AI Enhanced Human-Robot Collaboration in Smart Manufacturing

A Dissertation Proposal

Presented to

the faculty of the School of Engineering and Applied Science

University of Virginia

---

in partial fulfillment

of the requirements for the degree

Doctor of Philosophy

by

Tian Yu

November 2023

Acknowledgement

First and foremost, I would like to express my heartfelt gratitude to my advisor, Dr. Qing Chang, for her invaluable guidance and careful supervision over years. Without her long-standing support, it would not be possible for me to complete this journey. My appreciation also goes out to my Ph.D. advisory committee members Dr. Ye Sun, Dr. Tariq Iqbal, Dr. Haibo Dong, and Dr. Daniel Quinn.

I also want to thank all current and previous members of Intelligent System Lab at University of Virginia and Stony Brook University for their assistance and friendship throughout my Ph.D. study. In addition, I would like to offer my biggest thanks to my family, especially my parents and my wife, for their tremendous support, love, and sacrifice.

In loving memory of my father Jun Yu

To my parents, Jun Yu and Fang Wei, and my wife Anqi Liu

Abstract

In the era of Industry 4.0, smart manufacturing systems have witnessed unprecedented growth, ushering in a new era of productivity and flexibility. The integration of robots into manufacturing environments has been instrumental in achieving these advancements. However, the full potential of human-robot collaboration (HRC) is yet to be fully realized. This dissertation addresses novel approaches to enhance human-robot collaborative efficiency and reduce the burden of reprogramming through the application of reinforcement learning (RL)/ deep reinforcement learning (DRL) based task scheduling and robot learning from demonstration (LfD).

The first part of this research focuses on the development of an intelligent task scheduling framework for HRC leveraging RL. Traditional scheduling techniques often struggle to adapt to dynamic manufacturing environments and unforeseen disruptions. By applying DRL, our framework dynamically allocates tasks to human and robot workers, taking into account real-time factors such as task complexity, agent availability and task preferences. Through extensive simulation experiments, we demonstrate significant improvements in HRC efficiency and adaptability.

The second part of this dissertation explores the robot LfD in HRC. Reprogramming robots for new tasks has traditionally been a time-consuming and expert-dependent process. We propose an RL-based LfD approach that enables robots to learn new tasks from human demonstrations, reducing the reprogramming overhead. To be more specific, we develop a systematic way to capture kinematic features of human demonstrations, which can be further mapped to all semantically similar tasks. By combining LfD with RL, robots acquire the capability to generalize from demonstrations and adapt to variations in

the manufacturing process. This not only reduces reprogramming time but also increases the flexibility of smart manufacturing systems.

Contents

Chapter 1. Introduction

In smart manufacturing, Human-Robot Collaboration (HRC) stands as a pivotal frontier, driving innovation and efficiency in modern industrial systems [1]. As Industry 4.0 continues to shape the manufacturing landscape, the efficient integration of human and robot workers has garnered substantial attention, fueled by the promise of increased productivity and adaptability [2]. However, this expected HRC efficiency is often hampered by the challenges in task scheduling in complex and dynamic HRC environments and limitations within industrial robots that are predominantly pre-programmed and require expert reprogramming for tasks that exhibit even marginal deviations from their original tasks [3]–[7]. To address these challenges, two pivotal components come into focus: higher-level task scheduling and lower-level motion planning.

On one hand, while considerable research and applications have focused on task scheduling and planning in HRC systems, many of these studies have primarily addressed single-human-single-robot collaboration [8]–[10]. Challenges, particularly those related to optimal task scheduling for multiple agents with complicate task structures, remain largely unaddressed. Consequently, there is a growing trend towards multi-agent and multi-level task scheduling in HRC. Recent advances in reinforcement learning (RL) [11], [12] and game theory [13] have shown promise in addressing large-scale decision-making problems, particularly when modeling systems as multi-agent reinforcement learning (MARL) problems [14], [15]. Inspired by this progress, this dissertation conceptualizes the HRC assembly process as a chessboard game governed by the chessboard mapping rule and the game playing rule. In this approach, task constraints are embedded in the chessboard format. The task scheduling problem is formulated in a Markov Decision Process (MDP) paradigm, and a Mont Carlo Tree Search (MCTS) [16] and a deep MARL algorithm [17] are introduced to optimize assembly task completion time. These approaches are demonstrated to

enable efficient task scheduling in HRC assembly, even when dealing with a substantial number of tasks and complex task structures.

On the other hand, in the realm of robot motion planning, existing researches have been done by sampling-based methods [18], [19], optimization-based methods [20], [21], and reinforcement-learning (RL) based methods [22]. However, these methods often demand either expert knowledge in modeling the task or substantial data and computational resources to produce motion plans effectively. In recent years, the concept of learning from demonstration (LfD) has emerged as a promising avenue for robot motion planning within human-robot collaboration settings [23]–[26]. Nonetheless, contemporary LfD methods face significant hurdles in terms of scalability and adaptability. With the advancements of AI, RL/DRL are used in LfD [27]–[30]. However, most RL/DRL based methods lack task space understanding and require tons of demonstrations to train the learning policy, which incurs significant labor and computational time expenses. In this dissertation, a novel RL-based LfD (RL-LfD) method is proposed [31], which enables the robot to learn from one single or a few demonstrations for all semantically similar task instances. In such a method, the lower-level motion planner captures kinematic features of demonstrations in the task space and fits these features to different new tasks by a mapping function. RL is employed to gain insights of the higher-level task structure and identify appropriate demonstrations for learning purposes. This LFD technique is expanded to encompass a coordination scheme for the mobile manipulator. This coordination scheme integrates both manipulator LfD and mobile base motion planning, leading to enhanced execution precision and computational time savings. Furthermore, in order to leverage the strengths of both the task space-based method and the joint space-based method while mitigating their respective shortcomings, we introduce a hybrid motion planning approach. This approach combines the task space RL-LfD method with a joint space DRL method. Through the utilization of such a method, a significant increase in training efficiency is achieved, and the feasibility of the trajectory is guaranteed.

Chapter 2. RL-based Task Scheduling in HRC Assembly

2.1.Background

Industrial robots have been successfully used to perform repetitive tasks with a high precision. However, there are tasks, such as complicated assembly works, that are less structured and too complex to be fully automated and thus cannot be totally performed by robots. Moreover, evaluation of the performance and the flexible adjustment by the human are sometimes necessary, which makes it impossible to fully replace humans with robots. Therefore, human–robot collaboration (HRC) systems are developed in industry to take advantages of the abilities of both humans and robots [32]. Unlike ordinary industrial robotics where the environment is structured and known, in HRC systems, the robots interact with humans who may potentially have very different skills and capabilities [33]. Over the past two decades, a significant number of researches has been done on the design of the HRC in manufacturing systems to improve the safety, quality, and efficiency of the system [3], [4], [34]–[36]. Since its declaration in late 1990s [37], the collaborative robots or cobots have been playing an increasingly significant role in the HRC in manufacturing systems as the assistants for humans [38],[39]. However, the traditional viewpoint has been mostly focused on the development of the hardware [40],[41], which results in machine-driven collaboration with less consideration of the function flexibility of the cobots. These studies are human-centralized, in which the cobots are limited to the scheme designed by humans and programmed based on the human domain knowledge, guidance or expert experience. Cobots have no self-learning capability, let alone the ability of self-organizing and the superhuman decision making that may surpass the existing models. Therefore, to explore a better method to accomplish a given task with the minimum workload, it is desired to

develop human–robot systems in which the cobots can better cooperate with humans in a more autonomous way. The cobots will not be supposed to be regarded as only working tools anymore, but to have their own abilities to judge the system states with corresponding decision-making capability and are able to adapt themselves to various levels of the human operators.

The assembly work in manufacturing is significantly important, which integrates parts and components to realize the final products. The application of HRC in complex, continually changing and variety-oriented assembly processes is still limited. One of the main challenges is that the HRC assembly environment is complex and dynamic. Although there might be a list of assembly tasks, the tasks are not necessarily in sequential order and many tasks can be performed in parallel, and some shared tasks can be taken by either human operators or robots. With the uncertainties involving human operator's performance level and other random disruptions in plant floor (e.g., machine tool random failure), the decision on which tasks to be taken by what available resource, i.e. robots or human operators, and in what sequences, will have profound influence on the productivity of the assembly process. The current practice is mostly a manual process that heavily depends on human experience. The cobot is pre-programmed to perform repetitive tasks in limited assembly process. When assembly tasks change, the whole system must be reprogrammed by robotics experts, while operators working on the floor usually do not possess the expertise to reprogram the system. This requires the design of more intelligent cobot for the HRC in manufacturing assembly process.

A large amount of the latest existing HRC studies on efficiency and safety of the system are trajectory based, focusing on the human plan recognition and the prediction of human motions [42], [43]. However, most of the trajectory-based studies are human-centralized, aiming at developing an adaptive robot assistant on a lower level. Although the robot can be developed very smart and sensitive to human behaviors, how to improve the hybrid human-robot performance on a system level is still illusive. Therefore, it is essential to allocate and dispatch the

tasks in a systematic way combining robots' precision, velocity and predictability with humans' intelligence and skills to achieve a hybrid solution to optimize the system performance according to a given criterion (e.g., time and energy consumption). Several constraints have to be considered, such as the ability of the resources to perform a task, the availability of all the necessary tools, and the time required by each of the resource for the performing the tasks.

In HRC assembly, real-time planning and scheduling play a key role in the generation of a plan and its robust execution. In order to cope with the presence of human in the loop, the task plan is generally constructed at an abstract, high and discrete level and continuously evaluated to decide how and when to execute a planned task, considering temporal or causal constraints, spatial constraints and controllable or uncontrollable activities [44]. For larger task spaces and more complex task structures, the application of mathematical optimization to the job dispatching problem will be NP hard.

With the emerging of smart manufacturing of Industrie 4.0 in recent years, plenty of advanced approaches have been developed in system control and decision making [45], in which the application of machine learning (ML) is the most impressive part with increasingly powerful algorithms. For example, in scheduling, tons of research have been done using reinforcement learning (RL) [46]–[48]. In the past, the traditional RL methods are commonly limited to low-dimension problems. In recent years, game-theoretic and RL models and methodologies are widely applied to the multi-agent task scheduling problems [49]. It is believed that the equilibrium concept in game theories and multi-agent training methods are highly potent in dealing with multi-constraint and multi-agent optimization problems. For example, [50] introduces a Q-learning based approach for obtaining Nash equilibria in general-sum stochastic games. Although the proof of convergence of the algorithm is provided for games with finite game and action spaces, their approach is computationally infeasible for all but the simplest examples. Above all, existing approaches are usually applicable within a very small task space or restricted by prior expert's knowledge from static global

point of view, which cannot appropriately describe the dynamic process of task scheduling in HRC. It is well noted that the multi-agent task scheduling in HRC is an NP-hard problem [51], therefore, it will be extremely time-consuming to yield an optimal working sequence through traversal-based algorithms for a large task space and complex task structures.

Over the past few years, the development of deep neural networks (DNNs) [11] and significant advances of deep reinforcement learning (DRL) have been witnessed in lots of outstanding large-scale sequential decision-making problems [11], [12], [52], [53]. Notably, lots of successful DRL applications model the systems as MARL problems. Inspired by self-play algorithm of Alphago Zero [12], we formulate the manufacturing assembly process as a chessboard game with the specific assembly rules determined by the required constraints. By integrating RL and deep neural network, we can take up the challenge to study the decision making and workload scheduling for both humans and cobots in HRC manufacturing systems.

2.2.Problem Description

2.2.1. HRC Assembly Problem Description

In the human-robot collaborative assembly system, the assembly tasks can be distinguished into different categories based on the evaluation of their physical properties and assembly characteristics [24]. To take the advantages of both human and robots, tasks in the human-robot collaborative assembly system can be categorized into three types: type I representing the tasks can be done by humans only, type II representing the tasks can be done by robots only, and type III representing the tasks can be done by either humans or robots. In this research, we do not focus on the task categorization and assume that the task type is given for specific assembly processes. For example, in a desk assembly process as shown in Fig. 2.1, tasks such as placing screws can be treated as type I tasks because humans are more flexible and faster to place screws into assembly holes. As robots can fasten screws much faster than humans, screwing tasks can be

classified as type II tasks. Tasks like flipping and rotating are regarded as type III tasks as they can either be done by humans or robots.

Typically, given a product, the plan of assembly is generated with a very rough assembly sequence based on physical constraints due to the product design, experts' experience or just by the preference of designers. However, there are still many possibilities to improve the assembly efficiency such as when to assign the tasks to the proper agents (i.e., humans or robots) and whether a human or robot needs to take on a type III task. Subsequently, for HRC assembly, it is significant to develop an adaptive method that determines the optimal task scheduling policies according to real-time system states.

In this research, the problem to solve can be described as: under a multi-agent HRC assembly environment, develop a method to find the optimal real-time task scheduling policy, such that the overall completion time for the entire assembly is minimized.

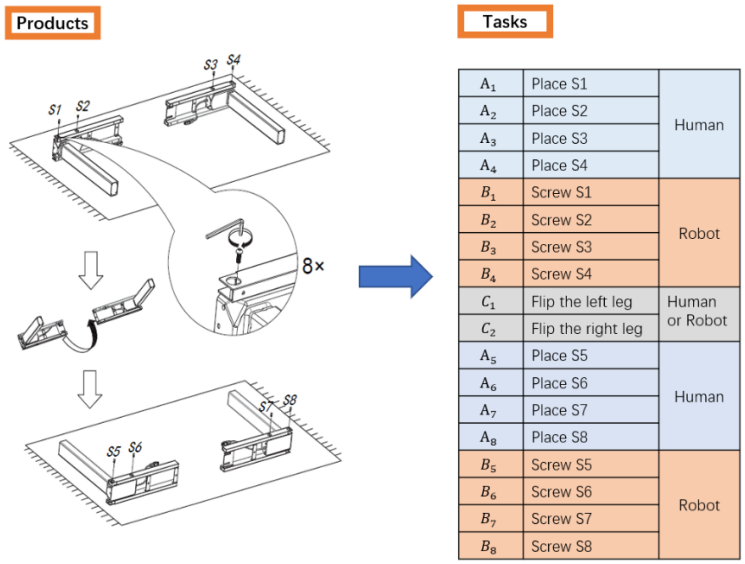


Figure 2.1. Diagram of decomposing the product in the human-robot collaborative assembly into tasks for humans, robots or both with a rough assembly sequence

2.2.2. Assembly Chessboard Game Framework

First of all, to solve the HRC task scheduling problem described above, the task information and constraints are intended to be formatted into a matrix. Using

this matrix as an input, the advanced deep learning method can be used to determine the optimal task sequence and task assignment, especially for type III tasks. Inspired by Alpha Go, the task structure of HRC assembly is formatted into a chessboard with three types of stones representing three types of tasks as shown in Fig. 2.2 (a). The relations of dependent tasks and/or concurrent tasks are embedded in the assembly chessboard using the chessboard mapping rules along with specific game playing rules, which can perfectly reflect the assembly process.

To take advantage of the assembly chessboard, the following assumptions are made in this research:

1. Each stone represents a minimum task or a subtask that can be done by only one agent at a time.
2. There are no conflicts among tasks or subtasks.

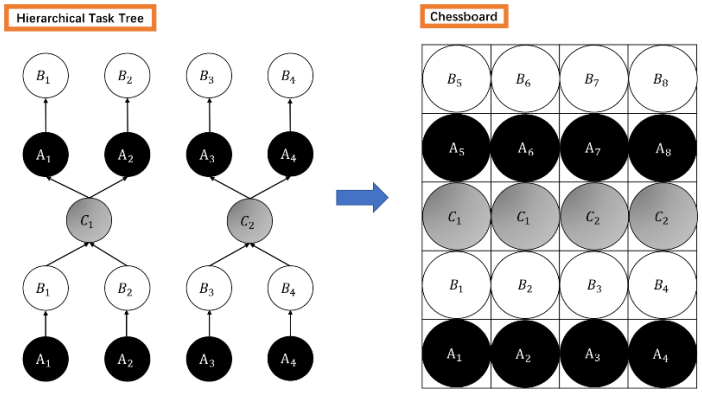
Chessboard mapping rules:

1. The chessboard has $w \times h$ grids, where the width w is determined by the number of parallel tasks at an assembly step that has the maximum parallel tasks among all assembly steps, and the height h is determined by the number of sequential steps for the entire assembly.
2. Black stones represent type I tasks.
3. White stones represent type II tasks.
4. Grey stones represent type III tasks.
5. By going through each branch of the hierarchical task tree, the stones representing corresponding tasks are mapped into the chessboard with one stone occupying one grid in the chessboard.
6. Tasks with no sequential constraints, are placed in the same row of the chessboard.

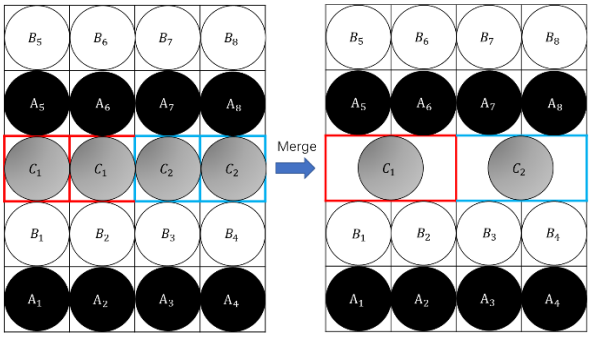
7. Tasks with sequential constraints, i.e., tasks of on the same branch of the task tree, are placed in the same column with prior tasks being set in the lower row of the chessboard.
8. After all tasks in the hierarchical task tree are mapped into the chessboard, adjacent stones representing the same task can be merged into a untied one.

Chessboard game playing rules:

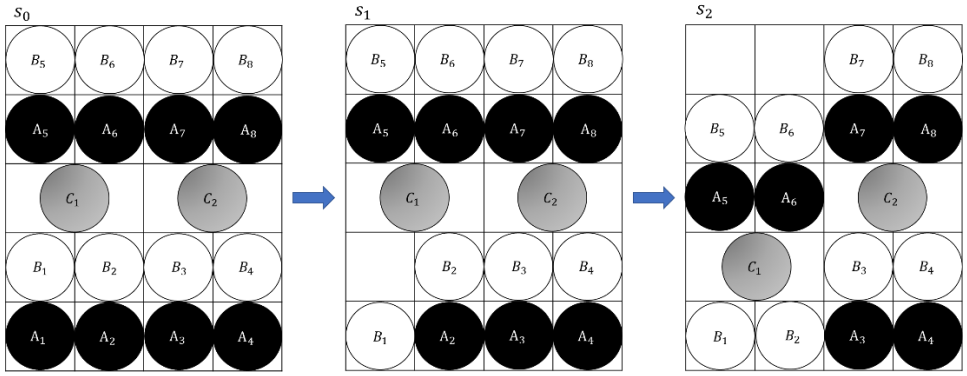
1. Players can only pick corresponding stones from the bottom row of the chessboard when they are available.
2. Each player can only pick one stone at a time.
3. The stone in the bottom row can only be picked when there are no stones representing the same task in upper rows.
4. Taking the stone away from the chessboard will cost the player the same associated time needed for the task that the stone represents.
5. When a stone in the bottom row is taken away, all upper stones within the same column will fall down one grid if the stones occupy the same size of the grid cells.
6. The game starts after mapping all assembly tasks into the chessboard with corresponding stones and ends until no stones left in the chessboard.



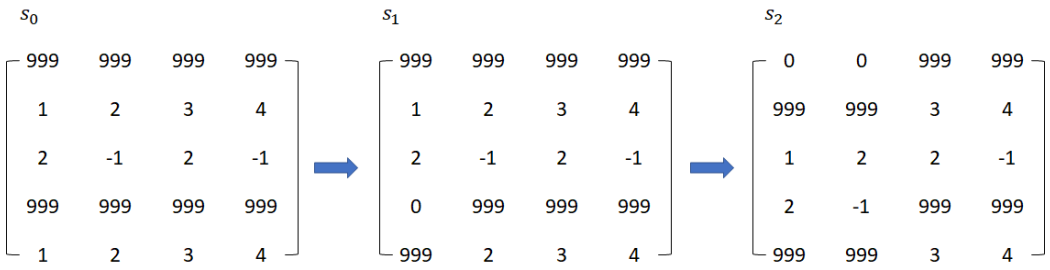
(a)



(b)



(c)



(d)

Figure 2.2. Diagrams of the format and transitions of the assembly chessboard. (a) Diagram of mapping the assembly process in Fig. 2.1 into the assembly chessboard based on the corresponding hierarchical task tree. (b) Diagram of how to merge the adjacent stones representing the same task in the same row into a united one. (c) The state of the assembly process and scheme of state transitions. (d) Task matrices of the human agent and the corresponding matrix transformation during the game playing.

To illustrate the chessboard mapping rules and game playing rules, the desk assembly shown in Fig. 2.1 is taken as an example. In the first part of assembly, human workers are supposed to place the screw S1, S2, S3 and S4 into assembly

holes so that robots can fasten them afterwards. Such placing-and-fastening tasks have sequential constraints. However, which screw to be assembled first has no specific sequential constraints. Therefore, according to the mapping rules, A_1 to A_4 representing task 1 to task 4 are parallel tasks and should be placed into the first row of the chessboard. As sequential tasks, B_1 to B_4 representing task 5 to task 8 are placed into the corresponding columns of A_1 to A_4 but in the second row on the chessboard. After screw S1 and S2 are fastened into assembly holes, the left leg of the desk needs to be flipped for screw S5 and S6 on the other side of the leg to be assembled. As the flipping work C_1 has sequential constraints with both B_1 and B_2, based on the mapping rules, it will be placed into the same column of both B_1 and B_2. In this way, C_1 occupies two grids in the third row. Same rules apply to C_2, the parallel task of C_1. The whole pattern of assembly chessboard is initialized as shown in the first figure of Fig. 2.2 (c).

After tasks are initialized in the chessboard, the HRC assembly process can be analogized by playing the chessboard game under the playing rules. For example, if the first task A_1 is picked and finished by a human operator, the corresponding stone A_1 will be removed. Consequently, the upper row stone B_1 will move down to the bottom row as shown in the second figure of Fig. 2.2 (c). It means that at this moment stone B_1 can be done at the same time or at the same level of other black stone tasks at the bottom row. Note that stone C_1 and all upper stones cannot move down at this time, since stone C_1 occupies two grid cells but only one stone below it falls down. Next, if stone A_2 is picked, then stone B_2 will move down one row to fill the bottom grid cell and consequently stone C_1 and all upper stones will also move down as shown in the third figure of Fig. 2.2 (c).

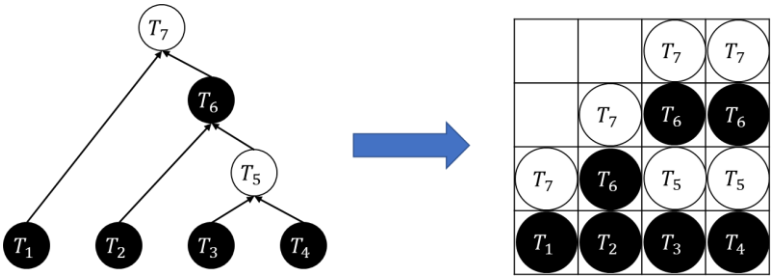
Applying the mapping rules and game playing rules above, the task structure at any time moment t can be formulated into a $w \times h$ state matrix s_t as shown in Fig. 2.2 (d). The assembly will start from the bottom row, i.e., the 5th row in Fig. 2.2 (d). Take one human agent as an example, the positive numbers in matrices in Fig. 2.2 (d) represent the completion time for corresponding subtasks. To bias tasks that might fit better for human operator's capability (type I tasks) or for

robot's advantages (type II tasks), the completion time of human agents for type II tasks are set to be a large number such as 999 minutes and the completion time of robot agents for type I tasks are also set to be 999 minutes. 0 in the matrix denotes that there is no task in corresponding position. -1 in the matrix denotes a bounding relation with its previous cell. For example, in Fig. 2.2 (c), stone C_1 takes two grids. Correspondingly, in matrix s_0, $s_0(3,1) = 2$ represents that it will take the human agent 2 minutes to finish task C_1. $s_0(3,2)$ is set as -1 since no task is needed in this cell but it is bounded with the task in $s_0(3,1)$ as a precondition for the tasks to be performed in the next row, i.e., row 2 in the example of Fig. 2.2 (d).

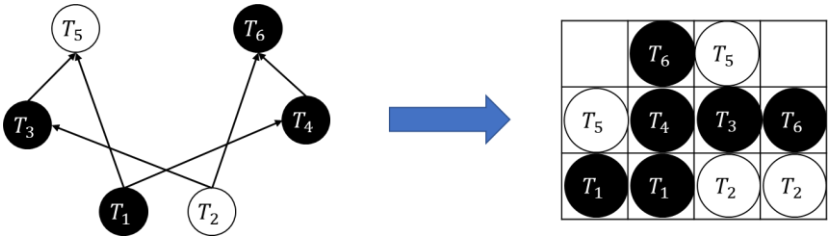
In some special cases, there exist different task dependency relations and mixed-logic relations for tasks across different task trees. Following the mapping rules, these special structures can also be mapped to chessboards, as shown in Fig. 2.3 (a) and (b). It is noticed that, in these scenarios, stones representing the same task may not be put into the same row, e.g., T_7 and T_6 in Fig. 2.3 (a), because of the special task structure. In these cases, one of stones representing the same task may fall to the bottom row during the game process while there are still stones representing the same task left in upper rows. In this situation, the game playing rule 3 works to prevent the stone to be taken away. For example, suppose that the working sequence for tasks in Fig. 2.3 (b) is $T_1 \rightarrow T_2 \rightarrow \cdots$. As shown in Fig. 2.3 (c), after T_1 is finished and the stone representing T_1 is taken away, the stone representing T_5 in the same column falls down to the bottom row. However, from the hierarchical task tree we know that T_5 can only be done after T_1 and T_3 are both finished. At this time, based on the game playing rule, the stone T_5 in the bottom row cannot be picked because there is still one stone representing T_5 in Row 3, Column 3. The same rule will be applied to T_6 when the stone representing T_2 is taken away.

Remark 1: Using chessboard mapping rules and game playing rules, most common assembly processes can be modelled using this assembly chessboard. Although there might be some limitations in representing the task conflicts and other complicated task relations, the approach proposed in this research is

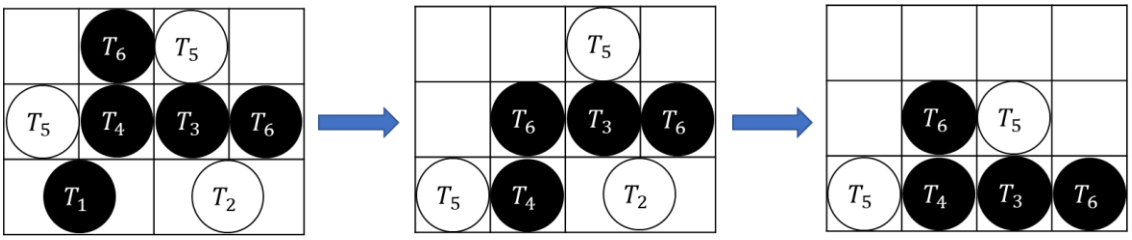
sufficient to address most assembly problems in the real world. More importantly, the proposed chessboard provides a systematic way to represent largely erratic assembly processes in an intuitive and concise manner, which lays the foundation for the HRC assembly task planning based on state-of-the-art deep learning techniques.



(a)



(b)



(c)

Figure 2.3. Mapping the tasks in the hierarchical task tree into the chessboard with (a) task dependency relations and (b) mixed logic relations across tasks of different hierarchical tasks trees. (c) Working flow of assembly tasks.

The mathematical notations and assumptions used in this research are defined as the following:

1. TK_u , $u = 1,2, \dots, U$, represents the u^{th} task with a total of U tasks.
2. Ag_i , $i = 1,2, \dots, n$, represents the i^{th} agent. $Ag_i =$

$\begin{cases} 0, & \text{the } i^{\text{th}} \text{ agent is a robot} \\ 1, & \text{the } i^{th} \text{ agent is a human} \end{cases}$

3. T_u^i, represents the i^{th} agent's average completion time on task TK_u.
4. $d_{TK_u}^i(t)$, represents the remaining time for an agent Ag_i to finish a task TK_u at time t. For example, if a task A_i has $T_{A_i} = 30\ min$, at time t it has already been done by the operator h for 10 min, then $d_{TK_u}^i(t) = 20\ min$.
5. $H_i(t) \in \{0,1\}$, represents the availability of the agent Ag_i at time t. If Ag_i is working on task TK_u, then H_i equals to 1. Otherwise, it is set to be 0, i.e.
6. $H_i(t) = \begin{cases} 0, & d_{TK_u}^i(t) = 0 \\ 1, & d_{TK_u}^i(t) > 0 \end{cases}$.
7. Humans have the priority of selecting actions over robots when they are available simultaneously.
8. Each agent can be assigned to only one task at a time.
9. Each task should be finished by only one agent.
10. At any time t, the information of each agent is transparent to all other agents.
11. The time each agent takes to complete a given task is random. Agent Ag_i's completion time for u^{th} task $T_u^i \sim \Pr(T_u^i = x)$ follows a probability distribution, e.g., normal distribution.

2.3.HRC Task Scheduling based on Single Agent RL

2.3.1. MDP Formulation

The formatting of the HRC assembly process to the assembly chessboard game playing provides great convenience for the task scheduling problem to perfectly fit the Markov Decision Process (MDP) paradigm, which is the most common framework for RL. The state, action and reward function of the MDP are defined as following.

In our task scheduling problem, at any time t the state of the chess game s_t consists of two parts. One is the task information on the chessboard such as the pattern of the chessboard and the remaining time of tasks on the chessboard, and the other is the availability of the human and robot. The state s_t can be defined as:

$$s_t = \left[p_1(t), \dots, p_Q(t), d_1(t), \dots, d_Q(t), H_1(t), .., H_M(t), R_1(t), \dots, R_N(t)\right] \quad (2.1)$$

where $p_i(t) = (x_i, y_i)$ is the position of the i^{th} task on the chessboard; $d_i(t) \in (0, T_i]$ is the remaining time of the i^{th} task; $H_i(t) \in [0, U]$ is the availability of the i^{th} human operator and $R_i(t) \in [0, U]$ is the availability of the i^{th} robot.

For each state s_t, each available agent can either pick a task to finish or just wait and take no actions. Thus, the actions of agent Ag_i at time t can be defined as:

$$a_t^i = \begin{cases} 0, & if\ Ag_i\ takes\ no\ actions \\ TK_u, & if\ Ag_i\ picks\ Tk_u \end{cases} \quad (2.2)$$

In this problem, the final goal for all agents is to work cooperatively to minimize the overall completion time of all tasks. Therefore, the reward function r_t^i for each agent Ag_i should be directly related to the completion time CT. The reward function is defined as:

$$r_t^i = \begin{cases} 0, & t < CT \\ -CT, & t = CT \end{cases} \quad (2.3)$$

From the initial state s_0 to the terminal state s_{end}, all the taken actions consist an action route l denoted as $\boldsymbol{a}_l = \{\, a(s_0), \dots, a(s_t), \dots, a(s_{end})\}$. Our goal is to find the optimal HRC policy to maximize the accumulated reward $r^l = \sum_{t=0}^{end} r_t$, so as to minimize the total job completion time.

It is noted that for route l, there are $O_l = \prod_{t=0}^{end} O_{s_t}$ possible combinations of humans' and robots' actions, in which O_{s_t} is the combinations of humans' and robots' actions in state s_t. For the example of the desk assembly, this O_l can be a huge number just like in the game of chess, the options of moves are around 35^{80}. For more complicated assembly jobs, such as automotive assembly, this can be an even larger number. Therefore, the assembly-chessboard game problem has an ultra-high dimension. A proper algorithm has to be developed to solve this problem both effectively and efficiently.

2.3.2. Monte Carlo Tree Search Convolutional Neural Network (MCTS-CNN) based Method for HRC Task Assembly

Based on the problem formulation in previous section, it has quite a few similarities to traditional board games, e.g. the game of Go. For example, it also has finite moves, and one agent's action inevitably affect the other's situation. Therefore, it is possible to leverage the successful solutions to traditional board games to solve our assembly-chessboard problem. In this section, we present the MCTS-CNN algorithm used in Alphago Zero to obtain the optimal HRC policy.

In the MCTS algorithm, the information of the system state s_t defined above is saved in each node s_t of the searching tree. For all legal actions $a_t \in a(s_t)$, the corresponding edges (s_t, a_t) are connected to the node s_t. The information saved in each state is a set as:

$$\{N(s_t, a_t), W(s_t, a_t), P(s_t, a_t)\} \tag{2.4}$$

in which, $N(s_t, a_t)$ is the visit count of the edge, $W(s_t, a_t)$ is the total action value and $P(s_t, a_t)$ is the probability in searching the edge (s_t, a_t). The searching process always starts from the root node s_0 of the searching tree and ends until a leaf node s_L is reached. And the action a is selected in the state s_t based on a variant of the PUCT algorithm, which can be defined as:

$$a = \arg\max_{a_t} \big(Q(s_t, a_t) + U(s_t, a_t)\big) \tag{2.5}$$

in which $Q(s_t, a_t) = W(s_t, a_t)/N(s_t, a_t)$ is the state value and $U(s_t, a_t) = cP(s_t, a_t)\sqrt{\sum_{b_t} N(s_t, b_t)}/(1 + N(s_t, a_t))$ is the exploration part to balance the exploitation part which is $Q(s_t, a_t)$, c is a constant that determines the level of exploration. When the leaf node is reached, it is always expanded and evaluated. Then, a reward v is backed up through each previous edge of the searching route. The visit count of these edges will be incremented as $N(s_t, a_t) = N(s_t, a_t) + 1$ and the action value will be updated as $W(s_t, a_t) = W(s_t, a_t) + v$. After a number of searches from the root, the agent takes an action in the root state s_0 based on the best policy $\pi_{opt}(a|s_0) = \max_{a_t}(N(s_0, a_t)/\sum_{b_t} N(s_0, b_t))$ proportional to the edge's visit count. Then, after the action is done, the system will transit to a new state s_1. Use this new state s_1 as the new root node and the subtree below node s_1 is retained as the new searching tree. All other parts of the previous searching tree are discarded. In this way, we can find the optimal policy for the agent scheduling in each state.

The assembly-chessboard game has an immense state space, and therefore the deep neural network helps tackle the dimensional issue with its powerful approximation ability. In addition, when the assembly job is represented with the chessboard, then the spatial correlation within the board is of great significance. Therefore, CNN is an ideal candidate to work with MCTS to solve the assembly-chessboard problem.

The CNN $f_\theta(*)$ takes all the information stored in the raw chessboard representation s as input, and outputs both move probabilities and a state value, i.e. $(p, v) = f_\theta(s)$. The vector of move probabilities p represents the probability of selecting each action. The value v is a scalar evaluation, estimating the task completion time starting from current state s. This neural network combines the roles of both policy network and value network into a single architecture.

The CNN is trained with the data gained from MCTS. In each state s_t, the MCTS search outputs policies π of taking actions. The actions selected by these policies are usually stronger than those selected from the CNN with the raw

probabilities. Therefore, MCTS can be regarded as a powerful policy improvement operator. The main idea of our RL algorithm is to use these search operators repeatedly in a policy iteration procedure, in which the neural network's parameters are updated to make the move probabilities and the state value $(p, v) = f_\theta(s)$ more closely match the improved search probabilities and the real task completion time; these new parameters are used in the next iteration$(p, v) = f_\theta(s)$ to more closely match the improved search probabilities and the real task completion time. Fig. 2.4 illustrates the training pipeline.

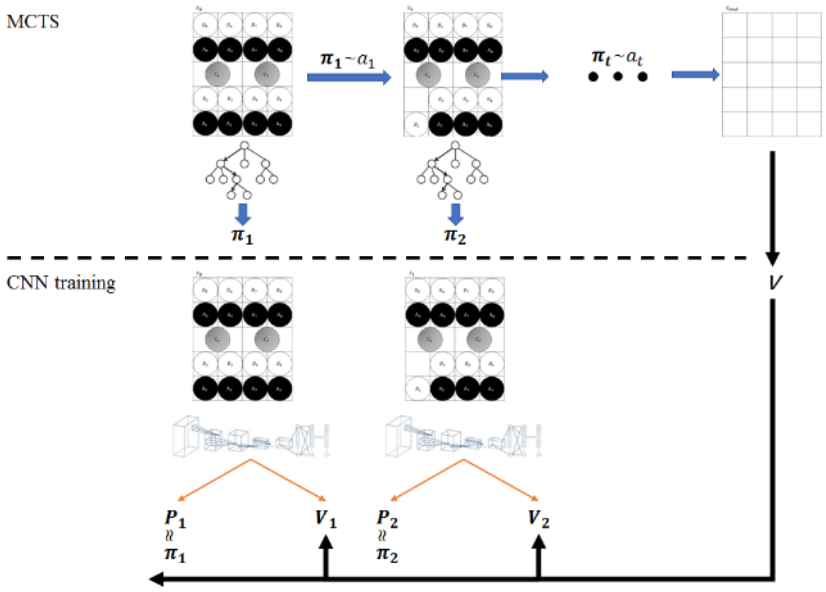


Figure 2.4. Diagram of the Monte Carlo Tree Search and the Convolutional Neural Network training the assembly chessboard.

2.3.3. Numerical Case Study

With the understanding of rules in the chessboard, we choose to assemble a height adjustable standing desk shown as an example. The job types are defined based on the following rational:

- Matching and placing work can only be done by humans;
- Screwing, drilling and gumming work can only be done by robots;
- Flipping and rotating work can be done by either humans and robots.

There are 27 robot-only tasks, 29 human-only tasks, and 4 human-or-robot tasks. After fitting the tasks into the chessboard according to the rough assembly sequence, an assembly chessboard is obtained with height $h = 15$ and width $w =$

8. There is one robot operator and one human operator cooperating with each other to complete the assembly job. The neural network architecture used in this case is defined as the followings:

- Input layer with size $h \times w \times d = 15 \times 8 \times 3$;
- Convolutional layer with 10 filters of kernel size 2×2 with stride 1 and Relu activation;
- Max-pooling layer with size 2×2;
- Convolutional layer with 10 filters of kernel size 2×2 with stride 1 and Relu activation;
- Max-pooling layer with size 2×2;
- Flattening layer;
- Dense layer of 128 units with Relu activation;
- First output from the dense layer: classification layer of size w with softmax activation function, and each indicates the probability choosing the w^{th} task;
- Second output from the dense layer: a single regression variable predicting the state value.

For MCTS, the maximum search depth is set to be 3 and maximum searches is limited to 30 times for one root node. The parameter for the UCT algorithm is chosen to be $c = 100$. We run the program in a PC with Intel Core I5-8400, UHD Graphics 630 and 12G RAM. The training progress is as shown in Fig. 2.5 with one specific completion time for each task. We can observe that the proposed algorithm steadily makes progress despite of some fluctuations during the first few iterations. The shortest completion time is reached at the sixth iteration. After the sixth iteration, the completion time remains steady as 95.

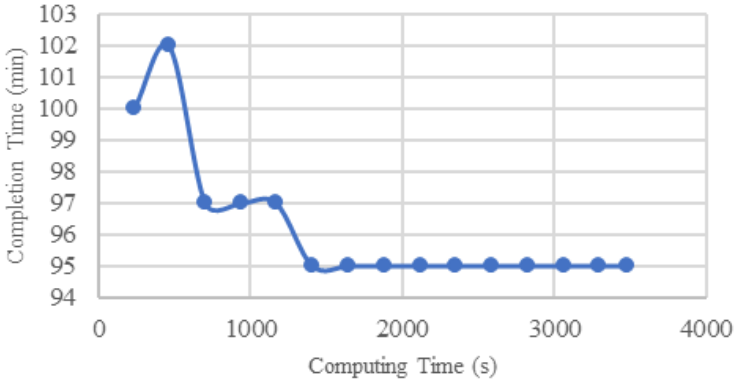


Figure 2.5. Diagram of the training process

For comparison purpose, two other approaches are considered in this case study. One is the exhaustive search, in which we traverse all possible routes; while the other one is dynamic programming (DP). It turns out that the exhaustive search is not feasible since the trajectories possibilities explodes and finally depletes the PC memory. This situation is almost inevitable for a large-scale planning problem since the state space explodes exponentially with decision steps. The exhaustive search program abruptly stops at step 55, and 5 more steps are yet to be traversed. We plot the total route numbers and computing time against the decision steps in Fig. 2.6.

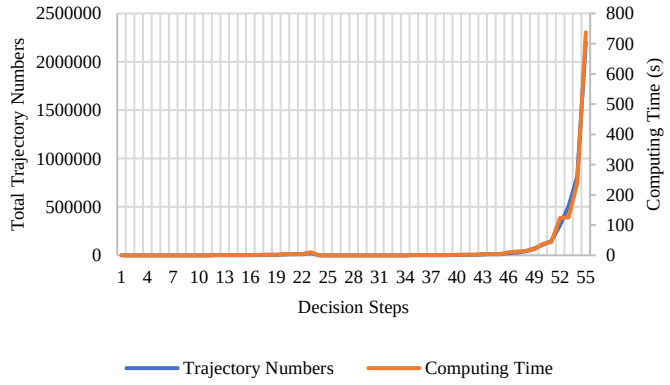


Figure 2.6. The computing time before depleting memory in exhaustive search

To make it computationally feasible, we limit the possible trajectories to 1,000 by randomly sampling. This approach mimics the scenario when both human operator and robot operator randomly choose tasks without considering the goal of minimizing the job completion time. However, as shown in Fig. 2.7, the result reveals that the shortest completion time is hardly achieved as only one out of 1,000 gives the completion time of 95. The average job completion time is 100.58, which is far more than the shortest completion time obtained by the proposed algorithm in this research. Therefore, the proposed algorithm is effective in optimizing the completion time and more powerful in saving computing time compared with exhaustive search.

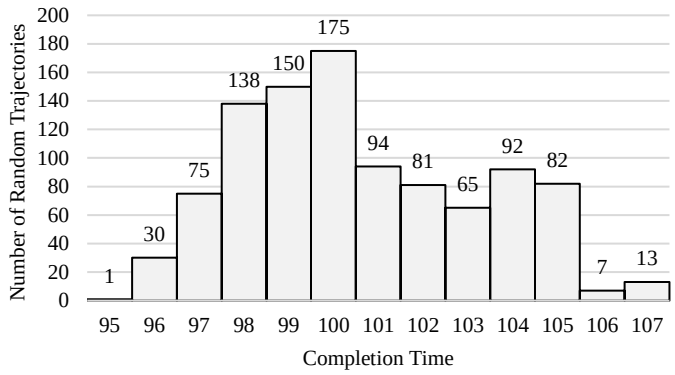


Figure 2.7. Histogram of the completion time with 1000 trajectories in total.

As for the other comparison approach, we compare our proposed approach with DP in the ability of obtaining the optimal policy in saving the completion time. As shown in Fig. 2.8 with one specific completion time for each task, we can observe that give enough computing time, both our proposed approach and DP will obtain the optimal policy in reducing the overall completion time of all tasks. While our proposed method in more efficient in achieving the best working sequence.

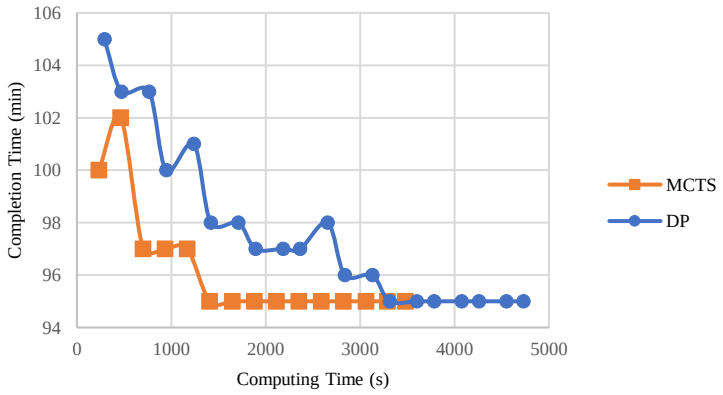
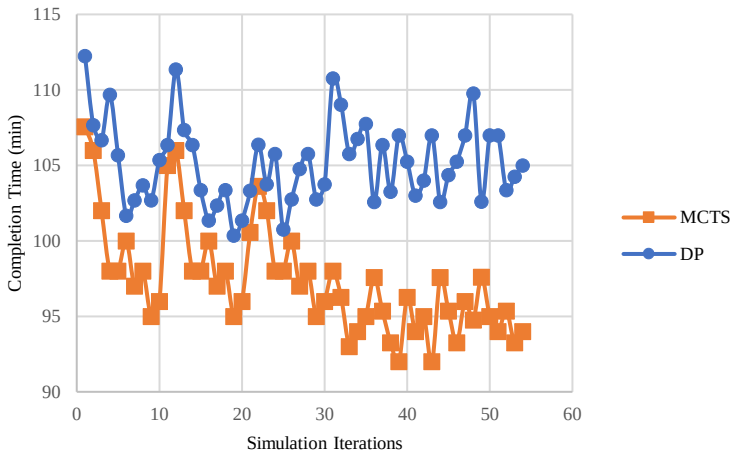


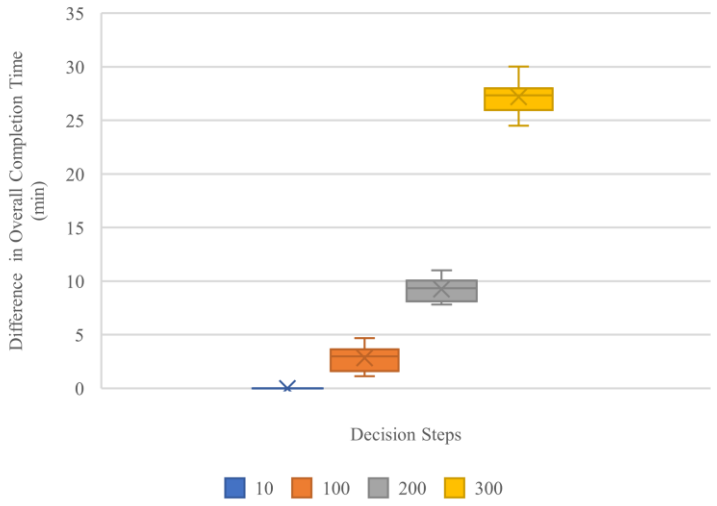
Figure 2.8. Comparison of CNN based MCTS method and DP in overall completion time under the scenario of one specific completion time for each task. Given enough computing time, both approaches will obtain the same overall completion time while the CNN based MCTS method performs better in saving computing time.

To demonstrate our proposed approach in real-time decision making, we also change the completion time of each task to a stochastic one following the normal distribution with the mean of the original specific completion time and the

standard deviation of 1 minute. To make comparison, both DP and our CNN based MCTS are given the same computing time (e.g. 1 second for each move). As shown in Fig. 2.9 (a). the overall completion time of our proposed approach converges at around 95 mins after 30 iterations. Although it takes much longer for the approach to converge than the specific completion time scenario, our approach shows the self-improvement in reducing the overall completion time. However, DP does not reflect the same properties as the CNN based MCTS method. We also investigate the completion time of both our proposed approach and DP given the same computing time with different size of decisions as shown in Fig. 2.9 (b). We can observe that our method shows more obvious advantages over DP in reducing the overall completion time with the increasing of the decision steps. Therefore, our method is demonstrated to be not only effective in real-time task planning under stochastic circumstances but also powerful in optimizing the overall completion time especially when the decision steps increase.



(a)



(b)

Figure 2.9. (a) Comparison of CNN based MCTS method and DP in overall completion time under the scenario of stochastic completion time for each task. Given the same computing time, the CNN based MCTS method performs better in reducing the overall completion time. (b) Differences of the overall completion time between CNN based MCTS method and DP under the environment containing 10, 100, 200 and 300 decision steps. The positive difference in overall completion time denotes the overall completion time obtained by DP is more than that of CNN based MCTS method.

2.4.HRC Task Scheduling based on MARL

2.4.1. MARL Framework in HRC Assembly

Based on the problem formulations, in the MARL framework, for every state s_t, each agent will select their own actions $a_t^1, \dots, a_t^n$ and receive corresponding rewards $r_t^i(s_t, a_t^1, \dots, a_t^n)$. These actions form a joint action $\boldsymbol{a_t} = (a_t^1, \dots, a_t^n)$, and consequently the state will transit to the next state s_{t+1} according to the transition probability $p(s_{t+1}|s_t, \boldsymbol{a_t})$ satisfying $\sum_{s_{t+1} \in S} p(s_{t+1}|s_t, \boldsymbol{a_t}) = 1$. With the goal of maximizing their own discounted accumulated rewards, each agent Ag_i select the action a_t^i under the policy π_t^i, which is the decision-making rule corresponding to the probability distribution of the agent's actions. Let $\pi^i = (\pi_0^i, \pi_1^i, \dots, \pi_t^i, \dots)$ be the policy of agent Ag_i for the whole game. Given an initial state s, the accumulated reward with discount factor $\gamma \in [0,1)$ can be represented

as:

$$v^i(s, \pi^1, \dots, \pi^n) = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}\left(r_t^i \middle| \pi^1, \dots, \pi^n, s_0 = s\right) \quad (2.6)$$

2.4.2. Equilibrium Strategies and Multi-agent Q-learning

In a Markov game, the Nash equilibrium is a joint policy in which each agent's policy is optimal considering the policies of other agents. Based on [50], in a Markov game, a Nash equilibrium point is defined as a tuple of n policies $(\pi_*^1, \dots, \pi_*^n)$ such that for all $s \in S$ and $i = 1, \dots, n$:

$$v^i(s, \pi_*^1, \dots, \pi_*^n) \geq v^i\left(s, \pi_*^1, \dots, \pi_*^{i-1}, \pi^i, \pi_*^{i+1}, \dots, \pi_*^n\right) \text{ for all } \pi^i \in \Pi^i \quad (2.7)$$

where Π^i represents all available policies for agent Ag_i.

To obtain the equilibrium policies in our HRC assembly problem, the multi-agent Q-learning method used in [50] is applied. According to [50], a Nash Q-value is defined as the expected accumulated discounted rewards when all agents follow specific Nash equilibrium policies. For agent Ag_i, the corresponding Nash Q-function Q_{t*}^i at state s_t can be expressed as:

$$Q_{t*}^i(s_t, \boldsymbol{a_t}) = r_t^i(s_t, \boldsymbol{a_t}) + \gamma \sum_{s_{t+1} \in S} p(s_{t+1}|s_t, \boldsymbol{a_t}) v_t^i(s_t, \pi_*^1, \dots, \pi_*^n) \quad (2.8)$$

where$(\pi_*^1, \dots, \pi_*^n)$ is the joint equilibrium policy and $r_t^i(s_t, \boldsymbol{a_t})$ is agent Ag_i's one step reward in state s_t under joint action $\boldsymbol{a_t}$.

Such extended Q-learning algorithm differs from single-agent Q-learning method in using next state's Q-values to updated current state's Q-values. In the multi-agent Q-learning, agents update their Q-values based on future Nash equilibrium payoffs, while in single-agent Q-learning, agents' Q-values are updated with their own payoffs. In other words, the agent has to observe its own reward as well as all other agents' rewards. At $t = 0$, our learning agent Ag_i learns its Q-value by forming an arbitrary guess, e.g., $Q_0^i(s, \boldsymbol{a}) = 0$ for all $s \in S$.

Then, at each time t, agent Ag_i updates its Q-value and calculates a Nash equilibrium $\pi_*^1(s_{t+1}) \cdots \pi_*^n(s_{t+1})$ for the state s_{t+1} according to:

$$Q_{t+1}^i(s_{t+1}, \boldsymbol{a_{t+1}}) = (1-\alpha_t)Q_t^i(s_t, \boldsymbol{a_t}) + \alpha_t\left[r_t^i + \gamma NashQ_t^i(s_{t+1}, \boldsymbol{a_{t+1}})\right] \quad (2.9)$$

where

$$NashQ_t^i(s_{t+1}, \boldsymbol{a_{t+1}}) = \pi_*^1(s_{t+1}) \cdots \pi_*^n(s_{t+1}) \cdot \ Q_t^i(s_{t+1}, \boldsymbol{a_{t+1}}) \quad (2.10)$$

is the payoff of agent Ag_i in state s_{t+1} for the selected equilibrium.

According to the definition of the Nash equilibrium, applying the equilibrium policy $\pi_*^1(s_{t+1}) \cdots \pi_*^n(s_{t+1})$, each agent Ag_i 's payoff will be maximized. Therefore, to obtain the equilibrium policy, each agent Ag_i need to know all other agents' $Q_t^j(s_{t+1}, \boldsymbol{a_{t+1}})$. However, at time t, other agents' Q-values are not given. Therefore, agent Ag_i has to build its own belief about other agents' Q-values. Same as updating its own Q-value, agent Ag_i guesses $Q_0^j(s, \boldsymbol{a}) = 0$ for all other agents at the beginning. With the propagation of the game, other agents' actions and rewards will be observed and used to updates its belief of all other agents Ag_j's Q-value as:

$$Q_{t+1}^j(s_{t+1}, \boldsymbol{a_{t+1}}) = (1-\alpha_t)Q_t^j(s_t, \boldsymbol{a_t}) + \alpha_t\left[r_t^j + \gamma NashQ_t^j(s_{t+1}, \boldsymbol{a_{t+1}})\right] \quad (2.11)$$

In addition, the proof of convergence of the optimality, i.e., the convergence of $(Q_t^1, \dots, Q_t^n)$ to $(Q_*^1, \dots, Q_*^n)$ has been provided in [50], which will not be elaborated in this research.

2.4.3. Applying DQN to Obtain Task Scheduling Policy

According to the discussion above, to obtain the equilibrium policy $(\pi_*^1, \dots, \pi_*^n)$, the agent Ag_i has to maintain and update n Q-values $(Q_t^1, \dots, Q_t^n)$. Let $|S_t|$ represent the number of all applicable states. Suppose that each agents' action space $|A_t^i|$ are the same, i.e., $|A_t^1| = \cdots = |A_t^n| = |A|$. The total number of entries in $Q_t^j(s_t, \boldsymbol{a_t})$ is $|S_t| \cdot |A|^n$. If all agents' Q-values are saved in a Q-table, the total memory space requirement is $n|S| \cdot |A|^n$. Therefore, the naive Nash-q method proposed in [11] cannot be put into practice when the state space is very

large. The problem with large state spaces is not just the memory needed for large tables, but the time needed to fill and search them accurately.

In recent years, the emergence of DQN algorithms have well addressed the scalability issue in Q-learning. It is a stable and scalable approach to complex and ultra-high-dimensional RL problems, such as Atari video games. In DQN, the Q-values are approximated with a neural network $\boldsymbol{\theta}$, i.e., $Q(s, a^1, \dots, a^n; \boldsymbol{\theta}) \approx Q(s, a^1, \dots, a^n)$. Instead of filling a large table, the DQN seeks to iteratively update the neural network parameters, which would well approximate the Q-values until the optimal policy $(\pi_*^1, \dots, \pi_*^n)$ is obtained eventually.

A critical question raised here is that can decentralized algorithms be faster than its centralized counterpart [25]? In this research, we combine DQN with MARL in Section 2.4.2 to generate a decentralized algorithm as shown in Algorithm 1. In the proposed DQN-MARL method, to determine the optimal joint actions $\boldsymbol{a_t}$ for each state s_t, agents are trained parallelly to learn the correlated policy $(\pi_*^1, \dots, \pi_*^n)$ based on the knowledge of other agents' rewards and actions. Suppose there are n available agents and m tasks at time t. Since all agents are trained parallelly, each agent only needs to consider its own possible allocations to tasks, which needs $O(nm)$ computational steps to go through all possible actions for all the agents. In the contrast, in our previous study [26], all allocations of agents' actions are considered in each step and controlled in a centralized fashion, which can be treated as a single-agent reinforcement learning (SARL) method. In the DQN-SARL method, it needs $O(m^n)$ computational steps to go through all possible actions. Therefore, the DQN-MARL method can outperform the DQN-based SARL (DQN-SARL) method in terms of computational efficiency, even if the same DQN are applied to the SARL method. For an ultra-high dimension task scheduling problem with complex task structures in an HRC system, both m and n could be a large number. Consequently, the computational efficiency of the DQN-MARL method can be more notably advantageous than that of the DQN-SARL method.

Algorithm 1

Input: Initialized chessboard information $\boldsymbol{p}(0)$, $\boldsymbol{d}(0)$, agent information $\boldsymbol{Ag}$

Output: $\boldsymbol{\theta}$

Initialize replay memory D to capacity N_{mem}

Initialize the neural network with random weights $\boldsymbol{\theta}$

While training time $t_{train} < T_{bound}$ **do**

For $t = 0,1, \dots, T$ **do**

Scan the availability of agents $\boldsymbol{H}(t)$

Find the legal action list $\boldsymbol{A}(s_t)$

With probability ϵ, select a random joint action $\boldsymbol{a_t}$

Otherwise select $a_t^i = \text{argmax}_{\boldsymbol{a_t} \in \boldsymbol{A}(s_t)} NashQ_t^i(s_t, \boldsymbol{a_t}; \boldsymbol{\theta})$

Each available agent executes action a_t^i and observe reward r_t^i

Observe next state s_{t+1}

Store transition sample $(s_t, \boldsymbol{a_t}, \boldsymbol{r_t}, s_{t+1})$ in replay memory D

Sample a minibatch of size b of transitions $(s_j, \boldsymbol{a_j}, \boldsymbol{r_j}, s_{j+1})$ from D

For each available agent, according to (7) and (8), set

$$y_j^i = \begin{cases} r_j^i & \text{if episode terminates at step } j+1 \\ r_j^i + \gamma NashQ_{j+1}^i\left(s_{j+1}, \text{argmax}_{\boldsymbol{a_{j+1}} \in \boldsymbol{A}(s_{j+1})} NashQ_{j+1}^i(s_{j+1}, \boldsymbol{a_{j+1}}; \boldsymbol{\theta}); \boldsymbol{\theta^-}\right) & \text{otherwise} \end{cases}$$

Perform a gradient descent step on $\left(y_j^i - NashQ_j^i(s_j, \boldsymbol{a_j}; \boldsymbol{\theta})\right)^2$

Every C steps, reset $\boldsymbol{\theta^-} = \boldsymbol{\theta}$

End for

episode=episode+1

End while

return $\boldsymbol{\theta}$

Procedure 1

Input: real-time states s_t

Output: joint task scheduling decision $\boldsymbol{a_t}$

Find the legal action list $\boldsymbol{A}(s_t)$

Run a forward propagation in neural network $\boldsymbol{\theta}$ to get $NashQ_t^i(s_t, \boldsymbol{a_t}; \boldsymbol{\theta})$

For each agent, find the optimal action as $a_t^i = \text{argmax}_{\boldsymbol{a_t} \in \boldsymbol{A}(s_t)} NashQ_t^i(s_t, \boldsymbol{a_t}; \boldsymbol{\theta})$

Output $\boldsymbol{a_t} = (a_t^1, a_t^2, \dots, a_t^n)$

After the offline training using Algorithm 1, the online playing is carried out to implement the DQN-MARL algorithm in practice. Given all the inputs and parameters of Algorithm 1 along with the chessboard simulator, the parameters $\boldsymbol{\theta}$ of the neural network will be obtained through the offline training. Using the trained neural network $\boldsymbol{\theta}$, the final decision-making process of each agents' task scheduling will be launched online based on Procedure 1. Since Procedure 1 only contains one forward propagation, the time for one step decision-making in the real-time HRC assembly is significantly short (less than 1 second).

2.4.4. Transferring Trained CNN to Broader Product Assembly with Different Task Scenarios

Regarding the neural network $\boldsymbol{\theta}$, the convolutional neural network (CNN), a type of neural network that is specialized in processing images or tensors, is adopted in the DQN-MARL algorithm. Since tasks in the task scheduling problem are mapped into the assembly chessboard, the usage of CNN is motivated by the fact that it is able to capture characteristic features (referred to as task features) from an image at different levels similar to a human brain. These features can also be reserved by CNN even if they are rotated, flipped or shifted [27]. This attribute of CNN is very significant in practice since structures of assembly tasks for various products in a workshop could be rather similar. These products are referred to as a product family in this research. Therefore, the trained CNN can be re-used for various products with similar task features to speed up the re-training for different product assembly involving different task scenarios. Because using the trained CNN, the Q-value for each state and action pair will be generated starting from a reasonable number rather than a random one. Consequently, the interaction of the task scheduling policy will be sped up to convergence with the pre-weighted Q-values.

For scheduling assembly tasks for various products (or product family), we can use the randomly generated patterns of the chessboard to represent a more general task scenarios accommodating various products assembly tasks. The randomly generated patterns can be used to train DQN to obtain more general task scheduling policies to deal with the change of task structures for various products assembly. However, this training process could be very time consuming. Therefore, a trained CNN from one product assembly can be partially re-used to different types of products assembly through adopting the selected trained weights based on the understanding of the task features. Fig. 2.10 describes this procedure. This method can largely speed up the re-training process for new product assembly with different task scenarios, which will be demonstrated through a case study in Section 2.4.5.

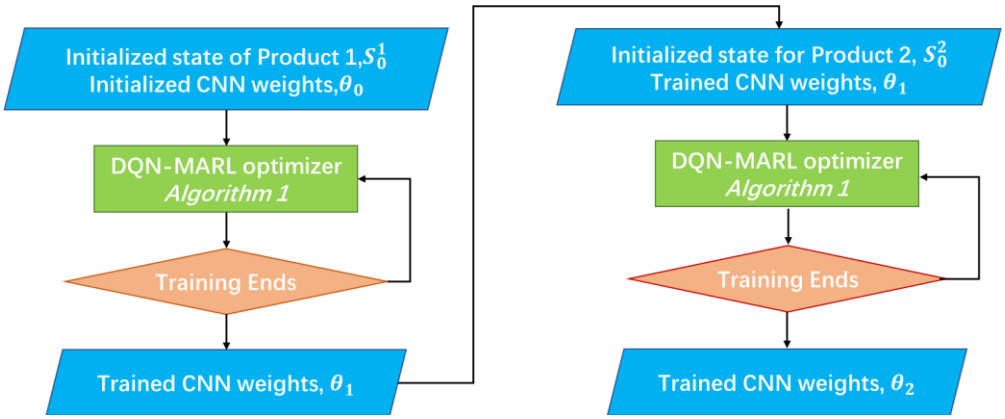


Figure 2.10. Flow chart for applying a trained CNN weights from one assembly product to the training process of a new assembly product.

2.4.5. Numerical Case Study

In order to validate the effectiveness of the proposed method, following numerical experiments and in-depth analysis are conducted based on the assembly of a height-adjustable desk shown in Fig. 2.11. The assembly chessboard including 29 type I tasks, 20 type II tasks and 4 type III tasks is generated randomly with each agent Ag_i's completion time for task TK_u follows a normal distribution $\mathcal{N}(T_u^i, \sigma^2)$ with $\sigma = 1$ min. For comparison, the naive Nash-Q method, dynamic programming (DP) and DQN-SARL method are used as alternative ways to solve the same task scheduling problem. In addition, the effectiveness of the proposed method for a broader product family assembly with similar task instances and a generalized case are also investigated. In this case study, two performance metrices are considered: (1) The overall completion time to finish the entire assembly tasks. (2) The time that the algorithm needs to converge to obtain the steady task scheduling policy. From the case study, three significant results can be concluded: (1) The proposed DQN-MARL method is effective in optimizing the task scheduling; (2) The proposed DQN-MARL method outperforms the naive Nash-Q, DP and DQN-SARL method in terms of time needed to obtain the optimal task scheduling policy, and such advantages become more prominent when the task space and agent number increases. (3) The well-trained CNN from the proposed method can be beneficial to obtaining

optimal task scheduling policies for a broader product family assembly with new task instances when task structures are similar.

For demonstration purpose, it is assumed that the average completion time of all human workers are identical and the same assumption works for all robots. To distinguish type I tasks from type II and type III tasks, the completion time of human agent for type II tasks is set as 999 minutes and robot agent for type I tasks as 999 minutes. Given the task parameters, all 53 assembly tasks are mapped into an 8×15 chessboard as shown in Fig. 2.11. Using this task structure, the completion time of each agent can be easily formatted into a completion time matrix and stack all matrices to form a n^{th} order tensor as the input layer of the CNN in the proposed method. The CNN architecture used in the case study is defined as followings:

- $15 \times 8 \times$ n input layer;
- Convolutional layer with 10 filters of kernel size 2×2 ;
- 2×2 Max-pooling layer;
- Convolutional layer with 10 filters of kernel size 2×2 ;
- 2×2 Max-pooling layer;
- Flattening layer;
- Dense layer with 128 units and ReLu activation;
- First output from the dense layer;
- Second output from the dense layer;

The capacity of the replay memory N_{mem} is $5{,}000$. The training batch size is $b = 32$. The target network replacement frequency C is 2,000 episodes. The reward discount rate $\gamma = 0.95$. The parameter for ϵ-greedy is at first set to be 0.8 and linearly diminished to 0.1 when it reaches 30,000 episodes and fixed to 0.1 for all subsequent episodes. The proposed algorithm is trained using TensorFlow

with 4 GPUs and 4 CPUs. The training time t_{train} is limited to T_{bound} =1200 minutes.

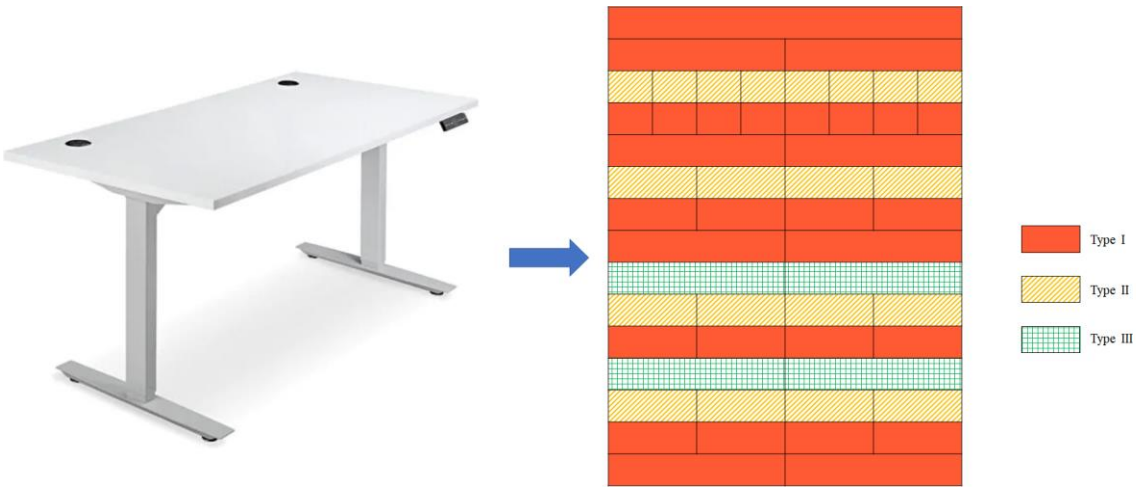


Figure 2.11. Diagram of tasks mapped into the chessboard. The red cells denote type I tasks. The yellow striped cells denote type II tasks. The green gridded cells denote type III tasks.

To evaluate the effectiveness of the proposed method in obtaining optimal HRC task planning policy, three other methods are considered: DQN-SARL method, naive Nash-Q learning method, and dynamic programming (DP) method, a commonly used value iteration algorithm in dealing with MDP problems. For DQN-SARL method, all allocations of human and robot workers are controlled by one agent, e.g., at each state s_t, the action a_t includes all the combinations and allocations of the available agents to the tasks in the bottom row of the chessboard. The corresponding reward $Q_t(s_t, a_t)$ is generated by a CNN with the same parameters as that for DQN-MARL method. For naive Nash-Q leaning method, as we mentioned in Section 2.4.2, the Q-values of each agent are saved in a Q table. For DP method, all possible actions of agents for each state are considered and evaluated in a centralized manner. The actions are optimized by backing up estimates of the optimally evaluated state values, which are always saved by updating the evaluation function.

In this section's case study, 100 chessboards are randomly generated with different initial completion time of the first 36 and the entire 53 tasks. Although the initial task structure of the chessboard is not changed, the completion time of

an agent to finish the task is randomly generated following the normal distribution $\mathcal{N}(T_u^i, \sigma^2)$. After certain training periods, the entire completion time for assembling the desk tends to be stable for each individual algorithm. Therefore, the policy is considered to be obtained for each algorithm, and will be used for online task scheduling on additional randomly constructed 100 chessboards.

First, four aforementioned algorithms are compared in the task scheduling for the first 36 tasks and the entire 53 tasks given 3 humans and 3 robots. The completion time during the training process for the four algorithms are shown in Figs. 2.12 (a) and (c). After the training periods, the trained policies are used for online implementation of the first 36 tasks and the entire 53 tasks for the four algorithms, as shown in Figs. 2.12 (b) and (d). From Figs. 2.12 (a) and (b), it is clear that DQN-based algorithms (i.e., DQN-SARL and DQN-MARL methods) outperform the naive Nash-Q method and the DP method in terms of both the computational efficiency during the training period and the optimal completion time during the online task scheduling. Although there is an obvious difference in the effective convergent time between DQN-SARL and DQN-MARL during the training period, the difference of the online completion time is not significant between the two methods. However, with the number of tasks increasing to 53 as shown in Figs. 2.12 (c) and (d), the DQN-MARL method clearly outperforms the DQN-SARL method in terms of both the training period needed in reaching a steady result and the optimal completion time during the online scheduling. Therefore, the proposed DQN-MARL method outperforms other three methods in both computational efficiency and the effectiveness for online optimizing the task scheduling, and this advantage becomes more notable with the increasing number of tasks.

Figures 2.12. Diagrams of the offline training (a), (c), (e) and corresponding online implementation process (b), (d), (f) with 3 humans and 3robots for the first 18, 36 and the whole 53 tasks using dynamic programing (square dot red line) naive Nash-Q (long dashed orange line), DQN-based SARL (solid blue line) and DQN-based MARL (dashed green line) method.

Next, we compare the four algorithms with different number of robot agents. Fig. 2.13 illustrates the training period of the entire 53 tasks with 1 human and an increasing number of robots from 1 to 4. It can be seen that more time is needed to find an optimal task scheduling policy with increasing number of agents. The proposed DQN-MARL method can still find an optimal policy in a shorter time comparing with the other three methods. Since an optimal task scheduling policy for any algorithm will take full advantage of agents' working time, fewer agents may lead to a scenario close to that of single agent due to the full utilization of agents. When the number of agents increases, the advantage of the DQN-MARL becomes more significant, as shown in Figs. 2.13 (a)-(d). It is noted that when

number of agents $n = 4$ and $n = 5$, it is hard for the non-DQN-based method to find an optimal scheduling policy. Moreover, it can be seen that the proposed DQN-MARL method is more effective and computationally efficient than the DQN-SARL method with the increasing number of agents.

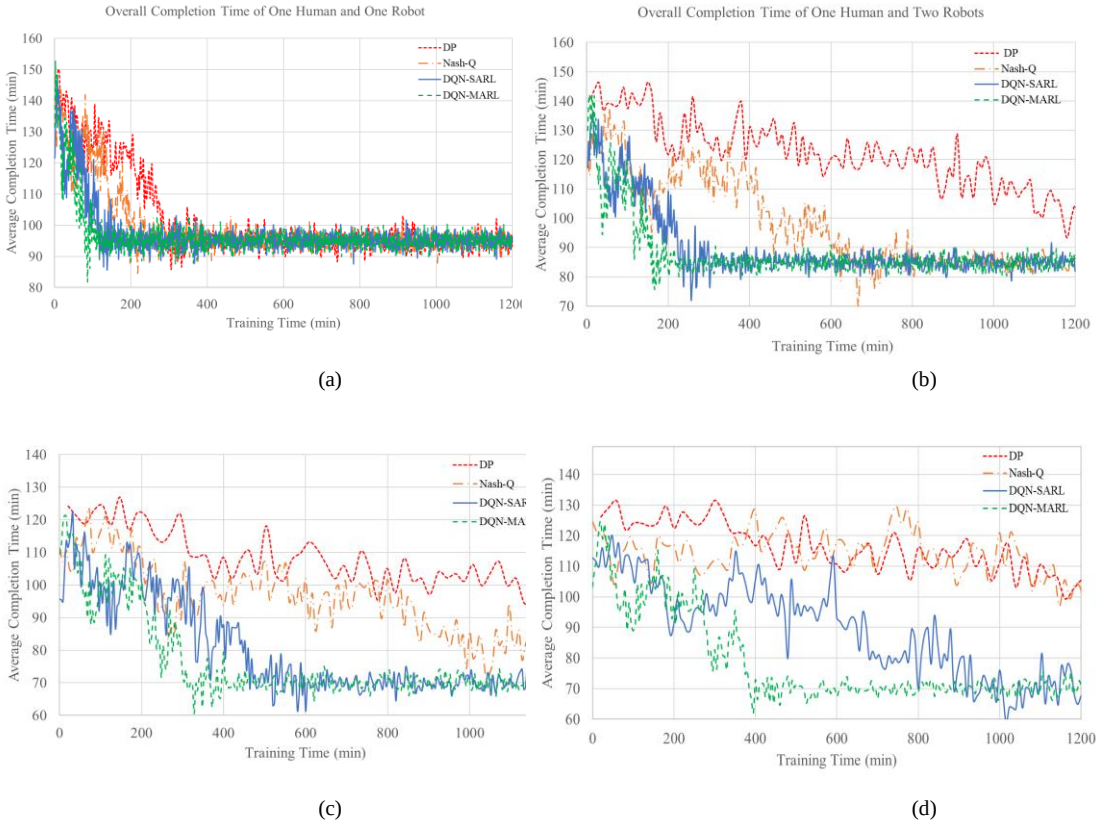
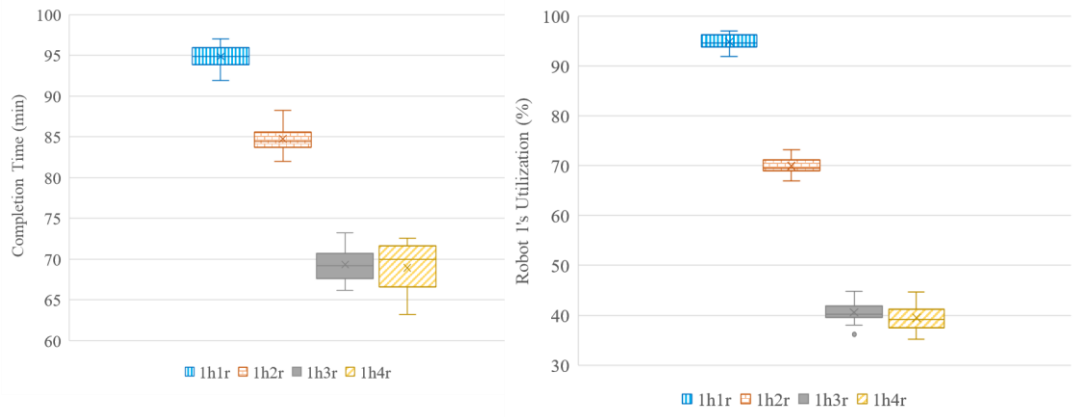


Figure 2.13. Training process of four methods with (a) one-human-one-robot, (b) one-human-two-robots, (c) one-human-three-robots and (d) one-human-four-robots.

In addition, we investigate the agent's utilization by implementing the trained DQN-MARL policy for increasing number of robot agents from 1 to 4. As shown in Figs. 2.14 (a) and (b), with the increasing number of robot agents, both the overall completion time and the Robot 1's utilization decrease as expected. It is also noticed that the completion time and the Robot 1's utilization of the one-human-three-robot scenario are nearly the same as that for the one-human-four-robot scenario. Therefore, for this product assembly, one-human-three-robot is the

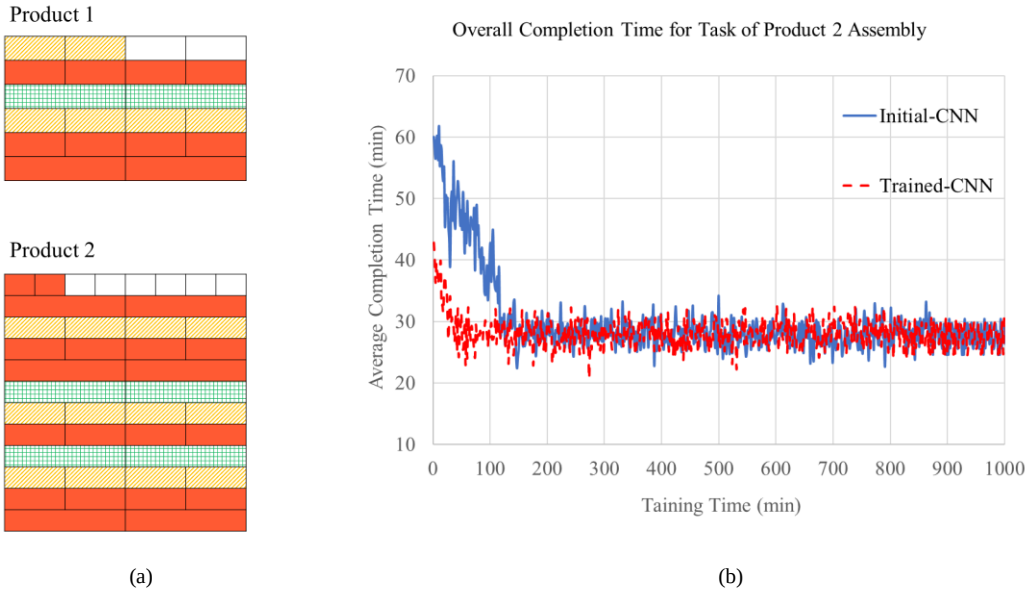
most efficient setting to complete the entire assembly tasks with minimum time. Additional robots are not necessary and costly.



(a) (b)

Figure 2.14. (a) Completion time of 100 online implementations under one-human-one-robot (1h1r), one-human-two-robot (1h2), one-human-three-robot (1h3r) and one-human-four-robot (1h4r) scenarios using DQN-MARL method. (b) utilization of the 100 online implementations using DQN-MARL method.

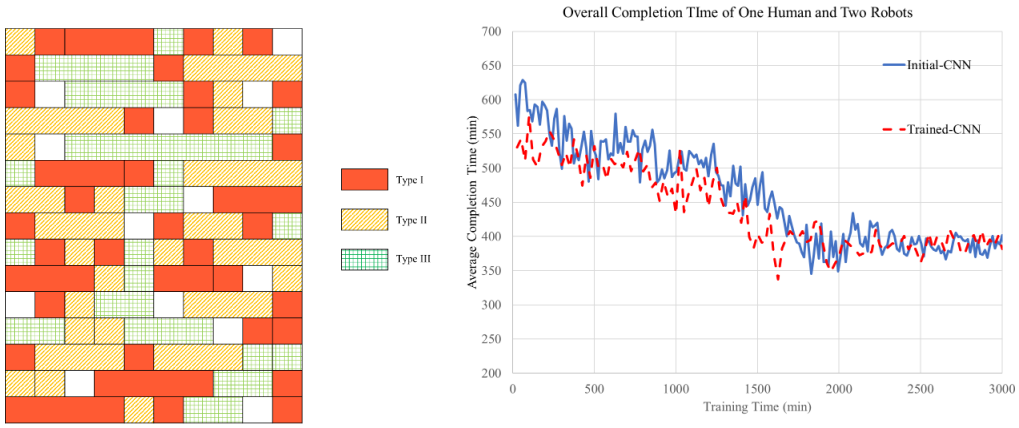
It is worth noting that previous experiments are based on the assembly of a real height-adjustable desk with simple task relations among tasks. One main concern is the effectiveness of the proposed method in a more general case especially for the aforementioned tasks with mixed logic relations. Therefore, we extend the case study with a chessboard containing 81 tasks generated randomly with various position and size of three types of tasks as shown in Fig. 2.15 (a). Agents are set to be 1 human and 2 robots. We compare the training process with and without the trained CNN obtained from previous case study with entire 53 tasks. From the training process shown in Fig. 2.15 (b), we can conclude that the DQN-MARL method is still effective in obtaining the optimal task scheduling policy for a general case with complex task structures.



(a) (b)

Figure 2.15. Training process of the first 18 tasks (a) and the first 36 tasks (b) with the trained CNN (red dotted line) and the initial CNN (blue line).

After the effectiveness of the DQN-MARL method is demonstrated given a general case, the effectiveness of the proposed method for similar products assembly are further explored. First, as shown in Fig. 2.16 (a) the chessboards of two products that contain similar task structures are constructed with the first 18 and 36 tasks. For the training process, 100 chessboards are randomly generated with different initial completion time of both Product 1 and Product 2. Following the flow chart in Fig. 2.10, after the training process of Product 1, the trained CNN is used for the training process of Product 2. In comparison, a CNN with all initial parameters is used for the training from random scratch of Product 2. From the result shown in Fig. 2.16 (b), one can find that by using parameters trained with Product 1, the time of reaching a steady task scheduling policy can be greatly reduced for the training of Product 2. Therefore, we can conclude that the proposed method is effective in obtaining the optimal task scheduling policy dealing with similar products.



(a) (b)

Figure. 2.16. (a) Diagram of the randomly generated chessboard with mixed task relations. (b) Training process with initial CNN and trained CNN.

To summarize, we can conclude that the proposed DQN-MARL method outperforms DQN-SARL, naive Nash-Q and DP method not only in terms of the computational efficiency of obtaining optimal task scheduling policies during the training period but also in the effectiveness of online optimizing the task scheduling, especially in the scenarios with large number of tasks and agents. In addition, the proposed DQN-MARL method can also be generalized to a fit the tasks with similar task structure or a completely different tasks with complicated task structures.

2.5.Summary

This research aims to solve task scheduling problems in the HRC environment. The HRC assembly is formatted as an assembly chessboard game by defining the task mapping rules and game playing rules. A CNN-MCTS method and a DQN-MARL method that is capable of obtaining correlated equilibrium solutions of task scheduling are developed. It is demonstrated that the proposed methods are effective in obtaining optimal task scheduling policies for complicated task structures and can be generalized to broader similar products assembly. Through extensive case studies, it becomes evident that the proposed Multi-Agent Reinforcement Learning (MARL) approach outperforms alternative machine

learning algorithms, including deep SARL, MARL, and baseline RL, especially when handling a large number of tasks involving multiple humans and robots.

2.6.Related Work

Part of the results presented in this chapter have been published in [16], [17].

Chapter 3. RL-based robot Learning from Demonstration (LfD)

3.1.Background

With the development of Industry 4.0, there is an increasing demand for robots to work adaptively and smartly with humans in industrial settings [54]. However, currently, motions of industrial robots are still largely preprogrammed to perform certain repetitive tasks. When tasks sightly change, robots need to be reprogrammed, which often requires considerable robotic expertise and time. This would significantly impair the efficiency of industrial robots that constantly encounter new tasks, and hence limit their use in quite a few industrial scenarios, e.g., small-batch manufacturing that processes highly customized products [55], [56]. Therefore, how to enable robots to collaborate with non-expert personnel and automatically plan adaptive motions for different tasks is a nontrivial and challenging research problem in today's robotics research [5]–[7]. In this research, our goal is to develop a scalable and adaptive motion planning method to automatically generate motion plans for new robotic manipulation tasks without manually reprogramming robots. To achieve this goal, we propose to represent a task by a sequence of critical constraints, and combine human demonstrations with motion planning algorithms to generate motion plans to fulfill those constraints.

In general, techniques for motion planning and robot control can be divided into Joint-space based approaches and Task-space based approaches [57], [58]. Joint space-based motion planning approaches handle the planning problem and compute the motion directly in the joint space of the robot. The strength of joint space methods lies in finding feasible paths that avoid obstacles [59]. However, handling task constraints in joint-space based planning approaches is quite

complicated [60] because they lead to nonlinear constraints in joint angles.Task space based planning approach is historically older than joint-space based approaches and rose out of the resolved motion rate control (RMRC) in [61]. Related to the task space based planning approaches are the operational space based control approaches [62], where the redundancy resolution may be done at either the velocity level or acceleration level.

In literatures, the use of human demonstration to teach a robot is often referred to as Learning from Demonstration (LfD). An important question in LfD is how to acquire demonstrations for learning, e.g., using vision-based sensors, kinesthetic, and data gloves or master-slave systems [30], [63]. In this research, we opt to use kinesthetic demonstrations for the following considerations. On one hand, for kinesthetic demonstrations, there is no correspondence issue between the kinematic structure of the demonstrating system and the follower robot. On the other hand, learning from kinesthetic demonstrations can potentially benefit from a large variety of existing approaches in learning motion from data, which can be classified as follows: (a) demonstrated trajectory decomposition [64], [65], (b) nonlinear regression techniques [7], [24], [66], and (c) dynamical systems based approach [67]–[69]

Specifically, the trajectory decomposition approaches [64], [65] use spline functions to decompose the trajectories. These methods ignore the noises in the demonstration, which may be nontrivial especially when the motion information is obtained through vision or teleoperation. Nonlinear regression techniques use statistical techniques to incorporate the uncertainty of sensing in the estimation [66]. However, these statistical approaches require multiple demonstrations. Furthermore, neither the statistical approaches nor the spline decomposition takes the kinetic transformations of the underlying task space, i.e., $SE(3)$, into account. The dynamical systems-based approach, or dynamical motion primitives (DMPs), on the other hand, can learn from single examples [67]. However, in these settings, most works assume that there is a dynamical system modeling each degree-of-freedom (DoF) of the end effector. The generalization of DMPs is predicated on

the region of attraction of the dynamical system used. In the case of orientations, there is no clear characterization of the region of attraction of the dynamical system on the group of rigid body rotations, i.e., $SO(3)$.

In this research, we develop a motion planning method that can enable the robot to learn from one or even multiple human demonstrations to generate adaptive motion plans for new manipulation tasks in a certain manufacturing environment. First, we define a syntax to specify the manipulation task in an assembly and loading/unloading scenario considering both explicit task constraints, e.g., critical configurations of the end-effector, and environment constraints, e.g., dimension and location of the obstacle. Next, we build a library to store human demonstrated features which are embedded in screw transformation throughout demonstrations. The same method can also be applied to abstract features of the manipulation task. A criterion to identify semantical similarity of a human demonstration and certain part of the manipulation task is defined. Based on this criterion, the appropriate features of human demonstrations will be mapped to the manipulation task. Therefore, to generate a motion plan for the manipulator to satisfy both explicit and implicit task constraints is equivalent to mapping appropriate features in the library to corresponding parts of the manipulation task. In this work, we formulate the motion planning problem in $SE(3)$ into a Markov Decision Process (MDP) framework and use the Q-learning method to train a general motion planning policy to generate adaptive motion plans in $SE(3)$ for different tasks in the same assembly and loading/unloading environment. Finally, inverse kinematics (IK) is used to calculate corresponding motion plans in the joint space to control the robot to execute learned motion plans.

3.2.Problem Description

In this research, a robot is required to do manipulation tasks with explicit and implicit constraints on end-effector configurations during a motion. We assume that the robot has basic capability to move its end-effector from one configuration

to another in the absence of any constraints. Our goal is to develop a method to enable the robot to use human demonstrations from a few common tasks stored in a scenario specific library and to plan point-to-point motions for other new tasks. The human demonstration library can be formed for each specific working scenarios, such as desk assembly library, warehouse sorting library etc. On human demonstrations, we take (one-time) kinaesthetic demonstration for one type of tasks and the information is observed by the screw transformation throughout the trajectory. Using such screw transformation, the feature of the manipulation task can be abstracted in the task space. To select the appropriate human demonstration to be learnt from for the manipulation task, criteria are built by comparing the screw transformation of the human demonstration and corresponding screw transformation of a few critical configurations of the task. Following the criteria, features of appropriate human demonstrations can be mapped to the new manipulation task in task space.

3.2.1. Mathematical Background

In this research, the joint space or configuration space is represented by $\mathcal{J}$, which is the set of all joint angles of the robot manipulator. $SE(3)$ denotes the Special Euclidean group of 3, which represents the task space contains all rigid body motions (i.e., rotations and translations) [70]. To describe configurations of the end-effector, we adopt dual quaternions since they can encode both rotation and translation in rigid body transformation. A dual quaternion $\boldsymbol{D}$ is defined as [70]:

$$\boldsymbol{D} = \boldsymbol{d}_r + \frac{1}{2}\epsilon \boldsymbol{d}_t \otimes \boldsymbol{d}_r \tag{3.1}$$

where $\epsilon \neq 0$, but $\epsilon^2 = 0$. In this definition, the pure translation of the rigid body is represented by the quaternion $\boldsymbol{d}_t$ which is denoted as:

$$\boldsymbol{d}_t = (0, \hat{\boldsymbol{t}}) \tag{3.2}$$

where $\hat{\boldsymbol{t}} = t_x\hat{\boldsymbol{i}} + t_y\hat{\boldsymbol{j}} + t_z\hat{\boldsymbol{k}}$ is the translation vector in $SE(3)$. The unit quaternion $\boldsymbol{d}_r$ representing the pure rotation of the rigid body can also be expressed as:

$$\boldsymbol{d}_r = \cos\left(\frac{\phi}{2}\right) + \hat{\boldsymbol{n}}\sin\left(\frac{\phi}{2}\right) \tag{3.3}$$

where $\hat{\boldsymbol{n}} = n_x\hat{\boldsymbol{i}} + n_y\hat{\boldsymbol{j}} + n_z\hat{\boldsymbol{k}}$ is a unit vector in $SE(3)$ representing the rotation axis, and ϕ is the rotation angle. Using this $\boldsymbol{d}_r$, any vector $\hat{v}$ can be rotated an angle ϕ about the axis $\hat{\boldsymbol{n}}$ by using the quaternion sandwich $\boldsymbol{d}_r\hat{v}\boldsymbol{d}_r^*$, and $\boldsymbol{d}_r^*$ is the conjugate of $\boldsymbol{d}_r$. For more quaternion manipulations, we refer readers to [71].

3.2.2. Specify Manipulation Tasks for the Robot

First of all, the manipulation task comprehensible to the robot needs to be specified. Some research [72], [73] describes a task, e.g., "open the door", using a symbolic vocabulary based on the high-level Planning and Domain Definition Language (PDDL), e.g., {preposition of the door, precondition: door closed, effect: door open}, to build a task library. These high-level commands lack detailed kinematics information and cannot be directly translated to actionable information for the lower-level robot manipulation. [74] sample reasonable modes for motions of a humanoid robot by specifying the manipulation as the starting and goal configurations, together with detailed transition configurations during the entire task in the task space. However, for a general manipulation task in practice, the specific transition from one configuration to another in both $SE(3)$ and $\mathcal{J}$ is not known to the robot. Therefore, in this research, the manipulation task is specified as a set of critical positions obtained from the known constraints based on task requirements (e.g., a specific goal position) and the environment constraints (e.g., the location of an obstacle), and the associated orientation tolerance for the end effector represented in $SE(3)$ at these positions.

Starting from an initial configuration of the end-effector, a manipulation task $\boldsymbol{TK}$ is defined as a sequence of n critical configurations:

$$\boldsymbol{TK} = \{con_1, con_2, \dots, con_n\} \tag{3.4}$$

where con_i, $i = 1, \dots, n$, is a tuple of two $< \boldsymbol{P}_i, \boldsymbol{\theta}_i >$. In this tuple, $\boldsymbol{P}_i = [x_i, y_i, z_i]^T$ is a vector in $SE(3)$ that specifies the position of con_i, $\boldsymbol{\theta}_i = [\theta_{i1}, \theta_{i2}, \theta_{i3}]^T$, $\theta_{l1} \le \theta_{i1} \le \theta_{u1}, \theta_{l2} \le \theta_{i2} \le \theta_{u2}, \theta_{l3} \le \theta_{i3} \le \theta_{u3}$, is a unit vector

in $SE(3)$ that defines Euler angles of con_i, and $\theta_{l1}, \theta_{l2}, \theta_{l3}, \theta_{u1}, \theta_{u2}, \theta_{u3}$ are the lower and upper bounds for corresponding Euler angles. For this task specification, only a few task-related explicit requirements are given. However, there might be implicit task constraints (e.g., maintain a certain orientation from one configuration to another) need to be complied by the robot. Our goal is to find a motion plan to satisfy all the task requirements.

For example, a task $\boldsymbol{TK}$ of transferring a cup of water shown in Fig. 3.1 can be specified by four critical configurations, $con_1, con_2, con_3, con_4$, as summarized in Table 3.1. In this task, the end-effector is required to move a cup of water from the starting configuration con_1 to the goal configuration con_4 while maintaining the cup upward. It is noted that, to move the cup without spilling water out, the pitch and roll angles of the end-effector need to be constrained within a certain range while the yaw angle does not need to be constrained. Therefore, γ in Table 3.1 can be any number from -2π to 2π, and θ_{i1} and θ_{i2} for each $\boldsymbol{\theta}_i$ are set to be 0 for simplicity. Furthermore, if the orientation of the end effector is required to remain the same during the entire transferring, then γ can be also specified to be 0.

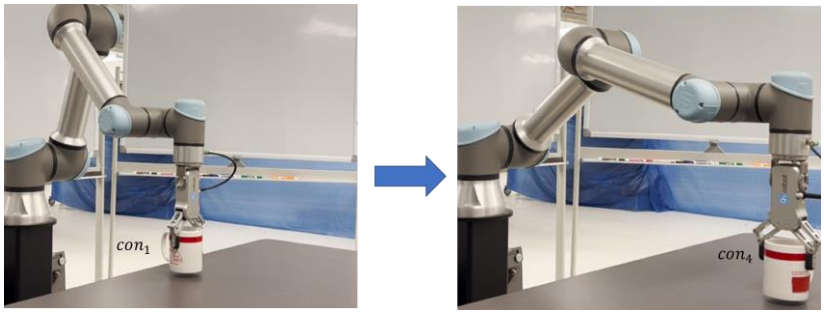


Figure 3.1. Critical configurations of a transferring task. The position of the end-effector is represented by the black dot and the orientation of the end-effector is represented by the triad.

Table 3.1. Critical Configurations of $\boldsymbol{TK}$

	con_1	con_2	con_3	con_4
$\boldsymbol{P}$	(-0.2,0,0.6)	(-0.3,0,0.6)	(-0.4,0,0.6)	(-0.5,0,0.6)
$\boldsymbol{\theta}$	(0,0,γ)	(0,0,γ)	(0,0,γ)	(0,0,γ)

In this research, we only provide a syntax, i.e., Eqn. (3.4), on how to specify a manipulation task. Note that the syntax can help us to automatically translate task requirements to quantitative specifications as shown in Table 3.1. The practical task specification is typically provided by outside sources and is quite a heuristic practice. If the user has sufficient knowledge on the requirements of a new task and the corresponding environment constraints, one may better define the critical positions and the associated end effector orientations. This will help the robot better understand the new task and facilitates a more effective learning in performing the new task.

3.2.3. Build the Library of Human Demonstrated Features

To facilitate the learning from human demonstration, we want to build a library to include primitive or common tasks for specific working scenarios. For example, for a workshop that works on assembling height-adjustable desks, the tasks of twisting screws clockwise and placing screws into assembly holes are common or primitive tasks and can be included in the library for this workshop or working scenario. To demonstrate these primitive tasks, the human operator can physically hold and move the robot's end-effector from the initial configuration to the goal configuration. Throughout the transformation trajectory, the explicit and implicit task constraints embedded in this kinesthetic demonstration can be recorded by many commercial robots such as UR robots in the joint space $\mathcal{J}$ as a time sequence of joint angles $\boldsymbol{\varphi}_{rec}$ as:

$$\boldsymbol{\varphi}_{rec} = \{\boldsymbol{\varphi}(1), \boldsymbol{\varphi}(2), \dots, \boldsymbol{\varphi}(m)\} \tag{3.5}$$

where each $\boldsymbol{\varphi}(i) = [\varphi_1(i), \varphi_2(i), \dots, \varphi_r(i)]^T, i = 1,2, \dots, m$ is the vector representing joint angles of the manipulator, r is the degrees of freedom (DOF) of the manipulator, the order of i is the time sequence of the configurations reached during the motion. Using the forward kinematics mapping $\mathcal{FK}: \mathcal{J} \rightarrow SE(3)$, the corresponding human demonstration in the task space will be obtained as $\boldsymbol{DP} = \{\boldsymbol{D}_1, \boldsymbol{D}_2, \dots, \boldsymbol{D}_m\}$, in which each $\boldsymbol{D}_j$ is a dual quaternion denoted as $\boldsymbol{D}_j =$

$\boldsymbol{d}_{rj} + \frac{1}{2}\epsilon \boldsymbol{d}_{tj} \otimes \boldsymbol{d}_{rj}$, $j = 1,2,\dots,m$, according to Eqn. (3.1). Therefore, $\boldsymbol{DP}$ represents the sequence of configurations of how a task is performed in $SE(3)$.

From the human demonstration $\boldsymbol{DP} = \{\boldsymbol{D}_1, \boldsymbol{D}_2, \dots, \boldsymbol{D}_m\}$, we compute the relative motion with respect to the final end effector configuration. Using the dual quaternion representation, the transformation δ_j between the final configuration (denoted by $\boldsymbol{D}_m$) and every other configuration $\boldsymbol{D}_i$ is:

$$\delta_j = \boldsymbol{D}_j^* \otimes \boldsymbol{D}_m \ , i = 1,2,\dots,m-1 \tag{3.6}$$

where $\boldsymbol{D}_j^*$ is the conjugate dual quaternion of $\boldsymbol{D}_j$ and $\otimes$ is the dual quaternion product. It is noticed that all implicit task constraints are embedded in the sequence of δ_j during the motion, which is referred to as the feature of the human demonstration. As such, the feature of the k^{th} human demonstration in the task space can then be described as:

$$\boldsymbol{HD}_k^{\delta} = \{\delta_1^{HD_k}, \delta_2^{HD_k}, \dots, \delta_{m-1}^{HD_k}\} \tag{3.7}$$

Therefore, the library consisting of h human demonstrated features in the task space can be denoted as:

$$\boldsymbol{LB} = \{\boldsymbol{HD}_1^{\delta}, \boldsymbol{HD}_2^{\delta}, \dots, \boldsymbol{HD}_h^{\delta}\} \tag{3.8}$$

Based on the definition of the manipulation task and the library of human demonstrations, the problem studied in this research can be described as follows: Given a library of human demonstrations $\boldsymbol{LB} = \{\boldsymbol{HD}_1^{\delta}, \boldsymbol{HD}_2^{\delta}, \dots, \boldsymbol{HD}_h^{\delta}\}$ and a new task $\boldsymbol{TK} = \{con_1, con_2, \dots, con_n\}$ in the task space $SE(3)$, develop a method to find motion plan $\boldsymbol{MP}$ in the joint space $\mathcal{J}$ for the new task by learning from human demonstrated features in $\boldsymbol{LB}$ such that the explicit task space constraints specified in each con_i in $\boldsymbol{TK}$ is satisfied. If no $\boldsymbol{MP}$ can be found, then a request for additional human demonstrations is made to satisfy all task-relevant constraints in $\boldsymbol{TK}$.

3.2.4. Build criteria for selecting appropriate human demonstration for the new task

In a previous work [75], we demonstrate a method of learning by demonstration, where the robot is shown the exact demonstration that needs to be learned from. In this research, we use a more advanced and realistic scenario that the human demonstrations are abstracted by features $\boldsymbol{HD}_k^{\delta}$ and stored in a library $\boldsymbol{LB}$. Therefore, when performing a task specified by $\boldsymbol{TK}$, we first need to identify the appropriate one or more $\boldsymbol{HD}_k^{\delta}$ to be learned from.

Let $\boldsymbol{tk}_s \subseteq \boldsymbol{TK}$ be a subset or the entire new task $\boldsymbol{TK}$ as:

$$\boldsymbol{tk}_s = \{con_i, con_{i+1}, \dots, con_w\}, \;\; i \geq 1, i \leq w \leq n \tag{3.9}$$

We need to determine a criterion that can facilitate the comparison of $\boldsymbol{tk}_s$ and a human demonstration $\boldsymbol{HD}_k^{\delta}$. Since $\boldsymbol{HD}_k^{\delta}$ represents the relative motion with respect to the final end effector configuration, in order to compare $\boldsymbol{HD}_k^{\delta}$ with $\boldsymbol{tk}_s$, the relative motion with respect to the final configuration in task $\boldsymbol{tk}_s$ needs to be obtained in a similar way to that of $\boldsymbol{HD}_k^{\delta}$. Based on Eqn. (3.1), the i^{th} configuration of the task $\boldsymbol{tk}_s$ can be written as:

$$\boldsymbol{D}_i = \boldsymbol{d}_{ri} + \frac{1}{2}\epsilon \boldsymbol{d}_{ti} \otimes \boldsymbol{d}_{ri}, i = 1,2, \dots, n \tag{3.10}$$

where $\boldsymbol{d}_{ti} = (0, \boldsymbol{P}_i)$, $\boldsymbol{P}_i$ is the position of the i^{th} critical configuration, $\boldsymbol{d}_{ri}$ is a unit quaternion that can be expressed as:

$$\begin{aligned} \boldsymbol{d}_{ri} = & \cos\left(\frac{\theta_{i3}}{2}\right)\cos\left(\frac{\theta_{i2}}{2}\right)\cos\left(\frac{\theta_{i1}}{2}\right) + \sin\left(\frac{\theta_{i3}}{2}\right)\sin\left(\frac{\theta_{i2}}{2}\right)\sin\left(\frac{\theta_{i1}}{2}\right) + \\ & \left(\sin\left(\frac{\theta_{i3}}{2}\right)\cos\left(\frac{\theta_{i2}}{2}\right)\cos\left(\frac{\theta_{i1}}{2}\right) - \cos\left(\frac{\theta_{i3}}{2}\right)\sin\left(\frac{\theta_{i2}}{2}\right)\sin\left(\frac{\theta_{i1}}{2}\right)\right)\hat{\boldsymbol{i}} + \\ & \left(\cos\left(\frac{\theta_{i3}}{2}\right)\sin\left(\frac{\theta_{i2}}{2}\right)\cos\left(\frac{\theta_{i1}}{2}\right) + \sin\left(\frac{\theta_{i3}}{2}\right)\cos\left(\frac{\theta_{i2}}{2}\right)\sin\left(\frac{\theta_{i1}}{2}\right)\right)\hat{\boldsymbol{j}} + \\ & \left(\cos\left(\frac{\theta_{i3}}{2}\right)\cos\left(\frac{\theta_{i2}}{2}\right)\sin\left(\frac{\theta_{i1}}{2}\right) - \sin\left(\frac{\theta_{i3}}{2}\right)\sin\left(\frac{\theta_{i2}}{2}\right)\cos\left(\frac{\theta_{i1}}{2}\right)\right)\hat{\boldsymbol{k}} \end{aligned} \tag{3.11}$$

where $\boldsymbol{\theta}_i = (\theta_{i1}, \theta_{i2}, \theta_{i3})$ is the orientation of the i^{th} critical configuration of the end-effector. Therefore, the transformation, $\delta_i^{tk_s}$, between the last critical configuration con_n and any other critical configuration con_i is defined as:

$$\delta_i^{tk_s} = \boldsymbol{D}_i^* \otimes \boldsymbol{D}_n, i = 1, \dots, n-1 \tag{3.12}$$

As such, the feature of the task $\boldsymbol{tk}_s$ can be represented as $\boldsymbol{tk}_s^{\delta} = \{\delta_1^{tk_s}, \delta_2^{tk_s}, \dots, \delta_{n-1}^{tk_s}\}$.

Based on Eqn. (3.1), using dual quaternions $\boldsymbol{u}_j$ and $\boldsymbol{v}_i$ to represent $\delta_j^{HD_k}$ and $\delta_i^{tk_s}$, the similarities between them are evaluated by the closeness/difference with respect to both rotation and translation. The closeness of their rotation can be evaluated using Euclidean distance as:

$$\alpha(\boldsymbol{u}_j, \boldsymbol{v}_i) = \min\{\|\boldsymbol{d}_r^{\boldsymbol{u}_j} - \boldsymbol{d}_r^{\boldsymbol{v}_i}\|, \|\boldsymbol{d}_r^{\boldsymbol{u}_j} + \boldsymbol{d}_r^{\boldsymbol{v}_i}\|\} \tag{3.13}$$

The difference between the translation direction of the δ_i and $\delta_j^{tk_s}$ is evaluated as the dot product of the normalized translation vector as:

$$\beta(\boldsymbol{u}_j, \boldsymbol{v}_i) = \frac{\boldsymbol{d}_t^{\boldsymbol{u}_j}}{|\boldsymbol{d}_t^{\boldsymbol{u}_j}|} \cdot \frac{\boldsymbol{d}_t^{\boldsymbol{v}_i}}{|\boldsymbol{d}_t^{\boldsymbol{v}_i}|} \tag{3.14}$$

Given tolerances Δ_α and Δ_β, if $\alpha(\boldsymbol{u}_j, \boldsymbol{v}_i) \leq \Delta_\alpha$ and $\beta(\boldsymbol{u}_j, \boldsymbol{v}_i) \geq \Delta_\beta$, then these two transformations are referred to as being "semantically similar".

According to a previous work [75], if two transformations are sufficiently close within a certain tolerance (the practical value is about 10 degrees according to [75]) in the task space, the corresponding solutions in the joint space using inverse kinematics can be uniquely determined by a small increment in joint angles. Since the human demonstration guarantees the feasible solution in the joint space without violating any joint limits, if the new task is "semantically similar" to a certain human demonstration based on Eqn. (3.13) and Eqn. (3.14), by learning from that human demonstration, the solution in the joint space can be

ensured. Therefore, a criterion of semantic similarity between a task and a demonstration can be defined.

Definition 1. Let p be the number of critical configurations in a task feature $\boldsymbol{tk}_s^{\delta}$, the task feature $\boldsymbol{tk}_s^{\delta}$ and a human demonstrated feature $\boldsymbol{HD}_k^{\delta}$ are semantically similar, denoted as $\boldsymbol{tk}_s^{\delta} \propto \boldsymbol{HD}_k^{\delta}$, if the same number of p configurations on $\boldsymbol{HD}_k^{\delta}$ can be allocated such that the following criterion is satisfied:

$$\forall \boldsymbol{v}_i \in \boldsymbol{tk}_s^{\delta}, i = 1, \dots, p, \exists \delta_l \in \boldsymbol{HD}_k^{\delta} \rightarrow \alpha(\boldsymbol{u}_j, \boldsymbol{v}_i) \leq \Delta_{\alpha} \text{ and } \beta(\boldsymbol{u}_j, \boldsymbol{v}_i) \geq \Delta_{\beta} \quad (3.15)$$

where $l = i, \dots, m$ and m is the last configuration in $\boldsymbol{HD}_k^{\delta}$.

For example, for the same task $\boldsymbol{TK}$ shown in Fig. 3.1, four critical configurations can be specified in Table 3.2 and corresponding features can be derived in Table 3.3 using Eqn. (3.12). Suppose there exists a library $\boldsymbol{LB}$ that includes features $\boldsymbol{HD}_1^{\delta}$ and $\boldsymbol{HD}_2^{\delta}$ summarized in Table 3.4, we want to compare these features with the feature of the task $\boldsymbol{TK}$. From Table 3.3 and Table 3.4, we find that, for each $\delta_i^{\boldsymbol{TK}}, i = 1,2$, we can find a δ_j in $\boldsymbol{HD}_1^{\delta}$, such that $\alpha(\boldsymbol{u}_j, \boldsymbol{v}_i) = 0$ and $\beta(\boldsymbol{u}_j, \boldsymbol{v}_i) \geq 0$. For $\delta_3^{\boldsymbol{TK}}$, we find δ_1 in $\boldsymbol{HD}_2^{\delta}$, such that $\alpha(\boldsymbol{u}_j, \boldsymbol{v}_i) = 0$ and $\beta(\boldsymbol{u}_j, \boldsymbol{v}_i) \geq 0$. Given $\Delta_{\alpha} = 0.5$ and $\Delta_{\beta} = 0$, based on Definition 1, the feature of the segment between con_1 and con_3 for task $\boldsymbol{TK}$ is semantically similar to that of $\boldsymbol{HD}_1$ and the feature of the segment between con_3 and con_4 for task $\boldsymbol{TK}$ is semantically similar to that of $\boldsymbol{HD}_2$.

Table 3.2. Critical Configurations of $\boldsymbol{TK}$

	con_1	con_2	con_3	con_4
$\boldsymbol{P}$	(-0.2,0,0.6)	(-0.3,0,0.6)	(-0.4,0,0.6)	(-0.5,0,0.6)
$\boldsymbol{\theta}$	(0,0,0)	(0, 0,0)	(0,0,0)	$(0,0,-\pi/2)$

Table 3.3. Features of $\boldsymbol{TK}$

	$\delta_1^{\boldsymbol{TK}}$	$\delta_2^{\boldsymbol{TK}}$	$\delta_3^{\boldsymbol{TK}}$
$\boldsymbol{d}_t^{\delta^{TK}}$	(0,1,0,0)	(0,1,0,0)	(0,1,0,0)
$\boldsymbol{d}_r^{\delta^{TK}}$	(1,0,0,0)	(1,0,0,0)	$(0.7,0,0,-0.7)$

Table 3.4. Features of Human Demonstrations

		HD_1	HD_2
δ_1	$d_t^{\delta_1}$	(0,1,0,0)	(0,0.7,0,-0.7)
	$d_r^{\delta_1}$	(0.7,0,0,-0.7)	(0.7,0,0.7,0)
δ_2	$d_t^{\delta_2}$	(0,1,0,0)	
	$d_r^{\delta_2}$	(0.9,0,0,-0.4)	

3.2.5. Map the feature of the selected human demonstration to the new task

In a previous work [75], to generate the point-to-point motion plan for similar task instances, an imitated path is built by transmitting the human demonstration in the task space to the new goal position and the ScLERP [76] is used to blend any current configuration of the end-effector into the imitated path. However, this method only considers the goal position of the new task instance as the explicit new task constraints and the implicit task constraints may not be satisfied during the blend in motion. Therefore, the method has the limit in performing similar tasks in close adjacent areas of human demonstrations. In this research, we adopt a different approach by using the selected feature $\boldsymbol{HD}_k^{\delta}$ based on the criterion in Eqn. (3.15), and mapping the feature to a new task or a segment of the new task.

Suppose $\boldsymbol{HD}_k$ and a new task $\boldsymbol{tk}_s$ are semantically similar based on Definition 1, i.e., $\boldsymbol{tk}_s \propto \boldsymbol{HD}_k$, we will use mapping $\boldsymbol{mp}_k^s$: $\boldsymbol{HD}_k \rightarrow \boldsymbol{tk}_s$, such that to finish the new task $\boldsymbol{tk}_s$, the robot can learn from $\boldsymbol{HD}_k^{\delta}$. Let $\boldsymbol{HD}_k^{\delta} = \{\boldsymbol{u}_1, \boldsymbol{u}_2, \dots, \boldsymbol{u}_{m-1}\}$, $\boldsymbol{tk}_s^{\delta} = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_{n-1}\}$, the first step is to align the translation vector $\boldsymbol{d}_t^{\boldsymbol{u}_1}$ in $\boldsymbol{HD}_k^{\delta}$ to the translation vector $\boldsymbol{d}_t^{\boldsymbol{v}_1}$ in $\boldsymbol{tk}_s^{\delta}$. Based on Eqn. (3.3), a unit quaternion $\boldsymbol{Q}$ representing rotating the vector $\boldsymbol{d}_t^{\boldsymbol{u}_1}$ to the vector $\boldsymbol{d}_t^{\boldsymbol{v}_1}$ is defined as:

$$\boldsymbol{Q} = \cos\left(\frac{\phi_{\boldsymbol{Q}}}{2}\right) + \hat{\boldsymbol{n}}_{\boldsymbol{Q}} \sin\left(\frac{\phi_{\boldsymbol{Q}}}{2}\right) \tag{3.16}$$

where $\hat{\boldsymbol{n}}_{\boldsymbol{Q}} = \frac{\boldsymbol{u}_1 \times \boldsymbol{v}_1}{|\boldsymbol{u}_1 \times \boldsymbol{v}_1|}$ and $\phi_{\boldsymbol{Q}} = \cos^{-1} \frac{\boldsymbol{u}_1 \cdot \boldsymbol{v}_1}{|\boldsymbol{u}_1||\boldsymbol{v}_1|}$. Since $\boldsymbol{HD}_k$ and $\boldsymbol{tk}_s$ are semantically similar, each $\boldsymbol{d}_t^{\boldsymbol{u}_1}$ of δ_j in $\boldsymbol{HD}_k, j = 1,2, \dots, m-1$ can be rotated using the

quaternion sandwich $\boldsymbol{Q}\boldsymbol{d}_t^{\boldsymbol{u}_1}\boldsymbol{Q}^*$. Based on Eqn. (3.16), we can align end-effector's translation encoded in the human demonstrated feature to that of the new task. Then, we can scale the vector $\boldsymbol{Q}\boldsymbol{d}_t^{\boldsymbol{u}_1}\boldsymbol{Q}^*$ with $\frac{|\boldsymbol{d}_t^{\boldsymbol{u}_1}|}{|\boldsymbol{d}_t^{\boldsymbol{v}_1}|}$. The final quaternion $\boldsymbol{d}_{tj}^{\boldsymbol{mp}_{ks}}$ that maps each end-effector's translation $\boldsymbol{d}_t^{\boldsymbol{u}_j}$ in $\boldsymbol{HD}_k$ to $\boldsymbol{tk}_s$ can be obtained as:

$$\boldsymbol{d}_{tj}^{\boldsymbol{mp}_{ks}} = \left(0, \frac{|\boldsymbol{d}_t^{\boldsymbol{u}_1}|}{|\boldsymbol{d}_t^{\boldsymbol{v}_1}|}\boldsymbol{Q}\boldsymbol{d}_t^{\boldsymbol{u}_j}\boldsymbol{Q}^*\right), j = 1,2,\dots,m-1 \tag{3.17}$$

Since the detailed transformation between con_i and con_{i+1} of the new task are unknown and unspecified, and more importantly, due to the rational in Remark 2, we can just use all the intermediate transformation in δ_i for the transformation between con_i and con_{i+1}. Let $\boldsymbol{d}_{rj}^{\boldsymbol{mp}_{ks}} = \boldsymbol{d}_r^{\boldsymbol{u}_j}, j = 1,2,\dots,m-1$, the mapping $\boldsymbol{mp}_{ks}$ can finally be derived as:

$$\boldsymbol{mp}_{ks} = \boldsymbol{d}_{\boldsymbol{r}}^{\boldsymbol{mp}_{ks}} + \frac{1}{2}\boldsymbol{d}_{\boldsymbol{t}}^{\boldsymbol{mp}_{ks}} \otimes \boldsymbol{d}_{\boldsymbol{r}}^{\boldsymbol{mp}_{ks}} \tag{3.18}$$

where $\boldsymbol{d}_{\boldsymbol{r}}^{\boldsymbol{mp}_{ks}} = \{\boldsymbol{d}_{r1}^{\boldsymbol{mp}_{ks}}, \boldsymbol{d}_{r2}^{\boldsymbol{mp}_{ks}}, \dots, \boldsymbol{d}_{rm-1}^{\boldsymbol{mp}_{ks}}\}$, $\boldsymbol{d}_{\boldsymbol{t}}^{\boldsymbol{mp}_{ks}} = \{\boldsymbol{d}_{t1}^{\boldsymbol{mp}_{ks}}, \boldsymbol{d}_{t2}^{\boldsymbol{mp}_{ks}}, \dots, \boldsymbol{d}_{tm-1}^{\boldsymbol{mp}_{ks}}\}$. Let $\boldsymbol{TK} = \{tk_1, tk_2, \dots, tk_p\}$, the final motion plan $\boldsymbol{MP}$ in task space for $\boldsymbol{TK}$ can be determined by

$$\boldsymbol{MP} = \boldsymbol{D}_n \otimes \boldsymbol{mp}_{ks}^*, s = 1,2,\dots,p \tag{3.19}$$

If the requirements of all $\boldsymbol{tk}_s$ of a new task $\boldsymbol{TK}$ can be covered by features of human demonstrations in $\boldsymbol{LB}$, then it is theoretically possible that there exists a set $\{\boldsymbol{HD}_{\boldsymbol{i}}^{\boldsymbol{\delta}}, \dots, \boldsymbol{HD}_{\boldsymbol{l}}^{\boldsymbol{\delta}}\} \subseteq \boldsymbol{LB}$, from which the robot can learn to finish $\boldsymbol{TK}$ by using mapping. This is made more rigorous in Theorem 1.

Theorem 1. Given a library of human demonstrations $\boldsymbol{LB} = \{\boldsymbol{HD}_{\boldsymbol{1}}^{\boldsymbol{\delta}}, \boldsymbol{HD}_{\boldsymbol{2}}^{\boldsymbol{\delta}}, \dots, \boldsymbol{HD}_{\boldsymbol{h}}^{\boldsymbol{\delta}}\}$ and a new task $\boldsymbol{TK} = \{\boldsymbol{tk}_1, \boldsymbol{tk}_2, , \dots, \boldsymbol{tk}_p\}$ in SE(3), it is always possible to find a set $\boldsymbol{HD}^{\boldsymbol{\delta}} = \{\boldsymbol{HD}_i^{\delta}, \dots, \boldsymbol{HD}_l^{\delta}\} \subseteq \boldsymbol{LB}$ such that $\forall\, \boldsymbol{tk}_s \subseteq \boldsymbol{TK}, s = 1,2,\dots,p, \exists \boldsymbol{HD}_k^{\delta} \in \boldsymbol{HD}^{\boldsymbol{\delta}}$ such that $\boldsymbol{tk}_{\boldsymbol{s}} \propto \boldsymbol{HD}_{\boldsymbol{k}}^{\boldsymbol{\delta}}$, if and only if $\forall\, \boldsymbol{tk}_s \subseteq \boldsymbol{TK}, s = 1,2,\dots,p, \exists \boldsymbol{HD}_j^{\delta} \in \boldsymbol{LB}$ such that $\boldsymbol{tk}_{\boldsymbol{s}} \propto \boldsymbol{HD}_j^{\delta}$

Proof

If $\forall\, \boldsymbol{tk}_s \subseteq \boldsymbol{TK}, s = 1,2,\dots,p, \exists \boldsymbol{HD}_j^{\delta} \in \boldsymbol{LB}$ such that $\boldsymbol{tk}_s \propto \boldsymbol{HD}_j^{\delta}$, then we can always find a set that includes $\boldsymbol{HD}_j^{\delta}$, i.e., $\boldsymbol{HD^{\delta}} = \{\boldsymbol{HD}_i^{\delta}, \dots, \boldsymbol{HD}_l^{\delta}\} \subseteq \boldsymbol{LB}$ that satisfy the above condition. To prove the sufficiency, suppose we can find a a set $\boldsymbol{HD^{\delta}} = \{\boldsymbol{HD}_i^{\delta}, \dots, \boldsymbol{HD}_l^{\delta}\} \subseteq \boldsymbol{LB}$, that $\forall\, \boldsymbol{tk}_s \subseteq \boldsymbol{TK}, s = 1,2,\dots,p$, $\exists \boldsymbol{HD}_k^{\delta} \in \boldsymbol{HD^{\delta}}$ such that $\boldsymbol{tk_s} \propto \boldsymbol{HD}_k^{\delta}$. Since $\boldsymbol{HD}_k \in \boldsymbol{HD^{\delta}} = \{\boldsymbol{HD}_i^{\delta}, \dots, \boldsymbol{HD}_l^{\delta}\} \subseteq \boldsymbol{LB}$, then sufficient condition must be true. ■

In the next section, we will discuss how to find a set $\boldsymbol{HD^{\delta}} = \{\boldsymbol{HD}_i^{\delta}, \dots, \boldsymbol{HD}_l^{\delta}\}$, meaning find the motion planning by learning from $\boldsymbol{HD}$ in the task space. This problem is formulated into a Markov Decision Process (MDP) problem and solved with Q-learning.

3.3. Obtaining the Optimal Motion Plan Trough Reinforcement Learning

To find a set of appropriate features in $\boldsymbol{LB}$ that can satisfy all task-relevant constraints in $\boldsymbol{TK}$, one has to go through all subsets $\boldsymbol{tk}_s$ of $\boldsymbol{TK}$ and evaluate each $\boldsymbol{HD}_k^{\delta}$ in the library $\boldsymbol{LB}$, which is an NP-hard problem [77]. Assume that there are x features stored in the library $\boldsymbol{LB}$ and y subsets of the new task TK, the computational complexity for searching exhaustively is $O(x^y)$, which would be huge if the number of human demonstrations and the constraints of the new task is large. For example, the library of "Assemble a height-adjustable desk" have 10 features including flipping, twisting, passing, etc. A new task "Place the screw into the assembly hole and fasten the screw" can have 20 subsets (such as passing horizontally to certain position, then change direction to another position, etc.). In this case, the computational complexity for exhaustive search is $O(10^{20})$. Therefore, we formulate the problem into an MDP and solve the problem using a model-free reinforcement learning (RL) algorithm.

3.3.1. MDP Formulation of the Problem

The most common framework for RL is MDP, which is a stochastic process that models the sequential decision making in uncertain environments. There are three components in an MDP, including state s, action a and reward function r. In a RL framework, an agent's objective is to find a policy π so as to maximize the sum of discounted expected rewards

$$v(s,\pi) = \sum_{t=0}^{T} \gamma^t E(r_t|\pi, s_0 = s) \tag{3.20}$$

where $v(s,\pi)$ is the value for state s under the policy π. Here $\pi = (\pi_0, \dots, \pi_t, \dots)$ is defined over the entire process. The standard solution to the problem above is through the $Bellman$ equation:

$$v(s,\pi^*) = \max_a [r(s,a) + \gamma \sum_{s'} p(s'|s,a) v(s',\pi^*)] \tag{3.21}$$

where $r(s,a)$ is the reward for taking action a at state s, s' is the next state, and $p(s'|s,a)$ is the probability of transiting to state s' after taking action a in state s. A solution π^* that satisfies the above equation is guaranteed to be an optimal policy. Before we can apply RL algorithms to obtaining the ultimate motion planning policy π^*, we need to first properly define the three key components s_t, a_t and r_t at time step t.

For a state space $\mathcal{S}(t)$, that contains the current configuration of the end-effector and all task segments, the state $s_t \in \mathcal{S}(t)$, is defined as:

$$s_t = [\boldsymbol{CF}_t, \boldsymbol{tk}_t] \tag{3.22}$$

where $\boldsymbol{CF}_t$ is the current configuration of the end-effector at t, $\boldsymbol{tk}_t = \{con_j, con_{j+1}, \dots, con_n\}$ is the subset of $\boldsymbol{TK}$ containing task constraints that the robot is going to satisfy.

Given a state s_t, all legal actions of identifying semantical similarities between human demonstrations and task segments form an action space $\mathcal{A}(s_t)$. The action $a_t \in \mathcal{A}(s_t)$ can be defined as:

$$a_t = [a_1(t), a_2(t), \dots, a_h(t)] \tag{3.23}$$

where each $a_i(t)$ is to identify a subsequent critical point con_k that $\boldsymbol{tk}'_t = \{con_j, \dots, con_k\} \subseteq \{con_j, con_{j+1}, \dots, con_n\}$, such that $\boldsymbol{tk}'_t \propto \boldsymbol{HD}_i$. Then $a_i(t)$ will take the index value of k. Therefore, $a_i(t)$ is defined as:

$$a_i(t) = \begin{cases} k, & \text{if } \boldsymbol{tk}'_t \propto \boldsymbol{HD}_i \\ 0, & \text{otherwise} \end{cases} \tag{3.24}$$

To evaluate the action at t, we apply the semantical similarity criteria defined in Section 3.2. The reward function r_t is defined as:

$$r_t = \begin{cases} -\sum_{l=j}^{k} \alpha\left(\delta_i, \delta_l^{\boldsymbol{tk}'_t}\right), & \text{if } \boldsymbol{tk}'_t \propto \boldsymbol{HD_i} \\ -\infty, & \text{otherwise} \end{cases} \tag{3.25}$$

where $\delta_l^{\boldsymbol{tk}'_t}$ is the l^{th} feature of the task segment $\boldsymbol{tk}'_t$. It is noted that after mapping, end-effector translation encoded in human demonstrated features will be aligned to the task segment while end-effector rotational features of demonstrations are preserved. Therefore, we only calculate the difference in end effector rotation between the selected human demonstration and the task segment to reward semantic similar demonstrations.

3.3.2. Applying Q-learning to Obtain the Optimal Motion Planning Policy

In order to obtain the optimal policy π^*, various algorithms have been proposed in the past, among which Q-learning is one of the most widely used algorithms [78]. The basic idea of Q-learning is that we can define a function Q:

$$Q(s, a) = r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) v(s', \pi) \tag{3.26}$$

such that $v(s,\pi^*) = \max_a Q^*(s,a)$. If we know $Q^*(s,a)$, then the optimal policy π^* can be found by simply identifying the action that maximizes $Q^*(s,a)$ under the state s. Starting with arbitrary initial values of $Q(s,a)$, the updating procedure of Q-learning is:

$$Q_{t+1}(s_t,a_t) = (1-\alpha_t)Q_t(s_t,a_t) + \alpha_t[r_t + \gamma \max_a Q_t(s_{t+1},a_t)] \tag{3.27}$$

where $\alpha_t \in [0,1)$ is the learning rate and $\gamma \in (0,1)$ is the discount factor. The training process is shown in Algorithm 3.1. After the training, the ultimate policy π^* is determined as:

$$\pi^*(a|s) = \begin{cases} 1, & \text{if } a = \arg\max_{a' \in A(s)} \{Q(s,a')\} \\ 0, & \text{otherwise} \end{cases} \tag{3.28}$$

where $A(s)$ is the set of all legal actions at state s in the Q-table $Q(s,a)$. The final motion plan $\boldsymbol{MP}$ is generated following the Algorithm 3.2. The overall framework of robot motion planning based on Q-learning by learning from human demonstrations is illustrated in Fig. 3.2.

Algorithm 3.1 Training of the RL-based Motion Planner in $\boldsymbol{SE}(3)$

Procedure 1 Mapping Features of the Human Demonstration to the New Task

Input: $\boldsymbol{tk}_s$, $\boldsymbol{HD}_k$

Output: $\boldsymbol{mp}_{ks}$

Initialize $\boldsymbol{mp}_{ks}$ with zeros

Compute each $\boldsymbol{v}_i$ using Eqn. (3.12)

Compute the rotation quaternion $\boldsymbol{Q}$ using Eqn. (3.16)

For $k = 1,2,\ldots,m-1$ **do**

Update each $\boldsymbol{d}_{tj}^{\boldsymbol{mp}_{ks}}$ using Eqn. (3.17)

$\boldsymbol{d}_{rj}^{\boldsymbol{mp}_{ks}} \leftarrow \boldsymbol{d}_r^{\boldsymbol{u}_j}$

End For

Compute $\boldsymbol{mp}_{ks}$ using Eqn. (3.18)

Output $\boldsymbol{mp}_{ks}$

End Procedure

Procedure 2 Training Process of Q-learning

Input: $\boldsymbol{TK}$, $\boldsymbol{LB}$, ϵ, γ

Output: $Q(s,a)$

Initialize $Q(s,a)$ randomly

Initialize $t = 0$

Initialize s_0 with $\boldsymbol{CF}_0 = \boldsymbol{D}_1^{tk_1}$ and $\boldsymbol{tk}_0 = \boldsymbol{TK}$

For $episode = 0,1,\dots,100$ **do**

While the last con_n of $\boldsymbol{TK}$ is not reached **do**

Choose a_t using policy derived from $Q(s,a)$ (e.g., ϵ-greedy)

Take action a_t, observe r_t

If $\boldsymbol{tk}_t' \propto \boldsymbol{HD_i}$ based on Definition 1

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[r_t + \gamma \max_{a'} Q(s_{t+1},a') - Q(s_t,a_t)\right]$$

$i \leftarrow$ index of the non-zero term of a_t

Invoke Procedure 1 with $\boldsymbol{HD}_i$ and $\boldsymbol{tk}_t'$ as inputs

$\boldsymbol{CF}_{t+1} \leftarrow \boldsymbol{D}_n^{tk_t'}$

$\boldsymbol{tk}_{t+1} \leftarrow \{con_{a_i(t)}, \dots, con_n\}$

$s_t \leftarrow s_{t+1}$

$t \leftarrow t+1$

Else

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[r_t + \gamma \max_{a'} Q(s_{t+1},a') - Q(s_t,a_t)\right]$$

Break

End if

End While

End For

Output $Q(s,a)$

End Procedure

Algorithm 3.2 Generating Motion Plan in $\boldsymbol{SE}(3)$

Input: $\boldsymbol{TK}$, $\boldsymbol{LB}$, $\epsilon, \gamma, Q(s,a)$

Output: $\boldsymbol{MP}$

Initialize s_0 with $\boldsymbol{CF}_0 = \boldsymbol{D}_1^{tk_1}$ and $\boldsymbol{tk}_0 = \boldsymbol{TK}$

For $t = 0,1,\dots,T$ **do**

Find legal action list $A(s_t)$ from $Q(s_t,a) \rightarrow a \in A(s_t)$

Find the optimal action as $a_t = \arg\max_{a \in A(s_t)} Q(s_t,a)$

Map $\boldsymbol{HD}_i$ to $\boldsymbol{tk}_t'$ according to a_t

$MP_t \leftarrow \boldsymbol{D}_n \otimes \boldsymbol{mp}_{ks}^*$ according to Eqn. (3.19)

End For

$\boldsymbol{MP} \leftarrow \{MP_0, MP_1, \dots, MP_T\}$

Output $\boldsymbol{MP}$

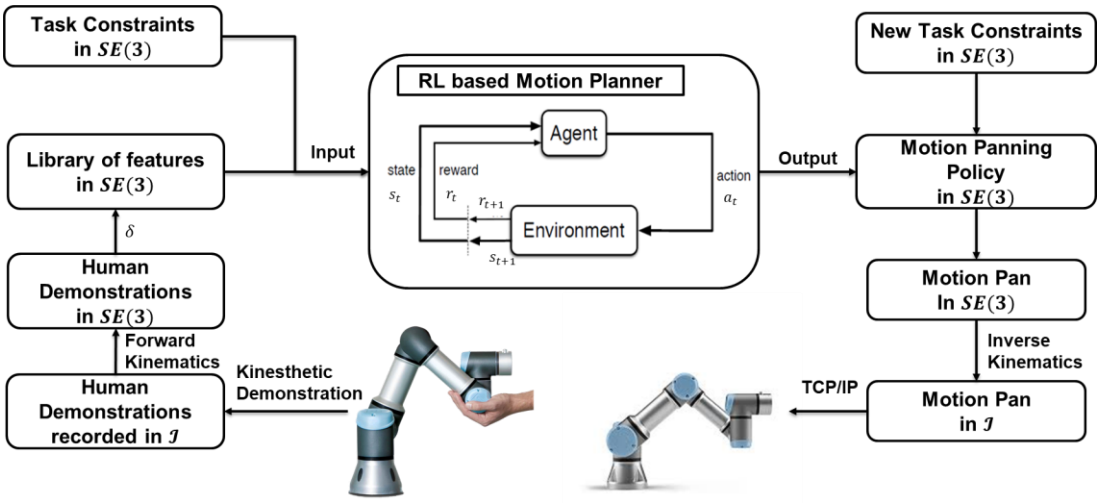


Figure 3.2. Framework of the RL based user-guided motion planning.

3.3.3. Computing the Motion Plan in the Joint Space

After the motion plan $\boldsymbol{MP}$ in the task space is generated, the corresponding motion plan in the joint space needs to be calculated through inverse kinematics. Based on a previous work [75], to calculate the sequence of joint angles in the joint space is to solve the equation

$$\dot{\boldsymbol{q}} = (\mathbf{J}^s)^T(\mathbf{J}^s(\mathbf{J}^s)^T)^{-1}\mathbf{J}_2\begin{bmatrix}\dot{\boldsymbol{p}}\\ \dot{\boldsymbol{r}}\end{bmatrix} \tag{3.29}$$

where $\dot{\boldsymbol{q}}$ is the joint rate vector, $\boldsymbol{p}$ is the position of the end-effector, $\boldsymbol{r}$ is the quaternion representing the orientation of the end-effector, $\mathbf{J}^s$ is the spatial Jacobian and $\mathbf{J}_2 = \begin{bmatrix}\mathbf{I}_{3\times3} & 2\boldsymbol{p}\mathbf{J}_1\\ \mathbf{0}_{3\times3} & 2\mathbf{J}_1\end{bmatrix}$ where $\mathbf{J}_1$ is the matrix transformation of the spatial angular velocity of the rigid body. Let $\dot{\boldsymbol{\gamma}} = [\dot{\boldsymbol{p}}\ \dot{\boldsymbol{r}}]^T$ and $\boldsymbol{B} = (\mathbf{J}^s)^T(\mathbf{J}^s(\mathbf{J}^s)^T)^{-1}\mathbf{J}_2$, Eqn. (3.29) can also be written as:

$$\dot{\boldsymbol{q}} = \boldsymbol{B}\dot{\boldsymbol{\gamma}} \tag{3.30}$$

Using Euler time-step to discretize this equation where h is a small time step,

$$\frac{\boldsymbol{q}(t+h) - \boldsymbol{q}(t)}{h} = \boldsymbol{B}\frac{\boldsymbol{\gamma}(t+h) - \boldsymbol{\gamma}(t)}{h} \tag{3.31}$$

or

$$\delta \boldsymbol{q} = \boldsymbol{B}\big(\boldsymbol{\gamma}(t+h) - \boldsymbol{\gamma}(t)\big) \tag{3.32}$$

In such way, for two consecutive configurations of the end-effector $\boldsymbol{\gamma}(t)$ and $\boldsymbol{\gamma}(t+h)$, the value of $\delta \boldsymbol{q}$ is determined. Therefore, starting from $\boldsymbol{q}(0)$, the joint angles $\boldsymbol{q}(t)$ at any time step t can be obtained.

In practice, an unavoidable problem when executing motion plans in the joint space is the joint limits. In previous studies, researchers usually handle joint limits with optimization or Nullspace of the Jacobian [79]. However, in this research, we still have limitations in handling joint limits because task constraints in $SE(3)$ always have the higher priority than joint limit constraints in $\mathcal{J}$. Since we map the feature of the normalized human demonstration to the new task in the task space, the motion plan obtained by Q-learning in $SE(3)$ cannot guarantee feasible solutions in $\mathcal{J}$ that do not violate joint limits. There're also some other methods in handling joint limits [80], we will explore these options in our future work.

3.4.Experiments and Validation

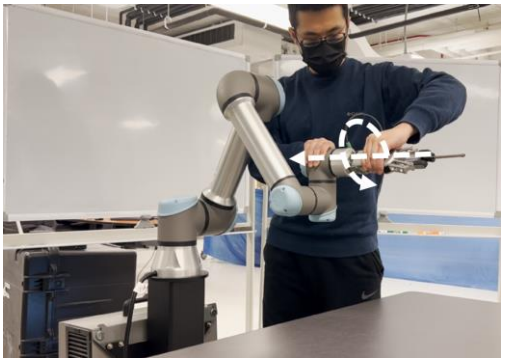
In order to validate effectiveness of the proposed method in generating motion plans for new tasks, multiple experiments are conducted on the UR5e platform. Our experiments consist of four steps: (1) Build an illustrative library of features of human demonstrations; (2) Specify new tasks in $SE(3)$; (3) Offline train the RL-based Motion Planner in $SE(3)$ and obtain motion plans in $\mathcal{J}$; (4) Execute motion plans in $\mathcal{J}$. In this case study, two performance metrices are considered: (1) The accumulated reward of the motion plan in $SE(3)$; (2) The successful rate of applying the motion plan in $\mathcal{J}$ for new tasks. From the case study, two significant results can be concluded: (1) The proposed RL-based user-guided motion planning method can benefit from sufficient knowledge in the proposed task specification scheme; (2) The proposed method is effective in combining different features of human demonstrations to generate motion plans for the new task; (3)

The proposed method is effective in requesting additional human demonstrations if no features in the library are semantically similar to the new task.

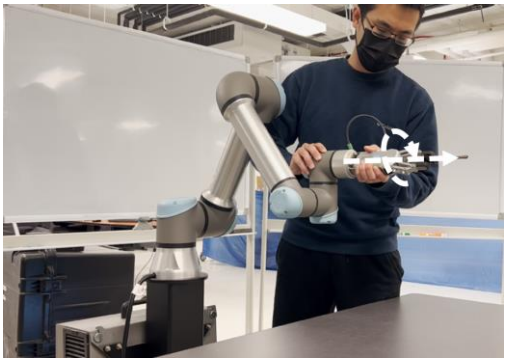
3.4.1. Setting up a Library of Features of User Demonstrations

To build a library including some common features in certain assembly and loading/unloading environment, we start with five illustrative tasks as shown in Fig. 3.3. The tasks are recorded in joint space $\mathcal{J}$ through kinesthetic demonstrations, including:

1) Screwing Task 1: Twist a screw driver 90 degrees clockwise;
2) Screwing Task 2: Twist a screw driver 90 degrees anti-clockwise;
3) Filling Task: Hold a cup horizontally and then turn down 90 degrees (representing certain orientation constraints);
4) Pouring Task: Hold a cup vertically and then turn up 90 degrees (representing certain orientation constraints);
5) Stacking Task: Stack one block from an initial location to a goal location;



(a)



(b)

(c)

(d)

(e)

Figure 3.3. Kinesthetic demonstrations of 5 most common tasks. (a) Screwing Task 1: The end-effector is required to twisting the screw driver 90 degrees clockwise while moving straight forward to the goal position. (b) Screwing Task 2: The end-effector is required to twisting the screw driver 90 degrees anti-clockwise while moving straight forward to the goal position. (c) Pouring Task: The end-effector is required to hold the cup horizontally at first, then turn down 90 degrees. (d) Filling Task: The end-effector is required to hold the cup vertical to the ground at first, then turn up 90 degrees. (e) Stacking Task: The end-effector is required to stack the small block up the big block. The orientation of the small block should be kept upward during the stacking.

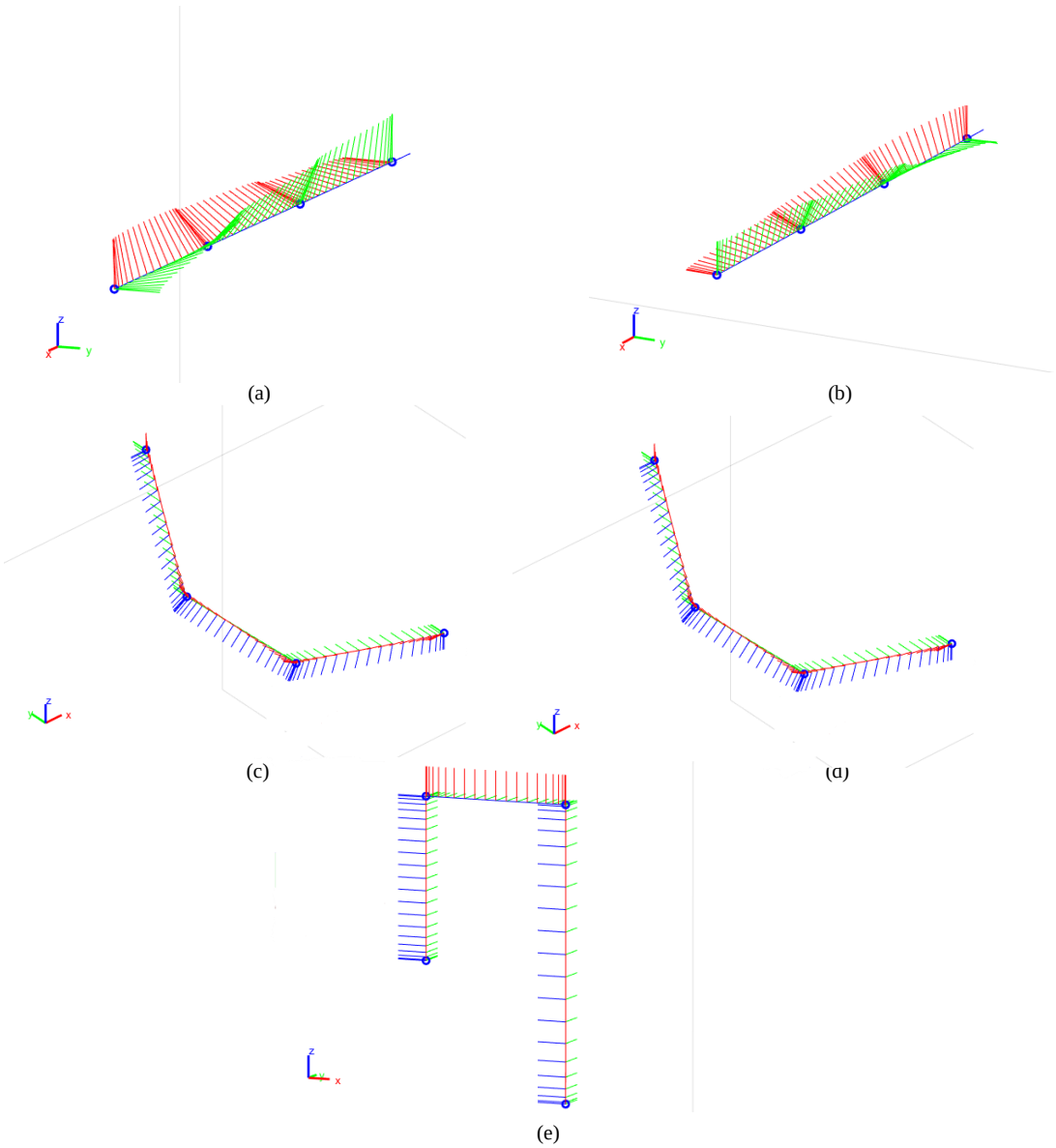


Figure 3.4. Human demonstrations in $SE(3)$. (a) Screwing task 1. (b) Screwing task 2. (c) Pouring task. (d) Filling task. (e) Stacking task.

Using forward kinematics, corresponding configurations of end-effector in $SE(3)$ are shown in Fig. 3.4. Using Eqn. (3.8), features of human demonstrations are saved as $\boldsymbol{LB} = \{\boldsymbol{HD}_1, \boldsymbol{HD}_2, \dots, \boldsymbol{HD}_5\}$ in the library shown in Table 3.5.

Table 3.5. Library of human demonstrated features

		HD_1	HD_2	HD_3	HD_4	HD_5
δ_1	$d_t^{\delta_1}$	(0,1,0,0)	(0,1,0,0)	(0,0.7,0,-0.7)	(0,-0.7,0,0.7)	(0,0.7,0,-0.7)
	$d_r^{\delta_1}$	(0.7,0,0,-0.7)	(0.7,0,0,0.7)	(0.7,0,0.7,0)	(0.7,0,-0.7,0)	(1,0,0,0)
δ_2	$d_t^{\delta_2}$	(0,1,0,0)	(0,1,0,0)	(0,0.5,0,-0.8)	(0,-0.8,0,0.5)	(0.0.7,0,0.7)
	$d_r^{\delta_2}$	(0.9,0,0,-0.4)	(0.8,0,0,0.5)	(0.8,0,0.5,0)	(0.8,0,-0.5,0)	(1,0,0,0)
δ_3	$d_t^{\delta_3}$	(0,1,0,0)	(0,1,0,0)	(0,0.2,0,-0.9)	(0,-0.9,0,0.2)	(0,0,0,1)
	$d_r^{\delta_3}$	(1,0,0,-0.2)	(0.9,0,0,0.3)	(0.9,0,0.2,0)	(0.9,0,-0.2,0)	(1,0,0,0)

3.4.2. Training the RL-based Motion Planner in SE(3)

In order to obtain a general motion planning policy in $SE(3)$ for the assemble and loading/unloading scenario with the features library, we implement Algorithm 3.1 to train a Q-table initialized with random Q values in Matlab using a 4-core 4.0GHz Intel Core i7 processor. Parameters for training are listed in Table 3.6. 20 new tasks are used during the training, each of which has four critical configurations. Positions of these critical configurations are randomly generated within a $50 \times 50 \times 50\ cm^3$ workspace. Corresponding Euler angles of each critical configuration are also randomly selected from a set $\{-\pi, -\frac{\pi}{2}, 0, \frac{\pi}{2}, \pi\}$. The total training episode for each new task is set to be 100. The total computation time is 1927.42 seconds.

To monitor the training process, accumulated rewards for each new task are recorded every two iterations. The average accumulated reward for all 20 new tasks is shown in Fig. 3.5. Although the training rewards are noisy before 50 episodes, the underlying trend is that the rewards are increasing with training episodes. It can be observed that the reward reaches a steady level after around 50 episodes. This indicates that a steady motion planning policy in $SE(3)$ that can map appropriate features of human demonstrations to new tasks with semantically similar features is generated for the assemble and loading/unloading scenario.

Table 3.6. Parameters for Training

Parameters	$\triangle \alpha$	$\triangle \beta$	γ	ε
Value	0.5	-0.9	0.9	0.8

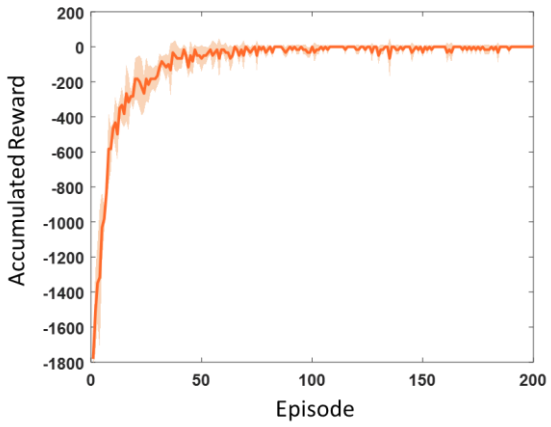


Figure 3.5. Training process for motion plans in $SE(3)$

3.4.3. Evaluation of the Trained Motion Planning Policy

To evaluate the performance of the trained motion plan policy in $SE(3)$ for the assemble and loading/unloading scenario, three new tasks, namely, a transferring task, a filling-and-pouring task, and an assembling task, are used as examples to demonstrate the method. The trained Q-table is used as the input to Algorithm 3.2 to generate motion plans for new unseen tasks in $SE(3)$. The inverse kinematics is used to calculate the final motion plan in joint space $\mathcal{J}$. For each task, 20 experiment trials are conducted to evaluate the successful executions.

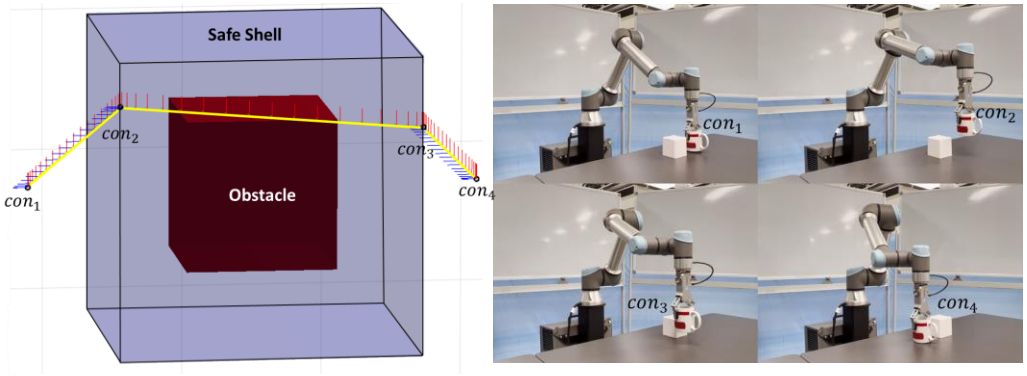
Transferring Task: In this task, the end-effector is required to transfer a cup of water while avoiding an obstacle shown as red cube in Fig. 3.6. The dimension of the obstacle is $20 \times 20 \times 20\ cm^3$ and the position of its center is $(-0.3, -0.1, 0.2)$. To avoid this obstacle, a safety protocol is assumed given as a $40 \times 40 \times 40\ cm^3$ safety shell (shown as transparent purple cube in Fig. 3.6 with the same center position as the cubic obstacle) that the manipulator cannot penetrate through. Knowing these environment constraints and the task requirement on moving the cup of water from the starting position to the goal position, a sample of user defined critical configurations, $con_1, con_2, con_3, con_4$

are summarized in Table 3.7, where $0.1 \leq z \leq 0.5$. In these critical configurations, con_1 and con_4 describe the end-effector starting and the ending positions and orientations, con_2 and con_3 are two intermediate critical configurations selected on edges of the safety shell. Note that different users may have different task specifications depending on their understandings of the task and environment constraints.

The result shows that all 20 trials are successfully executed, where each of the 20 feasible motion plans (shown as the path composed of the small triads) complies with the task requirement that the end-effector maintains the same orientation of $(0, -\pi/2, 0)$ to prevent the spill out. One of the 20 motion plans are demonstrated as the yellow line in Fig. 3.6 (a), where the orientations of the small triads along each path represent the orientations of the end-effector. Execution of the corresponding motion plan in the joint space is shown in Fig. 3.6 (b). It is noticed that the feature of the human demonstrated stacking task is learned and mapped for this transferring task. In this experiment, both explicit task constraints (position and orientation constraints of critical configurations) and implicit task constraints (keeping the orientation of the end-effector) are satisfied. This experiment demonstration the effectiveness of the trained motion plan policy, which can be used as motion plans to avoid the obstacles in $SE(3)$ if users have sufficient knowledge about the task and the environment and can properly infuse the knowledge in task specification based on the syntax we defined.

Table 3.7. Critical configurations of Task 3

	con_1	con_2	con_3	con_4
$\boldsymbol{P}$	(-0.5,0,0.3)	(-0.4,0.2,z)	(0,0.2,z)	(0.1,0,0.3)
$\boldsymbol{\theta}$	$(0,-\pi/2,0)$	$(0,-\pi/2,0)$	$(0,-\pi/2,0)$	$(0,-\pi/2,0)$



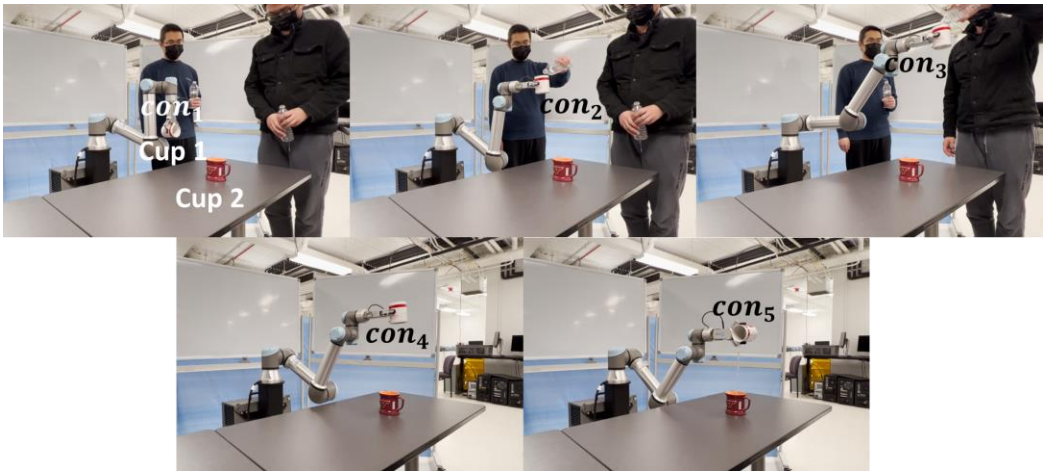
(a) (b)

Figure 3.6. Motion plans for the transferring task. (a) The motion plan in $SE(3)$. (b) Execution the motion plan in $\mathcal{J}$.

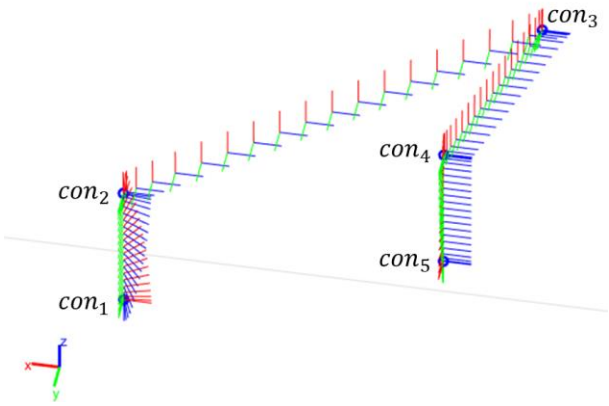
Filling-and-Pouring Task: As shown in Fig. 3.7 (a), in this task, the end-effector is required to fill water to Cup 1, then go through another two critical positions, and reach a goal position above Cup 2, and finally pour water to Cup 2. The location of Cup 2 is on the surface of a desk within a workspace of $20 \times 20\ cm^2$. We can specify 5 critical configurations based on the described task. Sample critical configurations from con_1 to con_5 are presented in Table 3.8, where $-\pi \leq \gamma \leq \pi$, $-0.5 \leq x \leq 0.7$, $-0.2 \leq y \leq 0$. It is noted that, in this task, only the orientation of the initial configuration and goal configuration are specified with specific Euler angles. When moving the cup from con_2 to con_4, the end-effector is only required upward without any specific constraints in the yaw angle.

Table 3.8. Critical configurations of TK

	con_1	con_2	con_3	con_4	con_5
$\boldsymbol{P}$	(-0.4,-0.1,0)	(-0.5,0.1,0.1)	(-0.7,-0.1,0.1)	$(x, y, 0.1)$	$(x, y, 0)$
$\boldsymbol{\theta}$	$(0,-\pi,0)$	$(0, -\pi/2,\gamma)$	$(0, -\pi/2,\gamma)$	$(0, -\pi/2,\gamma)$	$(\pi/2,0, -\pi/2)$



(a)



(b)

Figure 3.7. Generate motion plans for the filling-and-pouring task. (a) The motion plan in $SE(3)$ (b) The final execution of the motion plan in $\mathcal{J}$.

For 20 experiment trials with various locations of Cup 2, all experiments are successfully performed in $\mathcal{J}$. We use the motion plan in $\mathcal{J}$ and $SE(3)$ for one trial as an example as shown in Fig. 3.7 (a) and (b) to illustrate the result. It is noticed that the feature of 3 human demonstrated tasks, namely filling, stacking, and twisting, are learned and mapped to the segment between con_1 and con_2, the segment between con_2 and con_4, and the segment between con_4 and con_5, respectively. In this experiment, the proposed method can identify and compose the appropriate features in the human demonstration library to perform a new task.

Assembling Task: In this task, the end-effector needs to disassemble a screw from Assembly Hole 1, then place the screw into Assembly Hole 2, and finally fasten the screw. The location of Assembly Hole 1 and Assembly Hole 2 are $(0, 0.5, 0.6)$ and $(0.5, 0, 0.6)$ as shown in Fig. 3.8 (a). To transfer the screw from Assembly Hole 1 to Assembly Hole 2, the end-effector is required to hold the screw horizontally and turn 90 degrees anti-clockwise. Based on the task requirement, the critical configurations can be specified in Table 3.9.

Table 3.9. Critical configurations of the assembling task

	con_1	con_2	con_3	con_4
$\boldsymbol{P}$	(0,0.5,0.6)	(0.1,0.5,0.6)	(0.5,0.1,0.6)	(0.5,0,0.6)
$\boldsymbol{\theta}$	$(-\pi/2,0,\pi/2)$	$(0, -\pi/2,0)$	$(0, -\pi/2,\pi/2)$	$(\pi,0, -\pi/2)$

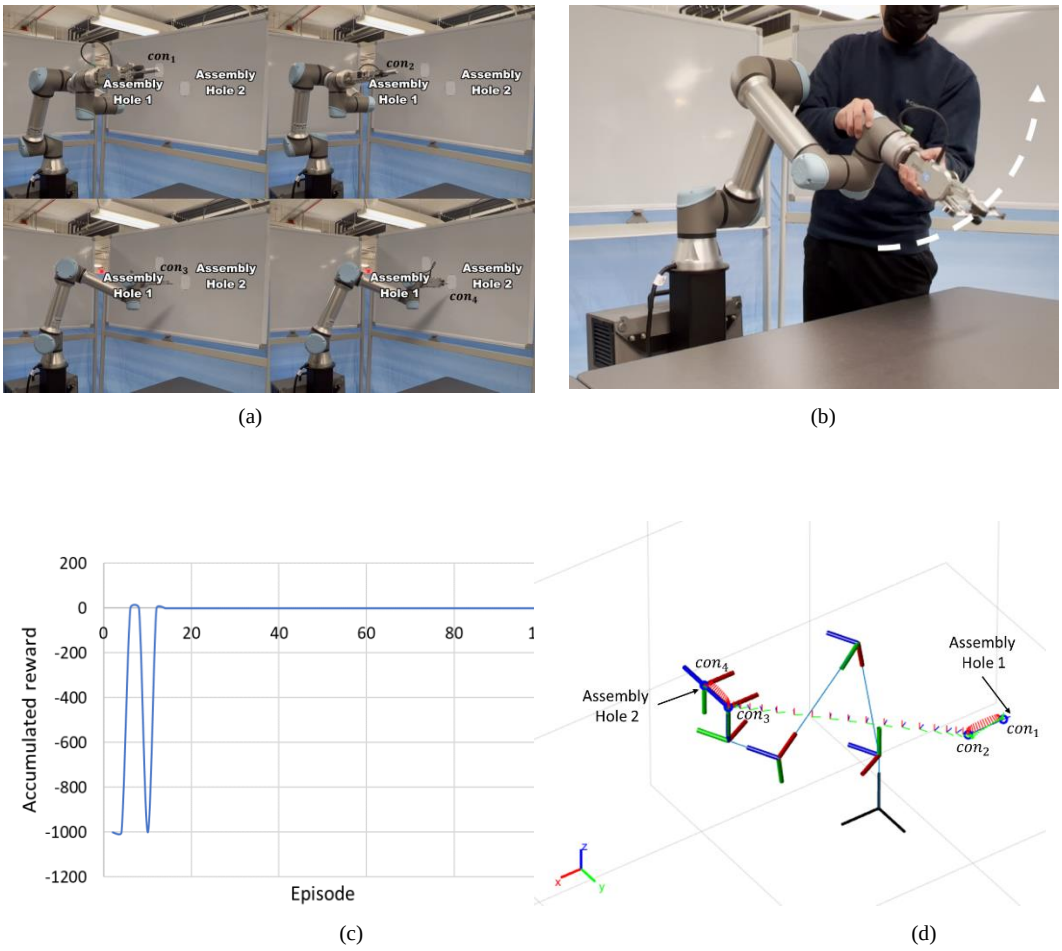


Figure 3.8 Generate motion plans for the assembling task after the feature of a new human demonstration is added to the library. (a) Critical configurations and the motion plan in $\mathcal{J}$ for the assembling task after an additional human demonstration is provided. (b) Additional human demonstration for picking up a span. (c) The training process of the assembling task after the feature of a new human demonstration is added. (d) Motion plan in $SE(3)$.

By applying the same trained general motion plan policy, no successful motion plan can be generated, which indicates additional human demonstrations are needed. A closer examination reveals that none of the five features saved in the library is semantically similar to the feature of the task segment between con_2 and con_3, which requires a 90-degree rotation about its body-fixed x-axis clockwise. Therefore, additional human demonstration is requested for this feature.

With this additional human demonstration shown in Fig. 3.8 (b) added in the library, the motion planning policy is retrained using Algorithm 3.1. As shown in

Fig. 3.8 (c), the accumulated reward reaches a steady value after around 10 iterations. Then by applying the newly trained policy, the motion plan in $SE(3)$ is generated as shown in Fig. 3.8 (d). The result shows that the features of the human demonstrated twisting task 1 and task 2 are learned and mapped to the task segment between con_1 and con_2, and the segment between con_3 and con_4, respectively. The feature of the newly added human demonstration is learned and mapped to the task segment between con_2 and con_3. Corresponding motion plan in $\mathcal{J}$ is shown in Fig. 3.8 (a).

To summarize, the case study results demonstrate the effectiveness of the proposed RL-based user-guided motion planning method in learning and mapping appropriate features of human demonstrations to new tasks and generating motion plans in the joint space for semantically similar tasks. The proposed method can also request additional human demonstrations when new task features cannot be found in the human demonstration library.

3.5.Summary

In this research, we present a novel method for robot learning from human demonstrations based on RL-based motion planning. A task specification scheme is first developed for users to provide necessary kinematic information about task and environment constraints. A human demonstration library for specific working scenarios is built through recording and storing the common actions by utilizing the existing recording capability for modern robots. By abstracting features from human demonstrations and tasks, the task-space RL-based motion planner can effectively identify, learn, and compose the appropriate demonstrated features to perform new tasks that comply with the task requirements and environment constraints. Followed by inverse kinematics, motion plans in joint space can be obtained.

3.6.Related Work

Part of the results presented in this chapter have been published in [31]

Chapter 4. Motion Planning and Task-Oriented Coordination Scheme for Mobile Manipulators

4.1.Background

Over the past decade, the mobile manipulator - a system created by integrating a robot manipulator onto a mobile base - has garnered considerable interest in the realm of industry [81]. As smart manufacturing continues to advance, there is an increasing expectation for robots to operate in dynamic, complex environments in close proximity to humans [82]. Nevertheless, the majority of current industrial mobile manipulators are designed to function solely in structured environments and lack autonomous capabilities [83]. To tackle this limitation, a significant amount of work have been conducted on motion planning of the manipulator [18]–[22] and the mobile base [84], [85]. However, due to the challenges in manipulator motion planning, precise localization of the mobile base and coordination between the mobile base and the manipulator, the implementation of these prevailing methods on mobile manipulators are yet to be fully explored [86].

For robot manipulator motion planning, current works can be categorized into sampling-based methods [18], [19], optimization-based methods [20], [21], and reinforcement-learning (RL) based methods [22]. However, these methods often require expert knowledge or a significant amount of data and computation to generate motion plans. In recent years, learning from demonstration (LfD) has become a promising approach for robot motion planning in human-robot collaboration environments [23]–[26]. However, scalability and adaptivity remain major challenges in current LfD methods. A scalable LfD technique has been proposed by the authors, enabling robots to generate adaptive motion plans from a single human demonstration [87]. However, most LfD methods are developed for

fixed-base manipulators, and joint limits can add further challenges when tasks exceed the manipulator's reachable range.

To handle joint limits of the manipulator, mobile manipulators often use a mobile base to mount the manipulator, which creates kinematic redundancy due to high degrees of freedom (DOF). While this flexibility is useful in complex environments, it complicates the motion planning process. Some existing methods treat the mobile manipulator as a redundant manipulator and calculate inverse kinematics (IK) for both the mobile base and manipulator simultaneously to track the desired end-effector trajectory [88], [89]. Nevertheless, as the DOF of the redundant manipulator increases, determining the exact IK solutions becomes more challenging and time-consuming [90], [91]. Other methods plan the motion of the mobile base and the manipulator separately [92]–[94]. However, how to maintain task constraints while moving the mobile base remains a challenging and unresolved issue.

In addition, due to the kinematics and dynamics of the wheels or other locomotion mechanism, the mobile base cannot move in any arbitrary direction and is subject to certain constraints on its movement. This non-holonomic nature limits the possible velocities and accelerations of the mobile base, which makes it difficult to achieve some types of motions, such as rotations or sideways movements [95]. Existing motion planning methods do not typically account for the notable disparity in accuracy between the mobile base and the manipulator in practical applications. This can create considerable challenges in end-effector locomotion and affect the execution of the task trajectory, especially in a sensor-less environment [86].

To address the aforementioned challenges, this research makes two primary contributions: Firstly, in joint space, a unique task-oriented manipulability is developed, which allows the manipulator's manipulability to be biased in accordance with task constraints (e.g., manufacturing task requirements on specific position). Secondly, a novel joint space coordination scheme for the mobile base and manipulator is developed, which is based on the task-oriented

manipulability and reachability of the manipulator. This coordination scheme not only handles the joint limits of our previous LfD method [87] but also enhances the accuracy of tracking the task end-effector trajectory by avoiding solving high DOF inverse kinematics and optimizing the movement of the mobile base, which is now well addressed in existing works.

4.2.Problem Description

In this research, a mobile manipulator that combines a robot manipulator and a mobile base is employed to finish tasks in a manufacturing environment. The task can be specified as a sequence of n critical configurations in the task space:

$$\boldsymbol{TK} = \{\boldsymbol{con_1}, \boldsymbol{con_2}, \dots, \boldsymbol{con_n}\} \tag{4.1}$$

where $\boldsymbol{con_i}$ is the i^{th} critical configuration that represents task requirements on both the position and orientation of the robot's end-effector. Taking advantage of the authors' previous work [87], the task end-effector trajectory $\boldsymbol{Traj} = \{\boldsymbol{D}'_1, \boldsymbol{D}'_2, \dots, \boldsymbol{D}'_m\}$ can be obtained in the task space by learning from a semantically similar human demonstration $\boldsymbol{DP} = \{\boldsymbol{D}_1, \boldsymbol{D}_2, \dots, \boldsymbol{D}_m\}$, where $\boldsymbol{D}_j$ and $\boldsymbol{D}'_j$ are dual quaternions that encode both the position and orientation of the end-effector.

During the execution of $\boldsymbol{Traj}$ in the joint space, manipulator's joint limits can pose a significant challenge, as the manipulator may not be able to reach configurations beyond its reachable range. To tackle this issue, existing methods usually consider the mobile manipulator as a redundant manipulator and calculate inverse kinematics, $IK: \boldsymbol{D}'_j \rightarrow < \boldsymbol{\theta}_j, \boldsymbol{B}_j >$, where $\boldsymbol{B}_j$ is the joint space configuration of the mobile base, for the entire system. However, since the complexity of the IK equations increases with the DOF of the manipulator, the numerical methods used to solve IK can become less accurate and more time-consuming [90], [91]. This can negatively impact task execution precision. Additionally, the non-holonomic nature of the mobile base can cause errors in end-effector locomotion when simultaneously moving the manipulator and the mobile base to reach the configuration $< \boldsymbol{\theta}_j, \boldsymbol{B}_j >$ [95].

Considering both task execution accuracy and computational efficiency, the goal of this research is to develop a coordination scheme for the mobile base and the manipulator, such that the trajectory obtained from and the movement of the mobile base, $\boldsymbol{MB} = \{\boldsymbol{B}_1, \boldsymbol{B}_2, \dots\}$ within a workspace $\boldsymbol{\mathcal{W}}$ can be optimized to compensate for the manipulator's joint limits. Using the $\boldsymbol{\vartheta}$ to denote the manipulator joint limits and $\boldsymbol{AE} = \sum_{j=1}^{m} \left| \boldsymbol{D}'_{jreal} - \boldsymbol{D}'_j \right|$, where $\boldsymbol{D}'_{jreal}$ is the real configuration of the end-effector, to denote the accumulated error during the task execution, the problem studied in this research can be described as follows: Given a new task specification and task end effector trajectory $\boldsymbol{Traj}$, develop a mobile manipulator coordination scheme to achieve the given new task with the highest accuracy, i.e.,

$$< \boldsymbol{\theta}^*, \boldsymbol{MB}^* >= \arg \min_{\boldsymbol{MB} \in \boldsymbol{\mathcal{W}}} \boldsymbol{AE} \quad s.t. \boldsymbol{\theta} \in \boldsymbol{\vartheta} \tag{4.2}$$

where $\boldsymbol{\theta}^*$ and $\boldsymbol{MB}^*$ are the optimized manipulator and mobile base trajectories obtained by the coordination scheme.

To address this challenge, we propose a mobile manipulator coordination scheme (introduced in Section 4.5) that seamlessly combines the scalability of the manipulator's LfD [87] with the flexibility of the mobile base. Within this coordination scheme, the high-accuracy manipulator's LfD trajectory is kept as much as possible, and the adjustment of the mobile base is triggered only when the manipulator encounters singularities, thereby optimizing the precision of task execution. It is worth noting that the task space LfD method not only captures kinematics features of one-time human demonstration but also incorporates a mapping function to adapt features of demonstration to all semantically similar tasks, which can differ in starting and goal positions. With this approach, the robot gains the capability to generate motion plans for numerous new task instances using just one demonstration. Furthermore, the robot can learn from demonstrations provided to other robots, even when those robots operate in different locations and manufacturing environments.

4.3.Task-oriented Mobile Manipulator Coordination Scheme

4.3.1. End-effector Trajectory Generation in the Task Space

To produce adaptable end-effector trajectories $\boldsymbol{Traj}$ for the task $\boldsymbol{TK}$ in the task space, the authors have opted for a learning from human demonstrations approach that was developed in their previous work [87] due to its scalability and flexibility. In order to make this research self-contained, the fundamental outcomes of the previous work on task space motion planning are presented without delving into technical intricacies.

Given the human demonstration $\boldsymbol{DP} = \{\boldsymbol{D}_1, \boldsymbol{D}_2, \dots, \boldsymbol{D}_m\}$, the transformation, $\boldsymbol{\delta}_i$, between the goal pose and every other pose is:

$$\boldsymbol{\delta}_i = \boldsymbol{D}_{i-1}^{*} \otimes \boldsymbol{D}_m \ , i = 2, \dots, n \tag{4.3}$$

where $\otimes$ represents dual quaternion multiplication and $\boldsymbol{D}^*$ denotes the conjugate of $\boldsymbol{D}$. Note that all implicit task constraints in a human demonstration are embedded in the sequence of $\boldsymbol{\delta}_i$ during the motion. Thus, the sequence of $\boldsymbol{\delta}_i$ represents a sequence of relative transformation and is referred to as a kinematic feature or semantics of a human demonstration:

$$\boldsymbol{HD} = \{\boldsymbol{\delta}_2^{\boldsymbol{HD}}, \dots, \boldsymbol{\delta}_m^{\boldsymbol{HD}}\} \tag{4.4}$$

Similarly, the feature of the task can be calculated as $\boldsymbol{TS} = \{\boldsymbol{\delta}_2^{\boldsymbol{TS}}, \dots, \boldsymbol{\delta}_n^{\boldsymbol{TS}}\}$. The similarity between the demonstrated $\boldsymbol{DP}$, and the new task, $\boldsymbol{TK}$, can then be calculated using similarity function, $\boldsymbol{\alpha}\left(\boldsymbol{\delta}_j^{\boldsymbol{HD}}, \boldsymbol{\delta}_i^{\boldsymbol{TS}}\right)$, which is defined based on Euclidean distance [76]. The semantic similarity criterion between the two is then defined as:

$$\forall \boldsymbol{\delta}_i^{\boldsymbol{TS}} \in \boldsymbol{TS}, \exists \boldsymbol{\delta}_j^{\boldsymbol{HD}} \in \boldsymbol{HD} \rightarrow \boldsymbol{\alpha}\left(\boldsymbol{\delta}_j^{\boldsymbol{HD}}, \boldsymbol{\delta}_i^{\boldsymbol{TS}}\right) \leq \Delta_\alpha \tag{4.5}$$

where Δ_α is a predefined tolerance value that can be determined based on the precision requirements of the specific task. If the human demonstration and the new task are semantically similar based on the criterion, the robot can generate a motion plan for the new task by using the human demonstration as a guidance. This is done by using a mapping function, $\boldsymbol{mp}: \boldsymbol{HD} \rightarrow \boldsymbol{TS}$, which aligns and enforces the features of the human demonstration onto the new task using a

quaternion sandwich operation [96]. Then, the task end-effector trajectory can be obtained as $\boldsymbol{Traj} = \{\boldsymbol{D}'_1, \boldsymbol{D}'_2, \dots, \boldsymbol{D}'_m\}$, where each $\boldsymbol{D}'_{i-1} = \boldsymbol{con_n} \otimes \boldsymbol{\delta}^{*\boldsymbol{HD}}_i, i = 2, \dots, m$, $\boldsymbol{\delta}^{*\boldsymbol{HD}}_i$ is the conjugate of $\boldsymbol{\delta}^{*\boldsymbol{HD}}_i$, and $\boldsymbol{D}'_m = \boldsymbol{con_n}$. For detailed derivation and proof, reader can refer to [87].

4.3.2. Development of Task-oriented Manipulability for the Manipulator

To solve the problem formulated in Section 4.2, in this section, a coordination scheme is developed, which takes advantage of the precise control of the manipulator and reduces the repositioning of the mobile base during the execution of the task. To ensure the task accuracy, the reachability of the manipulator is optimized to cover a maximized portion of the target end-effector trajectory. The mobile base is only mobilized when there is a need to compensate for the joint limits of the manipulator. To further improve the accuracy, a task-oriented manipulability that can bias the manipulability based on the task constraints is defined and compared with the conventional manipulability.

First, for a given end-effector configuration $\boldsymbol{D} \in \mathbb{R}^4$ and a mobile base pose $\boldsymbol{B} \in \mathbb{R}^2$, reachability of the mobile manipulator can be determined by checking the existence of solutions to the inverse kinematics $IK: \boldsymbol{D} \rightarrow \boldsymbol{\theta}\ s.t. \boldsymbol{\theta} \in \boldsymbol{\vartheta}$, which can be defined as:

$$\boldsymbol{Ra}(\boldsymbol{D}, \boldsymbol{B}) = \begin{cases} 1, & \text{if IK solutions exist} \\ 0, & \text{otherwise} \end{cases} \tag{4.6}$$

Therefore, within a 2D workspace $\boldsymbol{\mathcal{W}} \in \mathbb{R}^2$, a feasible region $\boldsymbol{\xi}$ can be defined as:

$$\boldsymbol{\xi} = \{\boldsymbol{\xi} \in \boldsymbol{\mathcal{W}} \mid \forall \boldsymbol{B} \in \boldsymbol{\xi} \rightarrow \boldsymbol{Ra}(\boldsymbol{D}, \boldsymbol{B}) = 1\} \tag{4.7}$$

This feasible region $\boldsymbol{\xi}$ includes all feasible mobile base poses $\boldsymbol{B}$ that make $\boldsymbol{D}$ reachable for the manipulator.

However, if only reachability constraint is considered to determine the desired mobile base pose, the searching space for finding a feasible solution could be

large and the singularity state $\boldsymbol{\theta}^*$ of the manipulator obtained by $IK: \boldsymbol{D} \rightarrow \boldsymbol{\theta}^*, \boldsymbol{B}$ cannot be avoided. To tackle these problems, manipulability is considered as a second constraint. The conventional measurement of manipulability [97] is given as :

$$man(\boldsymbol{\theta}) = \sqrt{\det[\boldsymbol{J}(\boldsymbol{\theta})\boldsymbol{J}^T(\boldsymbol{\theta})]} \tag{4.8}$$

where $\boldsymbol{J}(\boldsymbol{\theta}) \in \mathbb{R}^{m \times n}$ is the Jacobian matrix and $\boldsymbol{J}^T(\boldsymbol{\theta})$ is the transpose of $\boldsymbol{J}(\boldsymbol{\theta})$. This is a metric that quantifies the distance between a manipulator state $\boldsymbol{\theta}$ and a singularity state $\boldsymbol{\theta}^*$. Such a measurement can provide information about the overall movement ability of the end-effector, but it does not take into account the specific requirements of a task. For a specific task, the mobile manipulator is required to move towards the critical configuration, and the movement ability of the end-effector in that direction is more critical than its movement ability in other directions. To address this, a task-oriented manipulability can be defined which reflects the biased movement ability of the end-effector towards the task direction. Based on the critical configuration of the task, a unit vector $\widehat{\boldsymbol{k}} = \frac{\boldsymbol{X}_{con_*} - \boldsymbol{X}_{\boldsymbol{D}_c}}{|\boldsymbol{X}_{con_*} - \boldsymbol{X}_{\boldsymbol{D}_c}|}$ is defined, where $\boldsymbol{X}_{con_*} \in \mathbb{R}^3$ and $\boldsymbol{X}_{\boldsymbol{D}_c} \in \mathbb{R}^3$ are positions of the target critical configuration of the task and the current end-effector configuration. The projection of end effector velocity along the $\widehat{\boldsymbol{k}}$ direction can be written as:

$$e = \boldsymbol{J}(\boldsymbol{\theta})\dot{\boldsymbol{\theta}} \cdot \widehat{\boldsymbol{k}} \tag{4.9}$$

Considering a sphere of joint velocity $\dot{\boldsymbol{\theta}}^T\dot{\boldsymbol{\theta}} = 1$, Eqn. (4.9) can be rewritten as:

$$e^T\left(\widehat{\boldsymbol{k}}^T\boldsymbol{J}(\boldsymbol{\theta})\boldsymbol{J}^T(\boldsymbol{\theta})\widehat{\boldsymbol{k}}\right)^{-1}e = 1 \tag{4.10}$$

Let A_k=$\widehat{\boldsymbol{k}}^T\boldsymbol{J}(\boldsymbol{\theta})\boldsymbol{J}^T(\boldsymbol{\theta})\widehat{\boldsymbol{k}}$, then the value e is limited to a manipulability ellipsoid where lengths of the principal axes are eigenvalues of A_k. Similar as the conventional measurement in Eqn. (4.8), the task-oriented manipulability, $man'(\boldsymbol{\theta})$, is defined as:

$$man'(\boldsymbol{\theta}) = \sqrt{\det\left(\widehat{\boldsymbol{k}}^T\boldsymbol{J}(\boldsymbol{\theta})\boldsymbol{J}^T(\boldsymbol{\theta})\widehat{\boldsymbol{k}}\right)} \tag{4.11}$$

By considering task-oriented manipulability measure as a constraint in the motion planning process, it helps to ensure that the end-effector has sufficient movement ability in the direction required for the task, which can further improve the task trajectory segment covered by the manipulator.

4.3.3. Development of Coordination Scheme of the Mobile Base Motion and the Manipulator Motion

Given the task trajectory $\boldsymbol{Traj} = \{\boldsymbol{D'}_1, \boldsymbol{D'}_2, \dots, \boldsymbol{D'}_m\}$ and considering both the reachability and manipulability of the manipulator, the positions of the mobile base within the feasible region $\boldsymbol{\xi}$ need to be determined, such that for each segment of the target trajectory $\boldsymbol{Seg}_i = \{\boldsymbol{D'}_j, \boldsymbol{D'}_{j+1}, \dots, \boldsymbol{D'}_{j+\mu_i}\}$, the maximum length from $\boldsymbol{D'}_j$ to $\boldsymbol{D'}_{j+\mu_i}$ can be covered by the manipulator. This optimization problem can be formulated as follows:

$$\arg\max_{\boldsymbol{B}\in\boldsymbol{\xi}} \mu, \mu = 1, \dots, \mu_i \quad s.t.\ \boldsymbol{Ra}\left(\boldsymbol{D}_{j+\mu}, \boldsymbol{B}\right) = 1, man'(\boldsymbol{\theta}) > \Delta_{man_k} \quad (4.12)$$

where Δ_{man_k} is the tolerance of the task-oriented manipulability.

To solve this problem, starting from $\boldsymbol{D'}_1$, the solution involves searching within $2r \times 2r$ m^2 square workspace $\boldsymbol{\mathcal{W}}$ with a mesh grid size of $g \times g$ m^2, where r is the length when the manipulator is fully extended. Reachability $\boldsymbol{Ra}\left(\boldsymbol{D'}_j, \boldsymbol{B}\right)$ at each grid point is determined to form the feasible region $\boldsymbol{\xi}$. Within $\boldsymbol{\xi}$, only the grid points in $\widehat{\boldsymbol{k}}$ direction are selected to evaluate for the reachability and task oriented manipulability of the manipulator. Once the specific position $\boldsymbol{B}^* \in \boldsymbol{\xi}$ is found that can maximize the segment trajectory $\boldsymbol{Seg}_i$, the mobile base moves to the position $\boldsymbol{B}^*$ while maintaining the configuration of the end-effector $\boldsymbol{D'}_j$. After the mobile base stops, inverse kinematics is calculated for the manipulator to track $\boldsymbol{Seg}_i$. When the manipulator finishes tracking the segment $\boldsymbol{Seg}_i$, the mobile base is repositioned to maximize the next segment $\boldsymbol{Seg}_{i+1}$ until the goal configuration $\boldsymbol{D'}_m$ is reached. The overall coordination structure is shown in Fig. 4.1 and the coordination algorithm is shown in Algorithm 4.1.

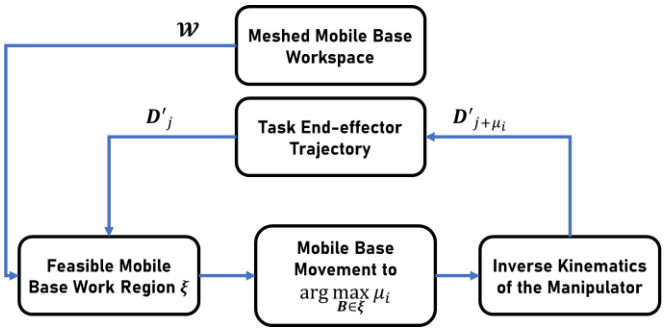


Figure 4.1. Coordination scheme for the mobile base and the manipulator.

Algorithm 4.1 Mobile Manipulator Coordination in the Joint Space

Input: EE trajectory $\{\boldsymbol{D'}_1, \boldsymbol{D'}_2, \dots, \boldsymbol{D'}_m\}$
Output: Sequence of mobile base poses $\boldsymbol{Sq}$, Sequence of joint angles $\boldsymbol{\theta}$
$j = 1$
While $i \leq m$ **do**
  Mesh a workspace $\boldsymbol{\mathcal{W}}$ with the center of $\boldsymbol{D'}_i$ using grid size $g \times g$ m^2
  $\boldsymbol{\xi} = \emptyset$
  For $\boldsymbol{B}$ **in** $\boldsymbol{\mathcal{W}}$
    If $\boldsymbol{Ra}(\boldsymbol{D}_j, \boldsymbol{B}) = 1$
      $\boldsymbol{\xi} = \boldsymbol{\xi} + \boldsymbol{B}$
    End If
  End For
  For $\boldsymbol{B}$ in $\widehat{\boldsymbol{k}}$ direction of $\boldsymbol{\xi}$
    If $man_k(\boldsymbol{\theta}) > \Delta_{man_k}$ do
      $j = j + 1$
    Else
      Break and output i
    End If
    $\boldsymbol{Sq} \leftarrow \{\boldsymbol{Sq}, \boldsymbol{B}\}$
  End For
  IK: $\boldsymbol{D'}_j, \boldsymbol{B} \longrightarrow \boldsymbol{\theta}_j$
  $\boldsymbol{\theta} \leftarrow \{\boldsymbol{\theta}, \boldsymbol{\theta}_j\}$
  Update $\boldsymbol{D}_j$
End While
Output $\boldsymbol{Sq}, \boldsymbol{\theta}$

4.4.Numerical Case Study

In this section, to simulate tasks in manufacturing settings, a material handling task and a painting task are specified in the task space. Simulation experiments are performed on a UR10-Husky virtual test bed shown in Fig. 4.2. The precision of the mobile base locomotion is set to $\pm$0.1m, and the accuracy of the manipulator is set to $\pm$0.001m. Three primitive actions are demonstrated by the

human operator on a UR5e manipulator. The performance of the proposed method is evaluated using two performance metrics: (1) The computing time to obtain the motion plan in the joint space; (2) The accumulated distance error between the target configuration in the end-effector trajectory and the current configuration of the end-effector during the execution, which is defined as:

$$\boldsymbol{AE} = \sum_{t=0}^{T} \left| \boldsymbol{X}_{real}(t) - \boldsymbol{X}_{traj}(t) \right| \tag{4.13}$$

where $\boldsymbol{X}_{real}(t)$ is the real-time position of the end-effector and $\boldsymbol{X}_{traj}(t)$ is the target position of the end-effector in the trajectory generated using LfD methods.

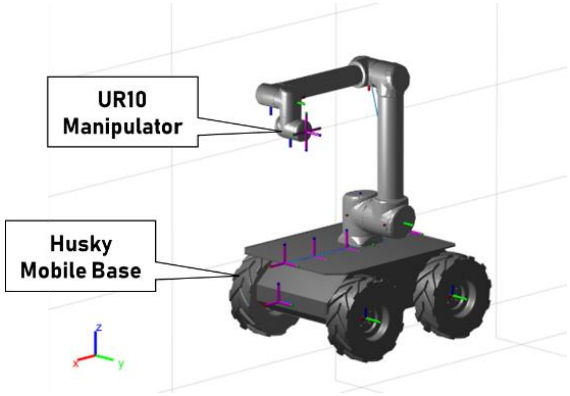


Figure 4.2. UR10-Husky virtual test bed.

In this research, three simulation experiments are conducted to evaluate the effectiveness and robustness of the proposed method. The first experiment explores the feasibility and scalability of the method with two different material handling tasks. The second experiment compares the performance of the proposed method using both conventional manipulability and task-oriented manipulability. The third experiment compares the performance of the proposed method with two other methods: (1) A duplication motion planning method that uses the manipulator to repeat the semantically similar primitive actions demonstrated by the human operator and relies on the mobile base to adjust the manipulator's movement to reach the desired end-effector trajectory for the new task; (2) An overall inverse kinematics method that considers the UR10-Husky mobile robot as an 8-DOF redundant manipulator and calculates inverse kinematics for this

redundant manipulator during the execution of the motion plan in the joint space. From the case study, the following three key conclusions can be drawn: (1) The proposed method is effective in producing adaptive end-effector trajectories through learning from one semantically similar primitive action; (2) The proposed method that uses task-oriented manipulability yields better results in computing time and accuracy compared with the conventional manipulability; (3) The proposed method has demonstrated superior accuracy in executing joint space motion plans compared with the other two methods.

4.4.1. Human Demonstrations

To build a library including some common tasks in a certain manufacturing setting, we start with 3 illustrative demonstrations as shown in Fig. 4.3. The tasks are recorded in the joint space through kinesthetic demonstrations, including one screwing task, one filling task, and one stacking task. Using forward kinematics, $\mathcal{FK}$, the joint space demonstrations are converted to task space trajectories, which are used as the input to the LfD algorithm in the simulation environment.

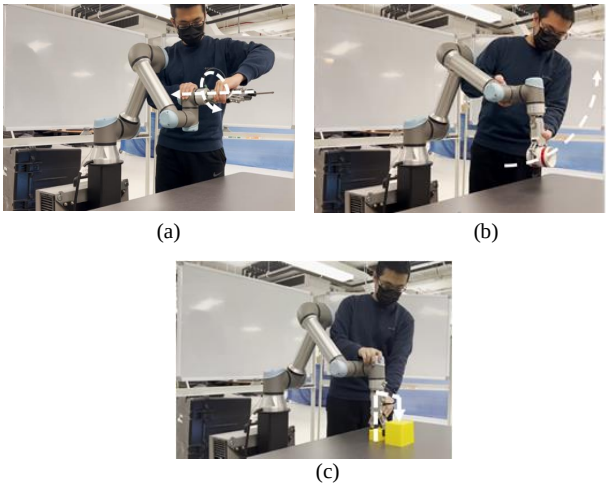


Figure 4.3. Kinesthetic demonstrations of 3 most common tasks. (a) Screwing task. (b) Filling task. (c) Stacking task.

4.4.2. Task Specification and End-effector Trajectories in the Task Space

In order to evaluate the performance of the execution of the motion plan in the joint space, a material handling task and a painting task are specified in Table 4.1 and 2, which are common tasks in manufacturing settings.

Material handling task: This task is designed to analogy material handling in a manufacturing setting, which involves short-distance movement within the confines of a building or between a building and a transportation vehicle. In this task, the end-effector is required to turn up 90 degrees from $\boldsymbol{con_1}$ and $\boldsymbol{con_2}$ and keep the orientation of the end-effector to transfer a cup of water from $\boldsymbol{con_2}$ to $\boldsymbol{con_3}$, then rotate the end-effector 90 degrees to reach $\boldsymbol{con_4}$. Based on the semantical similarity criteria defined in Eqn. (4.5), the feature of the filling task, stacking task and the twisting task will be mapped to the segment from $\boldsymbol{con_1}$ to $\boldsymbol{con_2}$, $\boldsymbol{con_2}$ to $\boldsymbol{con_3}$, and $\boldsymbol{con_3}$ to $\boldsymbol{con_4}$ respectively. The final end-effector trajectory in the task space is shown in Fig. 4.4 (a). Note that this task is designed to mimic potential pick-and-place industrial applications such as pegging a hole, assembling and disassembly a screw, where the end-effector configurations are specified in a certain range.

Painting task: In this task, the end-effector is required to maintain its orientation to paint a square by traveling from $\boldsymbol{con_1}$ to $\boldsymbol{con_5}$. This task is designed to mimic industrial applications such as painting, cutting, or welding through a specific routing with specified end-effector configurations. Based on the semantical similarity criteria, the feature of the stacking task is mapped to the segment from $\boldsymbol{con_1}$ to $\boldsymbol{con_2}$, $\boldsymbol{con_2}$ to $\boldsymbol{con_3}$, $\boldsymbol{con_3}$ to $\boldsymbol{con_4}$, and $\boldsymbol{con_4}$ to $\boldsymbol{con_5}$ respectively. The final end-effector trajectory in the task space is shown in Fig. 4.4 (b).

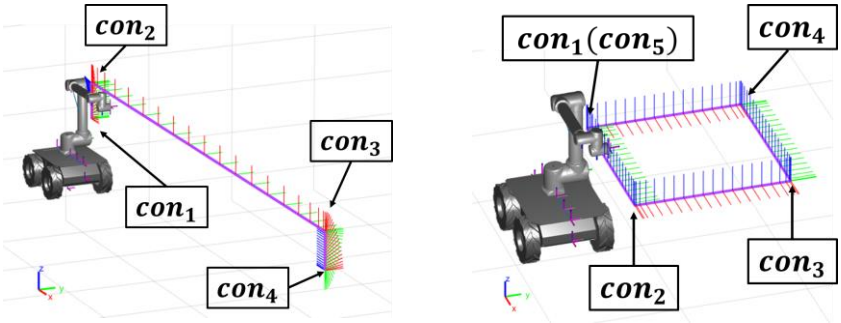
Table 4.1. Critical Configurations of the Material Handling Task.

	con_1	con_2	con_3	con_4
$\boldsymbol{P}$	(0.2,0.2,0.4)	(0.2,0.2,0.8)	(6.2,0.2,0.8)	(6.2,0.2,0.4)

$\boldsymbol{\theta}$	(0,0,0)	$(0,-\pi/2,0)$	$(0,-\pi/2,0)$	$(\pi/2,0,-\pi/2)$

Table 4.2. Critical Configurations of the Painting Task.

	con_1	con_2	con_3	con_4	con_5
$\boldsymbol{P}$	(0.2,0.2,0.6)	(1.4,0.2,0.6)	(1.4,1.4,0.6)	(0.2,1.4,0.6)	(0.2,0.2,0.6)
$\boldsymbol{\theta}$	(0,0,0)	(0,0,0)	(0,0,0)	(0,0,0)	(0,0,0)



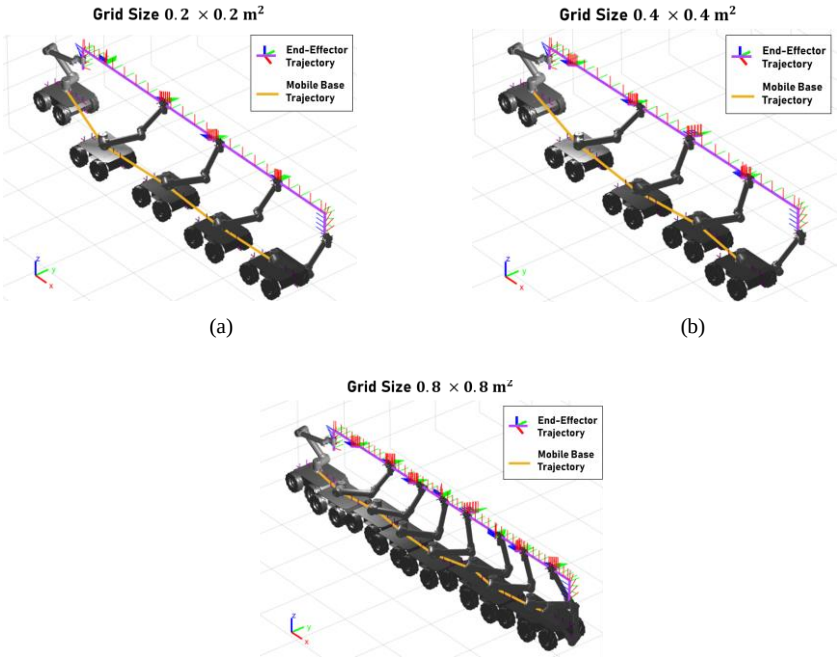
(a) (b)

Figure 4.4. End-effector trajectories in the task space. (a) Material handling task; (b) Painting task.

4.4.3. Determining the Grid Size for the Mobile Base Workspace

Given the target end-effector trajectory for the new task, the pose of the mobile base can be determined by utilizing Algorithm 4.1 to search through a meshed workspace. To determine the ideal grid size for the workspace that strikes a balance between accuracy and computational efficiency, the accumulated distance error and the computing time for various grid sizes are compared in this section. The tolerance Δ_{man_k} of task-oriented manipulability is set to be 0.5. Given the mesh grid size of $0.2 \times 0.2\ \mathrm{m}^2$, $0.4 \times 0.4\ \mathrm{m}^2$ and $0.8 \times 0.8\ \mathrm{m}^2$, 20 trials of experiment are conducted for each grid size and the statistic results are summarized in Table 4.3. A single mobile base trajectory is selected to represent the 20 trials for each grid size, where as shown in Fig. 4.5 (a)-(c), the trajectory of the end effector is denoted by the purple line with the red-blue-green corrdinates, and the trajectory of the mobile based is denoted by the yellow line. Starting from the initial configuration, the target mobile base poses selected within the workspace are represented by the Husky mobile base in Fig. 4.5. From Table 4.3,

it can be found that the accumulated error of the grid size $0.2 \times 0.2\ \mathrm{m}^2$ and $0.4 \times 0.4\ \mathrm{m}^2$ are close, while the accumulated error of the grid size $0.8 \times 0.8\ \mathrm{m}^2$ is nearly 50% more than that of the grid size $0.2 \times 0.2\ \mathrm{m}^2$ and $0.4 \times 0.4\ \mathrm{m}^2$. This happens because the grid size $0.8 \times 0.8\ \mathrm{m}^2$ is too large for the mobile base to select the optimal position that can maximize reachability of the manipulator. To compensate for the manipulator, the mobile base has to move more frequently, resulting in more errors in terms of distance. This can also be observed in Fig. 4.5 (a)-(c), where the movement of the mobile base with the grid size $0.8 \times 0.8\ \mathrm{m}^2$ is three times more than that of the grid size $0.2 \times 0.2\ \mathrm{m}^2$ and $0.4 \times 0.4\ \mathrm{m}^2$. As for the computational efficiency, searching with smaller mesh grids will cost more time and the added movement of the mobile base can also decrease the searching efficiency. Considering both the accumulated distance error and the computing time listed in Table 4.3, the grid size $0.4 \times 0.4\ \mathrm{m}^2$ is selected as the optimal balance between the accuracy and efficiency of the proposed method, which will be used in the following experiments.



(a) (b)

(c)

Figure 4.5. Trajectories of the end-effector and the manipulator for the material handling task with grid sizes of (a) $0.2 \times 0.2\ m^2$, (b) $0.4 \times 0.4\ m^2$, and (c) $0.8 \times 0.8\ m^2$.

Table 4.3. Performance of the Proposed Method on the Material Handling Task with Different Grid Sizes.

Grid Size	Accumulated Distance Error (95% CI)	Computing Time (95% CI)
$0.2 \times 0.2\ \mathrm{m}^2$	0.442m (0.427 – 0.457m)	2250.03s (2219.05 – 2281.01s)
$0.4 \times 0.4\ \mathrm{m}^2$	0.450m (0.438 – 0.462m)	871.64s (851.93-891.35s)
$0.8 \times 0.8\ \mathrm{m}^2$	0.684m (0.670 – 0.698m)	792.25s (772.57-807.93s)

4.4.4. Evaluation of the feasibility and scalability of the Task-oriented Motion Planning Method

In the first experiment, to evaluate the feasibility and scalability of the proposed method, two different material handling tasks are specified with different work space, different goal configuration, and different routings as shown in Table 4.4 and Table 4.5. For each task, 20 trials are conducted. Results show the task can be successfully finished using the proposed method for each trial. As shown in Fig. 4.6, features of the human demonstrated stacking task (keeping the orientation of the end-effector) are scaled and rotated based on the requirement of the passing segment ($\boldsymbol{con_2 \rightarrow con_3}$) of Material Handling Task 2 and Material Handling Task 3. Features of the human demonstrated filling and the twisting tasks can also be successfully learned through the mapping function to corresponding new task segments ($\boldsymbol{con_1 \rightarrow con_2}$ and $\boldsymbol{con_3 \rightarrow con_4}$) with different scales and environments without requiring additional demonstrations or human interference.

Table 4.4. Critical Configurations of the Material Handling Task 2.

	con_1	con_2	con_3	con_4
$\boldsymbol{P}$	(0.2,0.2,0.4)	(0.2,0.2,0.8)	(6.2,6.2,0.8)	(6.2,6.2,0.4)
$\boldsymbol{\theta}$	(0,0,0)	$(0,-\pi/2,0)$	$(0,-\pi/2,0)$	$(\pi/2,0,-\pi/2)$

Table 4.5. Critical Configurations of the Material Handling Task 3.

	con_1	con_2	con_3	con_4
$\boldsymbol{P}$	(0.2,0.2,0.4)	(0.2,0.2,0.8)	(0.8,0.2,0.8)	(0.8,0.2,0.4)
$\boldsymbol{\theta}$	(0,0,0)	$(0,-\pi/2,0)$	$(0,-\pi/2,0)$	$(\pi/2,0,-\pi/2)$

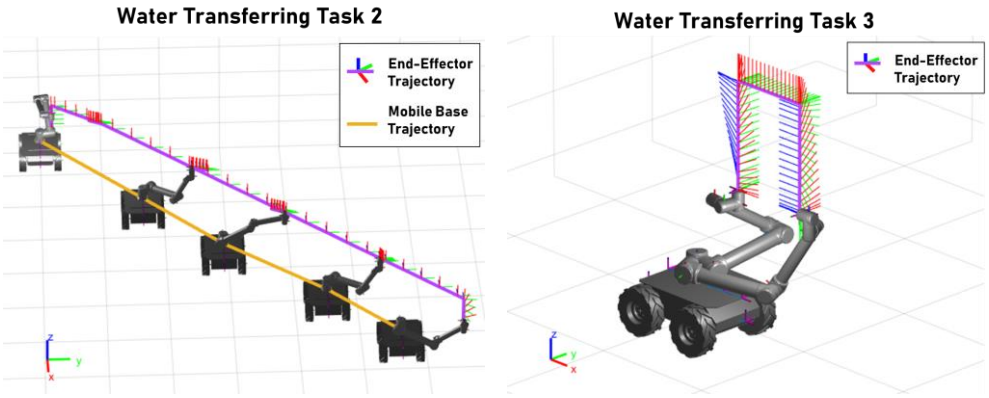


Figure 4.6. Trajectories of the end-effector and the manipulator for Material Handling Task 2 (left) and 3 (right).

4.4.5. Examining Proposed Task-oriented Manipulability

In the second experiment, the performance of the motion planning method is evaluated by comparing the results using both proposed task-oriented manipulability and conventional manipulability. The tolerance of both conventional manipulability and task-oriented manipulability are set to 0.5. As shown in Fig. 4.7, the number of mobile base movements using task-oriented manipulability is half of that using conventional manipulability. This occurs because conventional manipulability does not properly account for the specific requirements of the task and therefore considers some base poses to be more feasible than they actually are, leading to an underestimation of the manipulability score.

For instance, during the task segment between $\boldsymbol{con_2}$ and $\boldsymbol{con_3}$, the optimal movement for maximizing reachability of the manipulator would be for the mobile base to move forward to $\boldsymbol{con_3}$ as much as possible, while stretching the manipulator backward to maintain the end-effector at $\boldsymbol{con_2}$. However, when the

manipulator stretches backward, the conventional manipulability score drops sharply to zero when the manipulator reaches its limit, even though the manipulator can still move freely toward the critical configuration $\boldsymbol{con_3}$. This underestimation of manipulability using the conventional measurement causes the manipulator to not fully stretch, requiring the mobile base to move more frequently as shown in Fig. 4.7. From the statistical results in Table 4.6, it can also be found that the accumulated distance error is increased by involving additional mobile base movements and more time is consumed to calculate these poses. Therefore, the proposed method using task-oriented manipulability outperforms that of using conventional manipulability in both computational efficiency and accuracy.

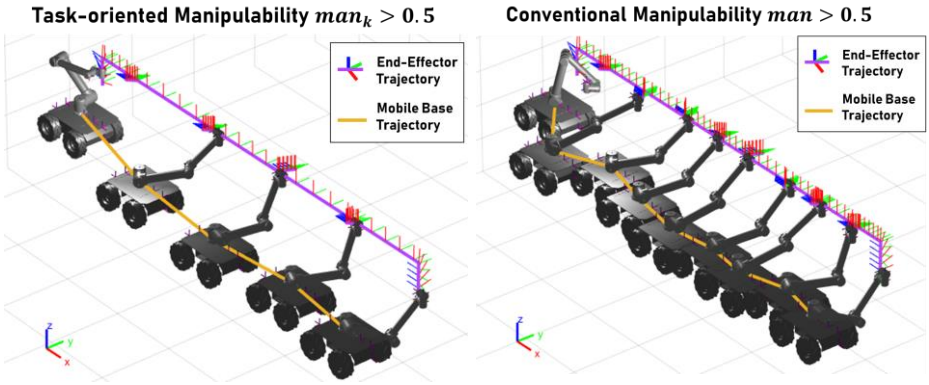


Figure 4.7. Trajectories of the end-effector and the manipulator for the material handling task with proposed task-oriented manipulability (left) and conventional manipulability (right).

Table 4.6. Comparison between Conventional Manipulability and Task-oriented Manipulability on the Material Handling Task.

Manipulability	Accumulated Distance Error (95% CI)	Computing Time (95% CI)
Conventional	$0.871\text{m}\ (0.835 - 0.907\text{m})$	$1987.82\text{s}\ (1964.15 - 2011.49\text{s})$
Task-oriented	$0.429\text{m}\ (0.411 - 0.447\text{m})$	$864.27\text{s}\ (842.15 - 886.40\text{s})$

4.4.6. Evaluation of the Efficiency and Accuracy of the Task-oriented Motion Planning Method

In the third experiment, a comparison is made between the computational efficiency and accuracy of the proposed method on the painting task and those of a duplication method and an overall inverse kinematics method. The following describes the comparison methods.

Duplication method: In this method, for the painting task, the motion of the manipulator fully duplicates the horizontal segment of the human demonstrated stacking task. The mobile base is used to compensate the manipulator for enabling the end-effector to track the target trajectory for the new task.

Overall inverse kinematics method: In this method, the whole mobile manipulator is regarded as an 8DOF redundant manipulator with two prismatic joints from the mobile base and six revolute joints from the manipulator. During joint space execution, inverse kinematics for the entire mobile manipulator is calculated using the Levenberg-Marquardt method [98].

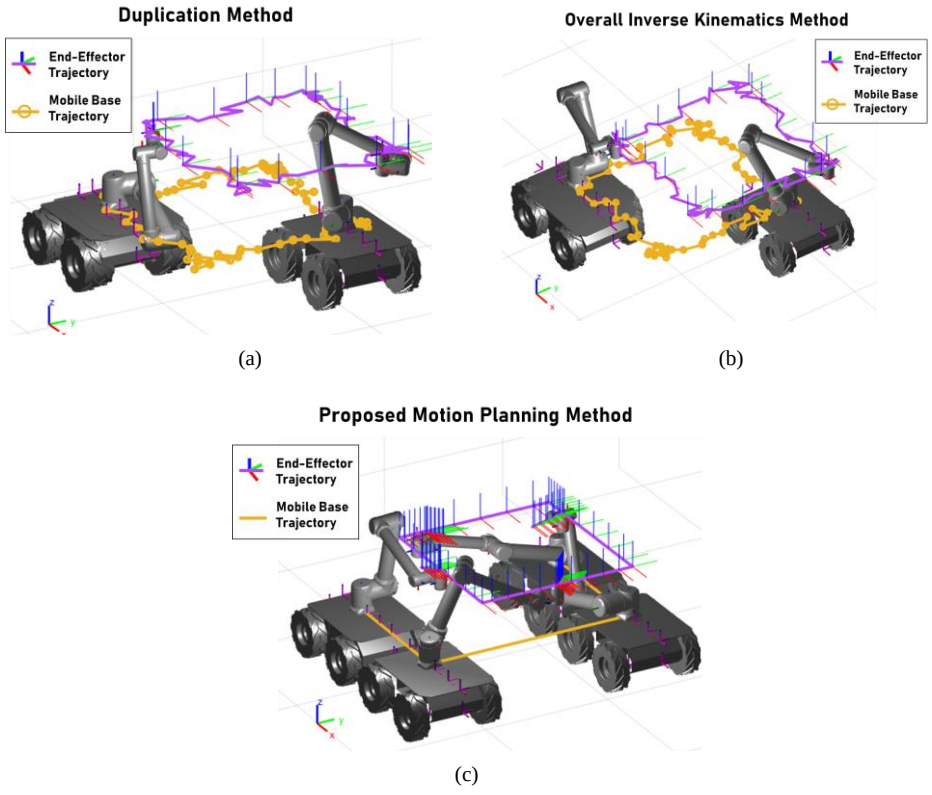


Fig. 4.8. Trajectories of the end-effector and the mobile base for the painting task using (a) the duplication method, (b) the overall inverse kinematics method and (c) the proposed method.

Trajectories of the end-effector and the mobile base obtained by different motion planning methods are shown in Fig. 4.8. For the duplication method and the overall inverse kinematics method, the repositioning of the mobile base is represented by the circle. As shown in Fig. 4.8 (a) and (b), the end-effector trajectories produced by the duplication method and the overall inverse kinematics method deviate significantly from the desired square shape required for the painting task. This is due to the lack of consideration for the significant accuracy difference between the mobile base and the manipulator and the proper coordination between the mobile base and the manipulator, resulting in simultaneous planning of both and additional movements of the mobile base. In contrast, the proposed method provides a coordination scheme that optimizes the movement of the mobile base to limited poses. In this way, most of the target trajectory is kept by the manipulator and the deviation in the distance is also

limited to only a few repositioning points of the mobile base. As a result, the end-effector trajectory obtained by the proposed method follows the square accurately. As listed in Table 4.7, compared with the duplication method and overall inverse kinematics method, the accumulated distance error is reduced by around 95% using our proposed method.

As for computing efficiency, the duplication method is significantly faster than the other two methods since it does not require solving either the optimization problem or inverse kinematics. However, since the duplication method completely duplicates the human demonstration on a specific manipulator, such a method cannot be generalized to other manipulators with different sizes and DOFs. The Levenburg-Marquardt method, which is used in the overall inverse kinematics method, has a high time complexity that grows as the DOF increases, making it the most time-consuming method compared to the other two. As a result, due to its low accuracy and computing efficiency, using a high DOF motion planning method like this should be avoided in practice.

Table 4.7. Performance of Different Motion Planning Methods on the Painting Task.

Method	Accumulated Distance Error (95% CI)	Computing Time (95% CI)
Duplication	8.828m (8.416 – 9.240m)	105.29s (99.81 – 110.77s)
Overall IK	9.156m (8.759 – 9.553m)	894.33s (873.44-915.22s)
Proposed	0.431m (0.710 – 0.752m)	839.10s (813.27-864.93s)

To sum up, the proposed scalable task-oriented motion planning method for mobile manipulators is effective in generating adaptive end-effector trajectories for new tasks by learning from human demonstrations. In addition, by utilizing a reasonable mesh grid size and task-oriented manipulability, the method is able to generate accurate motion plans in the joint space execution with high efficiency.

4.5.Summary

In this research, the we present a scalable motion planning and task-oriented coordination scheme for mobile manipulators. A LfD method is applied to enable the robot to generate adaptive end-effector trajectories in the task space. During the execution of the end-effector trajectory in the joint space, a coordination scheme is proposed to optimize the movement of the mobile base within a boundary based on the reachability and novel task-oriented manipulability of the manipulator. Case study results have validated the effectiveness of the proposed method in generating adaptive end-effector trajectories in the task space and improving accuracy in joint space execution. However, the proposed method still has limitations in its searching efficiency when trying to find the optimal positions of the mobile base.

Chapter 5. Hybrid Robot Learning for Automatic Robot Motion Planning in Smart Manufacturing

5.1.Background

In today's manufacturing settings, there is a clear shift towards agile, intelligent production with an enhanced role for robots [99]. Yet, the industry predominantly leans on pre-programmed robots, necessitating reprogramming even for minimal task adjustments. The associated time and costs in reprogramming these robotic systems present notable challenges [96], [100].

To address challenges in robot motion planning, researchers have developed two main approaches: joint space and task space planning. Joint space planning avoids singularities—including joint limits and obstacles—allowing precise control over individual joints for detailed motion planning [101], [102]. However, it is usually time-consuming and lacks task-level understanding, leading to potential inaccuracies in the end-effector's motion. Task space planning, on the other hand, focuses on task-centric operations. It is fast and adaptable to different robots but relies on inverse kinematics for joint angle calculations, posing challenges in avoiding singularities, collisions, and joint limits [27], [91]. Hence, a method combining the merits of both approaches while mitigating their limitations is essential.

In recent years, with the development and advancement of AI, Deep Reinforcement Learning (DRL) has become crucial for robot motion planning [103]. For instance, the authors propose a Soft Actor Critic (SAC) based method [104] for self-homing in industrital robotic cell. This method assumes a pre-sensed unknown environment, allowing for policy transfer without extra training.

It employs a multi-agent training setting, enhancing state space exploration, with agents sharing experiences and deploying policies collectively. While DRL methods like Deep Q-network (DQN) [105], [106], Deep Deterministic Policy Gradient (DDPG) [27], [107], and Proximal Policy Optimization (PPO) [108] are effective for tasks like pick-and-place and assembly, these methods also face challenges. These challenges include difficulty learning implicit task constraints, requiring extensive data and time, and lacking consistent stability and accuracy. Modifications to work cells or tasks also demand DRL agent retraining or finetuning [109], highlighting the need for ongoing refinement in DRL techniques.

Compared with modeling a task and planning the motion of a robot in the DRL-based robot motion planning, human operators are often more intuitive in performing the task. Learning from Demonstrations (LfD) offers a viable strategy for robots to execute similar tasks in Human-Robot Collaboration environments due to its intuitive nature. However, LfD methods currently face challenges with scalability and adaptability [23], [25]. The authors have introduced a scalable LfD technique, allowing robots to devise adaptive motion plans from a single demonstration [31]. Although this approach efficiently maps kinematic features to new tasks, its efficacy is limited in complex environments with varied obstacles. Robots often require search algorithms to navigate, making original demonstrations less applicable. Furthermore, task space LfD algorithms may encounter joint-space issues, including self-collisions, reachability, and manipulability limitations [29].

This research introduces a hybrid robot motion planning method, integrating the task space LfD [31] and joint space DRL-based approaches [104], aiming to capitalize on their respective strengths while mitigating their limitations. For a robot working in a narrow work cell, the proposed method initiates with a comprehensive feasibility map calculation, considering reachability, manipulability, and collision avoidance. A task space trajectory is then generated using a Hierarchical Reinforcement Learning-based LfD (HRL-LfD) method. The

infeasible segments of such a trajectory are identified via the feasibility map and adjusted into feasible regions using the DRL-based approach.

5.2.Problem Description

In this research, we delve into a 3D work cell scenario where a robotic manipulator is tasked with performing distinct assignments, such as material handling, painting, assembly, or inspection [110]. Depending on the particular task at hand, various constraints are typically imposed, encompassing factors like initial and target positions, as well as the requirement of specific orientations for the end effector. In manufacturing environments, work cells often exhibit intricate layouts comprising machines, robots, and various components. It is assumed that, the position and dimension of the workspace and each component in the workspace are observable. Also, the critical configurations of the new task are given. This research aims to create an effective automated motion planning solution for a robot manipulator, thereby mitigating the need for costly reprogramming when executing diverse tasks within a complex industrial environment.

The authors have previously developed two approaches with the same goal. The task space-based robot LfD approach [31] is effective at quickly and accurately learning from a single demonstration but struggles in complex robot environments with various obstacles, requiring search algorithms for navigation. Additionally, it operates in task-space motion planning, potentially encountering issues in joint-space. In contrast, the DRL-based motion planning approach [104] guarantees joint-space solutions but demands substantial data for agent training, involves time-intensive computations, and lacks inherent task-specific constraint adherence. Therefore, there is a need to develop a hybrid approach that combines the strengths of both methods while mitigating their respective weaknesses.

5.3.Hybrid Motion Planning Framework

In order to address the significant challenge of enabling efficient robot motion planning within complex manufacturing environments and to overcome the challenges posed by both methods (i.e., task-space-based LfD and joint-space-based DRL methods), a novel hybrid approach is introduced. This method aims to systematically integrate the two approaches, utilizing the LfD method for its scalability and task space understanding, and incorporating the DRL-based approach to ensure joint space feasiblity. The framework of this approach is shown in Fig. 5.1.

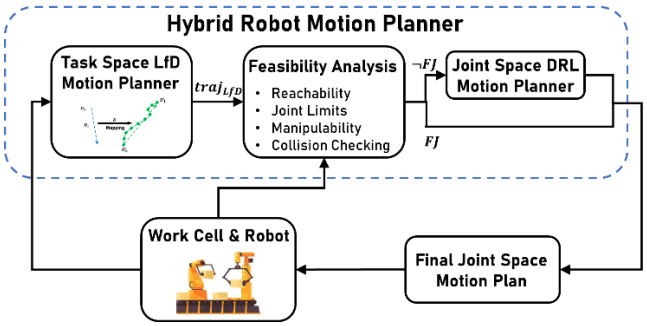


Figure 5.1. Framework of the hybrid motion planning method.

For a 3-D workspace, denoted as $\boldsymbol{WS}$, a joint-space feasibility study is carried out, resulting in the formation of a discrete feasibility map, $\boldsymbol{CM} \subset \boldsymbol{WS}$, where each specific pose in $\boldsymbol{CM}$ denotes an end effector configuration, including both position and orientation. $\boldsymbol{CM}$ is divided into two primary regions, the feasible region, $\boldsymbol{FR}$, and infeasible region, $\neg\boldsymbol{FR}$, which correspond to the joint-space analysis of each end effector configuration. For $\neg\boldsymbol{FR} \subset \boldsymbol{CM}$, three potential conditions may arise: the manipulator's end effector might be unable to reach a position, a self-conflict or collision with workspace obstacles may occur, or the manipulator's manipulability [97] (a measure of how close a manipulator is to singularity) may fall below a certain tolerance. Therefore, $\boldsymbol{FR} \subset \boldsymbol{CM}$ represents the desirable region for motion planning.

Subsequently, the feasibility map $\boldsymbol{CM}$ is employed as a filter for the trajectory generated by the task-space robot HRL-LfD method (will be introduced in Section 5.4), referred to as $\boldsymbol{traj}_{LfD}$. It is important to note that $\boldsymbol{traj}_{LfD}$ may encompass segments situated within both the feasible and infeasible regions. The

feasible segments can be represented as $\boldsymbol{FJ} \subseteq \{\boldsymbol{traj}_{LfD} \cap \boldsymbol{FR}\}$, while the infeasible segments can be represented as $\neg\boldsymbol{FJ} \nsubseteq \{\boldsymbol{traj}_{LfD} \cap \boldsymbol{FR}\}$. For the segments falling within $\boldsymbol{FJ}$, inverse kinematics will be calculated to obtain the joint angles to control the robot. For segments within $\neg\boldsymbol{FJ}$, a DRL approach, which will be introduced in Section 5.5 is utilized to guide these segments into the $\boldsymbol{FJ}$ category, resulting in a feasible trajectory $\boldsymbol{traj}_{\boldsymbol{DRL}}$.

5.4.Task-Space HRL-Based LfD

In manufacturing, manipulator motion planning can be complex, but human operators can intuitively demonstrate tasks. The authors have created a task-space LfD method for robot manipulators [31], enabling them to learn specific tasks like object grasping or relocation based on a single primitive skill. In this research, we expand our earlier LfD method to develop an HRL-based LfD.

First, a local motion planner is developed by using kinematic task-space planning that follows the implicit geometric constraints throughout a one-time human demonstration of a primitive skill. Then, built upon the local motion planner, a global planner is established through an HRL scheme, which can enable robots to automatically generate their motion plan for various tasks by intelligently combining a set of demonstrated primitive skills.

Local Motion Planner: The authors' recent work [14] has developed the local motion planner, which is briefly introduced here without delving into technical details for paper's self-containment.

Let $\boldsymbol{\mathcal{DP}} = \{\boldsymbol{D}_1, \boldsymbol{D}_2, \dots, \boldsymbol{D}_n\}$ represents a demonstrated primitive skill in the task-space, where $\boldsymbol{D}_i$ is a dual quaternion representation for the i^{th}configurations in time sequence during the motion. The transformation, $\boldsymbol{\delta_i}$, between the last pose and every other pose is:

$$\boldsymbol{\delta_i} = \boldsymbol{D}_{i-1}^{*} \otimes \boldsymbol{D}_n \ , i = 2, \dots, n \tag{5.1}$$

where $\otimes$ represents dual quaternion multiplication and $\boldsymbol{D}^*$ denotes the conjugate of $\boldsymbol{D}$. Thus, the sequence of $\boldsymbol{\delta_i}$ represents a sequence of transformation. Note that all implicit task constraints in a human demonstration are embedded in the sequence of $\boldsymbol{\delta_i}$ during the motion. This sequence can represent the features or semantics of human demonstrations. The feature of the k^{th} human demonstration in the task space can be represented in a time sequence can be represented as:

$$\boldsymbol{HD}_k = \{\boldsymbol{\delta}_2^{\boldsymbol{HD_k}}, \dots, \boldsymbol{\delta}_n^{\boldsymbol{HD_k}}\} \tag{5.2}$$

For a new task, $\boldsymbol{tk}$, a mapping operation, $\boldsymbol{mp_{HD \to tk}}$, is developed, which can align and enforce the feature of the demonstration to the task by using the quaternion sandwich operation [111]. The details of the mapping can be found in the authors' recent work [31] and is shown in Fig. 5.2.

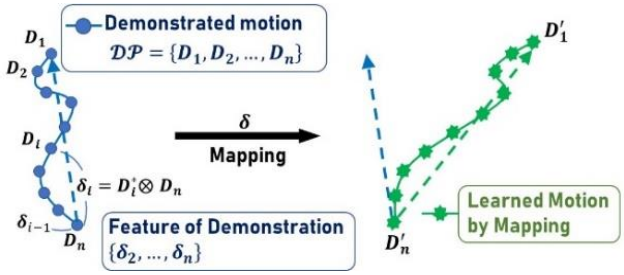


Figure 5.2. Mapping the demonstrated skill (blue line) to the new task with the new starting and goal configuration D_1' and D_n'.

Global Planner: However, manufacturing tasks such as assembly tasks are often complicated, which may include combinations of various primitive skills. Therefore, a library of h demonstrated primitive skills can be formed as:

$$\boldsymbol{LB} = \{\boldsymbol{HD}_1, \boldsymbol{HD}_2, \dots, \boldsymbol{HD}_h\} \tag{5.3}$$

For any task instance, a robot should be able to look at the library of demonstrations and be able to learn and combine the most suitable demonstrations by using the local motion planner. To do this, one may need to go through all possible subsets of new tasks and evaluate each $\boldsymbol{HD}_k$, $k = 1, 2, \dots, h$, in the library $\boldsymbol{LB}$, which is an NP-hard problem. The state space of the problem would be huge if the constraints of new tasks and the number of demonstrated skills are large. The problem can be formulated as a model-free reinforcement learning (RL)

problem in the Markov Decision Process (MDP) framework. In the context of the robot learning global planning, the RL problem is a tuple $< \boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R}, \boldsymbol{S}' >$, and is formulated as follows.

The state space $\boldsymbol{S}$ contains a set of system states, which include the current configuration of the end-effector and all possible task segments. Let $\boldsymbol{s}_t$ denote the system state at time t, $\boldsymbol{s}_t \in \boldsymbol{S}$, then $\boldsymbol{s}_t$ is defined as

$$\boldsymbol{s}_t = \langle \boldsymbol{EE}_t, \boldsymbol{ta}_t \rangle \tag{5.4}$$

where $\boldsymbol{EE}_t$ is the current configuration of the robot's end-effector at t, $\boldsymbol{ta}_t \sqsubseteq \boldsymbol{TK}$ is the possible task segments at t.

The action space $\boldsymbol{A}$ is the set of action pairs for the robot learner to decide on how to segment a new task and what demonstrations should be selected. The action $\boldsymbol{a}_t \in \boldsymbol{A}$, can be defined as

$$\boldsymbol{a}_t = < \boldsymbol{ts}_t, \boldsymbol{HD}_{t,l} > \tag{5.5}$$

where $\boldsymbol{ts}_t \in \boldsymbol{ta}_t$ and $\boldsymbol{HD}_{t,l} \in \boldsymbol{LB}$.

The reward function is determined based on Euclidean distance between the feature of the demonstration, $\boldsymbol{\delta}_i^{\boldsymbol{HD_i}}$, and the feature of a new task, $\boldsymbol{\delta}_l^{\boldsymbol{ts_t}}$, as in [31]. Let Δ_β be a predefined tolerance value, then:

$$r_t = \begin{cases} -\sum_{l=j}^{n} \beta\left(\boldsymbol{\delta}_i^{\boldsymbol{HD_i}}, \boldsymbol{\delta}_l^{\boldsymbol{ts_t}}\right), & \text{if } \beta\left(\boldsymbol{\delta}_i^{\boldsymbol{HD_i}}, \boldsymbol{\delta}_l^{\boldsymbol{ts_t}}\right) \le \Delta_\beta \\ -\infty, & \text{otherwise} \end{cases} \tag{5.6}$$

The standard RL approach needs to simultaneously determine the action pair $< \boldsymbol{ts}_t, \boldsymbol{HD}_{t,l} >$, and the state-action space grows exponentially with the number of features and task-relevant constraints. To alleviate the curse of dimensionality, an HRL scheme is developed (see Fig. 5.3), where the agent uses a two-level hierarchy consisting of a task-controller and a motion-controller with two inter-dependent networks.

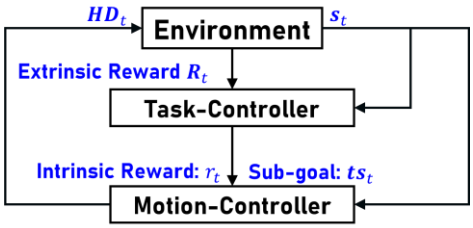


Figure 5.3. Human-in-the-loop hierarchical RL scheme.

The task-controller is to find a policy that specifies the subgoal, $\boldsymbol{ts}_t$, under the global state $\boldsymbol{s}_t$. The task-controller will determine how to segment the new task through estimating the value function $Q(\boldsymbol{s}_t, \boldsymbol{ts}_t)$, such that the extrinsic reward, R_t, can be maximized. the $Q(\boldsymbol{s}_t, \boldsymbol{ts}_t)$ function of the task-controller is estimated as:

$$Q(\boldsymbol{s}_t, \boldsymbol{ts}_t) = E_{\pi_Q}[R_t | \boldsymbol{s}_t = \boldsymbol{s}, \boldsymbol{ts}_t = \boldsymbol{ts}] \tag{5.7}$$

where π_Q is the global policy over subgoals, R_t is the extrinsic reward for the meta-controller and is defined as:

$$R_t = \sum_{t'=0}^{t} r_t \tag{5.8}$$

where r_t is the intrinsic reward for the next level critic-controller.

The objective of the task-controller is to find an optimal policy, $\pi^*_{\boldsymbol{ts}}$, such that R_t can be maximized. $\pi^*_{\boldsymbol{ts}}$ can be defined as:

$$\pi^*_Q(\boldsymbol{ts}|\boldsymbol{s}) = \begin{cases} 1, & \text{if } \boldsymbol{ts} = \arg\max_{\boldsymbol{ts}_t \subseteq \boldsymbol{TK}} \{Q(\boldsymbol{s}_t, \boldsymbol{ts}_t)\} \\ 0, & \text{otherwise} \end{cases} \tag{5.9}$$

The motion-controller is to continue to determine the policy on specifying the action $\boldsymbol{HD}_{t,l}$ under the global state $\boldsymbol{s}_t$ and the current subgoal, $\boldsymbol{ts}_t$, that is flown down from the task-controller. The motion-controller will decide pertinent demonstrated primitive skills by estimating a value function, $q(\boldsymbol{s}_t, \boldsymbol{ts}_t, \boldsymbol{HD}_{t,l})$, so that the intrinsic reward, r_t, can be maximized. The value function can be written as:

$$q(\boldsymbol{s}_t, \boldsymbol{ts}_t, \boldsymbol{HD}_{t,l}) = E_\pi[r_t | \boldsymbol{s}_t = \boldsymbol{s}, \boldsymbol{ts}_t = \boldsymbol{ts}, \boldsymbol{HD}_{t,l} = \boldsymbol{HD}_l] \tag{5.10}$$

where $\boldsymbol{ts}$ is the given subgoal from the task-controller in state $\boldsymbol{s}$, and π_q is the policy on how to select human demonstrations.

The intrinsic reward, r_t, is to compare semantic similarity between the human demonstrations, $\boldsymbol{HD}_{t,l}$, and the subgoal, $\boldsymbol{ts}_t$, using Eqn. (5.11). The internal critic checks if the subgoal is reached and provides an appropriate intrinsic reward to the controller. The optimal policy π_q^* of the critic-controller is defined as:

$$\pi_q^*(\boldsymbol{HD}_l | \boldsymbol{s}, \boldsymbol{ts}) = \begin{cases} 1, & \text{if } \boldsymbol{HD}_l = \arg\max_{\boldsymbol{HD}_{t,l} \in \boldsymbol{LB}} \{q(\boldsymbol{s}_t, \boldsymbol{ts}_t, \boldsymbol{HD}_{t,l})\} \\ 0, & \text{otherwise} \end{cases} \tag{5.11}$$

The algorithm of offline training and online execution of the HRL-LfD method is shown in Algorithm 5.1.

Algorithm 5.1

Procedure1 Offline Training of the HRL-LfD Method
Input: **TK**, **LB**
Initialize $H(\mathbf{s}, \mathbf{ts})$ and $q(\mathbf{s}, \mathbf{ts}, \mathbf{HD_l})$randomly
Initialize $\mathbf{s}_0$ with $\mathbf{EE}_0$ and $\mathbf{ta}_0$
For $t = 1, \ldots, T$ do
  For $t' = 1, \ldots, \tau$ do
    Compute r_t using Eqn. (5.6)
    Update $q_{t'}(s_t, \mathbf{ts}_t, \mathbf{HD}_{t',l})$using Eqn. (5.10)
  End For
  Update $\tilde{r}_t \leftarrow \sum_{t'=0}^{t} r_{t'}$ and update $H_t(s_t, \mathbf{ts_t})$
End For
Output $\mathbf{H}(\mathbf{s}, \mathbf{ta})$ and $\mathbf{q}(\mathbf{s}, \mathbf{ta}, \mathbf{HD_l})$

Procedure2 Oline Execution of the HRL-LfD Method
Input: **TK**, **LB**, $\mathbf{H}(\mathbf{s}, \mathbf{ta})$, and $\mathbf{q}(\mathbf{s}, \mathbf{ta}, \mathbf{HD_l})$
While $\mathbf{EE}_t$ is not the last configuration in **TK** do
Select the task segment $\mathbf{ts} = \arg\max_{\mathbf{ts_t} \subseteq \mathbf{TK}} \{Q(\mathbf{s}_t, \mathbf{ts}_t)\}$
Select the demonstration
  $\mathbf{HD}_l = \arg\max_{\mathbf{HD}_{t,l} \in \mathbf{LB}} \{q(\mathbf{s}_t, \mathbf{ts}_t, \mathbf{HD}_{t,l})\}$
Mapping $\mathbf{mp}_{\mathbf{HD_l} \to \mathbf{ts}}$ to calculate $\mathbf{traj}_{\mathrm{LfD}}$
Update $\mathbf{EE}_t$
End While
Output $\boldsymbol{traj}_{LfD}$

5.5. Joint Space DRL-Based Motion Planner

As mentioned earlier, in cases where the HRL-LfD trajectory enters an infeasible region, three possible scenarios may unfold: the manipulator's end effector may fail to reach a position, encounter self-conflict or workspace collisions, or experience reduced manipulability below a set tolerance, which constitutes a joint-space failure. To address this issue, the authors build upon our recent work [104] by applying a Deep Reinforcement Learning (DRL) approach. The motion planning problem is formulated as an MDP, and soft actor-critic (SAC) algorithm is used in the offline training of the joint space motion planner. SAC belongs to the family of model-free, off-policy Deep RL algorithms, which outperforms prior state-of-the art RL methods in continuous state-action problem settings [112]. After training, this motion planner is then implemented during real-time execution to rectify any unfeasible segments of the LfD trajectory.

5.5.1. MDP Formulation

Given the workspace, the robot and the target position, the primary components of the MDP can be defined as follows. The state space $\boldsymbol{S}$ includes all information regarding the robot and the environment conditions. The state $\boldsymbol{s_t} \in \boldsymbol{S}$ of the RL agent is defined as:

$$\boldsymbol{s_t} = \langle \boldsymbol{JP_t}, \boldsymbol{JO_t}, \boldsymbol{LV_t}, \boldsymbol{AV_t}, \boldsymbol{TP_t}, \boldsymbol{TO_t}, \boldsymbol{RL_t} \rangle \tag{5.12}$$

where $\boldsymbol{JP_t}, \boldsymbol{JO_t} \in \mathbb{R}^{3\times n}$, n is the degree of freedom, denotes the x, y, z positions and Euler angles of each joint, respectively; $\boldsymbol{LV_t}, \boldsymbol{AV_t} \in \mathbb{R}^{3\times n}$ denotes the linear and angular velocities of each joint, respectively; $\boldsymbol{TP_t}, \boldsymbol{TO_t} \in \mathbb{R}^3$ denotes the x, y, z positions and Euler angles of the end-effector, respectively; $\boldsymbol{RL_t} \in \mathbb{R}^{25}$ denotes the length of the ray that generated from the end-effector to the surface of the environment. This ray trace can reflect the real-time environment conditions. In this joint space motion planner, 25 rays are generated in different angles to the end-effector using a Pybullet library.

The action of the RL agent is defined as a vector of joint angles:

$$\boldsymbol{a_t} = [\theta_{1t}, \theta_{2t}, \dots, \theta_{nt}] \tag{5.13}$$

where $\theta_{it}, i \in 1, \ldots, n, n$ is the DOF, denotes the joint angle, which is limited within [-1, 1]. At every time instance t, $\boldsymbol{a_t}$ depicts a robot's configuration. The feasibility of this configuration plays a pivotal role in determining the reward. Moreover, navigating the robot towards the target region requires considering the gap between the end-effector and the intended goal as a significant reward component. The position of the end-effector can be calculated using forward kinematics $\mathcal{FK}$: $\boldsymbol{a_t} \rightarrow \boldsymbol{D_t}$. Let d_t represent the distance between the end effector and the object, and let r signify the radius of the target region relative to the goal position. Consequently, the reward for the SAC agent is described as:

$$\boldsymbol{R}_t = \begin{cases} 0, & if\ d_t \geq r\ or\ fea(\boldsymbol{D_t}) = 0 \\ 0.1 & otherwise \end{cases} \tag{5.14}$$

5.5.2. Offline Training and Online Execution of the Joint Space Motion Planner

For offline training, to train a general joint space motion planner that can generate a joint space feasible trajectory, an SAC method is applied. It is noted that by leveraging the aforementioned HRL-LfD, the DRL method does not need to learn from scratch by searching the whole work space. It only needs to learn the infeasible segments. After training, the trained policy, $\boldsymbol{\pi}^*$ (weights of neural networks) with the optimal average reward is used for the online execution.

For online execution, given the infeasible segment, $\neg \boldsymbol{FJ}_{ij} = \{\boldsymbol{D}_i, \boldsymbol{D}_{i+1}, \ldots, \boldsymbol{D}_j\}$, of the HRL-LfD trajectory, the starting and goal pose of the joint space motion planner is $\boldsymbol{D}_{i-1}$ and $\boldsymbol{D}_{j+1}$. Using the pair $\{\boldsymbol{D}_{i-1}, \boldsymbol{D}_{j+1}\}$ as the input, the output of the joint space motion planner is a feasible trajectory $\boldsymbol{traj}_{DRL}$ that moves the robot from $\boldsymbol{D}_{i-1}$ to $\boldsymbol{D}'_{j+1}$. The algorithm of offline training and online execution of the DRL method is shown in Algorithm 5.2.

Algorithm 5.2

Procedure1 Offline Training of DRL Method

Input: $\boldsymbol{WS}$, robot, starting and goal positions, radius of the target area, r, initial neural network weights, $\boldsymbol{\theta}$

While the episode does not terminate do

Observe the state s and select the action $a \sim \pi_\theta(\cdot\,|s)$

Execute a in the environment and get the next state s'

If s' is a feasible state
Calculate R_t using Eqn. (5.14)
Store $(s, a, r, s', done)$ in replay buffer $\mathcal{D}$
End if
If it is time to update the target neural network then
Randomly sample a batch of transitions from $\mathcal{D}$
Compute the loss function and update the policy
Update the target network's $\boldsymbol{\theta}$
End If
End While
Output: target network $\boldsymbol{\theta}$

Procedure2 Oline Execution of the DRL Method
Input: Robot starting pose, Goal Position, Neural Network $\boldsymbol{\theta}$
While end-effector position is not in the target area
Execute the action $a_t \sim \pi_\theta^*(\cdot\,|s_t)$
Append a_t to $\boldsymbol{traj}_{DRL}$
Update the current end-effector pose
End While
Output $\boldsymbol{traj}_{DRL}$

5.6. Joint Space Feasibility Analysis

This section presents a thorough feasibility study that combines the evaluation of reachability, joint limits, manipulability, and collision checking for the robot, resulting in the creation of a feasibility map. This map will serve as a filtering mechanism to streamline the process of hybrid motion planning. It will systematically determine whether to employ an HRL-LfD or a DRL-based method to automatically generate a manipulator motion plan.

While existing studies have discussed reachability [113], manipulability [114], and collision checking [115], most of them focus on individual problems. This research develops a holistic study to integrate all aspects and provide a feasibility map.

In this research, the configuration that integrates position and orientation of the end-effector in $SE(3)$ is represented as a dual-quaternion $\boldsymbol{D} \in \mathbb{R}^8$, which is an 8-dimensional real algebra isomorphic to the tensor product of the quaternions and the dual numbers [70]. Given a configuration $\boldsymbol{D}$ and the joint limits $\boldsymbol{B} \in \mathbb{R}^{\mathrm{n}}, n$ is the degrees of freedom (DOF), reachability of the manipulator can be determined by checking the existence of solutions to the inverse

kinematics $IK: \boldsymbol{D} \rightarrow \boldsymbol{\theta}\ s.t. \boldsymbol{\theta} \in \boldsymbol{B}$, where $\boldsymbol{\theta} \in \mathbb{R}^n$ is a vector of joint angles. It can be defined as:

$$\boldsymbol{RH}(\boldsymbol{D}, \boldsymbol{B}) = \begin{cases} 1, & \text{if IK solutions exsit} \\ 0, & \text{otherwise} \end{cases} \tag{5.15}$$

However, if only reachability constraint is considered some poses can be reached but will lose one DOF in translation or rotation. To tackle this problem, manipulability is considered as a second constraint. The conventional measurement of manipulability [97] is given as :

$$man(\boldsymbol{\theta}) = \sqrt{\det[\boldsymbol{J}(\boldsymbol{\theta})\boldsymbol{J}^T(\boldsymbol{\theta})]} \tag{5.16}$$

where $\boldsymbol{J}(\boldsymbol{\theta}) \in \mathbb{R}^{m \times n}$, m is the DOF of the end-effector and n is the DOF of the robot arm, $\boldsymbol{J}(\boldsymbol{\theta})$ is the Jacobian matrix and $\boldsymbol{J}^T(\boldsymbol{\theta})$ is the transpose of $\boldsymbol{J}(\boldsymbol{\theta})$. This is a metric that quantifies the distance between a manipulator state $\boldsymbol{\theta}$ and a singularity state $\boldsymbol{\theta}^*$. Such a measurement can provide information about the overall movement ability of the end-effector. By comparing the manipulability $man(\boldsymbol{\theta})$ with the manipulability of the initial pose of the robot $man(\boldsymbol{\theta}_0)$, the normalized manipulability can be defined as:

$$man'(\boldsymbol{\theta}) = \frac{man(\boldsymbol{\theta})}{man(\boldsymbol{\theta}_0)} \tag{5.17}$$

To evaluate the normalized manipulability, a threshold $\delta_{man'}$ is set based on specific task requirements, so that $man'(\boldsymbol{\theta}) \leq \delta_{man'}$.

Furthermore, it is essential to conduct collision check especially considering a manipulator's potential interactions with workspace obstacles or its own components. In this research, the collision detector in Pybullet [116] is utilized for these collision checks. In PyBullet, the Continuous Collision Detection (CCD) library first simplifies complex objects by decomposing them into simpler convex pieces, and then identifies overlaps of pieces, which furnishes detailed insights such as contact points, contact joint or link indices, and penetration depths. For a configuration $\boldsymbol{\theta}$, the collision index is denoted as $\boldsymbol{COL}(\boldsymbol{\theta})$, using CCD, if $\boldsymbol{COL}(\boldsymbol{\theta}) = 1$ then there is a collision identified, otherwise $\boldsymbol{COL}(\boldsymbol{\theta}) = 0$.

Therefore, the feasibility measurement of a configuration that integrates reachability, manipulability, and collision checking can be defined as:

$$fea(\boldsymbol{D}) = \begin{cases} 1, & if\ \boldsymbol{RH} = 1\ \&\ man' \geq \delta_{man} \&\ COL = 0 \\ 0 & otherewise \end{cases} \tag{5.18}$$

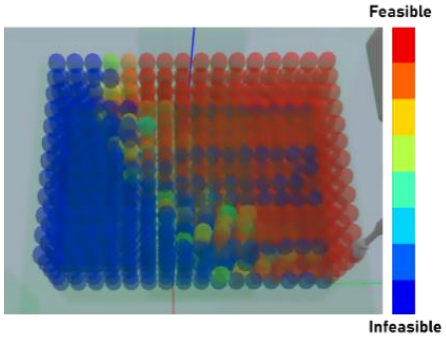


Figure 5.4. Schematic picture of the feasibility map.

Within the work cell environment $\boldsymbol{W}$, it is straightforward to define a 3-D workspace, $\boldsymbol{WS}$, according to task specifications, such as the precise painting of a component at a designated location. $\boldsymbol{WS}$ can then be discretized into small voxel, each with its center signifying the x, y, z position of the end effector. Within each voxel position, the orientation of the end effector in terms of α, β, γ can be further discretized within a task specific range of $[-\theta, \theta] \sqsubseteq [-\pi, \pi]$. Consequently, the discretized $\boldsymbol{WS}$ forms a tensor of rank 2, where each array corresponds to a unique end effector configuration. Using Eqn. (5.18), it becomes possible to assess the feasibility of each end effector configuration within $\boldsymbol{WS}$. This discretized feasibility assessment can be analogized to a "map", referred to as $\boldsymbol{CM}$, which divides the workspace $\boldsymbol{WS}$ into feasible and infeasible regions based on the criterion $fea\,(\boldsymbol{D})$. The map $\boldsymbol{CM}$ effectively encapsulating the end effector configuration along with its associated joint space feasibility status. Fig. 5.2 is a 3D illustration of a feasibility map where both feasibility and unfeasibility regions are color-coded. It is important to acknowledge that the "actual map" encompasses six dimensions, which include both position and orientation, making it impossible to visualize directly. Therefore, each voxel in Fig. 5.2 includes a specific position but includes a range of orientation that may or may not be in

feasible region, which is represented by transition color between red (represent feasible) and blue (infeasible).

5.7.Hybrid Motion Planning based on the Feasibility Map

For a task $\boldsymbol{tk}$, using the HRL-LfD method, which will be introduced in the next section, a task space trajectory, $\boldsymbol{traj}_{LfD} = \{\boldsymbol{D}_1, \boldsymbol{D}_2, \dots, \boldsymbol{D}_n\}$, can be generated. Starting from $\boldsymbol{D}_1$, by checking the feasibility of each configuration, the infeasible trajectory segment, e.g., $\neg\boldsymbol{FJ}_{ij} = \{\boldsymbol{D}_i, \boldsymbol{D}_{i+1}, \dots, \boldsymbol{D}_j\}, i, j \in \{2, .., n-1\}$ can be determined. The DRL method [104], is used to compensate for each $\neg\boldsymbol{FJ}_{ij}$.

For many tasks, such as achieving a proper grip for stacking, it is more crucial to mimic the latter part of the $\boldsymbol{traj}_{LfD}$ generated through LfD, rather than the initial stages. This assumption ensures the utilization of DRL motion planning for handling the infeasible segment, $\neg\boldsymbol{FJ}_{ij}$, since DRL-based method excel at ensuring joint-space feasibility but may not adequately address task-specific constraints. In an effort to minimize $\neg\boldsymbol{FJ}_{ij}$, we anticipate that the transition from $\boldsymbol{traj}_{DRL}$ to $\boldsymbol{traj}_{LfD}$ should occur early enough to capture the relevant portion of the constraints in the demonstration. The process begins at $\boldsymbol{D}_{i-1}$, where a feasible trajectory, $\boldsymbol{traj}_{DRL}$, is computed to reach a pose within close proximity to $\boldsymbol{D}_{j+1}$. subsequently, screw-linear interpolation [96] is employed to seamlessly merge $\boldsymbol{traj}_{DRL}$ to $\boldsymbol{D}_{j+1}$.

It is important to note that when learning from different demonstrations for the same task, $\boldsymbol{tk}$, various trajectory outcomes, $\boldsymbol{traj}_{LfD}$, can be achieved. To evaluate each $\boldsymbol{traj}_{LfD}$, a criterion needs to be established. Notably, the policy generated through the proposed DRL method is not deterministic, leading to varying $\boldsymbol{traj}_{DRL}$ lengths for identical tasks. On the other hand, each $\boldsymbol{traj}_{LfD}$ acquired through learning from a specific demonstration is deterministic in nature. The core concept behind the suggested hybrid approach is to maximize the utilization of LfD motion planning due to its computational efficiency and the generalizability inherent in the proposed LfD method. Consequently, minimizing

$\neg \boldsymbol{FJ}_{ij}$ serves as the prime criterion when selecting $\boldsymbol{traj}_{LfD}$, facilitating a highly efficient fusion of $\boldsymbol{traj}_{LfD}$ motion planning and $\boldsymbol{traj}_{DRL}$ motion planning in our hybrid approach.

The HRL-LfD and DRL motion planning method can be systematically integrated into a hybrid motion planning approach using the feasibility map. The offline training and online execution of the hybrid motion planning method can be shown as Algorithm 5.3. It is worth noting that, in offline training, a set of tasks are generated randomly to train the HRL-LfD motion planning policy, and the data of infeasible segment $\neg \boldsymbol{FJ}$ for each $\boldsymbol{traj}_{LfD}$ can be used to train the DRL policy. Since the task space HRL-LfD method is a scalable and adaptive method for different tasks, when task changes, the hybrid method does not need to be retrained. In addition, the set of $\neg \boldsymbol{FJ}$ represents the region in the workspace where the infeasibility usually happens. With such knowledge obtained from HRL-LfD, the DRL method does not need to learn from scratch by searching the whole work space. This method significantly shrinks the search space and boosts training efficiency. In online execution, the resulting trajectory, $\boldsymbol{traj}_{final}$, derived from this hybrid approach represents an optimized path that not only satisfies the task space constraints but also feasible in the joint space.

Algorithm 5.3

Procedure1 Offline Training of the Hybrid Method
Input: Demonstration Library $\boldsymbol{LB}$, $\boldsymbol{WS}$, Robot
Calculate the feasibility map $\boldsymbol{CM}$
Initialize a set of task $\boldsymbol{TK}$ with random starting and end positions
  For each $\boldsymbol{TK}$ do
    Call Procedure 1 in Algorithm 5.1 and save each $\neg \boldsymbol{FJ}$
  End For
Import all $\neg \boldsymbol{FJ}$ to Procedure 1 in Algorithm 3
Output: Trained Q-table of the HRL-LfD method and trained neural network of the DRL method
Procedure2 Online Execution of the Hybrid Method
Input: New Task $\boldsymbol{TK}$, Demonstration Library $\boldsymbol{LB}$, $\boldsymbol{WS}$, Robot
Call Algorithm 5.2 to calculate the task space trajectory $\boldsymbol{traj}_{LfD}$
For the infeasible segment $\neg \boldsymbol{FJ}$, call Algorithm 3 to calculate the joint space trajectory $\boldsymbol{traj}_{DRL}$
For the feasible segment, calculate inverse kinematics
$\boldsymbol{traj}_{final} \leftarrow$ Concatenate $\boldsymbol{traj}_{DRL}$ and the joint space $\boldsymbol{FJ}$
Output $\boldsymbol{traj}_{final}$

5.8.Experiments and Validation

This section presents simulated experiments conducted on a fluorescent penetrant inspection (FPI) task, which is the most widely used Non-Destructive Testing (NDT) method in the aerospace industry [117]. The performance of the proposed hybrid robot motion planning method is evaluated using two metrics: (1) computing time to achieve a steady motion planning policy in the offline training; (2) the success rate of the motion plan in the online execution. To assess the performance of the proposed method, one purely DRL-based robot motion planning method [104] is used for comparison. Based on the results of the case study, three key conclusions can be drawn: (1) the proposed hybrid robot motion planning method is effective in generating adaptive trajectories for different tasks; (2) in offline training, the proposed hybrid robot motion planning method outperforms the purely DRL-based method in training efficiency and generating feasible configurations; (3) when provided with same training time, the proposed hybrid robot motion planning method outperforms the purely DRL-based method in the success rate of achieving different tasks.

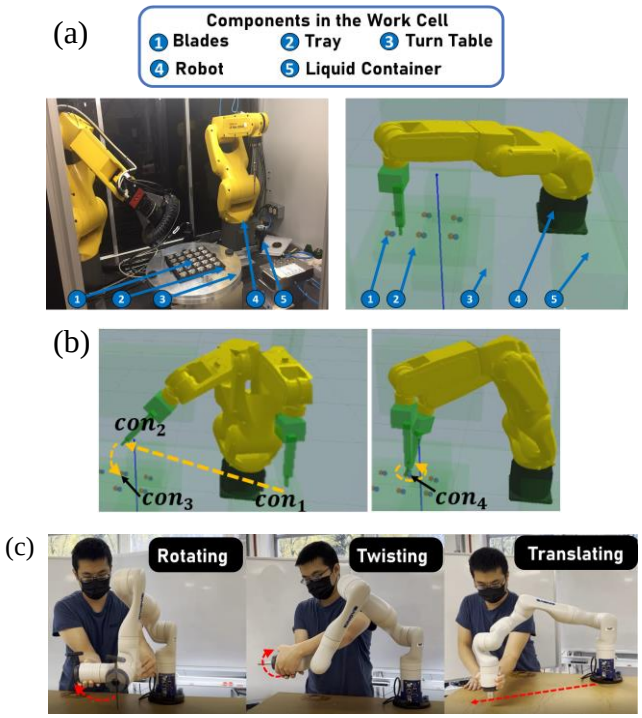


Figure 5.5. Fanuc LR Mate 200iD test bed and kinesthetic demonstrations. (a) Components in the FPI work cell. The simulated work cell (right) is set up in Pybullet, which is the digital twin of the real robotic work cell (left). (b) FPI tasks to be performed. (c) Kinesthetic demonstrations on a Kinova Gen3 robot.

5.8.1. Environment Settings

The experimental setup for the FPI task is depicted in Fig. 5.5(a). The task to be performed is shown in Fig. 5.5(c), where the Fanuc LR Mate 200iD robot is required to moves from the initial home configuration, con_1, to the hovering position, con_2, above the center of the tray, and then brush one blade from con_3 (an initial configuration on a blade) to con_4 (an ending configuration on a blade). A set of 20 fundamental skills commonly employed in FPI tasks are provided. Fig. 5.5(b) illustrates three examples of the demonstrated skills, including rotating, twisting, and translating on a Kinova Gen3 robot.

5.8.2. Offline Training of the Hybrid Motion Planning Method

All offline training is executed on an 8-core workstation processor paired with an Nvidia GPU. For effective offline training, 100 FPI tasks are generated with randomly chosen starting and goal positions in the work cell. This diverse task set allows us to evaluate the system's performance across various scenarios. First, Algorithm 5.1 is employed to train the task space HRL-LfD motion planner. After training, the feasibility analysis is conducted to identify infeasible segments. These segments' starting and ending configurations are then utilized as input data to train the joint space DRL motion planner using Algorithm 5.2. For comparison, a purely DRL-based method is trained within the entire workspace by following the same task constraints. During the offline training phase, the radius of the target region is set to a tolerance of 25 centimeters. Under these settings, the DRL-based method can learn a steady policy within an acceptable time frame.

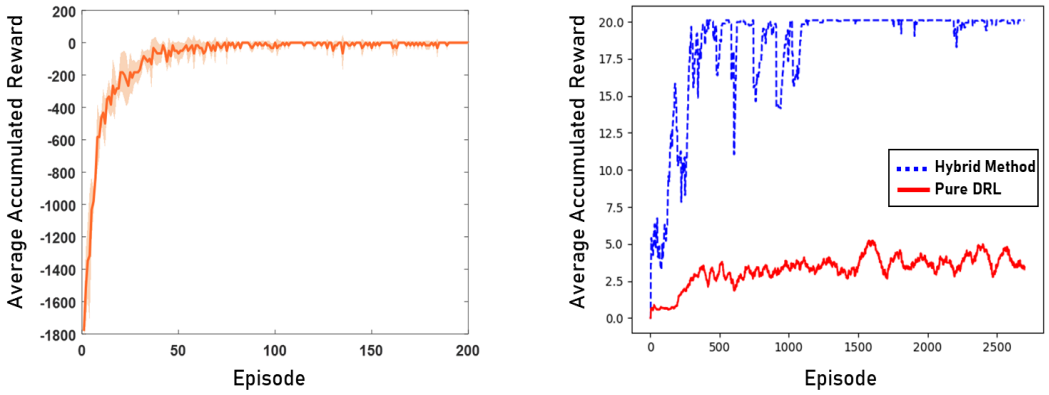


Figure 5.6. Offline training of the HRL-LfD method (left) and the offline training of the DRL method (right).

As shown in the left plot of Fig. 5.6, after around 100 episodes, the HRL-LfD motion planner can obtain a steady policy that optimizes the average accumulated reward. This high training efficiency is achieved as the proposed HRL-LfD method learns the kinematic features, which remain consistent across similar tasks despite variations in positions and orientations. In the right plot of Fig. 5.6, the learning curve of DRL in the hybrid approach reaches a stable policy after around 1250 episodes. In contrast, the purely DRL method struggles to converge. Furthermore, the average accumulated reward obtained using the purely DRL approach is notably lower compared to the hybrid method. This discrepancy can be attributed to the inherent difficulty of the purely DRL method in searching for feasible configurations, ultimately leading to reduced rewards. The delayed progress in training of the purely DRL method stems from its learning from scratch, where it searches the feasible motion planning policy within the whole space to comprehend position and orientation task constraints. The hybrid approach, however, leverages the strengths of both task space HRL-LfD and joint space DRL. The trained HRL-LfD policy plans the trajectory based on the task-level understanding and the demonstrated features of skills, which does not need to be retrained when tasks change. In addition, the knowledge of the infeasible segments transferred from the HRL-LfD method, and the feasibility map significantly reduces the searching space of the DRL based method and avoid

learning from scratch, resulting in the high efficiency and adaptivity in the offline training.

5.8.3. Online Execution of the Hybrid Motion Planning Method

Trained policies from both the proposed hybrid method and the purely DRL-based method are employed to execute ten distinct painting tasks on ten blades. For each task, a total of 100 execution trials are conducted. A trial is deemed successful when the robot can reach all essential configurations, covering both position and orientation. This includes reaching the container, as well as the initial and ending configurations on each blade, all within a 5-centimeter tolerance for positions and a 5-degree tolerance for each Euler angle in the critical orientation. Fig. 5.7 illustrates the success rates of both approaches, highlighting the significant superiority of the hybrid method in successfully completing all tasks compared to the DRL approach. This contrast arises from the fact that DRL-based training relies on a less stringent tolerance for achieving a target configuration to establish a relatively stable policy. Conversely, the hybrid method capitalizes on HRL-LfD to improve accuracy within the task space.

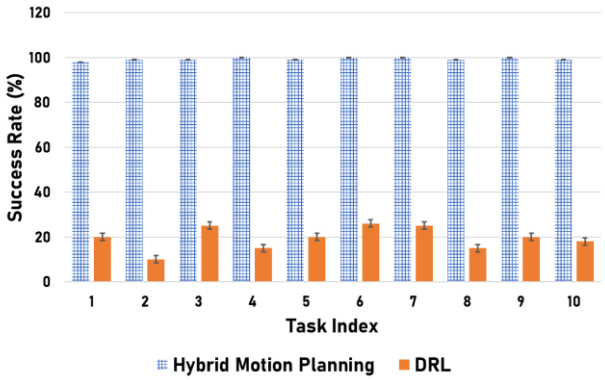


Figure 5.7. Online execution of the proposed hybrid motion planning method and the purely DRL method.

In conclusion, the proposed hybrid motion planning method demonstrates both effectiveness and efficiency in generating feasible trajectories that satisfy task requirements, thereby providing significant benefits over methods solely based on DRL.

5.9. Summary

This research presented a novel hybrid robot motion planning method that integrates the strengths of the task space HRL-LfD method and the joint space DRL-based motion planning approach while minimizing their inherent limitations. Through training of the DRL-based method using identified infeasible segments of the HRL-LfD trajectory, the proposed methodology ensures a significant improvement in training efficiency and the generation of a feasible motion planning policy.

Chapter 6. Conclusion and Future Work

Smart manufacturing systems depend on automation and robotics, whereas HRC contributes to increasing productivity and flexibility of today's and future factories. The concept of lean and hybrid automation with balanced introduction of human flexibility and robot efficiency are on the rise. However, the desired flexibility and human-robot co-existence have led to higher total complexity of manufacturing systems. This dissertation addresses key challenges in the realm of HRC in smart manufacturing by making significant contributions in two main areas: task scheduling and robot motion planning.

For task scheduling in complex HRC environments, the dissertation introduces a novel approach that conceptualizes the HRC assembly process as a chessboard game. Task constraints are embedded within this game structure, and advanced reinforcement learning techniques are employed to optimize assembly task completion. This approach demonstrates its efficiency even when dealing with a large number of tasks and complex task structures, showing promise for broader application in manufacturing systems.

In the field of robot motion planning, the dissertation presents a comprehensive solution to address the scalability and adaptability challenges of existing methods. It introduces an innovative RL-based Learning from Demonstration (LfD) method, enabling robots to learn from a limited number of demonstrations for various task instances. Additionally, a joint space coordination scheme for mobile manipulators enhances execution precision and computational efficiency. Furthermore, a hybrid motion planning approach combines the strengths of task space RL-LfD and joint space DRL methods, significantly increasing training efficiency while ensuring trajectory feasibility.

The dissertation makes contributions to the field of HRC in smart manufacturing with respect to the following aspects:

(1) Development of a novel framework to format the typical HRC assembly process as a chessboard game with specific mapping and gameplay rules. Such formatting not only simplifies the task scheduling problem by embedding constraints within the game rules but also allows for the utilization of powerful RL/MARL to address complex task scheduling in the HRC system, which is characterized by large state and action spaces.

(2) Development of a scalable and adaptable RL-LfD method for reducing reprogramming in smart manufacturing systems and collaboration with non-expert human operators. This approach allows for generalization across different initial and final configurations, task scales, and accommodates path disturbances while remaining adaptable to various robot platforms. It achieves scalability by requiring only a minimal number of examples from human operators and offers customization for diverse scenarios across manufacturing systems.

(3) Development of a unique joint space coordination scheme for the mobile base and manipulator. This coordination scheme not only aligns manipulation constraints with task requirements but also improves end-effector trajectory tracking accuracy while managing joint limits and optimizing mobile base movements based on innovative task-oriented manipulability.

(4) Development of a hybrid robot motion planning method that combines a task space LfD and a joint space DRL approach to leverage their strengths while addressing their limitations. In offline training, using the feasibility map, data from infeasible segments of the LfD trajectory train the DRL policy, significantly reducing the search space, improving training efficiency, and enhancing adaptability to different tasks compared to pure DRL methods. During online execution, the final motion plan ensures both joint space feasibility and adherence to task constraints, providing a comprehensive solution for robot motion planning in constrained environments.

In the era of Industry 5.0, a prominent paradigm shift emphasizes human-centered manufacturing. There is a growing aspiration to establish Human-Robot Collaboration systems in smart manufacturing that are not only safer and more efficient but also highly adaptable, with a primary focus on mitigating human factors. This dissertation suggests a few potential research directions to further this aim:

(1) Human Factors and Safety:

- Investigate human-robot interaction (HRI) dynamics to design more intuitive and safe interfaces for non-expert human operators working alongside robots in complex manufacturing environments.
- Develop safety protocols and mechanisms that ensure the physical and cognitive well-being of human workers in HRC settings, including collision avoidance, emergency stop procedures, and ergonomic considerations.
- Conduct comprehensive user studies and ergonomic assessments to refine the robot's behavior and workspace design, optimizing both task efficiency and human comfort.

(2) Multi-Robot Collaboration:

- Extend the research to address multi-robot collaboration scenarios, where multiple robots collaborate on tasks while ensuring safe and efficient interactions between them.
- Investigate methods for coordinating actions and task allocation among multiple robots in a way that maximizes overall system efficiency and minimizes conflicts.
- Develop techniques for adaptive task distribution among robots based on dynamic changes in task requirements and environmental conditions.

(3) Human-in-the-Loop Collaboration:

- Explore strategies for seamless collaboration between human operators and robots in dynamic HRC scenarios, including methods for human intervention and control when needed.
- Develop user-friendly interfaces that enable non-expert operators to provide high-level guidance and adjust robot behavior in real-time, ensuring flexibility and adaptability in changing manufacturing conditions.
- Study the cognitive workload of human operators when collaborating with robots and design interfaces that minimize cognitive fatigue and enhance decision-making in time-critical situations.

Bibliography

[1] D. Kofer, C. Bergner, C. Deuerlein, R. Schmidt-Vollus, and P. Heß, "Human–robot-collaboration: Innovative processes, from research to series standard," *Procedia CIRP*, vol. 97, pp. 98–103, Jan. 2021, doi: 10.1016/J.PROCIR.2020.09.185.

[2] X. Xu, Y. Lu, B. Vogel-Heuser, and L. Wang, "Industry 4.0 and Industry 5.0—Inception, conception and perception," *J. Manuf. Syst.*, vol. 61, pp. 530–535, Oct. 2021, doi: 10.1016/J.JMSY.2021.10.006.

[3] J. Krüger, T. Lien, A. V.-C. annals, and undefined 2009, "Cooperation of human and machines in assembly lines," *Elsevier*, Accessed: Feb. 29, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0007850609001760.

[4] N. Pedrocchi, F. Vicentini, M. Malosio, and L. M. Tosatti, "Safe human-robot cooperation in an industrial environment," *Int. J. Adv. Robot. Syst.*, vol. 10, Jan. 2013, doi: 10.5772/53939.

[5] D. Kragic, J. Gustafson, H. Karaoguz, P. Jensfelt, and R. Krug, “Interactive, Collaborative Robots: Challenges and Opportunities.,” in *IJCAI*, 2018, pp. 18–25.

[6] R. Laha, L. F. C. Figueredo, J. Vrabel, A. Swikir, and S. Haddadin, “Reactive cooperative manipulation based on set primitives and circular fields,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 6577–6584.

[7] J. Aleotti and S. Caselli, “Robust trajectory learning and approximation for robot programming by demonstration,” *Rob. Auton. Syst.*, vol. 54, no. 5, pp. 409–413, 2006.

[8] R. Maderna, M. Pozzi, A. M. Zanchettin, P. Rocco, and D. Prattichizzo, “Flexible Scheduling and Tactile Communication for Human-Robot Collaboration,” 2020.

[9] P. Tsarouchi, A. S. Matthaiakis, S. Makris, and G. Chryssolouris, “On a human-robot collaboration in an assembly cell,” *Int. J. Comput. Integr. Manuf.*, vol. 30, no. 6, pp. 580–589, 2017, doi: 10.1080/0951192X.2016.1187297.

[10] S. Pellegrinelli, A. Orlandini, N. Pedrocchi, A. Umbrico, and T. Tolio, “Motion planning and scheduling for human and industrial-robot collaboration,” *CIRP Ann.*, vol. 66, no. 1, pp. 1–4, 2017.

[11] D. Silver *et al.*, “Mastering the game of Go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484–489, 2016, doi: 10.1038/nature16961.

[12] D. Silver *et al.*, “Mastering the game of Go without human knowledge,” *Nature*, vol. 550, no. 7676, pp. 354–359, 2017, doi: 10.1038/nature24270.

[13] L. G. Apr, “Deep Q-Learning for Nash Equilibria: Nash-DQN ˚,” pp. 1–16, 2018.

[14] J. Su, J. Huang, S. Adams, Q. Chang, and P. A. Beling, “Deep multi-agent

reinforcement learning for multi-level preventive maintenance in manufacturing systems," *Expert Syst. Appl.*, vol. 192, p. 116323, Apr. 2022, doi: 10.1016/J.ESWA.2021.116323.

[15] S. Baer, J. Bakakeu, R. Meyes, and T. Meisen, "Multi-Agent Reinforcement Learning for Job Shop Scheduling in Flexible Manufacturing Systems," *2019 Second Int. Conf. Artif. Intell. Ind.*, pp. 22–25, 2019, doi: 10.1109/AI4I46381.2019.00014.

[16] T. Yu, J. Huang, and Q. Chang, "Mastering the working sequence in human-robot collaborative assembly based on reinforcement learning," *IEEE Access*, vol. 8, pp. 163868–163877, 2020, doi: 10.1109/ACCESS.2020.3021904.

[17] T. Yu, J. Huang, and Q. Chang, "Optimizing task scheduling in human-robot collaboration with deep multi-agent reinforcement learning," *J. Manuf. Syst.*, vol. 60, pp. 487–499, 2021.

[18] A. Shkolnik and R. Tedrake, "Path planning in 1000+ dimensions using a task-space Voronoi bias," in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 2061–2067.

[19] R. Seyboldt, C. Frese, and A. Zube, "Sampling-based path planning to cartesian goal positions for a mobile manipulator exploiting kinematic redundancy," in *Proceedings of ISR 2016: 47st International Symposium on Robotics*, 2016, pp. 1–9.

[20] Q. Yu *et al.*, "Base position optimization for mobile painting robot manipulators with multiple constraints," *Robot. Comput. Integr. Manuf.*, vol. 54, pp. 56–64, 2018.

[21] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *2011 IEEE international conference on robotics and automation*, 2011, pp. 4569–4574.

[22] R. Terasawa, Y. Ariki, T. Narihira, T. Tsuboi, and K. Nagasaka, “3D-CNN based heuristic guided task-space planner for faster motion planning,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 9548–9554.

[23] D. Mukherjee, K. Gupta, L. H. Chang, and H. Najjaran, “A survey of robot learning strategies for human-robot collaboration in industrial settings,” *Robot. Comput. Integr. Manuf.*, vol. 73, p. 102231, 2022.

[24] D. Kulić, W. Takano, and Y. Nakamura, “Incremental learning, clustering and hierarchy formation of whole body motion patterns using adaptive hidden markov chains,” *Int. J. Rob. Res.*, vol. 27, no. 7, pp. 761–784, 2008.

[25] M. Petrović, Z. Miljković, and A. Jokić, “A novel methodology for optimal single mobile robot scheduling using whale optimization algorithm,” *Appl. Soft Comput.*, vol. 81, p. 105520, Aug. 2019, doi: 10.1016/J.ASOC.2019.105520.

[26] J. Duan, Y. Ou, S. Xu, and M. Liu, “Sequential learning unification controller from human demonstrations for robotic compliant manipulation,” *Neurocomputing*, vol. 366, pp. 35–45, 2019, doi: 10.1016/j.neucom.2019.07.081.

[27] O. Kilinc and G. Montana, “Reinforcement learning for robotic manipulation using simulated locomotion demonstrations,” *Mach. Learn.*, vol. 111, no. 2, pp. 465–486, 2022, doi: 10.1007/s10994-021-06116-1.

[28] S. El Zaatari, Y. Wang, Y. Hu, and W. Li, “An improved approach of task-parameterized learning from demonstrations for cobots in dynamic manufacturing,” *J. Intell. Manuf.*, vol. 33, no. 5, pp. 1503–1519, 2022, doi: 10.1007/s10845-021-01743-w.

[29] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard, “Recent advances in robot learning from demonstration,” *Annu. Rev. Control. Robot. Auton. Syst.*, vol. 3, pp. 297–330, 2020.

[30] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, “A survey of robot learning from demonstration,” *Rob. Auton. Syst.*, vol. 57, no. 5, pp. 469–483, 2009.

[31] T. Yu and Q. Chang, “User-guided motion planning with reinforcement learning for human-robot collaboration in smart manufacturing,” *Expert Syst. Appl.*, p. 118291, 2022.

[32] A. Bauer, D. Wollherr, and M. Buss, “Human-robot collaboration: A survey,” *Int. J. Humanoid Robot.*, vol. 5, no. 1, pp. 47–66, Mar. 2008, doi: 10.1142/S0219843608001303.

[33] T. Fong, I. Nourbakhsh, and K. Dautenhahn, “A survey of socially interactive robots: Concepts, Design and Applications,” 2002, Accessed: Feb. 29, 2020. [Online]. Available: http://uhra.herts.ac.uk/handle/2299/3821.

[34] G. Michalos, S. Makris, J. Spiliotopoulos, … I. M.-P., and undefined 2014, “ROBO-PARTNER: Seamless human-robot cooperation for intelligent, flexible and safe operations in the assembly factories of the future,” *academia.edu*, Accessed: Feb. 29, 2020. [Online]. Available: https://www.academia.edu/download/46559479/ROBO-PARTNER_Seamless_Human-Robot_Cooper20160616-16031-18g6a59.pdf.

[35] N. Papakostas, G. Michalos, S. Makris, D. Zouzias, and G. Chryssolouris, “Industrial Applications with Cooperating Robots for the Flexible Assembly,” *Int. J. Comput. Integr. Manuf.*, vol. 24, no. 7, pp. 650–660, Jul. 2011, doi: 10.1080/0951192X.2011.570790.

[36] J. T. C. Tan, F. Duan, Y. Zhang, K. Watanabe, R. Kato, and T. Arai, “Human-robot collaboration in cellular manufacturing: Design and development,” *2009 IEEE/RSJ Int. Conf. Intell. Robot. Syst. IROS 2009*, pp. 29–34, 2009, doi: 10.1109/IROS.2009.5354155.

[37] J. Colgate, J. Edward, and M. Peshkin, “Cobots: Robots for collaboration with human operators,” *Citeseer*, 1996, Accessed: Feb. 29, 2020. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.7236.

[38] E. Helms, R. D. Schraft, and M. Hägele, “rob @ work : Robot Assistant in Industrial Environments.”

[39] R. D. Schraft, C. Meyer, C. Parlitz, and E. Helms, “PowerMate - A safe and intuitive robot assistant for handling and assembly tasks,” *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2005, no. April, pp. 4074–4077, 2005, doi: 10.1109/ROBOT.2005.1570745.

[40] K. Kosuge, S. Hashimoto, and H. Yoshida, “Human-robots collaboration system for flexible object handling,” *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2, no. May, pp. 1841–1846, 1998, doi: 10.1109/ROBOT.1998.677435.

[41] F. Mizoguchi, H. Hiraishi, and H. Nishiyama, “Human-robot collaboration in the smart office environment,” *IEEE/ASME Int. Conf. Adv. Intell. Mechatronics, AIM*, pp. 79–84, 1999, doi: 10.1109/aim.1999.803146.

[42] H. Modares, I. Ranatunga, F. L. Lewis, and D. O. Popa, “Optimized Assistive Human-Robot Interaction Using Reinforcement Learning,” *IEEE Trans. Cybern.*, vol. 46, no. 3, pp. 655–667, 2016, doi: 10.1109/TCYB.2015.2412554.

[43] C. Liu, T. Tang, H.-C. Lin, Y. Cheng, and M. Tomizuka, “SERoCS: Safe and Efficient Robot Collaborative Systems for Next Generation Intelligent Industrial Co-Robots,” 2018, [Online]. Available: http://arxiv.org/abs/1809.08215.

[44] R. Wilcox, S. Nikolaidis, and J. Shah, “Optimization of Temporal Dynamics for Adaptive Human-Robot Interaction in Assembly Manufacturing Terms of Use Creative Commons Attribution-Noncommercial-Share Alike Optimization of Temporal Dynamics for

Adaptive Human-Robot Interaction in Assembly Manufac," 2013, doi: 10.15607/RSS.2012.VIII.056.

[45] S. Wang, J. Wan, D. Li, and C. Zhang, "Implementing Smart Factory of Industrie 4.0: An Outlook," *Int. J. Distrib. Sens. Networks*, vol. 12, no. 1, p. 3159805, Jan. 2016, doi: 10.1155/2016/3159805.

[46] J. Shahrabi, M. A. Adibi, and M. Mahootchi, "A reinforcement learning approach to parameter estimation in dynamic job shop scheduling," *Comput. Ind. Eng.*, vol. 110, pp. 75–82, 2017, doi: 10.1016/j.cie.2017.05.026.

[47] J. A. Ramírez-Hernández and E. Fernandez, "Optimization of preventive maintenance scheduling in semiconductor manufacturing models using a simulation-based approximate dynamic programming approach," *Proc. IEEE Conf. Decis. Control*, pp. 3944–3949, 2010, doi: 10.1109/CDC.2010.5717523.

[48] J. Huang, Q. Chang, and N. Chakraborty, "Machine preventive replacement policy for serial production lines based on reinforcement learning," *IEEE Int. Conf. Autom. Sci. Eng.*, vol. 2019-Augus, pp. 523–528, 2019, doi: 10.1109/COASE.2019.8843338.

[49] G. Qu *et al.*, "Scalable Reinforcement Learning of Localized Policies for Multi-Agent Networked Systems," PMLR, Jul. 2020. Accessed: Apr. 14, 2021. [Online]. Available: http://proceedings.mlr.press/v120/qu20a.html.

[50] J. Hu and M. P. Wellman, "Nash Q-learning for general-sum stochastic games," *J. Mach. Learn. Res.*, vol. 4, no. Nov, pp. 1039–1069, 2003.

[51] X. Xue, Z. Li, D. Zhang, and Y. Yan, "A Deep Reinforcement Learning Method for Mobile Robot Collision Avoidance based on Double DQN," *IEEE Int. Symp. Ind. Electron.*, vol. 2019-June, pp. 2131–2136, 2019, doi: 10.1109/ISIE.2019.8781522.

[52] O. Vinyals *et al.*, "Grandmaster level in StarCraft II using multi-agent

reinforcement learning," *Nature*, vol. 575, no. August, 2019, doi: 10.1038/s41586-019-1724-z.

[53] J. Munk, J. Kober, and R. Babuska, "Learning state representation for deep actor-critic control," in *2016 IEEE 55th Conference on Decision and Control, CDC 2016*, Dec. 2016, pp. 4667–4673, doi: 10.1109/CDC.2016.7798980.

[54] S. El Zaatari, M. Marei, W. Li, and Z. Usman, "Cobot programming for collaborative industrial tasks: An overview," *Rob. Auton. Syst.*, vol. 116, pp. 162–180, 2019.

[55] M. Jurczyk-Bunkowska, "Using discrete event simulation for planning improvement in small batch size manufacturing system," in *Sustainable Production: Novel Trends in Energy, Environment and Material Systems*, Springer, 2020, pp. 19–43.

[56] T. C. Poon *et al.*, "A real-time warehouse operations planning system for small batch replenishment problems in production environment," *Expert Syst. Appl.*, vol. 38, no. 7, pp. 8524–8537, 2011, doi: https://doi.org/10.1016/j.eswa.2011.01.053.

[57] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE J. Robot. Autom.*, vol. 3, no. 1, pp. 43–53, 1987.

[58] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, 1996.

[59] D. Berenson, S. S. Srinivasa, D. Ferguson, and J. J. Kuffner, "Manipulation planning on constraint manifolds," in *2009 IEEE international conference on robotics and automation*, 2009, pp. 625–632.

[60] L. Jaillet and J. M. Porta, "Path planning under kinematic constraints by

rapidly exploring manifolds," *IEEE Trans. Robot.*, vol. 29, no. 1, pp. 105–117, 2012.

[61] C. A. Klein and C.-H. Huang, "Review of pseudoinverse control for use with kinematically redundant manipulators," *IEEE Trans. Syst. Man. Cybern.*, no. 2, pp. 245–250, 1983.

[62] J. Nakanishi, R. Cory, M. Mistry, J. Peters, and S. Schaal, "Operational space control: A theoretical and empirical comparison," *Int. J. Rob. Res.*, vol. 27, no. 6, pp. 737–757, 2008.

[63] Z. Zhu and H. Hu, "Robot learning from demonstration in robotic assembly: A survey," *Robotics*, vol. 7, no. 2, p. 17, 2018.

[64] J.-H. Hwang, R. C. Arkin, and D.-S. Kwon, "Mobile robots at your fingertip: Bezier curve on-line trajectory generation for supervisory control," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, 2003, vol. 2, pp. 1444–1449.

[65] Á. Madridano, A. Al-Kaff, D. Martín, and A. de la Escalera, "Trajectory planning for multi-robot systems: Methods and applications," *Expert Syst. Appl.*, vol. 173, p. 114660, 2021, doi: https://doi.org/10.1016/j.eswa.2021.114660.

[66] S. Calinon, F. Guenter, and A. Billard, "On learning, representing, and generalizing a task in a humanoid robot," *IEEE Trans. Syst. Man, Cybern. Part B*, vol. 37, no. 2, pp. 286–298, 2007.

[67] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: learning attractor models for motor behaviors," *Neural Comput.*, vol. 25, no. 2, pp. 328–373, 2013.

[68] E. Gribovskaya, S. M. Khansari-Zadeh, and A. Billard, "Learning non-linear multivariate dynamics of motion in robotic manipulators," *Int. J. Rob. Res.*, vol. 30, no. 1, pp. 80–117, 2011.

[69] A. Jokić, M. Petrović, and Z. Miljković, “Semantic segmentation based stereo visual servoing of nonholonomic mobile robot in intelligent manufacturing environment,” *Expert Syst. Appl.*, vol. 190, p. 116203, 2022, doi: https://doi.org/10.1016/j.eswa.2021.116203.

[70] J. M. Selig, *Geometric fundamentals of robotics*, vol. 128. Springer, 2005.

[71] L. F. da C. Figueredo, “Kinematic control based on dual quaternion algebra and its application to robot manipulators,” 2016.

[72] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez, “From skills to symbols: Learning symbolic representations for abstract high-level planning,” *J. Artif. Intell. Res.*, vol. 61, pp. 215–289, 2018.

[73] Z. Wang, C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez, “Active model learning and diverse action sampling for task and motion planning,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 4107–4114.

[74] K. Hauser and V. Ng-Thow-Hing, “Randomized multi-modal motion planning for a humanoid robot manipulation task,” *Int. J. Rob. Res.*, vol. 30, no. 6, pp. 678–698, 2011.

[75] R. Laha, A. Rao, L. F. C. Figueredo, Q. Chang, S. Haddadin, and N. Chakraborty, “Point-to-point path planning based on user guidance and screw linear interpolation,” in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 2021, vol. 85451, p. V08BT08A010.

[76] L. Kavan, S. Collins, C. O’Sullivan, and J. Zara, “Dual quaternions for rigid transformation blending,” *Trinity Coll. Dublin, Tech. Rep. TCD-CS-2006-46*, 2006.

[77] D. P. Bovet, P. Crescenzi, and D. Bovet, *Introduction to the Theory of Complexity*, vol. 7. Prentice Hall London, 1994.

[78] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*.

MIT press, 2018.

[79] F. Flacco, A. De Luca, and O. Khatib, “Control of redundant robots under hard joint constraints: Saturation in the null space,” *IEEE Trans. Robot.*, vol. 31, no. 3, pp. 637–654, 2015.

[80] S. Moe, G. Antonelli, A. R. Teel, K. Y. Pettersen, and J. Schrimpf, “Set-based tasks within the singularity-robust multiple task-priority inverse kinematics framework: General formulation, stability analysis, and experimental results,” *Front. Robot. AI*, vol. 3, p. 16, 2016.

[81] C. Schou, R. S. Andersen, D. Chrysostomou, S. Bøgh, and O. Madsen, “Skill-based instruction of collaborative robots in industrial settings,” *Robot. Comput. Integr. Manuf.*, vol. 53, pp. 72–80, 2018.

[82] J. Meng *et al.*, “Iterative-learning error compensation for autonomous parking of mobile manipulator in harsh industrial environment,” *Robot. Comput. Integr. Manuf.*, vol. 68, p. 102077, 2021.

[83] M. R. Pedersen *et al.*, “Robot skills for manufacturing: From concept to industrial deployment,” *Robot. Comput. Integr. Manuf.*, vol. 37, pp. 282–291, 2016.

[84] F. Gul, I. Mir, L. Abualigah, P. Sumari, and A. Forestiero, “A consolidated review of path planning and optimization techniques: Technical perspectives and future directions,” *Electron.*, vol. 10, no. 18, pp. 1–38, 2021, doi: 10.3390/electronics10182250.

[85] N. Botteghi, B. Sirmacek, K. A. A. Mustafa, M. Poel, and S. Stramigioli, “On Reward Shaping for Mobile Robot Navigation: A Reinforcement Learning and SLAM Based Approach,” 2020, [Online]. Available: http://arxiv.org/abs/2002.04109.

[86] T. Sandakalum and M. H. Ang Jr, “Motion planning for mobile manipulators—a systematic review,” *Machines*, vol. 10, no. 2, p. 97, 2022.

[87] T. Yu and Q. Chang, “Reinforcement Learning Based User-Guided Motion

Planning for Human-Robot Collaboration," *arXiv Prepr. arXiv2207.00492*, 2022.

[88] L. Petrović, J. Peršić, M. Seder, and I. Marković, "Cross-entropy based stochastic optimization of robot trajectories using heteroscedastic continuous-time Gaussian processes," *Rob. Auton. Syst.*, vol. 133, p. 103618, 2020.

[89] K. Watanabe, K. Kiguchi, K. Izumi, and Y. Kunitake, "Path planning for an omnidirectional mobile manipulator by evolutionary computation," in *1999 Third International Conference on Knowledge-Based Intelligent Information Engineering Systems. Proceedings (Cat. No. 99TH8410)*, 1999, pp. 135–140.

[90] J. Wang and A. Chortos, "Control Strategies for Soft Robot Systems," *Adv. Intell. Syst.*, vol. 4, no. 5, p. 2100165, 2022, doi: 10.1002/aisy.202100165.

[91] J. A. Abdor-Sierra, E. A. Merchán-Cruz, and R. G. Rodríguez-Cañizo, "A comparative analysis of metaheuristic algorithms for solving the inverse kinematics of robot manipulators," *Results Eng.*, vol. 16, no. August, 2022, doi: 10.1016/j.rineng.2022.100597.

[92] K. Vazquez-Santiago, C. F. Goh, and K. Shimada, "Motion Planning for Kinematically Redundant Mobile Manipulators with Genetic Algorithm, Pose Interpolation, and Inverse Kinematics," *IEEE Int. Conf. Autom. Sci. Eng.*, vol. 2021-Augus, pp. 1167–1174, 2021, doi: 10.1109/CASE49439.2021.9551546.

[93] N. Castaman, E. Pagello, E. Menegatti, and A. Pretto, "Receding Horizon Task and Motion Planning in Changing Environments," *Rob. Auton. Syst.*, vol. 145, p. 103863, 2021.

[94] A. Rastegarpanah, H. C. Gonzalez, and R. Stolkin, "Semi-autonomous behaviour tree-based framework for sorting electric vehicle batteries components," *Robotics*, vol. 10, no. 2, pp. 1–18, 2021, doi:

10.3390/robotics10020082.

[95] S. Thakar *et al.*, “A Survey of Wheeled Mobile Manipulation: A Decision-Making Perspective,” *J. Mech. Robot.*, vol. 15, no. 2, 2023, doi: 10.1115/1.4054611.

[96] R. Laha, A. Rao, L. F. C. Figueredo, Q. Chang, S. Haddadin, and N. Chakraborty, “Point-to-Point Path Planning Based on User Guidance and Screw Linear Interpolation,” *Proc. ASME Des. Eng. Tech. Conf.*, vol. 8B-2021, Nov. 2021, doi: 10.1115/DETC2021-71814.

[97] T. Yoshikawa, “Manipulability of robotic mechanisms,” *Int. J. Rob. Res.*, vol. 4, no. 2, pp. 3–9, 1985.

[98] T. Sugihara, “Solvability-unconcerned inverse kinematics by the Levenberg-Marquardt method,” *IEEE Trans. Robot.*, vol. 27, no. 5, pp. 984–991, Oct. 2011, doi: 10.1109/TRO.2011.2148230.

[99] Y. Lu *et al.*, “Outlook on human-centric manufacturing towards Industry 5.0,” *J. Manuf. Syst.*, vol. 62, pp. 612–627, Jan. 2022, doi: 10.1016/J.JMSY.2022.02.001.

[100] J. Zhang, H. Liu, Q. Chang, L. Wang, and R. X. Gao, “Recurrent neural network for motion trajectory prediction in human-robot collaborative assembly,” *CIRP Ann.*, vol. 69, no. 1, pp. 9–12, 2020, doi: 10.1016/j.cirp.2020.04.077.

[101] J. Yamada *et al.*, “Motion Planner Augmented Reinforcement Learning for Robot Manipulation in Obstructed Environments,” no. CoRL, pp. 1–15, 2020, [Online]. Available: http://arxiv.org/abs/2010.11940.

[102] X. Yang, Z. Ji, J. Wu, and Y. K. Lai, “An Open-Source Multi-goal Reinforcement Learning Environment for Robotic Manipulation with Pybullet,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 13054 LNAI, no. 201908440400, pp. 14–24, 2021, doi: 10.1007/978-3-030-89177-0_2.

[103] D. Han, B. Mulyana, V. Stankovic, and S. Cheng, “A Survey on Deep Reinforcement Learning Algorithms for Robotic Manipulation,” *Sensors*, vol. 23, no. 7, pp. 1–35, 2023, doi: 10.3390/s23073762.

[104] J. N. Karigiannis *et al.*, “Reinforcement Learning Enabled Self-Homing of Industrial Robotic Manipulators in Manufacturing,” *Manuf. Lett.*, vol. 33, pp. 909–918, 2022, doi: 10.1016/j.mfglet.2022.07.111.

[105] L. Lu, M. Zhang, D. He, Q. Gu, D. Gong, and L. Fu, “A Method of Robot Grasping Based on Reinforcement Learning,” *J. Phys. Conf. Ser.*, vol. 2216, no. 1, 2022, doi: 10.1088/1742-6596/2216/1/012026.

[106] X. Zhang, S. Jin, C. Wang, X. Zhu, and M. Tomizuka, “Learning Insertion Primitives with Discrete-Continuous Hybrid Action Space for Robotic Assembly Tasks,” *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 1, pp. 9881–9887, 2022, doi: 10.1109/ICRA46639.2022.9811973.

[107] N. Vulin, S. Christen, S. Stevsic, and O. Hilliges, “Improved Learning of Robot Manipulation Tasks Via Tactile Intrinsic Motivation,” *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 2194–2201, 2021, doi: 10.1109/LRA.2021.3061308.

[108] A. Franceschetti, E. Tosello, N. Castaman, and S. Ghidoni, “Robotic Arm Control and Task Training Through Deep Reinforcement Learning,” *Lect. Notes Networks Syst.*, vol. 412 LNNS, pp. 532–550, 2022, doi: 10.1007/978-3-030-95892-3_41.

[109] T. Z. Zhao *et al.*, “Offline Meta-Reinforcement Learning for Industrial Insertion,” *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 6386–6393, 2022, doi: 10.1109/ICRA46639.2022.9812312.

[110] A. Bauer, D. Wollherr, M. B.-I. J. of Humanoid, and undefined 2008, “Human–robot collaboration: a survey,” *World Sci.*, Accessed: Feb. 29, 2020. [Online]. Available: https://www.worldscientific.com/doi/abs/10.1142/S0219843608001303.

[111] R. Laha, A. Rao, L. F. C. Figueredo, Q. Chang, S. Haddadin, and N. Chakraborty, “Point-to-Point Path Planning Based on User Guidance and Screw Linear Interpolation,” *Proc. ASME Des. Eng. Tech. Conf.*, vol. 8B-2021, Nov. 2021, doi: 10.1115/DETC2021-71814.

[112] P. Christodoulou, “Soft Actor-Critic for Discrete Action Settings,” pp. 1–7, 2019, [Online]. Available: http://arxiv.org/abs/1910.07207.

[113] A. Makhal and A. K. Goins, “Reuleaux: Robot base placement by reachability analysis,” *Proc. - 2nd IEEE Int. Conf. Robot. Comput. IRC 2018*, vol. 2018-Janua, pp. 137–142, Apr. 2018, doi: 10.1109/IRC.2018.00028.

[114] D. Torielli, L. Muratore, and N. Tsagarakis, “Manipulability-Aware Shared Locomanipulation Motion Generation for Teleoperation of Mobile Manipulators,” *IEEE Int. Conf. Intell. Robot. Syst.*, vol. 2022-Octob, pp. 6205–6212, 2022, doi: 10.1109/IROS47612.2022.9982220.

[115] M. Zhou and X. Zhang, “Fast Collision Checking for Dual-Arm Collaborative Robots Working in Close Proximity,” *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 243–249, 2022, doi: 10.1109/ICRA46639.2022.9811857.

[116] E. Coumans and Y. Bai, “Pybullet, a python module for physics simulation for games, robotics and machine learning,” 2016.

[117] J. Karigiannis *et al.*, “Multi-robot system for automated fluorescent penetrant indication inspection with deep neural nets,” *Procedia Manuf.*, vol. 53, pp. 735–740, 2021, doi: 10.1016/j.promfg.2021.06.072.