TTGT components reduce significantly. Nevertheless, MRTAgent consistently outperforms the baseline methods in all scenarios.

The separation of Tasks (*Planner*) and Robot (*Executor*) agents in our framework enables scalability with varying task and robot counts. This setup also allows upgrading the *Executor* without retraining the entire system. Scenarios like robot failures—which reduce available robot count—can be managed by extending robot availability time (r_j, a feature in MRTAgent) to a large value, excluding the failed robot from selection. For instance, MRTAgent trained with 30 robots (see Table 3) is evaluated with only 25 available, without retraining in Table 4. This is achieved by assigning large values to r_j for the extra 5 robots, effectively preventing task allocation and enabling our framework to adapt to different robot counts without retraining.

Variable number of tasks: To evaluate the MRTAgent framework's generalizability, the planner and executor, initially trained for a fleet of 5 robots and 505 tasks (see Table 1), are tested on a larger number of tasks, specifically 2005 (see Table 5). As observed, MRTAgent consistently outperforms the baselines across a variable number of tasks without requiring retraining.

In Tables 1-5, the different rows represent the use of various datasets for evaluation. Specifically, in Table 1, the standard deviations reported for MRTAgent indicate slight fluctuations in performance when the model is evaluated using different random seeds. It is crucial to note that MRTAgent is trained with a periodically updated random task list, which prevents the agent from simply memorizing performance on a fixed set of tasks. Instead, this approach enables the agent to learn to adapt effectively to a wide range of scenarios. This is why there are small variations in the rewards across episodes, even as the agent gradually develops a consistent policy.

A note on the baselines: While FIFO is known for its simplicity and computational efficiency; BFO, despite common intuition, is one of the strongest baselines which, given a look ahead (LA), evaluates all task-robot pairs and selects the one with earliest possible execution. In fact, BFO is an optimal task allocation and assignment approach given the current state of the LA received in an online fashion. The reason why MRTAgent is able to outperform it is due to the fact that MRTAgent exploits the underlying distribution defining task generation to plan for tasks to appear in future despite it having access to the same causal information as the BFO.

6 CONCLUSION AND FUTURE WORK

This study introduces MRTAgent, a self-play motivated bi-level RL framework designed to enhance MRTA in modern warehousing environments. By optimizing operational costs and efficiency, MRTAgent addresses the increasing demands associated with online order fulfillment. The framework employs an optimal sequential task and robot selection process, coupled with a LQR based collision-free navigation algorithm. It effectively manages critical constraints such as robot dynamics, charging needs, and specific task requirements. Our approach demonstrates significant improvements over baseline methods across various test datasets. The validation of MRTAgent over datasets with distributional shifts and varying numbers of robots and tasks highlights its generalizability.

While our MRTAgent algorithm presents a promising approach, it is essential to acknowledge certain limitations that warrant attention in future research endeavors:

1. Single-Task Assumption: The algorithm currently assumes that robots are engaged in one task at a time, potentially limiting its applicability in scenarios where multitasking is prevalent.

2. No Contingency for Sudden Breakdowns: The model does not account for the sudden breakdown of a robot during operation. Once a task is allocated, our algorithm lacks the capability to reconsider or revert the decision in the event of an unforeseen robot malfunction.

3. Homogeneous Robots: We have made the simplifying assumption that all robots in the system are homogeneous, sharing identical values of characteristics such as velocity, acceleration, and load capacity. This assumption may not reflect the diversity present in real-world robot fleets.

4. Negligible Load/Unload Time Assumption: The algorithm assumes negligible time for loading/unloading operations after a robot reaches the task origin/destination. In reality, this may not hold true, and accounting for realistic loading and unloading times is a consideration for future enhancements.

Future work will focus on enabling robots to handle multiple tasks simultaneously, incorporating load/unload times, integrating heterogeneous robots, and implementing learning-based navigation, all of which will enhance the algorithm's effectiveness and applicability in diverse, practical scenarios.

REFERENCES

[1] Aakriti Agrawal, Amrit Singh Bedi, and Dinesh Manocha. 2023. Rtaw: An attention inspired reinforcement learning method for multi-robot task allocation in warehouse environments. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 1393–1399.

[2] Aakriti Agrawal, Senthil Hariharan, Amrit Singh Bedi, and Dinesh Manocha. 2022. DC-MRTA: Decentralized Multi-Robot Task Allocation and Navigation in Complex Environments. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 11711–11718.

[3] Javier Alonso-Mora, Andreas Breitenmoser, Martin Rufli, Paul Beardsley, and Roland Siegwart. 2013. Optimal reciprocal collision avoidance for multiple non-holonomic robots. In *Distributed autonomous robotic systems: The 10th international symposium*. Springer, 203–216.

[4] Ali Bolu and Ömer Korçak. 2021. Adaptive task planning for multi-robot smart warehouse. *IEEE Access* 9 (2021), 27346–27358.

[5] Yuxiao Chen, Ugo Rosolia, and Aaron D Ames. 2021. Decentralized task and path planning for multi-robot systems. *IEEE Robotics and Automation Letters* 6, 3 (2021), 4337–4344.

[6] Ayan Dutta, Swapnoneel Roy, O Patrick Kreidl, and Ladislau Bölöni. 2021. Multi-robot information gathering for precision agriculture: Current state, scope, and challenges. *IEEE Access* 9 (2021), 161416–161430.

[7] Alessandro Farinelli, Elena Zanotto, Enrico Pagello, et al. 2017. Advanced approaches for multi-robot coordination in logistic scenarios. *Robotics and Autonomous Systems* 90 (2017), 34–44.

[8] Yasuhiro Fujita, Prabhat Nagarajan, Toshiki Kataoka, and Takahiro Ishikawa. 2021. ChainerRL: A Deep Reinforcement Learning Library. *Journal of Machine Learning Research* 22, 77 (2021), 1–14. http://jmlr.org/papers/v22/20-376.html

[9] Brian P Gerkey and Maja J Mataric. 2003. Multi-robot task allocation: Analyzing the complexity and optimality of key architectures. In *2003 IEEE international conference on robotics and automation (Cat. No. 03CH37422)*, Vol. 3. IEEE, 3862–3868.

[10] Maria Gini. 2017. Multi-robot allocation of tasks with temporal and ordering constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 31.

[11] Refael Hassin and Adam Nathaniel. 2021. Self-selected task allocation. *Manufacturing & Service Operations Management* 23, 6 (2021), 1669–1682.

[12] Gaël Humbert, Minh-Tu Pham, Xavier Brun, Mady Guillemot, and Didier Noterman. 2015. Comparative analysis of pick & place strategies for a multi-robot application. In *2015 IEEE 20th conference on emerging technologies & factory automation (ETFA)*. IEEE, 1–8.

[13] Bilal Kartal, Ernesto Nunes, Julio Godoy, and Maria Gini. 2016. Monte Carlo tree search for multi-robot task allocation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 30.

[14] Alaa Khamis, Ahmed Hussein, and Ahmed Elmogy. 2015. Multi-robot task allocation: A review of the state-of-the-art. *Cooperative robots and sensor networks 2015* (2015), 31–51.

[15] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[16] Donald E Kirk. 2004. *Optimal control theory: an introduction*. Courier Corporation.

[17] Minghua Liu, Hang Ma, Jiaoyang Li, and Sven Koenig. 2019. Task and path planning for multi-agent pickup and delivery. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

[18] Hang Ma, Jiaoyang Li, TK Kumar, and Sven Koenig. 2017. Lifelong multi-agent path finding for online pickup and delivery tasks. *arXiv preprint arXiv:1705.10868* (2017).

[19] Kai-Chieh Ma, Zhibei Ma, Lantao Liu, and Gaurav S Sukhatme. 2018. Multi-robot informative and adaptive planning for persistent environmental monitoring. In *Distributed Autonomous Robotic Systems: The 13th International Symposium*. Springer, 285–298.

[20] Alejandro R Mosteo and Luis Montano. 2007. Comparative experiments on optimization criteria and algorithms for auction based multi-robot task allocation. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 3345–3350.

[21] Aritra Pal, Anandsingh Chauhan, Mayank Baranwal, and Ankush Ojha. 2024. Optimizing Multi-Robot Task Allocation in Dynamic Environments via Heuristic-Guided Reinforcement Learning. In *ECAI 2024 - 27th European Conference on Artificial Intelligence, 19-24 October 2024, Santiago de Compostela, Spain (Frontiers in Artificial Intelligence and Applications, Vol. 392)*. IOS Press, 1911–1918.

[22] Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V. Albrecht. 2021. Benchmarking Multi-Agent Deep Reinforcement Learning Algorithms in Cooperative Tasks. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS)*. http://arxiv.org/abs/2006.07869

[23] Sun-Oh Park, Min Cheol Lee, and Jaehyung Kim. 2020. Trajectory planning with collision avoidance for redundant robots using jacobian and artificial potential field-based real-time inverse kinematics. *International Journal of Control, Automation and Systems* 18, 8 (2020), 2095–2107.

[24] Martin L Puterman. 1990. Markov decision processes. *Handbooks in operations research and management science* 2 (1990), 331–434.

[25] Jorge Pena Queralta, Jussi Taipalmaa, Bilge Can Pullinen, Victor Kathan Sarker, Tuan Nguyen Gia, Hannu Tenhunen, Moncef Gabbouj, Jenni Raitoharju, and Tomi Westerlund. 2020. Collaborative multi-robot search and rescue: Planning, coordination, perception, and active vision. *Ieee Access* 8 (2020), 191617–191643.

[26] Oren Salzman and Roni Stern. 2020. Research challenges and opportunities in multi-agent path finding and multi-agent pickup and delivery problems. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. 1711–1715.

[27] Guillaume Sartoretti, Yue Wu, William Paivine, TK Satish Kumar, Sven Koenig, and Howie Choset. 2019. Distributed reinforcement learning for multi-robot decentralized collective construction. In *Distributed Autonomous Robotic Systems: The 14th International Symposium*. Springer, 35–49.

[28] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[29] Mark W Spong, Seth Hutchinson, and Mathukumalli Vidyasagar. 2020. *Robot modeling and control*. John Wiley & Sons.

[30] Changyun Wei, Ze Ji, and Boliang Cai. 2020. Particle swarm optimization for cooperative multi-robot task allocation: a multi-objective approach. *IEEE Robotics and Automation Letters* 5, 2 (2020), 2530–2537.

[31] Sean Wilson, Paul Glotfelter, Siddharth Mayya, Gennaro Notomista, Yousef Emam, Xiaoyi Cai, and Magnus Egerstedt. 2021. The robotarium: Automation of a remotely accessible, multi-robot testbed. *IEEE Robotics and Automation Letters* 6, 2 (2021), 2922–2929.

[32] Qinghong Xu, Jiaoyang Li, Sven Koenig, and Hang Ma. 2022. Multi-goal multi-agent pickup and delivery. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 9964–9971.

[33] Yang Yang, Li Juntao, and Peng Lingling. 2020. Multi-robot path planning based on a deep reinforcement learning DQN algorithm. *CAAI Transactions on Intelligence Technology* 5, 3 (2020), 177–183.