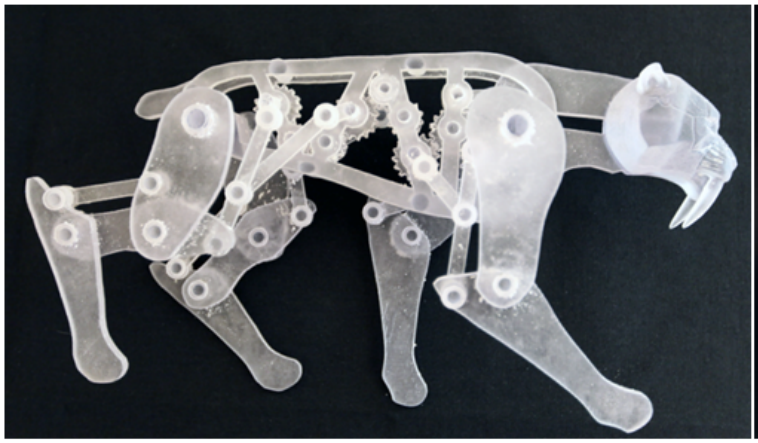
Task and Motion Re-Planning for Multi-Agent Systems




Wenkai Xuan

Master's Thesis

Computational Robotics Lab
ETH Zürich

Supervisors:
Miguel Zamora, Valentin N. Hartmann
Prof. Dr. Stelian Coros

May 15, 2025

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computational Robotics Lab

Acknowledgements

Chapter 0

I would like to thank Miguel and Valentin very much! Thanks for their guidance for me on this project. I have a great pleasure to share my happiness in finishing this project. During the past six months, I was proud to build the simulation environment on the *MuJoCo* platform and to train the network to guide the motion planning process.

After completing this thesis report, I am going to finish my master's program at ETH Zurich. Time passed swiftly—it feels like I had just bid farewell to my undergraduate life, and now I am preparing to say goodbye to my master's journey. Reflecting on nearly three years of master's study, the first year was particularly challenging, as it was my first time living abroad. I encountered numerous difficulties in adapting to a new lifestyle, especially in terms of eating habits and daily routines. Unlike my undergraduate years, where most aspects of life were taken care of by the university, studying in Switzerland required me to take full responsibility for daily living—buying groceries, cooking meals, and spending weekends with canteens closed. Additionally, the considerable distance between my home location and the university, which took about half an hour, was in great contrast to the convenience of undergraduate life, where the dormitories, the canteens, and the library were all within short distances. Initially, I found it extremely difficult to adjust, as the time and effort needed for daily life were significantly higher than what I was used to. This shift was particularly challenging for someone like me, who had previously devoted nearly all my time to studying. As a result, my academic performance during the first semester was terrible. Furthermore, language set another barrier. All courses were conducted in English, and daily communication often involved both English and even German. For someone entering an English-speaking environment for the first time, the adjustment period was inevitably demanding. It was not until the second and third semesters that I gradually adapted to both the academic and linguistic lifestyle.

Until the third semester, I had the opportunity to join the Computational Robotics Lab (CRL) through a semester project. I worked with Simon and Moritz on a project focusing on "deep learning accelerates grasp-optimized motion planning". The motion planning components were implemented entirely in C++, a language I had neither learned nor used before. Relying on my knowledge of C and the object-oriented programming concepts I had studied during the semester, I gradually learned to read and modify parts of the existing codebase and even add some new code. Prior to this experience, I would have considered it impossible to finish a C++-based project from scratch without any prior training. However, completing this project taught me that I was indeed capable of tackling such challenges—even if it required significantly more time and effort compared to a computer science student. As for the deep learning component, I was already familiar with implementing neural networks in Python. In retrospect, this project taught me a valuable lesson: it is neither necessary nor realistic to be completely prepared before solving on a new challenge. Just as I could not wait to become a C++ expert before starting this semester project, sometimes taking the first step is more important than waiting for the "perfect" moment—which may never come. In fact, excessive preparation may not even yield significantly better results.

This project was my first hands-on experience in the field of robotics, and it was through this work that I truly began engaging in robotics research. After completing this project and fulfilling my course credit requirements, I decided to write my master's thesis at CRL as well. Unlike the semester project, which focused on a single robotic arm performing the pick-and-place task, my master's thesis is about task assignment and motion planning for multi-agent systems. The thesis was supervised by Miguel and Valentin, to whom I am deeply grateful for their patience and guidance throughout the thesis. According to ETH regulations, the master thesis must last six months, which gives me the opportunity to work at a desk with a desktop in the CRL office. This experience was distinctly different from my previous study environments across ETH's HG, CHN, CAB buildings, or Hönggerberg campus. For the first time, I felt a sense of belonging. Although the workspace was only assigned to me for six to seven months, during this time it became a second home—with its own key, granting me access beyond my own living house. It was a place I could go whenever I needed a change from being at home. Indeed, over those few months, I spent nearly every weekend in the office, and for the first time, I no longer felt isolated. I would also like to express my sincere thanks to Yu Zhang, Haoyang Zhou, and Mingjie Li for their company and support in the lab. And of course, thanks to Ben, with whom I shared many happy conversations. Their presence made the process of the master thesis not only bearable, but enjoyable.

I would like also thanks for my friends, everyone graduating from University of Science and Technology of China, everyone in the Computational Science and Engineering master program, and everyone I met at ETH Zurich. Thank you for your company to make me feel less lonely.

Many thanks for my past self! Thank you for your efforts and wish you the best. Hope you can become the man you want to be in the future!

Abstract

Dealing with multiple robots comes with additional difficulties compared to single-robot settings: the motion planning for multiple agents must account for the movements of other robots, and task planning needs to consider not only when each task should be executed but also to which robot should be assigned to perform it. In performing prehensile manipulation, robots may make mistakes such as dropping objects during the grasping and moving process. At this moment, we must re-plan new motions for the whole robots-objects system. Considering these, we present a simulated workflow containing task and motion plans for multiple robots acting asynchronously in the same workspace and modifying the same environment. We consider prehensile manipulation in this workflow, and focus on various pick-drop-pick-and-place tasks. By executing these various tasks, we could obtain a simulated dataset. Training using these data to predict the makespan of a task sequence enables speeding up finding low-makepan sequences by ranking sequences before computing the full motion plan. By leveraging such a deep learning predictor, the motion planning process can be guided to produce shorter trajectories, thereby reducing the time required to perform tasks.

Contents

Introduction

Chapter 1

As robots become increasingly integrated into daily life, the research of robot collaboration recently catches people's attention. For complicated tasks, more robots are needed to cooperate with each other automatically in the workspace. The multi-agent systems are what we need to focus on. However, multi-agent systems come with many additional challenges, which is different from single agent systems, such as the task assignment between the robots in a single multi-agent system, and the increased difficulties of motion planning due to the other robots moving in the same workspace and the increasing degrees of freedom. Also, during operation, multi-agent systems may make mistakes: robots failing to grasp and steadily hold objects, resulting in objects being dropped during task execution. In this case, we have to consider the task and motion re-planning to solve this problem.

To achieve or even surpass the speed with which human perform tasks, we need to accelerate the task assignment and motion planning process for multi-agent systems. The early sampling-based motion planning approaches, like the Rapidly-exploring Random Trees (RRT) [1] or optimal Rapidly-exploring Random Trees (RRT*) [2] may have difficulties to be applied directly in multi-agent systems. Therefore, we should make some changes to these methods and apply some speed-up means. Some algorithms have employed heuristics to refine the RRT search, such as by biasing the sampling procedure [3] or formulating a series of sub-planning problems based on the current solution [4]. What's more, focusing techniques have been applied to restrict the search space of RRT* after an initial solution is found [5], [6], [7]. Although these methods can enhance the quality of the initial solution or accelerate convergence to the optimum, the underlying RRT-based search remains inherently unordered. Some algorithm also uses parallel computing to speed up the motion planning process [8].

Besides, learning-based approaches have gained attention recently. Learning-based approaches have the potential to improve the rate at which good solutions may be found. Learning-based method like [9] developed learned sampling strategies to accelerate planners like RRT* by focusing sampling where paths are likely. More directly, using neural networks to predict feasible trajectories to significantly reduce planning time [10] is also a good choice. In this paper, we aim to leverage the deep learning method to guide the motion planner to generate plans with small makespans.

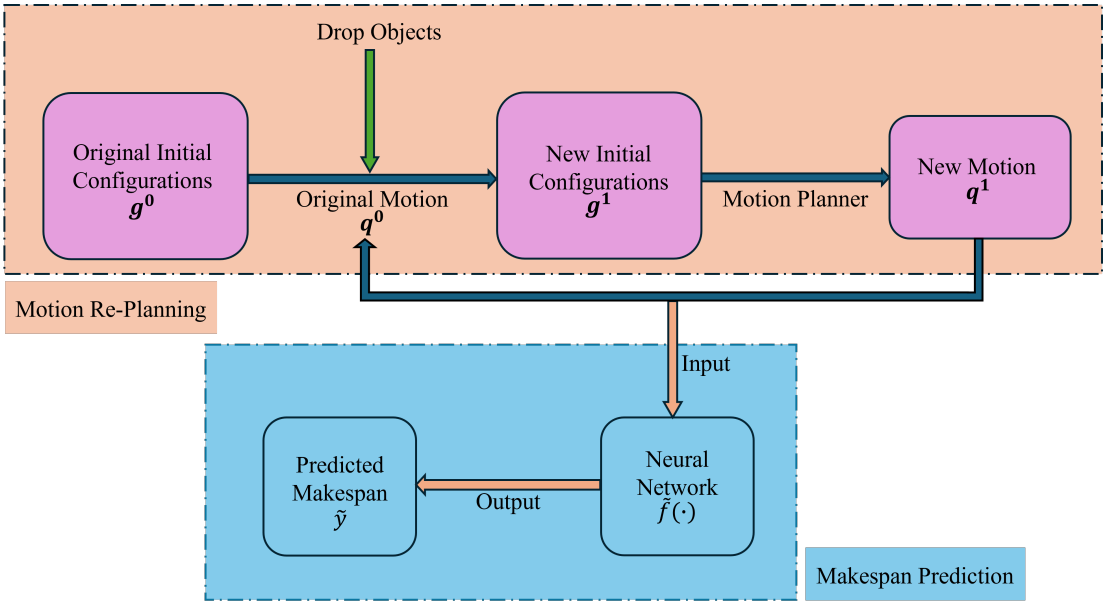


Figure 1.1: The pipeline of this paper. It mainly contains two parts: *Motion Re-Planning* and *Makespan Prediction*. *Motion Re-Planning* plans the new motions based on the new initial configurations, which correspond to the states of the original motions at the point where execution was disrupted due to objects drop. With enough replanned motions, we can train a neural network and use it to predict the makespan of the replanned motions(length of the motion trajectory) in *Makespan Prediction* part.

Based on the work of [11], we focus on the research of task and motion re-planning in our work. Our work mainly contains two parts: Motion Re-Planning and Makespan Prediction, as shown in Figure 1.1. Based on the initial task configurations $\mathbf{g}^0$, motion planner will give some motions $\mathbf{q}^0$ to solve the tasks. Motion re-planning refers to generating new motion trajectories $\mathbf{q}^1$ based on updated initial configurations $\mathbf{g}^1$, which arise when the original motions $\mathbf{q}^0$ are disrupted due to objects being dropped. And such new motion trajectories can also be treated as the original motions to generate other new motions. By generating original motions $\mathbf{q}^0$ and the re-planned motions $\mathbf{q}^1$, we can get enough data to train a neural network and use it to predict the makespan $\tilde{y}$ of the replanned motions(length of the motion trajectory). With this trained network, we can reversely use it to guide the motion planning to produce shorter trajectories, thereby reducing the time required to perform tasks.

1.1 Motion Re-Planning

In order to better analyze the conditions when robots dropping objects, we first imitates the real workspace in the virtual environment by a general purpose physics engine, *MuJoCo* (*Multi-Joint dynamics with Contact*) [12]. The robot model is *UR5e* [13] and the objects are described as boxes.

Our dataset contains four different base scenes of different difficulties: *"Random"* scene, with up to 4 robot arms with random base orientation and pose; *"Husky"* scene, with 2 robot arms on a husky base; *"Conveyor"* scene: a conveyor-like setting with 4 arms where objects have to be moved from the middle to the outside, and *"Shelf"* scene: a setting with a shelf and 2 arms.

1.2 Makespan Prediction

By executing various pick-drop-pick-and-place tasks, we generate a simulated dataset, on which a neural network is trained to predict the makespan of a task sequence. It can enable faster identification of low-makespan sequences by ranking them prior to the full motion planning computation.

Based on these datasets for each scene, we first train a makespan-predicting network for a specific scene separately and then train a unified network being able to predict the makespans of motion trajectories in each scene.

Related Work

Chapter 2

Task and Motion Planning (TAMP) addresses the task and motion planning problems simultaneously, aiming to determine what actions to take and how to execute them. In this paper, we focus on multi-agent task and motion planning as well as the learning method for TAMP.

2.1 Task and Motion Planning Methods

Sampling-based motion planning algorithms such as the Rapidly-exploring Random Trees (RRT) [1], optimal Rapidly-exploring Random Trees (RRT*) [2] and bi-directional Rapidly-exploring Random Trees (e.g., RRT-Connect [14]) have been the popularized solutions for generic motion planning problems, due to their efficiency in practice and completeness guarantee in theory.

Space-Time RRT* (ST-RRT*) is a probabilistically complete, bidirectional motion planning algorithm, which is asymptotically optimal with respect to the shortest arrival time [15]. Inspired by RRT-Connect [14] and RRT* [2], ST-RRT* introduced a motion planning algorithm designed for dynamic environments where obstacles may move over time, and agents must adhere to velocity constraints. It outperforms RRT-Connect [14] and RRT* [2] on both initial solution time, and attained final solution cost. ST-RRT* extends traditional sampling-based planners by incorporating the temporal dimension directly into the planning process. This method doesn't require a predefined arrival time. Instead, it incrementally expands the goal region in the time dimension, allowing the algorithm to discover feasible arrival times dynamically. Considering the velocity constraints, ST-RRT* employs a conditional sampling ensuring that only feasible states are sampled during the planning process. In order to improve the arrival time, ST-RRT* only preforms rewiring on the goal trees instead of start trees. This simplification on wiring reduces computational overhead while maintaining optimality guarantees.

Based on ST-RRT* [15], we solve multi-arm multi-task planning problem with the method introduced in [16]. It introduces the method of greedy descent with restarts for multi-robot task planning. This approach enables the optimization of the task sequence and assignment in multi-agent systems. The algorithm operates on a serialized sequence of tasks, which encodes the assignment of tasks to robots and the relative priorities between tasks. For each serialized sequence, ST-RRT* [15] is used for path planning and to evaluate its makespan. The path planning proceeds task by task according to the serialized order, treating previously computed paths as fixed constraints. The method implements a greedy descent search with random restarts to generate neighbor sequences, enabling the discovery of high-quality solutions while escaping local optima.

2.2 Learning in Task and Motion Planning

2.2.1 Learning-based Motion Planning

Numerous recent studies have introduced learning-based approaches that trade an extended offline training phase for accelerated online execution. Several of these methods enhance the efficiency of sampling-based motion planning by biasing the sampling distribution to guide the generation of new nodes [17], directly predicting the next node [10]. Learning-based method can accelerate the grasp-optimized motion planning by using the network-computed approximate motions as a starting point from which the optimizing motion planner refines to an optimized and feasible motion with few iterations [18]. Several studies have investigated the application of reinforcement learning techniques to motion planning [19], [20], [21], [22]. The method introduced in [20] combines multi-agent reinforcement learning (MARL) with expert demonstrations to learn a decentralized motion planning policy. This approach only applies expert demonstrations (from centralized BiRRT) when the current policy fails at a task. This selective approach allows the policy to learn from its own successes while getting guidance when needed. The system also continuously monitors the environment state and regenerates motion plans at each step, allowing adaptation to dynamic conditions.

Besides the usage of reinforcement learning in motion planning, our method use the motion planning data to predict the length of motion trajectory (makespan), based on the work of [11]. We focus on learning for multi-agent task and motion planning.

2.2.2 Large Language Model

Large language models (LLMs) have also been used to solve task and motion planning problems [23, 24]. Unlike the traditional learning methods requiring extensive training data for new tasks, LLM-involved approaches bridge this gap by combining the commonsense knowledge from LLMs with traditional robotic planning frameworks to allow robots to adapt to different task semantics, observation spaces, and workspace overlaps. They mainly use prompting to extract commonsense knowledge about semantically valid object configurations from an LLM and instantiates them with a task and motion planner in order to generalize to varying scene geometry [23, 24]. The LLM leverages the strong priors from language for the high-level communication in multi-agent systems. The workflow begins with a natural language service request. The system then processes this request to generate and execute a plan.

This novel method leverages large language models not only in single-robot rearrangement tasks [23, 25] but also in multi-robot collaboration [24, 26].

The method

Chapter 3

We first build the platform of the robot-object scenes in *MuJoCo*. Then we simulate and visualize the robots picking objects and objects dropping process on this platform. When the objects drop on the ground and stay statically, we output the current states of robots and objects. These states will then be used as new initial configurations, based on which we can use the motion planner to plan robots' next motion to pick and place the objects.

3.1 Motion Simulation and Re-Planning

In this section, we use *MuJoCo* to simulate and visualize the motions of robots and objects.

3.1.1 Motion Simulation

Four different base scenes need to be simulated and visualized: *"Random"*, with up to 4 arms with random base pose and up to 4 objects; *"Husky"*, with 2 arms on a husky base; *"Conveyor"*, a conveyor-like setting with 4 arms moving objects from the middle to the outside, and *"Shelf"* a setting with a shelf and 2 arms. In the *"Random"* and *"Conveyor"* scenes, all robots and objects are on the table/ground(Figure 3.1/Figure 3.2). The *"Husky"* and the *"Shelf"* scenes consist of more complicated configurations than the other two scenes. For the *Husky* scene, two robots are attached to a husky base and objects are on the table in front of the robots. In the *Shelf* scene two robots are positioned on the table/ground, while the objects are placed on a shelf in front of them. The task requires transferring these objects from the shelf to the table/ground. In these scenes, we randomize the number of robots, the number of objects, the size of objects, and their start and target locations to generate different initial configurations. In the *"Random"* scene, we additionally randomize the robots base positions and orientations. Based on these configurations, we will get motion trajectories to perform the tasks. The base scenarios are shown in Figure 3.1 and Figure 3.2.

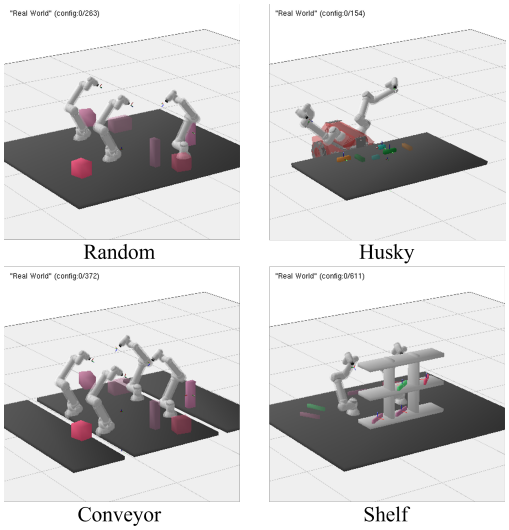


Figure 3.1: Four base scenes: *Random*, *Husky*, *Conveyor*, *Shelf* [11].

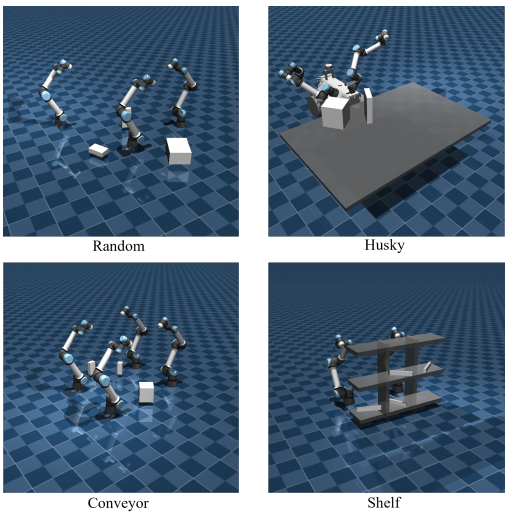


Figure 3.2: Four base scenes: *Random*, *Husky*, *Conveyor*, *Shelf* in *MuJoCo*.

In addition to the simulation environment introduced in [11], we built these scenes in *MuJoCo* with the robot models of *MuJoCo Menagerie* [13], as shown in Figure 3.1 and Figure 3.2 respectively. The husky base is still in the *Husky* scene, and the husky base model is built according to the work of [27] (Figure 3.2).

In these four scenes, we first import the original tasks and robots' states. During the tasks execution, we randomly select a moment when robots are grasping objects, then we make robots drop the objects. At this moment, the objects will be thrown away and finally stay on the ground, and meanwhile the robots stay statically. These final states of robots and objects are stored and will be used as a new initial configurations for the motion planner (Figure1.1 *Motion Re-Planning*). All of these motions are simulated and visualized in *MuJoCo*.

In order not to throw objects too far, we set an invisible wall around the objects with a specific range for each scene, within which robots can pick the dropped objects. At the dropping-object moment, it could be common that several robots are picking several objects at the same time. So, one of these objects could be thrown away while others are still grasped by their robots.

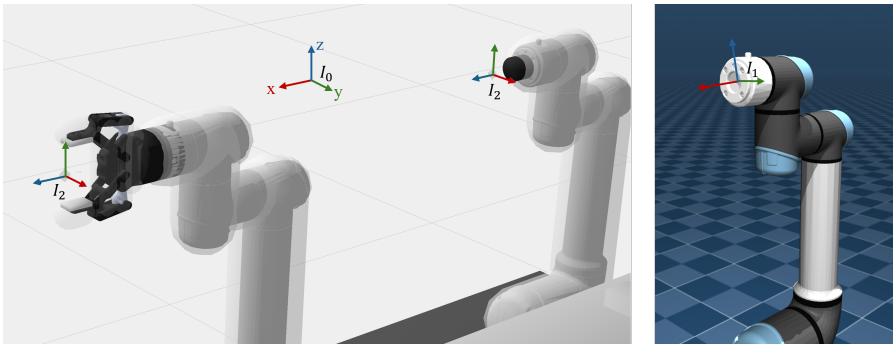


Figure 3.3: The representation of the coordinate systems. There are two types of robots grasper: *gripper* and *vacuum*

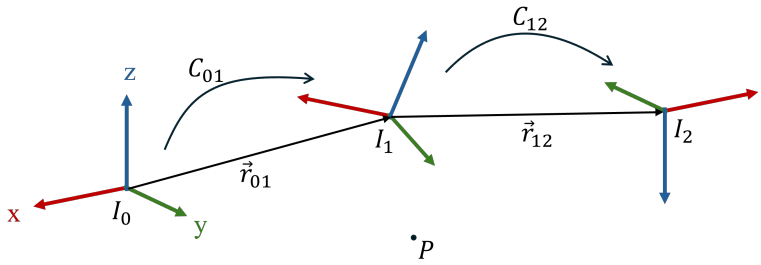


Figure 3.4: The transformation among the coordinate frames. We represent the position and orientation of the object at point P in these coordinate systems.

3.1.2 Relative Coordination

Considering the case in which robots still grasping objects at the dropping-moment and in order to make the motion planning easier, we calculate the relative coordinate of object to its attaching robot if it is grasped by a robot. Here we also consider the frame transformation between the *MuJoCo* simulation environment and the motion planner simulator.

In Figure 3.3, I_0 is the world coordinate system, I_1 is the end-most *geomesh* coordinate system of the robot arm, and I_2 is the end-effector coordinate system, the target coordinate system. We aim to get the coordinate of object in the end-effector frame, which belongs to the attaching robot arm of the object.

Let's symbolize the above coordinate systems transformation in Figure 3.4. So the coordinate transformation is to represent the coordinate of point P in system I_2 from I_0. We have:

$$\begin{aligned} {}_0\vec{r}_P &= C_{01} \cdot_1 \vec{r}_P +_0 \vec{r}_{01} \\ {}_1\vec{r}_P &= C_{12} \cdot_2 \vec{r}_P +_1 \vec{r}_{12} \\ \Rightarrow {}_0\vec{r}_P &= C_{01} \cdot [C_{12} \cdot_2 \vec{r}_P +_1 \vec{r}_{12}] +_0 \vec{r}_{01} \\ &= C_{01} \cdot [R_x R_y \cdot_2 \vec{r}_P +_1 \vec{r}_{12}] +_0 \vec{r}_{01} \\ \Rightarrow {}_2\vec{r}_P &= (R_x R_y)^{-1}[C_{01}^{-1}({}_0\vec{r}_P -_0 \vec{r}_{01}) -_1 \vec{r}_{12}] \end{aligned} \tag{3.1}$$

where ${}_j\vec{r}_P$ is the position of point P in frame I_j; ${}_i\vec{r}_{ij}$ is the translation vector between two frames I_i and I_j represented in the frame I_i; C_{ij} is the rotation matrix between two frames I_i and I_j.

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}, \quad R_y = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

There is still an slight adjustment of the frame I_1 along the x-axis:

$$\hat{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(-0.188) & -sin(-0.188) \\ 0 & sin(-0.188) & cos(-0.188) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.98238 & 0.18689 \\ 0 & -0.18689 & 0.98238 \end{bmatrix}$$

That is, $C_{12} = \hat{R}_x R_x R_y$, and the final position of point P in frame I_2 shoud be:

$${}_2\vec{r}_P = (\hat{R}_x R_x R_y)^{-1}[C_{01}^{-1}({}_0\vec{r}_P -_0 \vec{r}_{01}) -_1 \vec{r}_{12}]$$

For the two types of robot graspers, we have two different offsets Δr, i.e. ${}_1\vec{r}_{12} = [\Delta r, 0, 0]$:

$$\Delta r_{vacuum} = 0.104, \quad \Delta r_{gripper} = 0.169$$

${}_0\vec{r}_P$, ${}_0\vec{r}_{01}$ and C_{01} can be gotten from the data states in *MuJoCo*.

The final orientation of object at point P in frame I_2 is:

$${}_2C_P = (C_{01}C_{12})^{-1} \cdot_0 C_P = (C_{01}\hat{R}_x R_x R_y)^{-1} \cdot_0 C_P$$

where ${}_iC_P$ is the orientation of object at point P in frame I_i.

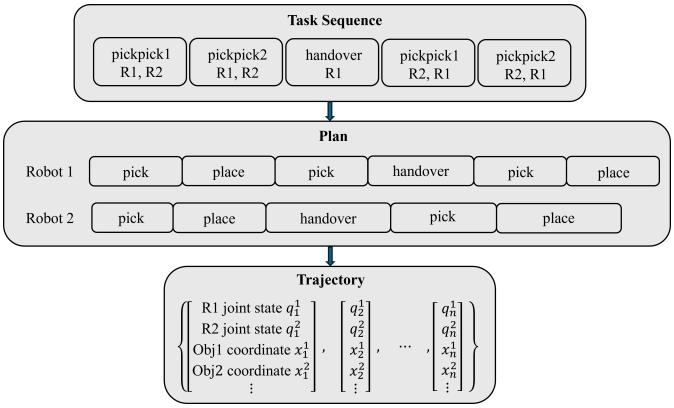


Figure 3.5: The procedure of motion planning.

3.1.3 Original Planning

With the initial configuration $\mathbf{g}^0$ of the robot-object system, the motion planner can generate several motions for the entire system to execute. Those motion plans are called *"Original Motions"* $\mathbf{p}^0 = [p_i^0]_{i=0}^n$. For one specific motion p_i^0, it contains a task sequence π, a plan, and a discrete trajectory. The task sequence π includes primitive tasks assigned to each participating robot. The plan outlines the actions taken by each individual robot and the corresponding time at which they occur. The discrete trajectory consists of sets of robot joint states $\mathbf{q}$, object position coordinates $\mathbf{x}$, and some other states such as robot end-effector positions (Figure 3.5). The planner first generates a task sequence π based on the initial and goal states of the robots and objects. With this task sequence π, the plans for each robot are produced. Finally, it will get the trajectories for robots and objects(Figure 3.5).

3.1.4 New Planning

During the execution of pick-and-place operations, robots may encounter some errors, such as dropping objects during the grasping or movement process. In these cases, it is necessary to re-plan new motions for the entire robot-object system.

After objects dropping and being static, put the current states of robots and objects as the new initial configuration $\mathbf{g}^1$ and the target configuration of robot-object system into the motion planner and we can get the new motions $\mathbf{p}^1$.

3.1.5 Motion Re-Planning

For one specific motion trajectory, it consists of a series of discrete motion steps, each of which contains the states of the robot-object system in the scene. We first select the motion steps that at least one robot is grasping an object. Within these grasping steps, we randomly select n steps $\{i_i, i_2, \cdots, i_n\}$ as dropping steps.

Algorithm 1 Algorithm for Objects Dropping

```
for j in range(n) do
  while execute the original motion do
    if current step == i_j then
      Randomly choose an object which is grasped by a robot.
      if number of grasped objects > 1 then
        Set other grasped objects attached to its grasping robots.
      end if
      Break
    end if
  end while
  Simulate the object dropping process and output the final static states of objects
  and robots.
end for
```

According to Algorithm1, with the final static states of objects and robots, we can form the initial configurations for motion re-planning. Based on these initial configurations of each dropping step, we plan new motion trajectories to finish the tasks of the system. Combing these new motions and the original motions, we generate a dataset file. And we use such data files to train a makespan predicting neural network.

For one motion plan data, it mainly contains seven elements: obj_file, $robot\_file$, $metadata$, $plan$, $scene$, $sequence$, $trajectory$, as shown in Table 3.1.

3.2 Makespan Prediction

In this section, we utilize the data obtained by the above motion replanning to train a neural network. We train the network for the different scenes separately and train one unified network based on the mixed dataset from these different scenes.

Element	*Ingredient*
obj_file	Configurations of all the objects in the scene
$robot\_file$	Configurations of all the robots in the scene
metadata	Metadata of the motion
plan	Motion plans to achieve the final goal
scene	Scene data of current motion
sequence	Task sequence to achieve the final goal
trajectory	Motion trajectory of all the robots and objects in the scene

Table 3.1: Data Format. The details of the motion data.

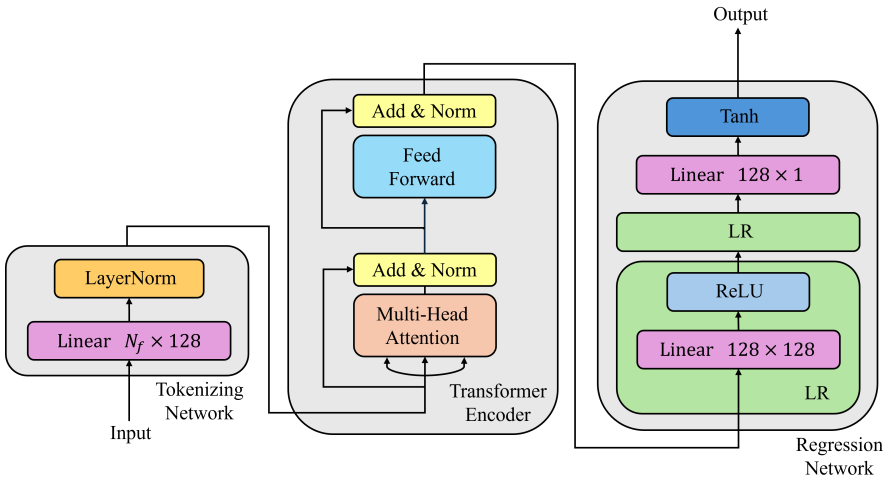


Figure 3.6: The architecture of the neural network.

3.2.1 Neural Network Architecture

The architecture of the neural network is shown in Figure 3.6. The input of the network are the features of robots, objects, and scenes, and the number N_f is the dimension of the feature sequence. The output of the network is a number within the range of (-1,1).

When we enter the input into the network, they first are converted into tokens after going through the tokenizing network, the Transformer Encoder maps an input sequence of symbol representations to a sequence of continuous representations [28], and then this sequence is regressed to be one single number by the regression network.

The tokenizing network is a fully connected network with a $(N_f \times 128)$ linear layer, followed by layer normalization, where N_f depends on the dimension of the input. Transformer Encoder layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, positionwise fully connected feed-forward network. It employs a residual connection around each of the two sub-layers, followed by layer normalization. That is, the output of each sub-layer is $LayerNorm(x+Sublayer(x))$, where $Sublayer(x)$ is the function implemented by the sub-layer itself [28]. The final regression network is composed of three sub-layers. Each of them is fully connected with a linear layer and an activation function. The first two sub-layers use $ReLU$ as the activation function and the activation function of the last sub-layer is *Tanh*.

3.2.2 Input and Output of the Neural Network

To construct the input of the network, we define a sequence s that encodes the features of the robot-object system: the identifications of a task τ and two robots R_1, R_2, the base positions $\mathbf{b}_1, \mathbf{b}_2$ and joint angles θ_1, θ_2 of two robots, the identification o, the size a_o, the initial and goal poses $\mathbf{p}_o^{initial}, \mathbf{p}_o^{goal}$ of an object. These features represent the configurations of the scenes.

$$s = [\tau, R_1, R_2, o, \mathbf{b}_1, \mathbf{b}_2, \theta_1, \theta_2, \mathbf{p}_o^{initial}, \mathbf{p}_o^{goal}, a_o] \tag{3.2}$$

where $\tau \in \mathcal{T}$ (τ is an element of the set of tasks $\mathcal{T}$), two involved robots are denoted as R_1, R_2, for each robot $R_i, i \in \{1, 2\}$ the base position is $\mathbf{b}_i \in \mathbb{R}^3$, the joint angles is $\theta_i \in \mathbb{R}^6$, the object is represented by $o \in \mathcal{O}$ ($\mathcal{O}$ is the set of possible objects), the size of the object o is $a_o \in \mathbb{R}^3$ with length, width and length of the object, the poses are $\mathbf{p}_o^{initial}, \mathbf{p}_o^{goal} \in SE(3)$.

A single task in a task sequence π involves at most two robots to pick and place one object. For the completion of a single task, at most two robots and one object are involved; specifically, the task is characterized by its identifier, the two participating robots, and the associated object. For these two robots, we should know the base positions of them and their initial joint states. For the involved object, initial and goal positions and orientations are needed. The size of the object should also be considered. Concatenating all these features for each task in one task sequence π, we can generate a matrix describing the feature of the entire task sequence π. This matrix is the input of the neural network, that is, $S = [s_1^\top, s_2^\top, ..., s_{|\mathcal{T}|}^\top, \mathbf{0}_{N_f, 8-|\mathcal{T}|}]^\top$, N_f is the length of the vector 3.2.

For a specific scene, the input of the network is S. However, in order to classify different scenes when training a unified network, the input is adjusted as:

$$\begin{aligned} S^* &= [s_1^{*\top}, s_2^{*\top}, ..., s_{|\mathcal{T}|}^{*\top}, \mathbf{0}_{N_f^*, 8-|\mathcal{T}|}]^\top \\ s^* &= [\tau, R_1, R_2, o, \gamma, \mathbf{b}_1, \mathbf{b}_2, \theta_1, \theta_2, \mathbf{p}_o^{initial}, \mathbf{p}_o^{goal}, a_o] \end{aligned} \tag{3.3}$$

where the scene is denoted as $\gamma \in \Gamma$, Γ is the set of all four scenes, and N_f^* is the length of s^*.

The output of the network is a number $\hat{y}$ within the range $(-1, 1)$.

3.2.3 Loss Evaluation

The loss function of the network is Mean Square Error (MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(X_i - \hat{X}_i)^2 \tag{3.4}$$

where X_i represents the true data and $\hat{X}_i$ represents the output of the network.

For the MSE loss curve, X_i should be the normalized makespan of a plan calculated by the following function:

$$Y \to y: \quad y = \frac{Y - Y_{min}}{Y_{max} - Y_{min}}(y_{max} - y_{min}) + y_{min} \tag{3.5}$$

where Y is the original makespan; y is the normalized makespan; Y_{max} represents the maximum makespan in the training dataset plus 400, $Y_{min} = 0$, $y_{max} = 1$, $y_{min} = -1$.

By scaling the makespan Y into the range of $(-1, 1)$, we can compare it with the output of the network, which is also within the range of $(-1, 1)$. So, the MSE loss is calculated by $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ with y_i being the normalized makespan of plans in the dataset and $\hat{y}_i$ being the output of the network.

Correspondingly, we can also scale the output of the network into the magnitude of the normal makespan values:

$$\hat{y} \to \hat{Y}: \quad \hat{Y} = \frac{\hat{y} - y_{min}}{y_{max} - y_{min}}(Y_{max} - Y_{min}) + Y_{min} \tag{3.6}$$

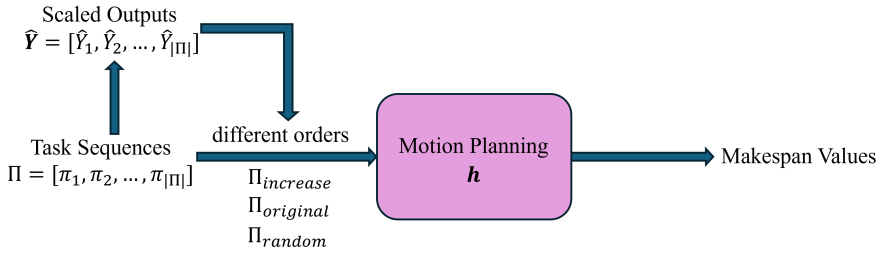


Figure 3.7: Test evaluation with different orders.

where $\hat{y}$ is the output of the network, $\hat{Y}$ is the denormalized predicted makespan of the network, and other variables are the same as the ones shown in equation 3.5.

In the training loop, we deal with dataset by batches, so in practice, n in equation 3.4 is the size of a batch. Based on this, we set other two loss evaluations called average scaled loss and average normalized loss:

$$e_{scaled} = \frac{1}{n}\sum_{i=1}^{n}|Y_i - \hat{Y}_i| \tag{3.7}$$

$$e_{normalized} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3.8}$$

where n is the batch size; as for the ith plan in the batch, $y_i, \hat{y}_i \in (-1, 1)$ are the normalized makespan and the network predicted output, Y_i, $\hat{Y}_i$ are the real makespan and the scaled network output.

3.2.4 Test Evaluation

Besides the loss, we next use the makespan values given by the motion planner to evaluate the predicted output of the network.

As shown in Figure3.7, firstly, we randomly sample a set of configuration data from the testing data. Based on this set of data, motion planner can generate sequences of task $\Pi = [\pi_1, \pi_2, ..., \pi_{|\Pi|}]$, helping robots to achieve the goals. For each sequence π_j, we can obtain the input to the network S_i (or S_i^*) and the output $\hat{y}_i$ (or $\hat{y}_i^*$) correspondingly. Secondly, we convert the output of the network $\hat{y}_i$ (or $\hat{y}_i^*$) into the magnitude of real makespan values $\hat{Y}_i$ (or $\hat{Y}_i^*$) by the function 3.6. Thirdly, as we have the network outputs $\hat{\mathbf{Y}} = [\hat{Y}_1, \hat{Y}_2, ..., \hat{Y}_{|\Pi|}]$, we order its elements in an ascending order $\hat{\mathbf{Y}}_{increase}$ and in a random order $\hat{\mathbf{Y}}_{random}$. Their corresponding sequences are therefore ordered as $\Pi_{original}$, $\Pi_{increase}$ and Π_{random}. Finally, we plan for these sequences with the motion planner and we will get the makespan values of the motion plans. Considering the random factor in motion planning, we perform motion planning experiments for 10 times and we get 100 motion plans at most for the task sequences Π each time. Based on the makespan values of these plans, we can plot the *"Makespan-Time"* figures for the sequences in the order of $\Pi_{original}$, $\Pi_{increase}$ and Π_{random} in each experiment (See Chapter 4).

3.2.5 Dataset

We use the data of four scenes. For *Conveyor* scene, we have (1324+1) sets of data; for *Husky* scene, we have (588+1) sets of data; for *Random* scene, we have (1109+1) sets of data; for *Shelf* scene, we have (730+1) sets of data, where 1 represents the original trajectory. For one set of motion plan data, it mainly contains seven elements: obj_file, $robot\_file$, $metadata$, $plan$, $scene$, $sequence$, $trajectory$, as shown in Table 3.1. A set of data is defined by one initial configuration and always contains multiple values for each element. The *Conveyor* scene is composed of 4 robots and 2 objects on the ground, the *Husky* contains 2 robot arms on a husky base and 2 objects on the table, the *Random* scene has 4 robots and 3 objects on the ground, and the *Shelf* scene consists of 2 robots on the ground and 3 objects on the shelf.

As we drop the objects at some moment of the trajectory and replan based on the new initial configurations, the makespan is calculated from a new initial configuration to a goal configuration. We get the distributions of makespans for four scenes in Figure 3.8. With all re-plans data mixed, we have the distribution of makespan in Figure 3.9.

With the previous motions added to the replans, we get the full plans which represent the motions from the original initial configurations to the goal configurations. According to this, the distributions of full plans for four scenes are shown in Figure 3.10. From Figure 3.8 and Figure 3.10, we can find the makespan values of plans for *Shelf* scene are obviously larger than other three scenes, which is due to its complex scene setting.

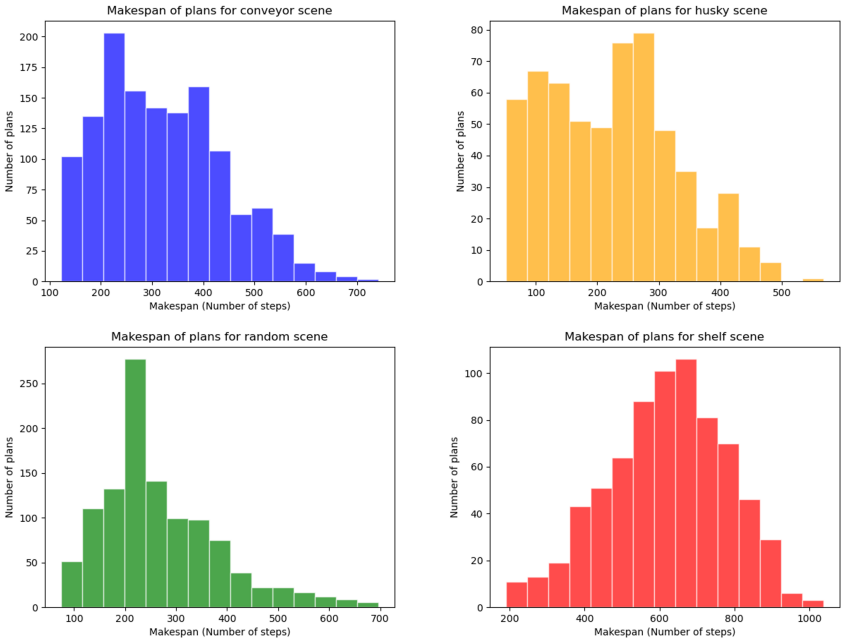


Figure 3.8: Makespan distributions of re-plans for four different scenes.

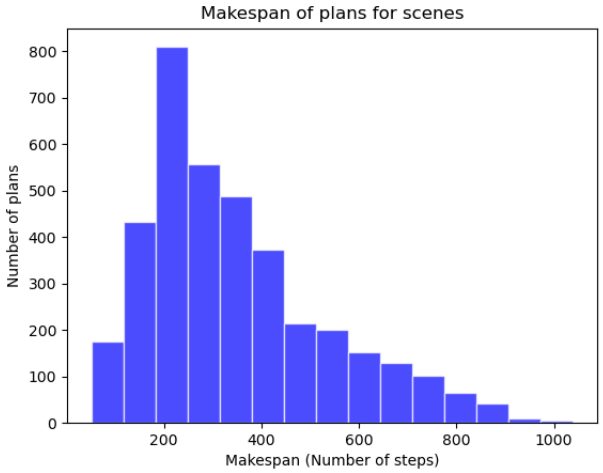


Figure 3.9: Makespan distribution of re-plans for all scenes together.

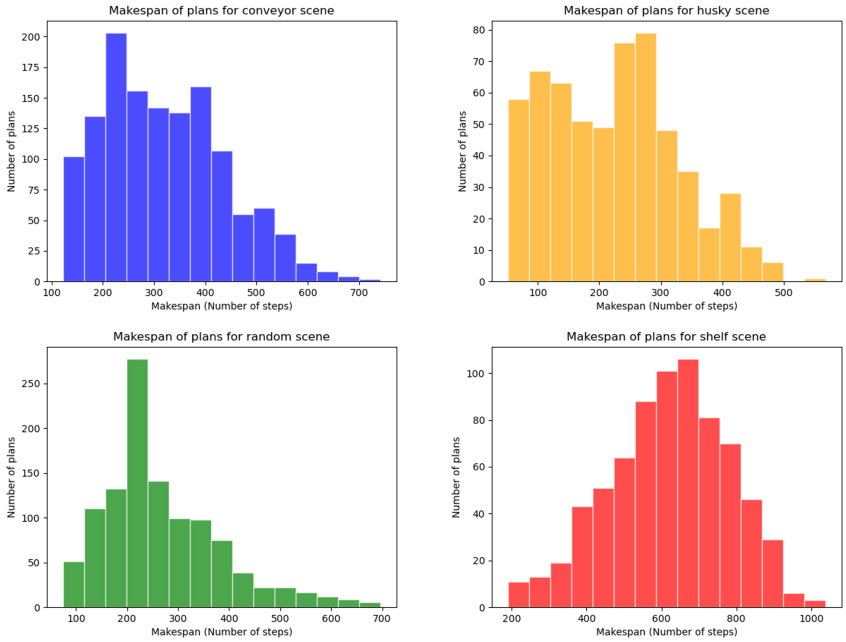


Figure 3.10: Makespan distributions of full plans for four different scenes.

For each scene, we partition the dataset by allocating 80% for training and the remaining 20% for testing. A scene-specific network is trained using only the training data from its corresponding scene and evaluated on its respective testing set. To train a unified network, we aggregate the training data from all four scenes, and evaluate the model using the combined testing datasets from the same scenes.

Results

Chapter 4

4.1 Motion Re-Planning

In order to illustrate the motion re-planning, we have made records of several samples in the four scenes. We randomly select one plan for each scene in the corresponding dataset. The results are shown in the videos [29], with the original plan trajectory and the replan trajectory for each scene.

Take *Random* scene for example, a demonstration of motion replanning is shown in Figure 4.1. As Figure 4.1 shows, there are four robots and three objects in this configuration. For the original plan trajectory, four robots should take three objects from the initial positions at time t_0 to the goal positions at time t_3 shown in Figure 4.1. However, the error happens at time t_1, as *Object 2* is dropped on the ground. Therefore, the multi-robot system has to plan a new trajectory to move all the objects to their goal positions.

At t_0 in Figure 4.1, all the objects are in their initial positions. Then, the robots will grasp the objects and move them to the goal locations following the original motion plan. At time t_1, *Robot 3* does not grasp *Object 2* steadily and makes the object fall onto the ground, while *Robot 0* and *Robot 2* are still holding *Object 3* and *Object 1*, respectively. At this moment, this is the new initial configuration for the motion planner and it will generate several new motions for robots. The graph of the time step t_2 demonstrates that with the new plan, *Robot 3* is grasping *Object 2* again and *Robot 0* and *Robot 2* are moving *Object 3* and *Object 1* to their goal positions. At the end step t_3, all the objects are put at their goal locations and all the robots are back to their default pose.

4.2 Makespan Prediction

We first demonstrate the results of the makespan prediction for each scene and then the unified network performance.

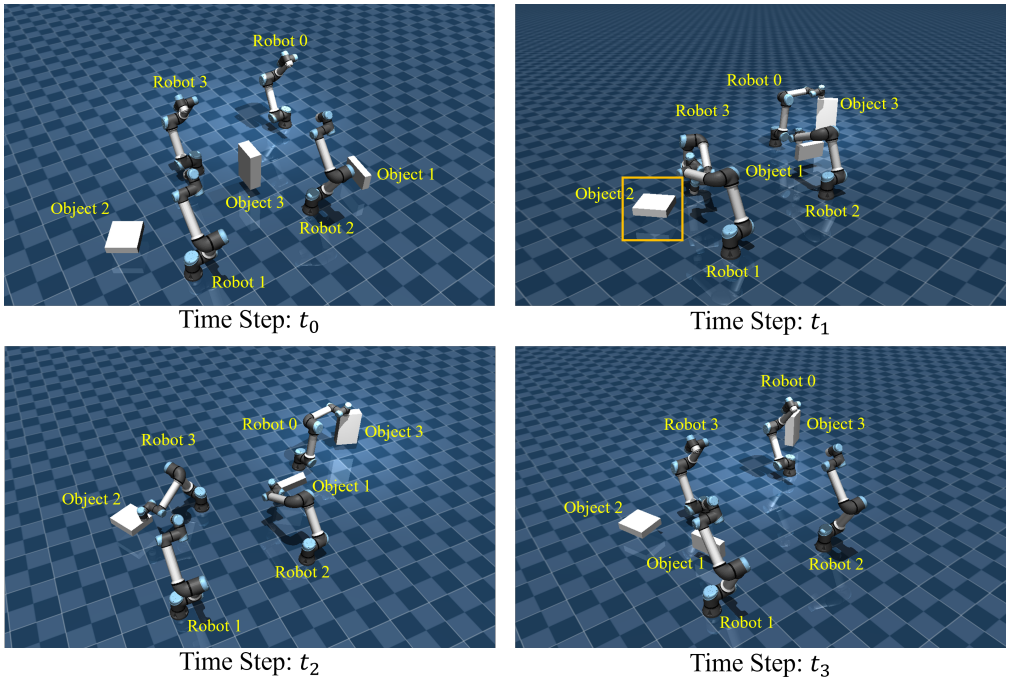


Figure 4.1: A demonstration of motion replanning for *Random* scene at different time steps.

Table 4.1: Parameters of neural network

Parameters	Values
Learning rate	10^{-3}
Number of epoches	10
Number of evaluation per epoch	5
Batch size	5
Size of dataset	M
Size of training dataset	$0.8M$
Size of testing dataset	$(M - 0.8M)$
Dropout probability	0.1
Optimizer	RAdam
Learning Rate Scheduler	Linear

Due to the different initial configurations, the number of new motion plans varies, but with the maximum as 100.

For the loss curves in the figures of each scene, "Loss/train" plot is the result of the MSE loss (equation 3.4) with X_i as the normalized makespan, $\hat{X}_i$ as the network output. As for the "Loss/avg_unorm_train" and "Loss/avg_norm_train" figures, they represent the results of equations 3.7 and 3.8 with the training data, respectively. Correspondingly, "Loss/avg_unorm_test" and "Loss/avg_norm_test" figures represent the results of equations 3.7 and 3.8 with the testing data. The *x-axis* "step" of these figures is defined by the cumulative number of batches. For each batch, we compute the mean square error, average scaled loss, and average normalized loss using the data for which our trained network provides predictions and the motion planner generates corresponding makespan values.

As we mentioned in section 3.2.4, we execute motion planning for the task sequences in different order: $\Pi_{increase}$, $\Pi_{original}$, and Π_{random}, related to the plots named with *"seq_predicted_order"*, *"seq_original_order"*, and *"seq_random_order"*. In the makespan curves presented for each scene, the *y-axis* is the value of makespan, while the *x-axis* is the cumulative computing time. In each graph, ten different colored curves correspond to ten experiments and we get 100 plans at most for the sequences Π in each experiment. We use the motion planner to produce makespan of plans for the given task sequences Π and we record them in the figure with name beginning with *"Makespan"*. As for the figures with name of *"MinMakespan"*, they record the minimal values of makespans we ever have had until the current computing step. So, they will have several flat lines and some slopes. The flat lines mean that the minimal makespan we have found is still α and we have not found another smaller makespan value, and the slope means that the minimal makespan we have found is changing from α to β. Taking *"(e).Makespantest_seq_random_order_Conveyor"* in Figure 4.3 for example, it shows the makespan of plans for test sequences in random order Π_{random} for the *Conveyor* scene with the number of robots being 4 and the number of objects being 2, according to the analysis in section 3.2.5.

Theoretically, all ten curves in a plot should be identical, as they are based on the same sequence Π. However, difference between them reveals the inherent variability of the motion planner, resulting in different motion plans and makespans for the same task sequence. Theoretically, the plan makespans related to ascending order sequences $\Pi_{increase}$ should be a monotonic increase with respect to the cumulative computing time, if the predicted results are accurate. However, since the variability in motion plans exists, the plot of ascending order sequences should exhibit an overall positive trend in case of accurate network output. In addition, the minimum plan makespan related to the ascending order sequences $\Pi_{increase}$ is supposed to be a flat line with respect to the cumulative computing time in each experiment. But in practice, the minimum makespan plots demonstrate flat lines with slopes.

Figure 4.2: Loss curves of *Conveyor* scene with 15 dropping steps. The light green curves are plotted during the training process. The dark green ones are the smoothed curves by Exponential Smoothing.

4.2.1 Conveyor Scene

For *Conveyor* scene, we randomly select $n = 15$ dropping steps and get 1324 sets of new motion plans totally. With the original motion plan added, $M = 1325$, we use 1060 sets of plans to train a network and 265 sets as the testing data. The performance of the network is shown in Figure 4.2. The testing result is shown in Figure 4.3.

The loss curves in Figure 4.2 all go down. The MSE training loss decreases to less than 0.05, ultimately converging near zero in (a), indicating effective training convergence. The average scaled training loss converges to less than 40 in (b), and the average normalized loss converges to 0.05 in(c). For the testing loss, the average scaled loss is less than 50 after 10k steps in (d), and the average normalized loss is less than 0.1 after 8k steps in(e). All losses exhibit convergence; but the training MSE loss, the testing average scaled loss, the testing average normalized loss converge more rapidly than the training average scaled loss and normalized loss.

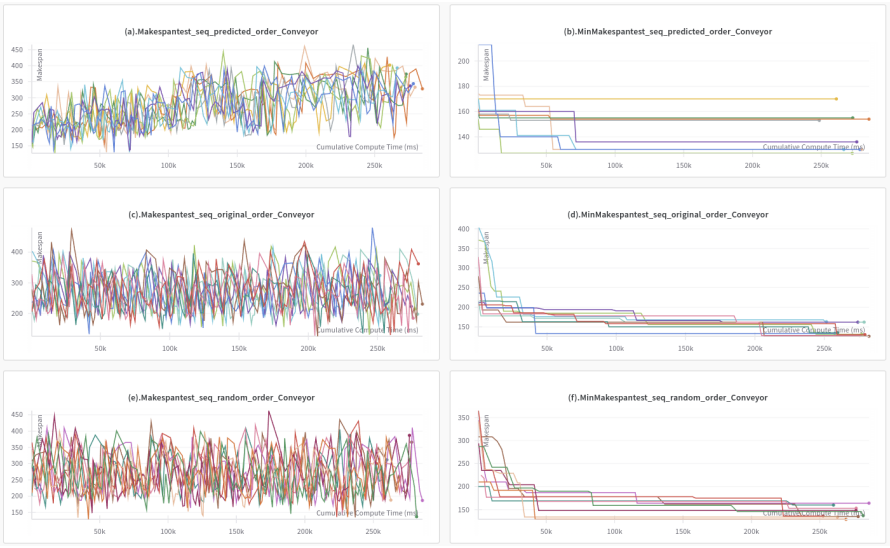


Figure 4.3: Makespan curves of *Conveyor* scene. The *x-axis* is the cumulative computing time in *ms* and the *y-axis* is the makespan.

Considering the makespan curves of *Conveyor* scene in Figure 4.3(a), the *"Makespan-Time"* curves of sequences in the predicted order, i.e. $\Pi_{increase}$, exhibit a generally monotonic increasing trend, reflecting the inherent increasing property of makespans in $\Pi_{increase}$. While the individual curves display local fluctuations with both increases and decreases, the collective trend demonstrates a general upward progression. For the *"Minimum Makespan-Time"* curves of of *Conveyor* scene in Figure 4.3(b), most become flat within the first 25% of the cumulative computing time, with several of them flat at the start. Ideally, if the network predicted output is accurate, Figure 4.3(b) should be horizontal lines according to the ascending order sequences $\Pi_{increase}$. However, with the variability of motion planning existing, the network predicts well with a monotonic increasing trend in Figure 4.3(a) and the curves rapidly flattening in Figure 4.3(b).

The *"Makespan-Time"* curves for both the original sequences $\Pi_{original}$ and the random sequences Π_{random} display relatively flat trends, while the time points at which their corresponding *"Minimum Makespan-Time"* curves plateau are uniformly distributed throughout the cumulative computation timeline. The fluctuations of the *"Makespan-Time"* curves, along with the uniformly distributed flattening time points of the *"Minimum Makespan-Time"* curves, reflect the inherent variability of the motion planning process.

The cumulative computation time across experiments is similar, indicating that the motion planner generates plans with consistent efficiency for the sample in the *Conveyor* scene.

Consequently, the network exhibits satisfactory performance on the *Conveyor* scene, as indicated by the testing results.

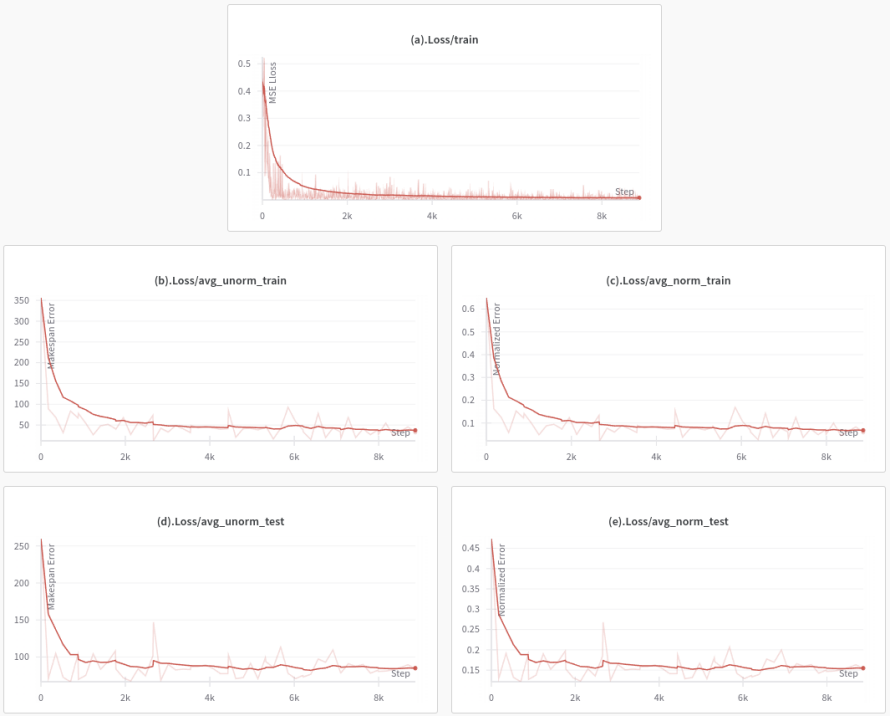


Figure 4.4: Loss curves of *Random* scene with 25 dropping steps. The light red curves are plotted during the training process. The dark red ones are the smoothed curves by Exponential Smoothing.

4.2.2 Random Scene

The *Random* scene demonstrates 4 robots picking and placing 3 objects. Therefore, we randomly select $n = 25$ dropping steps and get 1109 sets of new motion plans plus one original plan. The performance of the network is shown in Figure 4.4. The testing results are shown in Figure 4.5.

Like the MSE loss curves of *Conveyor* scene, the MSE loss of *Random* scene in Figure 4.4(a) also converges to zero as the step grows. All four other loss curves in Figure 4.4 go down. The smoothed average scaled loss curves in Figure 4.4(b) and (d) go down to less than 50 with respect to the step. The smoothed average normalized loss curves in Figure 4.4(c) and (e) steadily decrease and approach a number with training loss lower than 0.1 in (c) and testing loss lower than 0.15 in (e). Compared with the results of *Conveyor* scene, the average scaled training loss and the average normalized loss show more clear convergence in *Random* scene.

Figure 4.5: Makespan curves of *Random* scene. The *x-axis* is the cumulative computing time in *ms* and the *y-axis* is the makespan.

The curves in Figure 4.5(a) demonstrate a clear uniform upward progression, indicating similar growth dynamics among the evaluated cases. The lines in Figure 4.5(b) turn flat within the first 50 seconds of the cumulative computing timeline. Similarly to the cases in *Conveyor* scene, the curves in Figure 4.5(c), (d), (e), and (f) show the inherent variability.

4.2.3 Husky Scene

There are 2 robots attached to a husky base and 2 objects on the table in the *Husky* scene. We randomly select $n = 25$ dropping steps and get 588 sets of new motion plans in addition to the original plan. The performance of the network is shown in Figure 4.6. The testing results are shown in Figure 4.7.

Compared to the network performance in the *Conveyor* scene, the loss curves of the *Husky* scene converge faster and more clearly. The MSE loss curve converges to 0 in Figure 4.6(a), the average scaled training loss converges to less than 30 in Figure 4.6(b), and the average normalized training loss in Figure 4.6(c) converges to lower than 0.05. As illustrated in Figure 4.6(d), the average scaled testing loss converges to approximately 40, whereas in Figure 4.6(e), the average normalized testing loss approaches a value of 0.08.

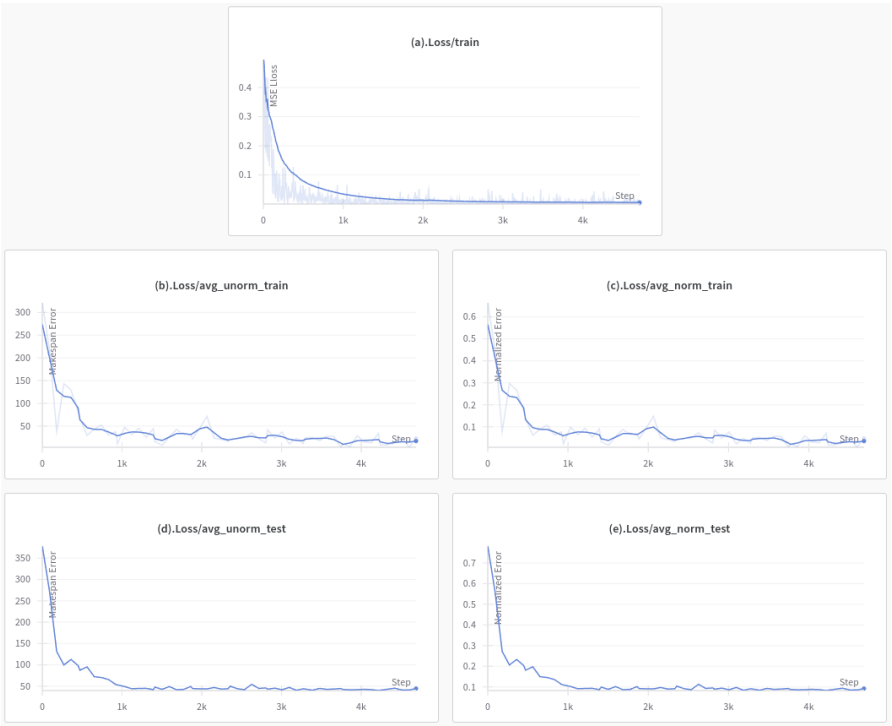


Figure 4.6: Loss curves of *Husky* scene with 25 dropping steps. The light blue curves are plotted during the training process. The dark blue ones are the smoothed curves by Exponential Smoothing.

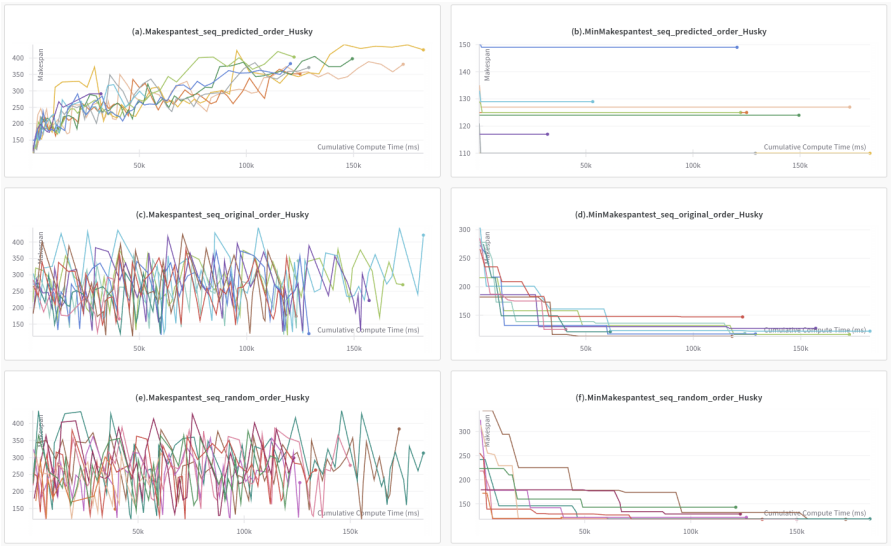


Figure 4.7: Makespan curves of *Husky* scene. The *x-axis* is the cumulative computing time in *ms* and the *y-axis* is the makespan.

Take a look at the testing makespan curves in Figure 4.7. The curves in Figure 4.7(a) show an increase trend and meanwhile, the *"Minimum Makespan-Time"* lines in Figure 4.7(b) turn flat quickly, almost in the first 10 seconds of cumulative computation time. The other two graphs of the sequences in the original and random order in (c) and (e) exhibit no increasing trend but a horizontal trend instead. Their corresponding graphs (d) and (f) describe lines turning horizontal at equally distributed time points. Graphs (c), (d), (e), and (f) shows the randomness of the motion planner and graphs (a) and (b) demonstrate the effectiveness of the neural network.

The *Husky* scene demonstrates a scene with 2 robot arms attached to a husky base and 2 objects on the table in front of the husky. Compared to the *Conveyor* and *Random* scenes, robots have more constraints in this scene. And the number of new plans used to train the network is less than the one in *Conveyor* and *Random* scenes. Due to these constraints, the planning results given by the motion planner are less, leading to sparse *"Makespan-Time"* plots.

4.2.4 Shelf Scene

With 2 robots on the ground and 3 objects on the shelf, we randomly select $n = 25$ dropping steps and get 730 sets of new motion plans and one original plan totally. The performance of the network is shown in Figure 4.8. The testing results are shown in Figure 4.9.

The loss curves in Figure 4.8 perform badly on convergence. The curves in Figure 4.8(a),(d), and (e) can be found convergent. The MSE loss in (a) decreases to 0 but with a lot of noise. The average scaled loss in (d) converges to approximately 60 and the average normalized loss in (e) goes down to around 0.09. Meanwhile, the loss curves in Figure 4.8(b) and (c) do not exhibit a clear trend of convergence. We can find they are going down and may converge to a certain number, but maybe because of the lack of data, they are not very clear within the current timeline. The *Shelf* scene has more complex setting than other scenes since it consists of an additional barrier, the shelf. It will be more difficult for the motion planner to generate feasible plans than in other scenes, which leads to the plans in the *Shelf* scene with more computing time and larger makespan values.

In contrast with the loss curves in the training process, the positive slope of the test curves is very clear in Figure 4.9(a). What's more, the lines in Figure 4.9(b) turn stable rapidly, within the first 20 seconds of the computing time. Similarly, the curves in Figure 4.9(c), (d), (e), and (f) represent the proper performance of the motion planner and the network. These satisfactory performance results may be attributed to the wide range and significant variation in makespan values observed in the *Shelf* scene.

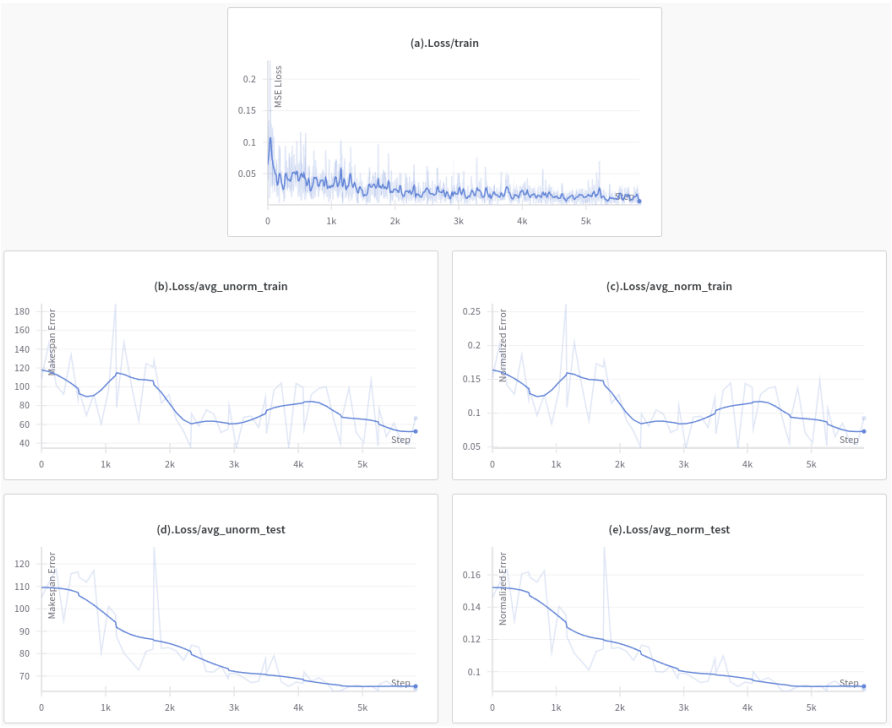


Figure 4.8: Loss curves of *Shelf* scene with 25 dropping steps. The light blue curves are plotted during the training process. The dark blue ones are the smoothed curves by Exponential Smoothing.

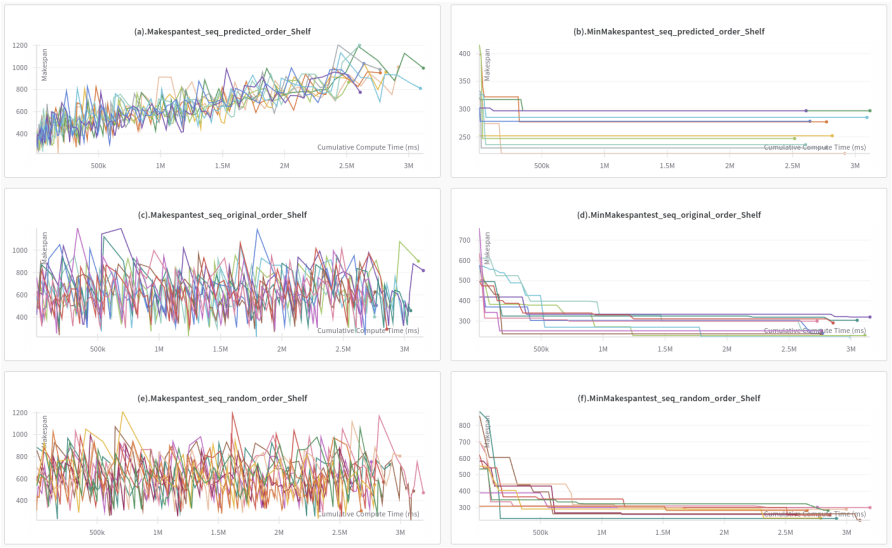


Figure 4.9: Makespan curves of *Shelf* scene. The *x-axis* is the cumulative computing time in *ms* and the *y-axis* is the makespan.

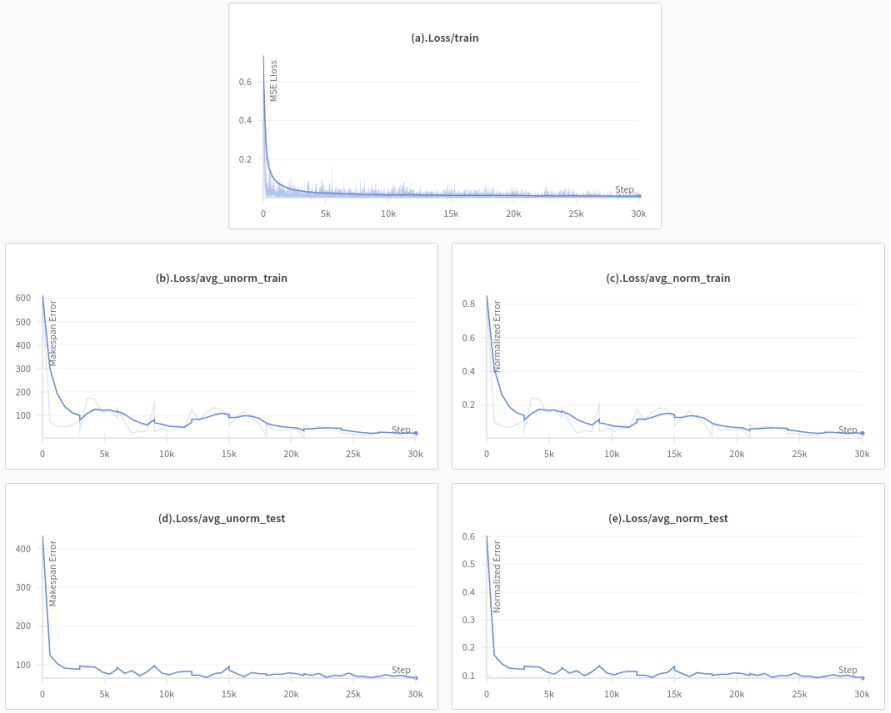


Figure 4.10: Loss curves of the unified network. The light blue curves are plotted during the training process. The dark blue ones are the smoothed curves by Exponential Smoothing.

4.2.5 Unified Scene

Besides the above scene-specific networks, we introduce the unified network with its performance in each scene in Figure 4.10. Based on this unified network, the test results for each scene are shown in Figures 4.11, 4.12, 4.13, 4.14.

In Figure 4.10(a), we find the MSE loss drops rapidly and converges to zero. The average scaled loss curves fall and approach the value of makespan lower than 50 on training (4.10(b)) and testing (4.10(d)). Similarly, the average normalized loss curves in 4.10(c) and 4.10(e) go down to the value less than 0.1.

The *"Makespan-Time"* curves of the unified network for four scenes all exhibit the increase trend on the ascending order sequences $\Pi_{increase}$ and the *"Minimum Makespan-Time"* curves all become flat quickly. From these, the performance of the unified network is satisfactory.

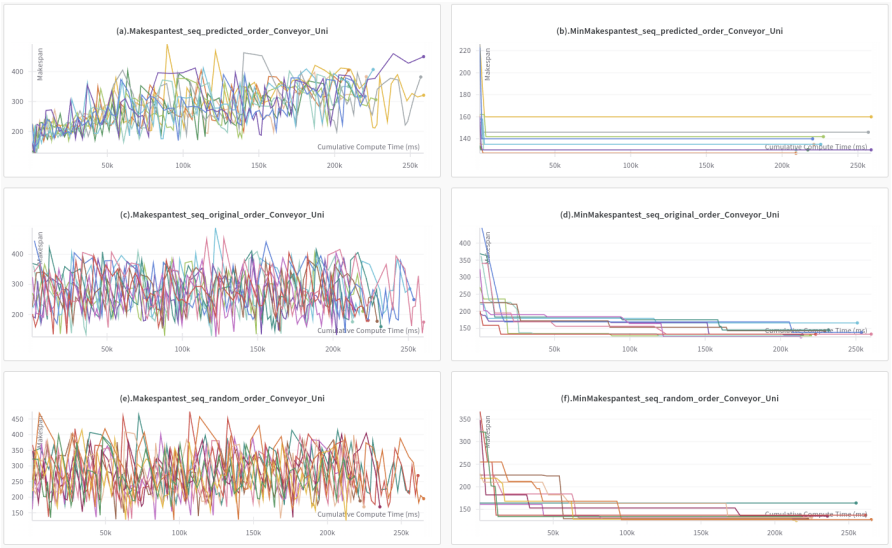


Figure 4.11: Makespan curves of the unified network for *Conveyor* scene. The *x-axis* is the cumulative computing time in *ms* and the *y-axis* is the makespan.

Figure 4.12: Makespan curves of the unified network for *Husky* scene. The *x-axis* is the cumulative computing time in *ms* and the *y-axis* is the makespan.

Figure 4.13: Makespan curves of the unified network for *Random* scene. The *x-axis* is the cumulative computing time in *ms* and the *y-axis* is the makespan.

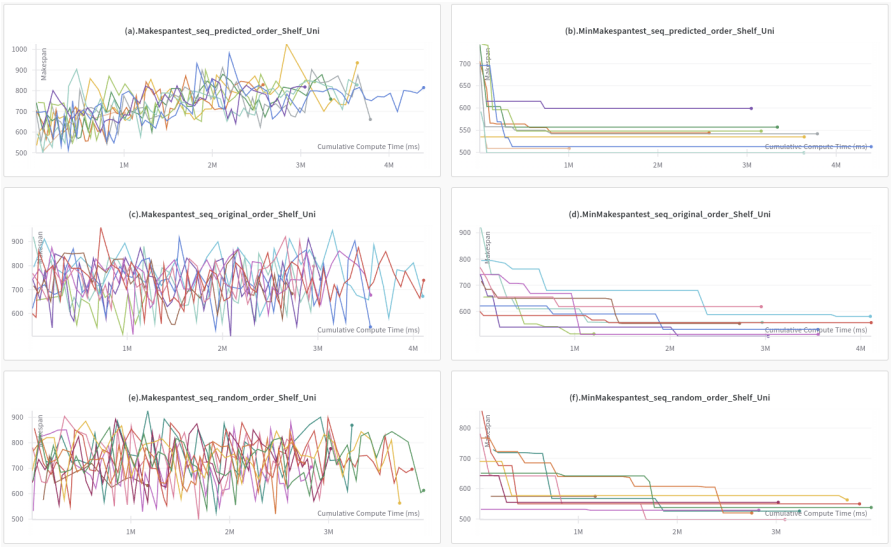


Figure 4.14: Makespan curves of the unified network for *Shelf* scene. The *x-axis* is the cumulative computing time in *ms* and the *y-axis* is the makespan.

Next, we compare the test results of the unified network with the ones of specific-scene networks. For the *Conveyor* scene, the slope of the increase trend in Figure 4.3(a) is larger than the one in Figure 4.11(a). The slope of the increase trend in Figure 4.5(a) is larger than the one in Figure 4.13(a) of the *Random* scene. However, the increase trend slope for the *Husky* scene in Figure 4.7(a) is almost the same as the one in Figure 4.12(a). Finally, the increase trend slope for the *Shelf* scene in Figure 4.9(a) is also larger than the one in Figure 4.14(a). Comparing these curves Figure 4.11(a), 4.12(a), 4.13(a), 4.14(a) with the previous single-scene testing curves Figure 4.3(a), 4.7(a)(a), 4.5(a), 4.9(a), the trend of the testing outputs of the unified network are basically less obvious than the single-scene testing outputs. That's to say, the predicted makespans of the unified network are more closer to each other than the ones of the scene-specific networks. This performance may be attributed to inter-scene data influence, where data from different scenes interact and potentially affect the network's overall performance.

The unified network may also give obvious makespan increase trends. But this can only be proofed with more experiments.

Discussion

Chapter 5

This work simulates the re-planning motions in *MuJoCo* and utilizes the learning-based method for task assignment and motion planning. Based on both the simulation results and the learning results, we found the possibility on using the simulation data to learn a policy that predicts the makespan for a given task sequence in order to speed up the search for a low makespan multi-agent task and motion plan.

For the simulation and visualization in *MuJoCo*, we could add more details to the scenes, like the different robot grasper models and consider more different types of base scenes. We could also add the torque control to the joints of the robot arms. At the present, the motion planner is still based on the simulation environment introduced in [16]. We could transfer the motion planner into *MuJoCo* in the next step, since *MuJoCo* introduces more interactive features for users.

Besides the simulation part, in the learning part of this work, the training loss curves of all the scenes and even the unified scene show rapid convergence, except for the *Shelf* scene. This could be due to the complicated scene settings and the lack of the data for the *Shelf* scene. Therefore, generating more data of the *Shelf* scene and considering more complex motions and more base scenarios could be the next work. For the testing results, we mainly focus on the increase trend of the *"Makespan-Time"* plot. According to the results in Chapter 4, we can clearly find that the neural network performs well for all the scenes as well as the mixed configuration. All the *"Makespan-Time"* curves for the ascending-order sequences $\Pi_{increase}$ exhibit a clear upward progression. All the *"Minimum makespan-Time"* curves turn flat quickly, which means that the sequence π with the minimum makespan is at the front part of the whole set of sequences Π. This performance also confirms the ascending order of $\Pi_{increase}$. We are not using the network to predict a very accurate makespan of a motion plan, but to give a feasible makespan ranges, based on which we use this increase trend to guide the motion planner to choose the most time-saving plans with small makespan values. The next step could be connecting this trained network to the motion planner and guiding the planner to plan short motion trajectories to perform tasks under different configurations. In this case, multi-agent systems would be able to generate motion trajectories with makespan normally less than 600 for *Shelf* scene and less than 300 for the other three scenes, as illustrated in Figure3.10.

However, we only tested the simulation-only validation: The whole system was extensively tested in simulation, but not on physical robot hardware. In the future, we could also use this learning policy with real-world configurations.

Bibliography

[1] S. M. LaValle, "Rapidly-exploring random trees : a new tool for path planning," *The annual research report*, 1998. [Online]. Available: https://api.semanticscholar.org/CorpusID:14744621

[2] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," 2011. [Online]. Available: https://arxiv.org/abs/1105.1186

[3] C. Urmson and R. Simmons, "Approaches for heuristically biasing rrt growth," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, vol. 2, 2003, pp. 1178–1183 vol.2.

[4] D. Ferguson and A. Stentz, "Anytime rrts," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, pp. 5369–5375.

[5] B. Akgun and M. Stilman, "Sampling heuristics for optimal motion planning in high dimensions," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 2640–2645.

[6] M. Otte and N. Correll, "C-forest: Parallel shortest path planning with superlinear speedup," *IEEE Transactions on Robotics*, vol. 29, no. 3, pp. 798–806, 2013.

[7] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 2997–3004.

[8] J. Pan and D. Manocha, "Gpu-based parallel collision detection for fast motion planning," *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 187–200, 2012.

[9] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," *CoRR*, vol. abs/1709.05448, 2017. [Online]. Available: http://arxiv.org/abs/1709.05448

[10] A. H. Qureshi, A. Simeonov, M. J. Bency, and M. C. Yip, "Motion planning networks," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 2118–2124.

[11] V. N. Hartmann, M. A. Z. Mora, and S. Coros, "TAPAS: A dataset for task assignment and planning for multi agent systems," in *RSS 2024 Workshop: Data Generation for Robotics*, 2024. [Online]. Available: https://openreview.net/forum?id=VqQPHz76JH

[12] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[13] K. Zakka, Y. Tassa, and MuJoCo Menagerie Contributors, "MuJoCo Menagerie: A collection of high-quality simulation models for MuJoCo," 2022. [Online]. Available: http://github.com/google-deepmind/mujoco_menagerie

[14] J. Kuffner and S. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, 2000, pp. 995–1001 vol.2.

[15] F. Grothe, V. N. Hartmann, A. Orthey, and M. Toussaint, "St-rrt*: Asymptotically-optimal bidirectional motion planning through space-time," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2022.

[16] V. N. Hartmann and M. Toussaint, "Towards computing low-makespan solutions for multi-arm multi-task planning problems," ICAPS Workshop on Planning and Robotics, 2023.

[17] A. H. Qureshi and M. C. Yip, "Deeply informed neural sampling for robot motion planning," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6582–6588, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:52875738

[18] J. Ichnowski, Y. Avigal, V. Satish, and K. Goldberg, "Deep learning can accelerate grasp-optimized motion planning," *Science Robotics*, vol. 5, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:227067930

[19] H. Zhang, E. Heiden, S. Nikolaidis, J. J. Lim, and G. S. Sukhatme, "Auto-conditioned recurrent mixture density networks for learning generalizable robot skills," in *IEEE International Conference on Robotics and Automation*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:84186967

[20] H. Ha, J. Xu, and S. Song, "Learning a decentralized multi-arm motion planner," 2020. [Online]. Available: https://arxiv.org/abs/2011.02608

[21] S. Luo and L. Schomaker, "Reinforcement learning in robotic motion planning by combined experience-based planning and self-imitation learning," *Robotics and Autonomous Systems*, vol. 170, p. 104545, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889023001847

[22] R. Trauth, A. Hobmeier, and J. Betz, "A reinforcement learning-boosted motion planning framework: Comprehensive generalization performance in autonomous driving," in *2024 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, Jun. 2024, p. 2413–2420. [Online]. Available: http://dx.doi.org/10.1109/IV55156.2024.10588750

[23] Y. Ding, X. Zhang, C. Paxton, and S. Zhang, "Task and motion planning with large language models for object rearrangement," 2023. [Online]. Available: https://arxiv.org/abs/2303.06247

[24] Z. Mandi, S. Jain, and S. Song, "Roco: Dialectic multi-robot collaboration with large language models," 2023. [Online]. Available: https://arxiv.org/abs/2307.04738

[25] S. Wang, M. Han, Z. Jiao, Z. Zhang, Y. N. Wu, S.-C. Zhu, and H. Liu, "Llm3:large language model-based task and motion planning with motion failure reasoning," 2024. [Online]. Available: https://arxiv.org/abs/2403.11552

[26] Z. Wang, S. Cai, G. Chen, A. Liu, X. Ma, and Y. Liang, "Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents," 2024. [Online]. Available: https://arxiv.org/abs/2302.01560

[27] C. Wang, Q. Zhang, Q. Tian, S. Li, X. Wang, D. Lane, Y. Petillot, and S. Wang, "Learning mobile manipulation through deep reinforcement learning," *Sensors*, vol. 20, no. 3, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/3/939

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[29] Wenkai Xuan, "Conveyor, husky, random, shelf scenes | motion re-planning | mujoco," https://youtube.com/playlist?list=PLoTeV4reqxJkiIM6ZK_HAVYujVkRW5In0&si=WxpbnoHvHajMLTZG, 2025.