Online Motion Planning for Robot Manipulators in Dynamic Environments

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik des
Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Xi Huang

Tag der mündlichen Prüfung: 05. Juni 2025
1. Referent: Prof. Dr.-Ing. Tamim Asfour
2. Referent: Prof. Dr.-Ing. Torsten Kröger

Abstract

Online Motion Planning for Robot Manipulators in Dynamic Environments

This thesis focuses on the development of motion planning algorithms for dynamic environments. A dynamic environment is defined as one that evolves over time, necessitating that planning algorithms be efficient in terms of planning and replanning and capable of adapting to the robot's current state and to potential changes in the environment. To address these challenges, this thesis proposes three contributions, each targeting a different aspect of motion planning in dynamic settings.

In order to achieve efficient planning and replanning with short planning time and small variations, the first research question is formulated to address the bottleneck of speeding up motion planning methods. Since collision checking is a major bottleneck for rapid motion planning, a motion planning method is proposed to identify and eliminate unnecessary collision checks during planning. This method uses a precomputed deterministic roadmap to capture the collision-free space of the static environment. It then performs a heuristic-informed search on the roadmap and introduces a novel safe zones concept for edge examination to find a feasible solution. When exploring the precomputed roadmap for a solution, the heuristic-informed search minimizes the number of edge examinations by prioritizing edges based on heuristics derived from path costs in the roadmap. The concept of safe zones utilizes the spatial relation between the robot and the environment to identify regions that do not require collision checks. As a result, the method achieves both a reduced number of edge examinations and a decrease in collision checks for each examination.

Despite the significant speedup by reducing the number of collision checks, the planning time naturally increases with problem complexity and remains unbounded. The second contribution is a method to generate subgoals for decomposing complex motion planning problems into small, easily solvable subproblems. Iteratively planning the subproblems ensures short planning time in dynamic environments. This method begins with a pipeline to collect a dataset of suitable subgoals, which can be planned within a specific bounded time. Using this dataset, a conditional generative model captures the distribution of subgoals based on the given planning problems. During inference, a time estimator acts as a critic to evaluate whether the generated subgoals can meet the desired time constraints and lead the robot toward the final goal. Based on this evaluation, the proposed method makes informed decisions to select a suitable subgoal for planning.

The contributions above solely consider the current geometric state of dynamic environments and overlook the temporal aspect. This short-sightedness over the time horizon can result in frequent replanning and local-minima issues. The third contribution addresses this issue by using episodic reinforcement learning to implicitly account for potential

changes in the environment over time. The learned policy refines reference trajectories based on the dynamic environment. Three trajectory refinement strategies based on B-spline movement primitives are introduced to modify the reference trajectories while ensuring smooth trajectory transitions. The planning results of the contributions above can be used as reference trajectories. This spatiotemporally aware method achieves superior task performance compared to methods that solely depend on geometric information and other spatiotemporal planners.

Comprehensive evaluations and ablation studies in simulation and real robot experiments are conducted to demonstrate the effectiveness and limitations of the proposed methods.

Zusammenfassung

Online-Bewegungsplanung für Robotermanipulatoren in Dynamischen Umgebungen

Diese Dissertation beschäftigt sich mit der Entwicklung von Bewegungsplanungsalgorithmen für Robotermanipulatoren in dynamischen Umgebungen. Dynamische Umgebungen verändern sich im Laufe der Zeit, weshalb Planungsalgorithmen sowohl effizient als auch anpassungsfähig sein müssen. Insbesondere ist es erforderlich, dass sie in der Lage sind, schnell auf Änderungen zu reagieren und sich an den aktuellen Zustand des Roboters sowie an Veränderungen in der Umgebung anzupassen. Zur Lösung dieser Herausforderung werden drei zentrale Beiträge vorgestellt, die jeweils unterschiedliche Aspekte des Bewegungsplanungsproblems adressieren.

Der erste Beitrag zielt auf eine Beschleunigung der Bewegungsplanung ab, indem unnötige Kollisionsprüfungen während des Planungsprozesses vermieden werden. Die vorgeschlagene Methode basiert auf einer vorab berechneten, deterministischen Roadmap, die den kollisionsfreien Raum der statischen Umgebung erfasst. Anschließend wird eine heuristikgestützte Suche auf der Roadmap durchgeführt, und ein neuartiges Konzept von Sicherheitszonen für die Kantenauswertung eingeführt, um eine realisierbare Lösung zu finden. Beim Durchsuchen der vorab berechneten Roadmap minimiert die heuristikgestützte Suche die Anzahl der Kantenauswertungen, indem Kanten basierend auf Heuristiken priorisiert werden, die aus Pfadkosten in der Roadmap abgeleitet sind. Das Konzept der Sicherheitszonen nutzt die räumliche Beziehung zwischen dem Roboter und der dynamischen Umgebung, um Bereiche zu identifizieren, in denen keine Kollisionsprüfungen erforderlich sind. Dadurch kann sowohl die Anzahl der Kantenprüfungen als auch die Gesamtzahl der Kollisionsprüfungen deutlich reduziert werden.

Da die Planungszeit trotz der beschriebenen Optimierungen mit wachsender Problemkomplexität weiterhin unbeschränkt ansteigen kann, wird im zweiten Beitrag ein Verfahren vorgestellt, das komplexe Bewegungsplanungsprobleme in kleinere, einfach lösbare Teilprobleme unterteilt. Zu diesem Zweck wird zunächst ein Datensatz geeigneter Zwischenziele erstellt, die innerhalb eines festen Zeitrahmens lösbar sind. Anschließend wird ein konditionales generatives Modell trainiert, das auf Basis der Problemstellung passende Zwischenziele vorschlägt. Ein Zeitschätzer bewertet in der Inferenzphase die vorgeschlagenen Zwischenziele im Hinblick auf ihre Planbarkeit innerhalb des gegebenen Zeitlimits und ihre Eignung zur Annäherung an das Endziel. Auf dieser Grundlage wird ein geeignetes Zwischenziel für die weitere Planung ausgewählt.

Die oben genannten Beiträge berücksichtigen ausschließlich den aktuellen geometrischen Zustand dynamischer Umgebungen und vernachlässigen den zeitlichen Aspekt. Diese Kurzsichtigkeit in Bezug auf den Zeithorizont führt zu häufiger Neuplanung und leidet un-

ter lokalen Minima in dynamischen Umgebungen. Der dritte Beitrag geht dieses Problem an, indem episodisches Reinforcement Learning eingesetzt wird, um potenzielle Veränderungen in der Umgebung über die Zeit hinweg implizit zu berücksichtigen. Die erlernte Policy verfeinert Referenztrajektorien unter Berücksichtigung der zeitlichen Entwicklung der Umgebung. Hierfür werden drei Strategien zur Trajektorienverfeinerung eingeführt, die auf B-Spline-Bewegungsprimitive basieren und einen glatten Übergang zwischen Trajektorienabschnitten gewährleisten. Die in den ersten beiden Beiträgen berechneten Trajektorien dienen hierbei als Referenz. Das raumzeitlich bewusste Verhalten der Methode führt zu einer deutlich verbesserten Aufgabenleistung im Vergleich zu rein geometrischen oder klassischen raumzeitlichen Planungsverfahren.

Die Wirksamkeit und Grenzen der vorgeschlagenen Methoden werden durch umfangreiche Evaluierungen und Ablationsstudien sowohl in Simulationen als auch an realen Robotersystemen nachgewiesen.

Contents

1 Introduction

Humans live in a dynamic world that is constantly changing, which requires us to adapt our behavior continuously to stay safe. This adaptation often needs to happen within very short timeframes. Treating the reaction process as a whole, a delayed reaction time can result in undesired and potentially fatal consequences in scenarios such as driving in crowded traffic. The average human reaction time is approximately 220 milliseconds, including[1] receiving stimulus (20-50 ms), processing (70-100 ms), and finally initiating actions (50-150 ms) [80, 145, 109]. The fastest human reaction time is close to 100 ms [32]. Similarly, in robotics, the reacting and planning pipeline can be divided into three stages, i. e., perception, processing, and response. This thesis only focuses on the processing part of the reaction. To enable robots to operate effectively in a dynamic world, they should show a shorter reaction time than humans. In extreme cases where objects are moving at high speeds, an even shorter overall reaction time is needed to reduce the disparity in the environment before and after the three stages of reaction. For instance, if an object is moving at a speed of 2 m/s and the robot has an overall reaction time of 50 milliseconds, the object will have traveled 100 millimeters during this processing time. This mismatch can cause potentially fatal consequences. However, efficient planning of collision-free motions for robots with high degrees of freedom (DoF), e. g., articulated manipulators, at a millisecond level of time remains a significant challenge.

In scenarios like most production lines, robots are programmed to navigate the surrounding static environment for one specific task efficiently. When these robots are in operation, human workers are not allowed to enter or share space with them, as illustrated in Figure 1.1a. Assuming the environment remains stationary, previous developments in motion planning methods have focused on designing algorithms to find solutions that optimize specific objectives, such as minimizing energy consumption or minimizing execution time. Depending on the algorithms and the complexity of the problems, the planning process usually takes seconds to hours [157]. Planning must be restarted from the beginning once the surrounding environment or task context changes. When deploying robots into our daily lives, as shown in Figure 1.1b, or in more dynamic production lines such as logistics, their surrounding environments change continuously and are sometimes even unpredictable. It is crucial that the algorithm can consistently provide solutions within a bounded time frame. Algorithms designed with the assumption of stationary environments often fail to generalize and perform effectively in such dynamic contexts.

From this point onward, the term *dynamic environments* refers to the environments that change continuously and can sometimes be unpredictable. In dynamic environments, planners must be able to adapt in real-time and meet the following requirements to ensure safe and effective performance:

- **Bounded reaction time** at the millisecond level, enabling rapid responsiveness to changes in the environment.

[1] For simplicity, the reaction process is only divided into three stages in this thesis.

- **Spatiotemporal awareness** to involve consideration of potential environmental changes over time, allowing the system to adjust its motion strategy in advance and minimize unnecessary replanning efforts.
- **Current robot state awareness** to facilitate smooth motion transitions.

(a) Isolated environment **(b)** ARMAR-6 [5] in a daily scenario[2]

Figure 1.1: Robots in isolated and open environments. (a) An industrial robot is operating in an isolated cell with fences and safety monitoring equipment. (b) ARMAR-6 is performing a handover task collaborating with human workers.

1.1 Research Questions

This thesis focuses on developing motion planning algorithms for articulated manipulators with high DoF in dynamic environments while addressing the aforementioned requirements. Three research questions are formulated to fulfill one or some of the requirements. Before formulating the research questions, a brief review of different categories of motion planning methods and their capabilities is provided. Current planning methods can roughly fit into these three categories: sampling-based methods, optimization-based methods, and reaction control methods.

Sampling-based methods can be used in a plug-in manner to offer solutions to planning problems without the need to carefully specify the problem and define cost functions. The output of a sampling-based method is usually a collision-free geometric plan. Recent methods in this category handle spatiotemporal planning problems with full knowledge of the environment along the time horizon [44] or use parallelization to accelerate the sampling and collision checking [162]. The spatiotemporal planning methods [44] usually require several seconds of planning budgets and fail to provide millisecond-level reaction time. On the other hand, methods based on parallelization [162] fulfill the requirement of millisecond-level reaction time but consider solely the geometric state of the environment and the robot and return a set of geometric waypoints as solutions. These solutions are not guaranteed to respect the current robot joint position and velocity limits, and can be

[2]This is a snapshot from the video: `https://youtu.be/hVprn7XwRkk`. Throughout the remainder of this thesis, figures are original unless stated otherwise.

infeasible. Therefore, all requirements in Table 1.1 are marked with $-+$, indicating these requirements can be partially met by some methods in the category.

Optimization-based methods take a reference trajectory as an initial solution and improve it incrementally according to handcrafted objective functions [167]. This category of methods can directly output trajectories and account for current robot states and desired kinematic constraints. While most optimization-based methods have a relatively long and unbounded computation time, recent advances such as STORM [12] and CuRoBo [159] provide good online optimization performance. STORM handles dynamic environments by explicitly modeling them in the cost function and updates the final policy by rolling out sampled trajectories[12]. Due to the limited horizon of the sampled trajectories, only a short duration of environmental changes can be considered. On the other hand, CuRoBo [159] is limited to so-called semi-dynamic environments, meaning that it only considers the current state of the environment. Although CuRoBo demonstrates good online performance, its computation time highly depends on the random seeds [52] and remains unbounded. Therefore, the fulfillment for spatiotemporal awareness is marked as $-$ and bounded reaction time as $-+$ in Table 1.1 for the optimization-based methods.

Reactive control methods such as potential fields [77] and velocity fields using Dynamical Systems (DS) [76] can be fast and deterministic due to their low computational complexity. The outcome of these methods is feasible control commands based on the current state of the robot. Therefore, the requirements for the current robot state awareness and the bounded reaction time are met. However, these methods only consider the current state of the environment and do not meet the requirement for spatiotemporal awareness.

	Sampling-Based	**Optimization-Based**	**Reactive Control**
Bounded Reaction Time	$-+$	$-+$	$+$
Spatiotemporal Awareness	$-+$	$-$	$-$
Current Robot State Awareness	$-+$	$+$	$+$

Table 1.1: Fulfillment of requirements for motion planning methods. Only methods for dynamic environments are considered. The symbols $-$ and $+$ indicate whether methods in the category meet the requirement, while $-+$ denotes that the requirement is partially met by methods in the category.

None of these three categories of methods can fully meet the requirements stated above. Compared to methods from the other categories, sampling-based motion planning methods can be easily plugged into diverse scenarios as a global planner without much modeling effort, and Table 1.1 shows their potential to fit in dynamic environments. Using sampling-based methods as the foundation, this thesis formulates three research questions to overcome the limitations of traditional motion planning methods and, thus, fulfill the requirements. Addressing these questions leads to three main contributions in this thesis, termed Contribution 1, Contribution 2, and Contribution 3, illustrated in Figure 1.2.

Q1 What is the main bottleneck in accelerating robot motion planning, and how can it be overcome?

- **Contribution 1** describes a method that reduces the number of collision checks during planning - a significant bottleneck that slows down rapid motion generation

in dynamic environments. The proposed method addresses this bottleneck from three different angles. First, a deterministic roadmap is computed offline, which provides admissible heuristics for the online planning phase. In the online phase, a heuristics-informed search and a concept of safe zones are developed to reduce the number of edge examinations and the number of collision checks per edge examination, respectively. These three components lead to a significant decrease in collision checks, resulting in a speedup of 7 times on average for most complicated test planning problems. This method can further be used as a base planner in the following contributions. Other than the significant speedup, the results in the evaluation also indicate that the planning time increases with the complexity of the planning problem and remains unbounded.

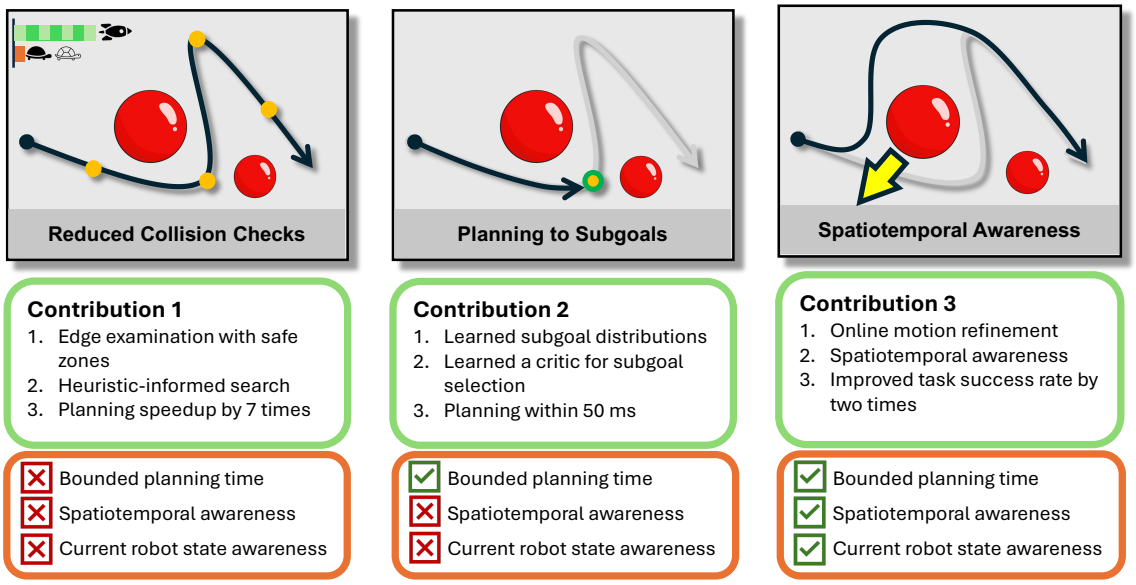
Q2 Can identifying and reaching subgoals be a more efficient strategy in dynamic environments?

- **Contribution 2** introduces a method for generating subgoals that decompose complex motion planning problems into small, easily solvable subproblems. By iteratively planning these subproblems, the approach ensures a short, bounded planning time in dynamic environments and guides the robot to approach the final goal incrementally. This method first establishes a data collection pipeline for suitable subgoals. With the collected dataset, a generative model is trained to capture the distribution of the subgoals. During inference, a learned critic is deployed to evaluate whether the generated samples fulfill the requirement of bounded planning time while being goal-oriented. The proposed method effectively decomposes complex problems requiring more than 1 second of planning time into small problems requiring less than 0.05 seconds.

Q3 How can planned solutions be adjusted to account for changes in the environment along the time horizon?

- **Contribution 3** proposes a method for refining trajectories to account for potential changes in the environment along the time horizon, addressing the limitations of Contribution 1 and Contribution 2. To grant spatiotemporal awareness regarding dynamic environments, this method uses episodic reinforcement learning to implicitly encode an understanding of how environments evolve into the learned policy. Additionally, trajectory refinement strategies based on B-spline-based movement primitives are developed to ensure smooth motion transitions, considering the robot's current positions and velocities. As a result, this method effectively combines sampling-based motion planning with episodic reinforcement learning, demonstrating superior task success rates compared to methods that do not incorporate spatiotemporal awareness.

While Contribution 1 and Contribution 2 together meet the first requirement regarding reaction time, Contribution 3 meets the other two requirements regarding proactive consideration of potential environment variations and smooth motion transitions.



(a) Contribution 1 **(b)** Contribution 2 **(c)** Contribution 3

Figure 1.2: Illustration of the contributions.

1.2 Outline of the Thesis

In the following, the structure of this thesis will be briefly introduced to give an overview of the big picture.

Chapter 2 first provides a broad review of robot motion planning methods and highlights how the focus of developing motion planning algorithms has shifted and evolved over time. Subsequently, it further delves into the topic of planning in dynamic environments, showing how different categories of methods, such as sampling-based and optimization-based methods, change their paradigm to embrace this challenge.

Chapter 3 addresses the bottleneck of speeding up motion planning methods for constantly changing environments. Being aware that the huge amount of collision checks is the bottleneck hindering rapid motion planning, a novel path planning method is proposed to identify and eliminate unnecessary collision checks during planning. As a result, this method demonstrates an average speedup of more than 7x in complex scenarios, making it possible to conduct robot motion planning in milliseconds. Despite the speedup, the proposed method does not address the constraints regarding a limited budget for planning.

Chapter 4 tackles the constraints regarding limited budget by decomposing a complex, long-horizon planning problem into smaller, manageable ones while still being goal-oriented. A generative model is used to sequentially generate a batch of subgoals, which can incrementally guide the robot to the final goal. To ensure the capability of planning within the time constraints, these constraints are implicitly encoded into the generative model and are explicitly considered using a time estimator as a critic.

Chapter 5 proposes a method to take into account how the dynamic environment evolves along the time horizon, while the method proposed in the previous chapters only considers the current state of the environment. The proposed method leverages reference trajectories as priors and uses reinforcement learning to generate refinements over the trajectories.

Multiple refinement strategies are discussed in this chapter and evaluated in simulation and real-world experiments.

Chapter 6 concludes the thesis by summarizing the key takeaways and highlighting the most important results of the work. At the end of each proposed component, contributions and limitations are discussed in detail. Based on these discussions, possible research ideas are proposed to tackle further challenges.

2 Related Work

This chapter reviews the state of the art in robot motion planning in the following order. First, Section 2.1 gives an overview of the different categories of methods for general motion planning problems. Then, Section 2.2 describes how the methods from these categories address the planning problem in the context of dynamic environments. Details of how machine learning changes the field of motion planning and how the end-to-end learning method conceptually differs from the classical motion planning methods are included in this part as well. Finally, Section 2.3 clarifies the relation between this thesis and other existing methods.

2.1 Overview of Motion Planning

Motion planning methods have been proposed and developed over the decades. This section reviews some significant categories of motion planning methods, i. e., sampling-based methods [74, 92, 86], optimization-based methods [140, 70, 146, 164, 167], and movement primitive (MP) methods [98, 144, 121]. As an overview, this section does not distinguish the methods by their capability to apply in dynamic environments. More discussion on dynamic environments is given in Section 2.2.

2.1.1 Sampling-Based Methods

The concept of motion planning using a reference point in the collision-free space was first introduced around 1979 [106]. This concept was later developed to plan in the configuration space for arbitrary kinematics. It has been proven that finding the shortest path in the configuration space among polyhedral obstacles is NP-hard [15]. This gave rise to sampling-based methods. The sampling-based perspective of motion planning methods was originally introduced [6] to fix the local minima problem caused by artificial potential fields [77]. While this method samples actions to escape the local minima, modern sampling-based methods evolved to approximate collision-free regions in the configuration space. Two major groups of sampling-based motion planning methods are single-query and multi-query planning methods. **Single-query** methods grow trees by sampling random configurations until the trees connect the start and goal configurations. Once the start-goal query is changed, the trees must grow from scratch. Methods like rapidly-exploring random trees (RRT) [92, 86], expansive-space trees (EST) [51, 128], fast-marching trees (FMT) [65], batch informed trees (BIT) [39] belong to this category. **Multi-query** methods usually construct a graph using random sampling to approximate the feasible set in the configuration space. This graph can be used for multiple start-goal queries and saved locally for the next usage [71, 178]. Methods such as probabilistic roadmaps (PRM) [74] and its variants [48, 27, 29] belong to this category.

The focus of the early phase of sampling-based methods is to have an efficient planner that is *probabilistically complete*. A planner is probabilistically complete if it can find the solution given an infinite time budget in case the solution exists. Techniques such as lazy collision checking [48, 27], biased sampling strategies [90, 88, 62], and deterministic sampling strategies [14, 53] are proposed to improve the planner's performance in terms of planning time while keeping the algorithm probabilistically complete.

Later on, the focus of developing sampling-based planners has been shifted to *optimality*. With rewiring operations with respect to heuristics on a graph or a tree, the planners can be extended to optimal planners, such as the graph-based planner PRM* [73], the tree-based planner RRT* [73], FMT* [65], BIT* [39], and the bi-directional tree-based planner AIT* [157].

With the increasing use of data-driven and machine learning techniques, several methods have been proposed to enhance or replace components in the sampling-based motion planning pipeline. It is important to note that the techniques discussed in this thesis are built on sampling-based methods and closely integrate with machine learning techniques to facilitate planning in dynamic environments. A detailed review of methods that combine sampling-based motion planning with machine learning is provided to visualize the different possibilities to apply machine learning to motion planning methods.

Data-Driven The early stage of leveraging the data does not involve machine learning. Lightning [11] collects the previous planning examples and saves them in a database for later retrieval. As a new planning query is requested, the result from a previous similar planning problem is retrieved from the database and then modified for the new query. Instead of saving the experience directly in the database, Thunder [24] constructs and maintains a sparse graph based on the experience and achieves much faster computation.

Later on, components using supervised learning and reinforcement learning were actively developed and integrated into sampling-based pipelines.

Reinforcement Learning PRM-RL [33] combines reinforcement learning and PRM for long-range navigation tasks. As a local planner, the RL agent navigates the robot based on the noisy sensor inputs and provides connectivity information for the PRM. This connectivity information indicates the capability of the RL agent to travel between two points and is used for graph construction instead of the trivial straight-line interpolation. This is different from kinodynamic planning methods [127]. While kinodynamic sampling-based methods solve optimal control problems to connect two samples, the edges in PRM-RL do not explicitly represent system dynamics. Other than connectivity information, the RL agent has also been used to provide the heuristics for connecting the nodes in RRT [22]. While the connectivity and heuristics are handcrafted features, APES [95] uses a generator-critic framework to bias the sampling distribution, which can directly optimize the planner's performance. This framework is very similar to the action-critic framework in reinforcement learning. The methods above use RL to replace a component in sampling-based methods. The reverse also holds. DDPG-MP [69] puts motion demonstrations from sampling-based planners into a replay buffer to guide the exploration in case the RL agent fails in an episode during training. This idea is close to the idea proposed in Contribution 3, where the reference trajectories serve as good priors for exploration.

Supervised Learning Compared to reinforcement learning, the marriage between supervised learning and sampling-based methods is more successful. The exploration has been in biasing the sampling distribution [96, 61, 88, 91, 95], lazy collision checking

by learning to prioritize the exploration [180, 18, 182, 18] and estimating the possibility of collision [56, 57, 183] or clearance [75], construct roadmaps with critical samples [62, 88, 91], planning in encoded latent spaces [59, 18] and predict next samples in an auto-regressive manner [134, 135, 137, 136, 155]. Supervised learning is more powerful than reinforcement learning in this context, which can be attributed to its ability to encode tasks and environments. The following introduces methods of each category mentioned above to illustrate the idea.

Sampling from **biased distributions** conditioned on the task context usually results in a significant speedup compared to uniform sampling. Repetition sampling [96] learns Gaussian Mixture Models (GMM) from previous solutions and applies the learned distribution to similar tasks. GMM-based Multi-RRTs (GMMM-RRT) [184] build trees inside each component of the mixture models and finally connect them. Other than GMMs, a conditional variational auto-encoder (CVAE) [79] can be learned from optimal paths and used to generate task-relevant samples [61]. Compared to GMMs, CVAE does not explicitly express the sample distribution but encodes it in a latent space. Conditioning on the task context, such as the initial states, goal region, and obstacles, the samples from the latent spaces can be mapped to biased samples. LEGO [88] fuses the idea of critical PRMs and learned distribution. It identifies a set of bottleneck nodes representing the regions difficult for a uniform sampler to cover. This is similar to the surface sampler and the Gaussian sampler. The learned CVAE is trained to capture the distribution of the bottleneck nodes.

Collision Estimation GMM can be used to represent the distribution of the collision region in configuration space as well [56, 57]. The components in the mixture model can be updated during the planning. Some approaches learn to reject the sample instead of bias the distribution to the desired region [183]. ClearanceNet [75] is a neural network that predicts the clearance, i. e., the minimum distance between the robot and the environment, and can provide gradients to shift the samples away to enlarge the minimum distance. With the features of batched operation and gradients coming from a neural network, a parallelized algorithm, CN-RRT, achieves a significant speedup.

The **prioritized exploration** works closely with collision estimation. A transformer-based graph neural network (GNN) is used to predict whether an edge in a random geometric graph is collision-free or not [180]. The prediction prioritizes the search and exploration of the graph, resulting in fewer collision checks. The graph-based explorer [180] suffers from the problem that the exploration does not take the global accumulated cost of the path into account. GraphMP [182] addresses the problem by introducing two GNN-based modules to estimate the probabilities of collision and the heuristics for A* graph search.

Roadmap Construction The concept of connectivity is further used to construct critical PRMs [62], where the samples of high connectivity are seen as critical samples. A neural network is trained and used to evaluate the connectivity of the samples. SV-PRM [91] treats the samples in a PRM as particles and uses particle-based variational inference [104] to update the distribution of the samples for more homogeneous coverage of the feasible set. Strictly speaking, SV-PRM is more of a maximum a posteriori (MAP) problem than a supervised learning problem. The information collected from the observation is used to update the posterior distribution. However, to show a big picture of how machine learning improves different components in the sampling-based motion planning pipeline, SV-PRM is introduced together with the learned critical roadmaps.

Planning in Latent Space Learned Latent RRT (L2RRT) [59] uses an autoencoding model, a dynamic model, and a collision-checking model to construct an RRT-based pipeline. With the encoded states using the encoding model, the steering and collision checking operations are conducted in the latent space. Planning in latent space has been extended to trajectory optimization [3]. Neural Exploration-Exploitation Trees (NEXT) [18] is a meta-neural path planning algorithm that uses a neural prior to guide the selection and expansion of tree-like planning approaches. The high-dimensional planning space is mapped to a 3D discrete latent space.

Learning Differential Fields An alternative representation of the desired distribution is the cost-to-go map [58], which is a field indicating the cost to reach the goal. The trajectory from the start to the goal is generated by following the slope of the field. A similar idea is the implicit signed distance field in joint space [83]. NTFields [114] leverage a physics-informed network to mimic the Eikonal equation, which approximates wave propagation and solves the shortest arrival time problem. The Eikonal equation can be applied to high-dimensional space, and the motion planning problem can be solved by following the gradients of the field.

Auto-regressive planning refers to the machine-learning-based methods that iteratively generate waypoints, which are then used as inputs for subsequent iterations until a solution connecting the starting point to the goal is achieved. These methods now utilize sampling-based motion planners to generate datasets for training, rather than relying on them at the inference stage. DeepSMP [134] learns a contractive auto-encoder to extract features from point clouds. These features are then fed to a deep neural sampler to predict the next samples on the optimal path. On inference, the auto-encoder and the neural sampler work in a stochastic auto-regressive manner. The auto-regressive mechanism is further improved by Long Short Term Memory (LSTM) [50] and achieves fixed-time inference [10] at a level of seconds for a 6-DoF planning problem. A follow-up work MPNet [137, 136] uses active continuous learning to accelerate the learning process. A bi-directional tree and trajectory refinement techniques, building on the learned auto-regressive model, are proposed to make the neural planner usable. This learning pipeline is further extended to planning on constraint manifolds [136] and MPC [100]. SIMPNet [155] uses a GNN to encode the kinematic chain of robots and then fuses it with the working encoding using cross-attention. The fused representation is used to generate samples in a bidirectional auto-regressive manner, similar to MPNet [135]. As a result, a computation time of seconds is reported in the paper, which cannot be directly applied to dynamic environments. The subgoal learning in Contribution 2 is related to this category. However, the key difference is that Contribution 2 aims to generate samples that can be planned within a desired bounded time with a sampling-based planner involved, while the methods in this category sequentially construct a complete path without utilizing a sampling-based planner. This distinction allows the method introduced in Contribution 2 to return a solution that is guaranteed to be collision-free within a short time frame.

The focus of sampling-based motion planning shifts from probabilistic completeness to optimality and finally to machine learning-based methods. Machine learning can be applied to various components of the sampling-based motion planning pipeline.

2.1.2 Optimization-Based Methods

Trajectory optimization uses gradient information to update the initial trajectory regarding user-defined objective functions while fulfilling the constraints. The general formalism of trajectory optimization for a path of T time steps $\boldsymbol{x}_{0:T} = [\boldsymbol{x}_0, \dots, \boldsymbol{x}_T]$ with $\boldsymbol{x}_\cdot \in \mathbb{R}^N$ can be written as

$$\min_{\boldsymbol{x}_{0:T}} f(\boldsymbol{x}_{0:T}), \quad \text{s.t.} \quad g(\boldsymbol{x}_{0:T}) \leq 0, \quad h(\boldsymbol{x}_{0:T}) = 0, \tag{2.1}$$

where $f(\cdot)$ denotes the objective function, $g(\cdot)$ defines the inequality constraints and $h(\cdot)$ defines equality constraints. Based on this formalism, some trajectory optimization methods are proposed to reduce the problem complexity and accelerate the optimization. In the following, methods based on gradient-based optimization [140, 186, 167, 146, 147, 165, 142, 141], gradient-free optimization [70, 93] and planning as inference are reviewed.

Gradient-Based Optimization Second-order derivatives contribute to faster convergence of the optimization. CHOMP [140, 186] uses covariant gradients with respect to the Hessian to achieve fairly fast convergence regarding the objective function composed of the collision avoidance and trajectory smoothness. Covariant gradients do not depend on the selection of coordinate systems [167] and provide a reliable step size and step direction for the optimization. TrajOpt [146, 147] consider continuous-time collision checking and use sequential convex optimization to solve the non-convex optimization problem in Eq. 2.1. Compared to CHOMP, TrajOpt requires fewer iterations to converge. KOMO [165], a framework for robot motion optimization, is the first to introduce an anytime version of the Augmented Lagrangian. This framework is further used in work such as logic-geometric programming (LGP) [166] and BITKOMO [72]. Inspired by KOMO, RieMO [141] is a motion optimization framework integrating first-order Riemannian geometry of the workspace. The geometry of the workspace, including obstacles, can be better represented by Riemannian metrics. Different Gaussian-Newton approximations and their impact on the optimization are investigated [142] as well. Other than Gaussian-Newton methods, Stein variational gradient descent can be applied to trajectory optimization [130].

Gradient-Free Optimization STOMP [70] generates a set of noisy trajectories and computes their costs in every iteration. These costs are used to update the candidate solution, and no gradients are required. A similar idea of stochastic optimization can be seen in model predictive path integration (MPPI) methods [175, 12]. Stochastic trajectory optimization has been extended to be multimodal and can generate diverse solutions [116]. Optimal transport is used to accelerate the trajectory optimization based on particles [93], where a gradient-free update rule, the Sinkhorn Step, shifts the particles toward low-cost regions.

Planning as Inference Trajectory optimization can be viewed as probabilistic inference [164, 168]. Instead of optimizing a trajectory, the methods of this category condition the distribution of trajectories on the desired constraints and infer the posterior distribution. GPMP [111] considers smooth continuous-time trajectories as samples from a Gaussian process (GP). Unlike the discrete-time formulation Eq. 2.1, only a few states are used to parameterize the trajectory, and GP interpolates these states. GPMP2 [31] uses factor graphs and numerical optimizations to accelerate GPMP. The factor graph can be modified for replanning at run-time while the goal changes. The changes regarding the start and goal position have a relatively small impact on the factor graph. However, when the environment changes, the benefit of using the factor graph from previous planning

does not hold anymore. Gaussian Variational Inference Motion Planning (GVI-MP) [181] optimizes the trajectory distributions by using variational inference to approximate the posterior distribution. Compared to GPMP, GVI-MP is a motion planning formalism that considers uncertainties.

2.1.3 Movement Primitives

Movement Primitives (MPs) [144, 98, 121] are motion generators that are broadly used in learning-based methods such as imitation learning [185] and reinforcement learning [117, 118]. The fundamental idea of MPs is to learn a weight vector for a set of predefined basis functions to parameterize a trajectory. Dynamic Movement Primitives [144] introduces a force term to a second-order spring-damper system with goal attractions. Parameterizing the force term leads to parameterization of the trajectories. Probabilistic Movement Primitives (ProMPs) represent the weight vector with a probabilistic model, enabling probabilistic inference conditioned on the environmental context and online updates. While ProMPs can model uncertainty and conduct conditional inference based on new observations, they fail to support the desired boundary conditions to achieve smooth trajectory switching. Probabilistic Dynamic Movement Primitives (ProDMPs) unify the DMPs and ProMPs by employing the closed-form solution instead of numerical integrations to represent how the force term in DMPs influences the trajectory, resulting in probabilistic modeling of the trajectories while supporting the smooth transitions.

Movement primitives are central to Episodic Reinforcement Learning (ERL) methods. Early works on ERL employed black-box optimization techniques to evolve parameterized controllers, such as small MLPs [174, 63, 43]. Episodic Policy Learning [82] first proposed using Movement Primitives as parameterized policies for ERL. This approach reduces the search space dimensions, shifting from the neural network parameter space to the smaller weight space of MPs (typically ranging from 20 to 50). Beyond improving sample efficiency, MPs also enable the generation of smooth trajectories [118] and more efficient exploration [99]. Early MP-based ERL methods [125, 1, 129] cannot deal with task variations. To address this limitation, follow-up works such as [2] and [17] proposed using linear models to predict MP parameters conditioned on task context. A recent work, *Deep Black-Box Reinforcement Learning* (BBRL) [117] further extends MP-based ERL with neural network policies and on-policy policy gradient updates with differentiable trust region layers. Building upon BBRL, MoRe-ERL employs contextual ERL to learn smooth motion residuals and further improve the sample efficiency by leveraging the prior knowledge from sampling-based motion planners.

Other than ERL, movement primitives have been deployed for residual learning methods. The idea of using RL to learn motion residuals was first introduced by [152] and [68]. This approach leverages the generalization capability of RL to enhance traditional feedback controllers, such as the PID regulator, to handle complex dynamics like contacts and friction, which are often difficult to model. Learning motion residuals simplifies the RL problem, reducing the demand for samples and making real-world applications more feasible [68, 139]. Recent works have also explored incorporating MPs into residual RL [16, 26], where MPs generate reference trajectories, while step-based RLs are used to learn the motion residual. However, step-based residual RL still suffers from similar limitations to step-based RL, including a lack of smoothness in generated motions and a heavy reliance on dense Markovian reward.

In Contribution 3, the proposed method uses ERL and B-spline-based movement primitives [102] to compute motion refinements for reference trajectories in dynamic environments, showing superior sample efficiencies and task success rates.

2.2 Motion Planning in Dynamic Environments

The previous section introduced multiple categories of methods for solving motion planning problems. This section delves deeper into methods suitable for dynamic environments. First, a clear line is drawn to distinguish the methods operating in dynamic environments from those that cannot. The methods that can consistently return feasible results within 50 ms are categorized as *dynamic-environment-capable*. Then, an additional distinction is introduced to separate methods incorporating predictions of future system evolution from those that rely solely on the current state.

This section reviews methods from the following categories: sampling-based methods [162, 176, 149, 179], optimization-based methods [171, 123, 41, 107, 159, 169], geometric methods [143, 97, 20, 170], signed distance field [83, 101, 103, 35], dynamic system [76, 55, 84, 38, 34] and end-to-end methods [105, 78]. While reviewing these methods, the focus is on analyzing where the speedup comes from. By having their capabilities analyzed, the uniqueness of the approaches proposed by this thesis will be demonstrated later in Section 2.3.

2.2.1 Sampling-Based Methods

A very straightforward approach to speeding up motion planning is parallelization. The parallelization does not conceptually change the algorithm but uses a more reasonable scheduler [150] or more efficient data structure [162, 138, 176] to achieve a significant speedup.

MPAccel [150] proposes a Spatial Aware Scheduler (SAS) and Cached Early-exit Collision Detection Unit (CECDU). SAS distributes the resources for collision checking while verifying an edge. CECDU uses a hierarchy to conduct collision checking based on primitives and precise collision checking if required.

Vectorized Sampling-Based Planning (VSBP) [162] utilizes Single Instruction/Multiple Data (SIMD) to speed up sampling-based motion planning on CPUs. The vectorization includes forward kinematics, collision checking, and an edge validation scheme. This is a general speedup that can be applied to most sampling-based methods. The method proposed in this dissertation can also benefit from this approach.

While VSBP focuses on the parallelization of the planning algorithm, Collision-Affording Point Tree (CAPT) [138] is a novel data structure designed to support parallelization in the perception part. CAPT is a spatial data structure that represents 3D point clouds. It inherits the SIMD parallelism and refines k-d trees to achieve collision checking against a point cloud with thousands of points at more than 60 FPS. Together with VSBP, online motion planning using sensor data on a CPU is feasible.

Fully Connected Informed Trees (FCIT*) [176] applies vectorized edge verification in VSBP to a fully connected graph, achieving anytime almost-surely asymptotic optimality

at the price of sacrificing the speed. The optimality is rather spatial optimality, neglecting the spatiotemporal aspect.

All methods mentioned above can be used in semi-dynamic environments, meaning that they are fast enough to constantly replan, but only with respect to the current state of the environment.

2.2.2 Optimization-based Methods

Optimization-based methods can be enhanced to be dynamic-environment-capable from diverse aspects. Other than parallelization using GPUs [159, 70], these methods achieve efficient computation by introducing waypoints to concentrate on smaller problems [123, 169, 107] or using neural signed distance fields [103, 101].

Parallelization

While the parallelization of sampling-based methods is realized using efficient data structures, the parallelization of optimization-based methods is usually done using GPUs.

CuRoBo [159] is an optimization-based framework based on GPU-parallelism. Given an initial configuration and goal pose in SE(3), CuRoBo parallelizes optimizations for IK and trajectory optimization over multiple random seeds. To have a good solution, the number of seeds is usually set to more than 1000. DiffusionSeeder [52] uses Diffusion [49] to generate seed trajectories for optimization, resulting in faster convergence and higher success rate. Other than Gaussian-Newton methods, sampling-based MPC can be implemented on GPUs as well. Stochastic Tensor Optimization for Robot Motion (STORM) [12] is a parallelized MPPI or sampling-based MPC method. The roll-out of the sampled trajectories runs on GPUs, and the collected costs are used to update the actual robot trajectory. Additionally, a mapping between the raw sensor data and the collision-checking-related cost function is learned, making this method capable of directly integrating perception into the control loop. As a global planner, CuRoBo finds a complete path to the desired goal, while STORM can get stuck in local minima.

Signed Distance Fields

Signed distance fields map points in a specific space to the distances between these points and the corresponding closest objects. Methods like MPPI need efficient distance computation to include the collision objective in the optimization [83]. Therefore, neural signed distance fields provide a huge boost for this purpose due to their parallelization on GPUs.

Regularized Deep Signed Distance Fields (ReDSDF) [103] are signed distance fields at any scale based on the assumption that the distance to the object center can approximate the distance while the point of interest is far from the object. In the close-up case, the joint configuration of the object is considered for the fine-grained distance field. Composite signed-distance field [35] takes object tracking into account and fuses the predicted signed distance field of dynamic objects along the time horizon into the static distance field, resulting in efficient collision avoidance. Neural Joint Space Implicit Signed Distance Function [83] uses a neural network to predict the shortest distance between the

robot and a point in the Cartesian space. The learned neural network can be used as collision avoidance constraints or a cost function in a quadratic programming inverse kinematics or a joint space sampling-based MPC. The formulation is similar to STORM; therefore, it inherits the same problem as STORM. Another approach combining configuration space distance field and MPPI is RAMP [172]. RAMP uses MPPI as a global planner and introduces an additional local vector-based approach for trajectory following. Establishing the distance field in robot joint configuration space, configuration space distance field (CDF) [101] directly estimates the minimum joint movements to hit the object. These joint movements indicate the gradients on the field. The CDF is designed for manipulation tasks. In the grasping scenario, the gradients directly guide the robot to approach the objects.

Problem Decomposition

Decomposing the high-dimensional space into a hierarchical tree structure, Park et al. sequentially optimize the trajectory in each tree node [123]. An incremental approach based on ITOMP [122] is used to handle the dynamic obstacles for a short time horizon. The trajectory is optimized according to the hierarchy and will be aborted once the time budget is up. This process is repeated until the goal is reached. The concept proposed here is to generate sub-trajectories instead of solving the complete problem, which is very similar to that introduced in chapter 4. The difference is that this method decouples all joints in a hierarchy. While the robot movement is strongly correlated, this assumption can lead to a huge amount of local refinements. This approach can address the spatiotemporal aspect of the environment only for a very small time window.

SecMPC [169] decomposes an overall manipulation control problem into solving sequential waypoints, timing the waypoints, and refining a short receding-horizon path. It assumes that the task and motion planning (TAMP) plan is computed offline. While the waypoints and their timing are optimized over the whole execution, accounting for long-term constraints, the short receding-horizon path is refined to reactively avoid collision.

SPARROWS [107] is a receding-horizon trajectory planner based on a differentiable reachability set and an exact signed distance function. The receding-horizon mechanism is very similar to the concept of subgoal introduced in chapter 4. Although the authors claimed that this method can be used for online planning, the signed distance field is computed offline. This indicates that this algorithm cannot be used for dynamic environments where the signed distance field changes over time.

Some methods combine the decomposition and the parallelization [64, 7]. Via-Point-based Stochastic Trajectory Optimization (VP-STO) [64] deploys a Gaussian distribution to iteratively sample via-points and uses the sampled via-points as references to simplify the stochastic trajectory optimization problem. The distribution is updated using an evolution-based optimization method, Covariance Matrix Adaptation (CMA-ES), at runtime. The fundamental idea in VP-STO is close to the subgoals learning in Contribution 2, implying that VP-STO is not able to address the spatiotemporal aspect of the planning problem either.

2.2.3 Reactive Control

Reactive control methods provide instant robot commands based on the current state of the robot [77, 76]. Therefore, the requirements regarding millisecond-level reaction time and current robot state awareness are fulfilled by nature. The following reviews the recent reactive methods based on dynamical systems (DS). Compared to other reactive methods, methods based on DS provide a strong theoretical guarantee for goal attractions.

Dynamical Systems

Using a similar formulation modeling the flow around an object, real-time collision avoidance can be achieved using Dynamical Systems (DS) [76]. The modeled flow indicates the velocity of the robot. Instead of precomputing a trajectory for the robot, the methods based on DS compute a velocity field or control field. Given the robot's position, the desired control command is computed in real-time, making these methods dynamic-environment-capable. Similar to potential fields, velocity-field methods have a generally short computation time, and their performance depends on the construction of the fields. It is worth mentioning that the avoidance behavior is only defined by the task space, making it not applicable in complicated scenarios [76, 55, 38].

Concave objects are a common cause for local minima in potential-field [177] and velocity-field methods [84]. To avoid these local minima, Rotational Obstacle Avoidance Method (ROAM) [55] extends the velocity field approach to concave objects using general non-linear dynamics. However, the avoidance is still limited to the task space. On-manifold planning methods [38] model the environments as a manifold, on which the end-effector trajectory can be derived. However, this method cannot be applied to joint space trajectories.

To scale the DS-based avoidance policy to joint space, a sampling-based MPC approach is used to construct obstacle-tangential velocity components in a receding horizon manner [84]. This method uses DS to compute joint space motion and activates the velocity component based on sampling-based MPC as an avoidance policy when necessary. However, the sampling-based MPC requires explicit modeling to account for potential changes in the environment over time, which makes it challenging to adapt effortlessly to different problems.

Generally, methods based on DS can provide good performance regarding collision avoidance in dynamic environments due to their capability of instant responses and awareness of current robot states. However, the spatiotemporal awareness remains unexplored.

2.2.4 Geometric Methods

Geometric methods use Riemannian geometry and Riemannian metrics to replace commonly used Euclidean metrics. Riemannian geometry is a branch of differential geometry that studies smooth manifolds. The metrics used to measure distance, angles, and other geometric properties are called Riemannian metrics [94]. Riemannian metrics offer a unified way to represent the motion in the task space and joint configuration space.

RieMO [141] is a motion optimization framework integrating first-order Riemannian geometry of the workspace. The geometry of the workspace, including obstacles, can be

better represented by Riemannian metrics. An optimization framework based on Riemannian metrics is proposed to investigate how to properly utilize the gradient information in a motion optimization problem. Riemannian Motion Policy (RMP) [143] is a mathematical object that pairs diverse policies with a Riemannian metric and fuses them into a second-order dynamical system. For example, suppose there are multiple motion policies, and each can avoid an obstacle in the workspace. In that case, RMP composes these policies using the Riemannian metrics and uses PullBack operations to map the resulting motion to joint configuration space. RMPflow [20] and RMP2 [97] use a tree hierarchy to represent and weigh these motion policies, making it very flexible to compose diverse policies [170]. Strictly speaking, RieMO is not a dynamic-environment-capable method. Starting with RMP, the time-consuming computation of low-level policies can be distributed to different machines, and the composition runs in real time. Riemannian geometry has also been employed in Imitation Learning (IL) [81, 8].

2.3 Relation to State of the Art

Contribution 1 While methods reviewed in Section 2.2.1 use parallelization to accelerate the collision checking and distance computation [162, 138] and neural signed distance fields [83] conduct batched operations on GPUs, all these methods strive to increase the capacity of collision checks per time unit. This benefits the method proposed in Contribution 1 rather than conflicting, since the goal of Contribution 1 is to reduce the accumulated amount of the distance computation. Theoretically, improved capacity per time unit can further shorten the planning time using the proposed method in Contribution 1.

Contribution 2 The idea of decomposing a complex problem into small ones is used from different angles to shorten the computation time in optimization-based methods [169, 7, 107]. The general concept is to generate sub-trajectories instead of solving the complete problem, which is very close to the method proposed in Contribution 2. Different from these methods, Contribution 2 aims to generate samples for sampling-based planners that can be planned within the desired bounded time.

Contribution 3 The most obvious feature in Contribution 3 is the implicit consideration of potential environment variations over time. Sampling-based methods such as ST-RRT* [44] conduct spatiotemporal planning when full knowledge of the environment is available. However, this assumption regarding full knowledge is, in most cases, not realistic. For sampling-based MPC methods such as STORM [12], the dynamics of the objects require explicit modeling to take into account possible changes in the environment over time. By taking only a short time window of state predictions into account, the approaches can still suffer from the problem of short-sightedness. RMPflow [19] integrates both position and velocity as the state of the environment to determine which obstacles should be considered in the current control loop.

3 Speeding up Motion Planning by Reducing Collision Checks

Newell's time scale of human action indicates that the time scale for deliberate action is 100 milliseconds [113], encompassing perception, processing, and action. Planning belongs to the processing stage. Given the time required for the subsequent stages, the algorithm must plan a feasible trajectory within a few milliseconds. However, typical motion planning algorithms require up to several *seconds* to compute a solution [115]. To address the gap between the required milliseconds and the actual seconds taken, this chapter explores the following research question:

What is the main bottleneck in accelerating robot motion planning, and how can it be overcome?

The bottleneck is well known in the literature [48, 75]. For a robot manipulator with high DoF, collision checking uses up more than 95% of the computation time of the sampling-based motion planning methods [48], making it the primary bottleneck to speed up and achieve online motion planning. Collision checking is an essential module for sampling valid configurations and examining the connection between two configurations in sampling-based methods. For an ordinary sampling-based motion planning problem, a large number of collision checks are needed to find a collision-free path. While one single collision check takes several microseconds [120, 119], this adds up to seconds of computation time.

To mitigate this computational burden, two main strategies can be employed: (a) reducing the total number of collision checks, denoted by n_{cc}, and (b) reducing the computational time required per collision check. This chapter focuses on strategy (a). While some other works [120, 75, 159, 112, 25] try to solve the problem using option (b), the methods from option (a) can benefit from them, leading to even greater overall efficiency. In this chapter, to enable the sampling-based motion planning methods to respond to the changes in the environment quickly, a **H**euristics-**I**nformed **R**obot **O**nline (HIRO) planning method is proposed to address the primary bottleneck of fast motion planning [1]. This method consists of (1) an offline step computing a deterministic roadmap regarding the static environment and (2) an online step conducting a heuristic-informed search over the pre-computed roadmap. For a static environment, the deterministic roadmap represents the collision-free joint configurations of the environment and must be computed only once and stored locally. The heuristic-informed search prioritizes edges with a high probability of being in the final solution and employs a novel concept, safe zones, to examine their validity. This proposed method significantly reduces the number of collision checks required to solve motion planning problems, enabling faster motion planning.

[1] HIRO was originally published in IROS'22 [53].

The total number of collision checks n_{cc} is correlated to two factors, i.e., the number of evaluated edges n_{ee} and the average number of checks per edge n_{ce}. The *key insight* of this chapter is that n_{ee} can be optimized by informed heuristics for the graph search, and n_{ce} can be reduced by introducing the concept of safe zones. Using a precomputed roadmap for the static environment, HIRO reduces the complexity of a single collision check and allocates computational resources to the dynamic part, thereby resulting in a further speedup.

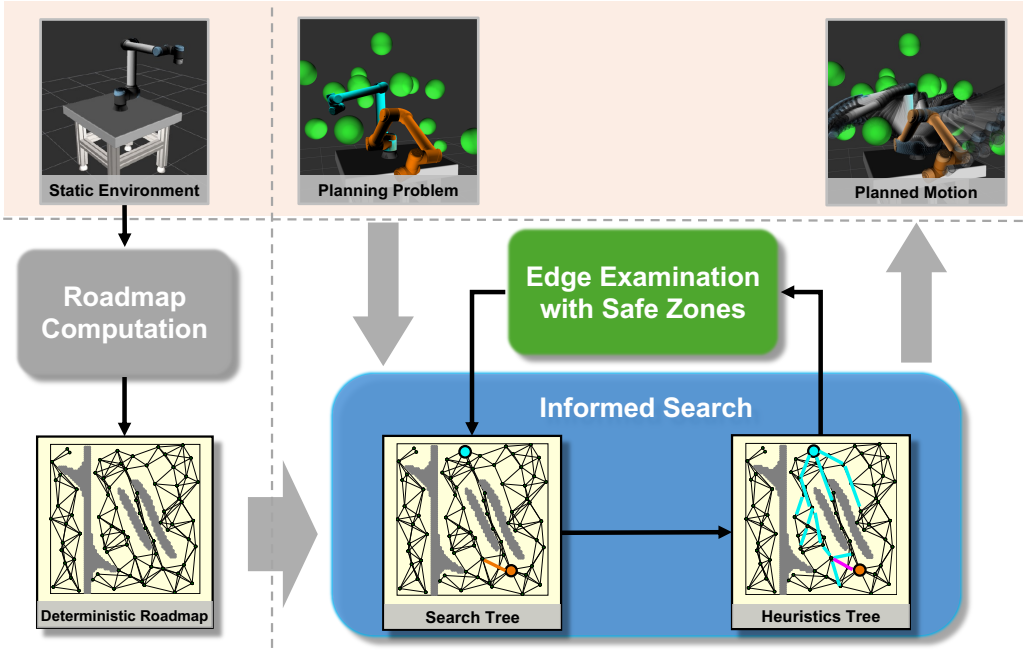


Figure 3.1: Pipeline of the HIRO. The deterministic roadmap is computed offline. Given a new planning problem with new obstacles in the workspace, HIRO performs a heuristics-informed search on the roadmap and examines edges using safe zones until a solution is found.

The high-level pipeline of the proposed method is illustrated in Figure 3.1. In the offline preparation phase, HIRO generates a deterministic roadmap regarding the static environment. The static part of the environment comprises permanently present objects, such as the table on which the robot is mounted. The dynamic component of the environment, including manipulable objects and humans within the workspace, is treated separately during the online phase. This precomputed roadmap can be stored locally and loaded only once prior to execution. Then, during the online planning phase, HIRO performs a heuristics-informed search across this roadmap. The heuristics estimate lower bounds on the final path cost, guiding the search to prioritize exploring edges likely relevant to the optimal solution. This process involves examining edges with safe zones. The so-called safe zones outline the regions in configuration space that are certain to be collision-free, allowing the examination to move on to the uncertain parts of the edge. The concept of safe zones improves the efficiency of the examination compared with the standard procedure, which requires breaking down the edge at a fixed resolution and sequentially checking the areas. In dynamic environments, HIRO keeps colliding nodes and edges in the roadmap by temporarily deactivating them for the current search, enabling the exact roadmap to be used as conditions change.

HIRO is evaluated across a range of planning problems of varying complexity, achieving an average speedup of 7.6x compared to the best baseline in the most challenging group of problems. Furthermore, the method is applied to a real-world scenario with moving objects, requiring rapid trajectory adjustments to avoid collisions and reach target goal regions. This requires the method, as a global planner, to be reactive and responsive. It is worth noting that this planner is sufficiently fast for dynamic environments by accounting for only the current state of the environment.

This chapter is structured as follows: Section 3.1 briefly introduces the concepts of robot task and configuration space and standard edge examination procedures, Section 3.3 depicts the offline phase computing deterministic roadmaps, and Section 3.4 describes the heuristic-informed graph search on the precomputed roadmap. Section 3.5 details the mathematical foundation of safe zones and how to apply safe zones to edge examination. Finally, Section 3.6 exhibits detailed evaluation results both in simulation and with robots. Discussions of the limitations of this approach are given at the end of the chapter.

3.1 Preliminaries

Before introducing the safe zone concept and the corresponding edge evaluation pipeline, this section briefly reviews fundamental concepts in motion planning, including the task and configuration spaces, collision checking, and edge examination.

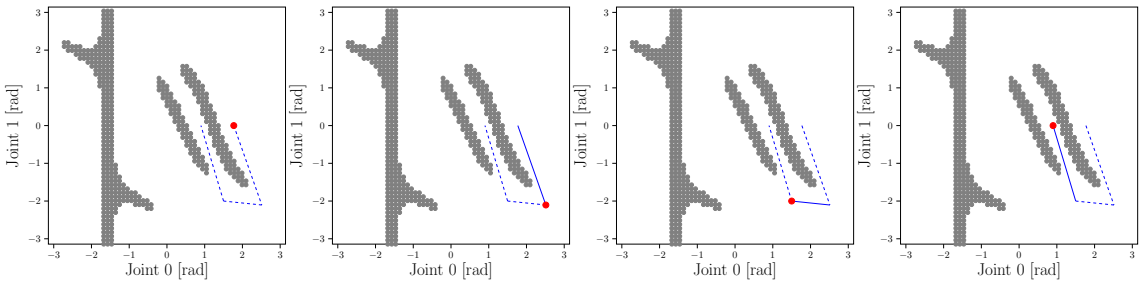
3.1.1 Task and Configuration Space

In robotics, task space and joint configuration space are two essential ways to describe a robot's pose and motion [151]. The task space, also known as Cartesian space, is the space in which the robot operates. In contrast, the joint configuration space represents all possible combinations of the robot's joint parameters. These parameters correspond to joint angles for a manipulator with revolute joints. Due to the rotation of revolute joints, linear displacements in configuration space usually result in nonlinear motion in task space, see Figure 3.2. Additionally, infeasible regions in the task space correspond to very different representations in the joint space, complicating motion planning between two joint configurations. For example, as shown in Figure 3.2, the smallest circular obstacle in the task space can correspond to the largest infeasible regions in the joint configuration space on the left.

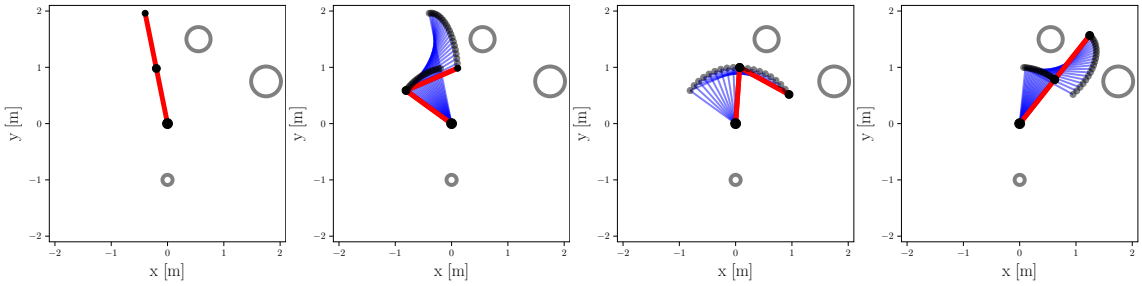
3.1.2 Collision Checking and Distance Computation

Collision-checking and distance-computation methods are two overlapping categories. Distance computation methods compute the distance between two objects, thereby implicitly indicating whether a collision occurs. On the other hand, some methods over-approximate object volumes using primitive shapes, quickly returning collision status without computing the exact distance. Generally, distance computation incurs some overhead relative to binary collision checking.

In robot manipulation tasks, Bounding Volume Hierarchies (BVHs) are commonly used for binary collision checking [151]. Typical types of bounding volume include Axis-



(a) Configuration space



(b) Task space

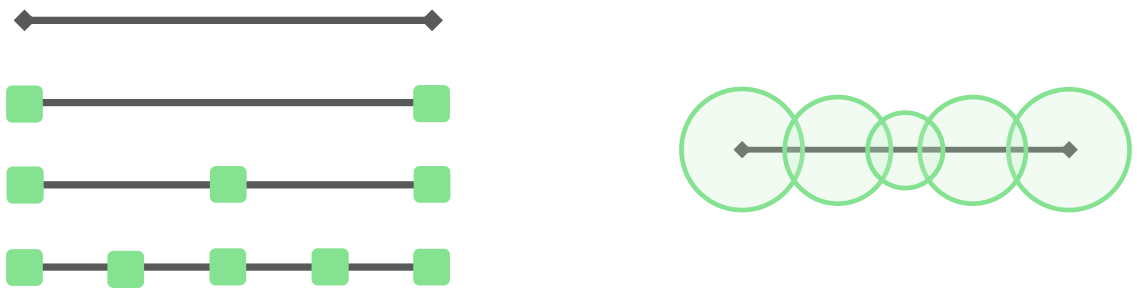
Figure 3.2: Configuration space and task space of a 2-DoF robot. Gray circle-shaped obstacles in the task space correspond to irregularly shaped regions in the configuration space. The red dot in the configuration space represents the current configuration of the red robot in the task space. Linear changes in the configuration space (blue solid lines) result in non-linear motions in the task space. Swept volumes of the robot in task space, obtained by following the line in configuration space, are shown in blue.

Aligned Bounding Box (AABB) and Oriented Bounding Box (OBB) [151]. A commonly used distance computation algorithm is the Gilbert-Johnson-Keerthi (GJK) method [42]. BVH structures can be used as a pre-processing step for GJK. Instead of checking all pairs of objects directly, BVH quickly eliminates unlikely collisions and selects the close pairs for further refinement. GJK is implemented as the default solver for both collision checking and distance computation in widely used collision-checking libraries, such as FCL [119] and Coal [120]. Using the Coal implementation, both binary and distance computations via GJK can be performed in 3 μs, with no significant differences observed [120]. Therefore, no further distinction between these two categories will be made in this thesis.

3.1.3 Standard Edge Examination

Edge examination can be performed in two ways: discrete and continuous examination. Discrete examination checks validity along the edge at a fixed resolution, which can lead to false positives. While continuous examination does not have such a problem, it is more computationally expensive.

Discrete examination is not performed sequentially from one end to the other. Instead, given valid edge endpoints, the examination begins at the midpoint of the edge. This approach originates from the discovery that the longer edge has a higher chance of colliding and the collisions are more likely to occur near the midpoint of the edge, as it is the most



(a) Discrete checking **(b)** Continuous checking

Figure 3.3: Illustration of discrete and continuous examination. Discrete examination verifies the middle point of the segments for better efficiency, where the green cubes indicate the examination points. When an edge is verified to be valid using discrete checking, it is still possible that a segment between two cubes is invalid, while the continuous checking in (b) does not have such a problem.

uncertain region if endpoints are verified collision-free [161]. The collision checking in MoveIt! [23] and VAMP [162] follow a similar concept. A visualization of discrete and continuous examination methods is provided in Figure 3.3.

3.2 Problem Description

A motion planning problem for manipulators can be defined by the start robot configuration $\boldsymbol{q}_s$, the goal configuration $\boldsymbol{q}_g$, and the surrounding environment $\mathcal{S} = \{\mathcal{S}_s, \mathcal{S}_d\}$. The surrounding environment is limited to objects that are within reach of the robot manipulator. It is composed of the static environment $\mathcal{S}_s$ and the dynamic environment $\mathcal{S}_d$, where $\mathcal{S}_s$ refers to the objects whose states remain constant over time and $\mathcal{S}_d$ contains all other objects in $\mathcal{S}$ that do not belong to $\mathcal{S}_s$.

To enable rapid planning in dynamic environments, a graph-like roadmap $\mathcal{G}$ is used to cover and represent the collision-free space of the static environment. During the online planning phase, the start and goal configurations, i.e., $\boldsymbol{q}_s$ and $\boldsymbol{q}_g$, are attached to $\mathcal{G}$, and then a collision-free path $\boldsymbol{q}_{1:N} = \{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_N\}$ considering $\mathcal{S}_d$ is searched within $\mathcal{G}$, subject to constraints $\boldsymbol{q}_1 = \boldsymbol{q}_s$ and $\boldsymbol{q}_N = \boldsymbol{q}_g$. Higher-order boundary conditions, such as velocity and acceleration, are not considered in this path-planning problem but can be addressed through online trajectory-generation algorithms [85]. The objective is to minimize the total number of distance computations, and collision checks n_{cc}, thereby reducing the planning time t_p.

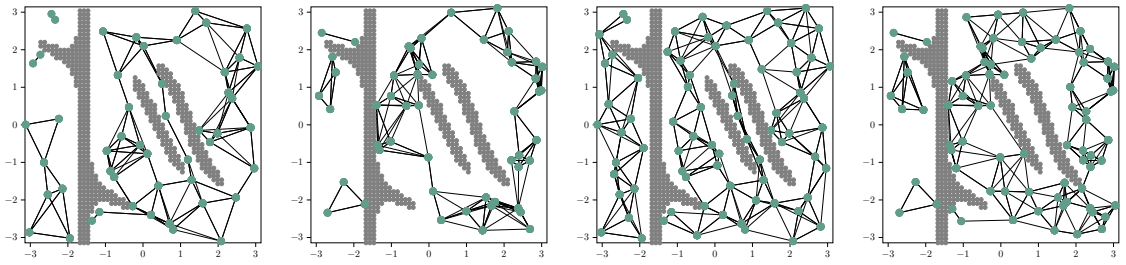
3.3 Precomputed Deterministic Roadmap

In the offline phase, a graph-like roadmap[2] $\mathcal{G}$ is constructed to represent the collision-free space of the static environment. The resulting roadmap is the foundation for online pathfinding. Ideally, the roadmap should provide a solution path for any arbitrary planning

[2]The term roadmap follows the convention of probabilistic roadmaps. It is interchangeable with the graph in this chapter.

request, if one exists. This requires that the roadmap $\mathcal{G}$ covers the high-dimensional configuration and avoids disconnected subgraphs. A dense graph can fulfill this requirement. However, dense graphs can lead to increased computation and, thereby, unnecessarily long planning time. Hence, the roadmap must achieve sufficient coverage using fewer samples where possible.

A deterministic roadmap uses quasi-random sequences such as the Halton sequence [46] to replace the uniform sampler. Halton sequences generate numbers in the interval $(0, 1)$ using prime numbers as bases. Halton sequences require fewer samples to cover the configuration space than uniform sampling. The difference between the Halton sequences and the uniform sampling is shown in Figure 3.4. Specifically in this example, Halton sequences cover most of the configuration space of a 2-DoF robot effectively with only 50 well-distributed samples. Halton sequences are increasingly popular for trajectory optimization tasks [12, 159].



(a) Halton-50 **(b)** Uniform-50 **(c)** Halton-75 **(d)** Uniform-75

Figure 3.4: Roadmaps using Halton sequences and uniform sampling with 50 and 75 samples, respectively. To emphasize the advantages of Halton sequences, three circular obstacles, same as in Figure 3.2, are placed as **static** environments. Roadmap (a) with 50 samples using the Halton sequence shows better coverage than roadmap (d) with uniformly sampled 75 samples.

Roadmaps are determined by three hyperparameters: the selection of prime numbers for the random seed, the number of connecting neighbors, and the maximum connection length. The random seeds determine the sampled nodes in the graph, and the other two hyperparameters rule the connections among nodes. The key difference between deterministic roadmaps and uniformly sampled roadmaps lies in the sampled nodes. The connectivity-related hyperparameters are the same in both roadmaps. Therefore, no further discussion regarding the optimization of the number of connecting neighbors and the maximum length of connections will be made in this thesis.

A precomputed roadmap based on Halton sequences underpins the informed search introduced in Section 3.4. Useful heuristics significantly improve the efficiency of graph searches, especially when the edge evaluations are computationally expensive. Using admissible heuristics[3], algorithms such as A* [47] have proven to be much more efficient than the Dijkstra algorithm [28] [4]. The core benefit of using precomputed roadmaps is that they provide actual path costs regarding the static environment without additional costly computations.

The second advantage of using deterministic roadmaps is their ability to cover the free configuration space. In dynamic environments, roadmaps must cover the collision-free

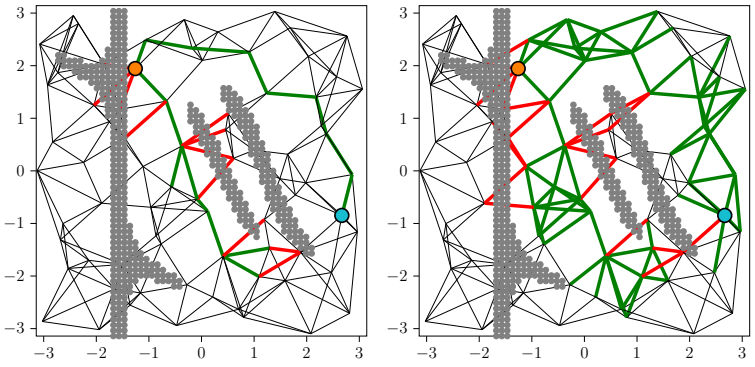
[3] Heuristics are called admissible if they don't overestimate the cost.
[4] A* applies admissible heuristics on top of the Dijkstra algorithm.

space with low discrepancy. A lower discrepancy indicates that the samples cluster less and cover the space more homogeneously. This property is also desired in optimization-based methods [12, 159]. As shown in Figure 3.4, Halton sequences achieve better coverage with fewer samples compared to uniform sampling, reducing graph complexity while maintaining low discrepancy. Although adding more samples reduces the discrepancy, they also increase search complexity. This is especially important in high-dimensional configuration space [5].

3.4 Heuristics-Informed Search

After a deterministic roadmap is computed offline, an informed search is conducted on the roadmap to find feasible paths between $\boldsymbol{q}_s$ and $\boldsymbol{q}_g$ online while optimizing the number of examined edges n_{ee}. The resulting paths are guaranteed to be collision-free and feasible with respect to both static and dynamic environmental constraints. Optimizing n_{ee} reduces computational resources during planning, thereby reducing the planning time.

Informed graph search is essentially a variant of A* search. A visualization of both methods is shown in Figure 3.5. The key difference is the heuristics used for the exploration. The informed graph search comprises two tree structures, i. e., a search tree $\mathcal{T}_S$ and a heuristic tree $\mathcal{T}_H$. Admissible heuristics obtained from $\mathcal{T}_H$ help the search tree $\mathcal{T}_S$ prioritize the edge examination. As the exploration and examination proceed, more information regarding the current state of the environment becomes accessible. Based on this information, the heuristics tree $\mathcal{T}_H$ will be updated. This section first introduces the heuristics used for searching the precomputed roadmap and then explains the details of the prioritized search based on these heuristics.



(a) Heuristics-informed search **(b)** A* using L2 distance

Figure 3.5: Comparison of exploration using heuristics informed search and A* using L2 distance as heuristics on a roadmap with 75 nodes. The start node is marked in orange, and the goal node is marked in cyan. The obstacles shown in gray were unknown at the time the roadmap was generated. The edges examined during the search are color-coded: red indicates invalid edges, and green signifies valid edges. Both search methods yield the same solution path. While only 27 edges have been examined in (a), 94 edge examinations are needed in (b).

[5] A thorough comparison of the standard probabilistic roadmap and deterministic roadmap can be found in literature [14, 66].

3.4.1 Heuristics for Roadmap Exploration

The proposed informed graph search uses the heuristic tree $\mathcal{T}_H$ to compute the path cost in the precomputed roadmap of the static environment as heuristics. The $\mathcal{T}_H$ is initialized with a root node at the goal region and expands to the start configuration on the graph with the A* algorithm *without* conducting collision checking. The objective is to find the shortest path from the goal to the start configuration in the graph. As unforeseen obstacles appear in the workspace, a feasible path between the start and goal configurations will not become shorter. Because dynamic environmental conditions can only increase path cost, the path cost in the static environment serves as a lower bound for the path cost in dynamic environments. Hence, they are admissible heuristics.

Slightly different from A*, where the heuristics are assigned to nodes, the heuristics in this proposed method are assigned to edges. Assigning heuristics to edges allows the priority queue $\mathcal{Q}$ to maintain edge rankings rather than node rankings. This avoids expanding all edges of a node, which entails expensive collision checking, thereby reducing computational cost. However, this does not affect the computation of heuristics. For each edge, the heuristics

$$h_e = \{n_{e,reach}, c_{e,reach}\}. \tag{3.1}$$

are composed of the number of edges to reach the goal $n_{e,reach}$ and the length of the shortest connection to the goal $c_{e,reach}$. These two values describe the effort to reach the goal and can be used to prioritize the exploration of $\mathcal{T}_S$.

It is worth noting that running A* multiple times can lead to substantial time consumption as the number of runs increases, even though collision checking is not involved in this step. Therefore, instead of repeating the A* algorithm from scratch, $\mathcal{T}_H$ grows and updates incrementally to provide heuristics for the other nodes.

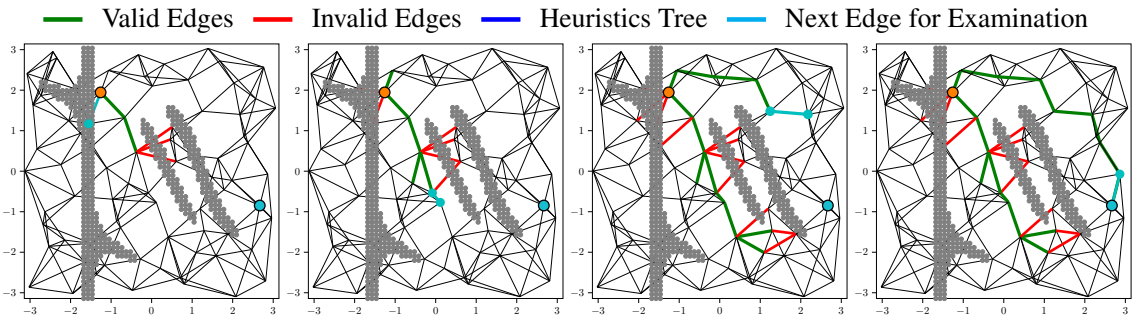
3.4.2 Informed Search

The informed graph search comprises two tree structures, i.e., a search tree $\mathcal{T}_S$ and a heuristic tree $\mathcal{T}_H$. Given a planning problem with the start $\boldsymbol{q}_s$ and the goal $\boldsymbol{q}_g$, two corresponding nodes v_s and v_g are added to the precomputed roadmap $\mathcal{G}$. The search tree $\mathcal{T}_S$ explores from v_s, and the $\mathcal{T}_H$ uses v_g as its root node.

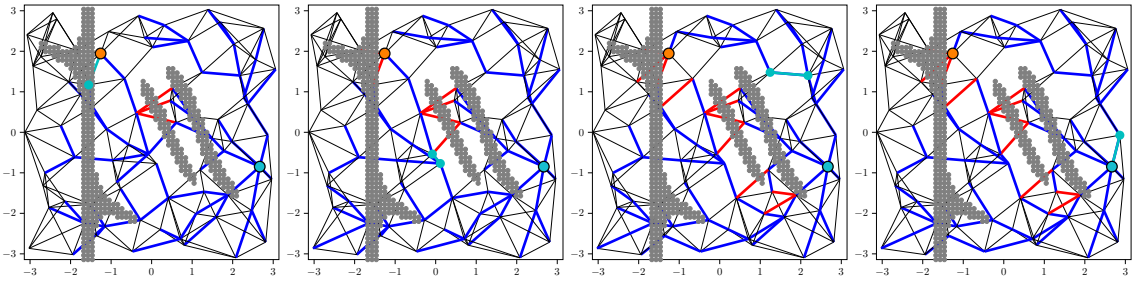
With the heuristics h_e provided by $\mathcal{T}_H$, the search tree $\mathcal{T}_S$ explores toward the goal configuration v_g. The exploration is an iterative process, which maintains a priority queue $\mathcal{Q}$ of edges and takes an edge from $\mathcal{Q}$ for examination at each iteration. The exploration is prioritized by the ranked f-score in $\mathcal{Q}$

$$f_e = \{n_{e,reach}, c_{e,reach} + c_{e,come}\}, \tag{3.2}$$

where $c_{e,come}$ indicates the cost-to-come traveling from v_s to this edge e in $\mathcal{T}_S$ and $n_{e,reach}$, $c_{e,reach}$ are the heuristics introduced in Eq. (3.1). When f_e is put in the priority queue $\mathcal{Q}$, the edges are ranked in lexicographic order. That is, the edges are first ranked by $n_{e,reach}$ and then by $c_{e,reach} + c_{e,come}$. A smaller number indicates a higher priority and gets out of the queue earlier. The number $n_{e,reach}$ indicates the surrogate efforts to reach the final



(a) Exploration of the search tree $\mathcal{T}_S$



(b) Update of the heuristic tree $\mathcal{T}_H$

Figure 3.6: Snapshots of a heuristics-informed search. The orange node represents the start point, while the cyan node represents the goal. The gray obstacles were **unknown** when constructing the roadmap. The heuristic tree $\mathcal{T}_H$ rewires and updates over time. In both (a) and (b), the edge with the best-estimated cost in $\mathcal{T}_H$ is marked in cyan, indicating which edge will be examined next. Note that the heuristics tree updates in the second and third plots as the search progresses.

goal, and $c_{e,reach} + c_{e,come}$ refers to the approximate path cost between the start and goal using this edge as a via point. Put differently, the exploration first considers the edges that need less effort to reach the goal. If multiple edges indicate the same effort, the one with the lowest estimated path cost, $c_{e,reach} + c_{e,come}$, will be selected for examination using safe zones. A visualization is shown in Figure 3.10. Due to the concept of safe zones introduced in section 3.5, collision checking is not conducted with fixed resolution along the edge and, therefore, is decoupled from $c_{e,reach}$. The number $n_{e,reach}$ is a better approximation of the effort. If the edge is considered valid in the iteration, the search tree $\mathcal{T}_S$ expands the child node v_c of this edge. During exploration, the heuristics h_e in $\mathcal{T}_H$ are updated based on the currently available knowledge in the environment and influence the ranking in $\mathcal{Q}$. The exploration iterates until the search tree $\mathcal{T}_S$ reaches the goal region or the priority queue is empty. Details regarding the exploration of $\mathcal{T}_S$ and updates of $\mathcal{T}_H$ are described in the following.

Algorithm 1 depicts a high-level procedure of the heuristics-informed search. It starts with initializing the search tree $\mathcal{T}_S$ at the start node v_s, the heuristics tree $\mathcal{T}_H$ at the goal node v_g, and the priority queue $\mathcal{Q}$ (Algorithm 1, line 1,2). Additionally, an empty closed set $\mathcal{S}_{closed}$ is declared. The closed set includes the three kinds of nodes: (1) the nodes that have been expanded and added to $\mathcal{T}_S$; (2) the nodes that are not valid regarding the current state of the environment; and (3) the nodes that don't have any chance to be part of the resulting path, e.g., all outgoing edges are not valid.

After the initialization, $\mathcal{T}_S$ expands the start node v_s and pushes all edges connected to v_s to the priority queue $\mathcal{Q}$ without performing any collision checking (Algorithm 1, line 3). During the expansion described in Algorithm 2, $\mathcal{T}_H$ is responsible for providing heuristics to rank the edges in $\mathcal{Q}$. Then, the search iterates until the problem is solved or the termination conditions are met (Algorithm 1, lines 4 - 26). The loop terminates when the priority queue is empty or a solution is found. Other early-termination conditions, such as the maximum number of iterations or the maximum computation time, can also be used. Within the iteration loop, the priority queue $\mathcal{Q}$ automatically sorts the edges according to the f_e. For every iteration, $\mathcal{Q}$ pops out the edge e with the best f_e (Algorithm 1, line 5). The best f_e refers to the lowest $n_{e,reach}$ and $c_{e,reach} + c_{e,come}$. If the child node v_c of the edge e is already in the closed set $\mathcal{S}_{closed}$, it indicates that v_c either has already been expanded or is confirmed to be impossible to be on the final path. Hence, there is no need to expand or examine this node. In this case, the iteration moves on and skips this edge (Algorithm 1, lines 8-10). Otherwise, if v_c is not in $\mathcal{S}_{closed}$, we first examine whether v_c is valid regarding the dynamic environment (Algorithm 1, line 11) and then examine the edge e (Algorithm 1, line 16). If v_c is not valid, it indicates that all nodes having v_c as the predecessor node in $\mathcal{T}_H$ need to find a new predecessor as shown in Algorithm 4. By finding new predecessors, $\mathcal{T}_H$ and the heuristics in $\mathcal{T}_H$ are updated accordingly. After the updates, the iteration continues from the beginning and considers the next best edge (Algorithm 1, line 14). With v_c being valid, the edge e is examined using safe zones (Algorithm 1, line 16), which is the most expensive step of the algorithm. Upon the edge being valid, we expand the child node v_c further (Algorithm 1, line 20) as described in Algorithm 2. Otherwise, the heuristics need to be updated.

As described in Algorithm 2, expanding a node v is computing the f-scores for outgoing edges of v and pushing them to the priority queue $\mathcal{Q}$. The outgoing edges are those whose nodes are not in $\mathcal{S}_{closed}$. The f-scores are composed of $c_{e,come}$ and the heuristics. The computation of the heuristics is shown in Algorithm 3. It is worth noting that the heuristics are computed using dynamic programming as $\mathcal{T}_H$ grows, and they are stored as properties on edges and nodes.

Once a node or an edge has been examined as invalid, the heuristic tree must be updated (1, line 13, 24). The update step described in Algorithm 4 takes all the neighbors $v_{\{nns\}}$ of the input node v in $\mathcal{T}_H$ and selects the one with the best-estimated cost. If no neighbor is connected to $\mathcal{T}_H$, $\mathcal{T}_H$ grows toward v_c and finally updates the heuristics. If no valid neighbors exist in the roadmap, this node will be added to the closed set and will never be visited again. Updating the heuristics guides the search to avoid spending time on edge evaluations that do not contribute to the result, given the current knowledge of the environment. One key difference between our method and LazyPRM is that we update the heuristic rather than the roadmap when new environmental information becomes available.

Finally, upon successful termination of the search, the graph search yields a set of waypoints as described in Algorithm 5. So far, the waypoints are not executable for robots yet. An additional module is needed to convert the waypoints to a trajectory with velocity profiles and time stamps of fixed resolution [126, 89]. Because the trajectory does not exactly follow the edges and waypoints [89], a backtracking and correction strategy is necessary to ensure safety. As the first step of the correction, the signed distances of the contact points are used to recover the invalid trajectory to a valid configuration. If no appropriate correction is found, the edge caused by this problem will be deactivated, and

we will seek an alternative solution. Only a few new edges should be searched since the invalid edges and nodes have already been deactivated.

Algorithm 1: Informed Graph Search

$\mathcal{T}_S \leftarrow v_s$, $\mathcal{T}_H \leftarrow v_g$
$\mathcal{S}_{closed} \leftarrow \{v_s\}$, $\mathcal{Q} \leftarrow [\,]$, $solved \leftarrow$ False
Expand(v_s, $\mathcal{G}$, $\mathcal{Q}$, $\mathcal{T}_S$, $\mathcal{T}_H$)
while !$solved \wedge$!$\mathcal{Q}$.empty() **do**
  $e \leftarrow \mathcal{Q}$.pop() // Get prioritized edge
  $v_p \leftarrow$ getParentNode(e)
  $v_c \leftarrow$ getChildNode(e)
  if $v_c \in \mathcal{S}_{closed}$ **then**
    Continue // No value to expand
  end
  if !isNodeValid(v_c) **then**
    $\mathcal{S}_{closed} \leftarrow \{\mathcal{S}_{closed}\} \cup \{v_c\}$
    updateHeuristics(v_c, $\mathcal{G}$, $\mathcal{T}_H$)
    Continue;
  end
  if isEdgeValid(e) **then**
    if inGoalRegion(v_c) **then**
      $solved \leftarrow$ True;
    end
    Expand(v_c, $\mathcal{G}$, $\mathcal{Q}$, $\mathcal{T}_S$, $\mathcal{T}_H$)
    v_c.setPredecessor(v_p, $\mathcal{T}_S$)
    $\mathcal{S}_{closed} \leftarrow \{\mathcal{S}_{closed}\} \cup \{v_c\}$
  else
    updateHeuristics(v_c, $\mathcal{G}$, $\mathcal{T}_H$)
  end
end
return getSolution(v_s, v_g, $\mathcal{T}_S$)

Algorithm 2: Expand

Input: v, $\mathcal{G}$, $\mathcal{Q}$, $\mathcal{T}_S$, $\mathcal{T}_H$
for *each* e in getOutEdges($v, \mathcal{G}$) **do**
  $n_e, c_{e,reach} \leftarrow$ getHeuristics(e, $\mathcal{T}_H$)
  $c_{e,come} \leftarrow$ getDistToRoot(e, $\mathcal{T}_S$)
  $f_e \leftarrow$ getFScore(n_e, $c_{e,\{reach,come\}}$)
  $\mathcal{Q}$.push(e, f_e)
end

Algorithm 3: getHeuristics

Input: e, $\mathcal{G}$, $\mathcal{T}_H$
$v_c \leftarrow$ getChildNode()
if !isInTree(v_c, $\mathcal{T}_H$) **then**
  growHeuristicsTree(v_c, $\mathcal{G}$, $\mathcal{T}_H$)
end
$d \leftarrow$ getDistToRoot(v_c, $\mathcal{T}_H$) + e.length()
$n \leftarrow$ getEdgeCountsToRoot(v_c, $\mathcal{T}_H$)
return n, d

Algorithm 4: updateHeuristics

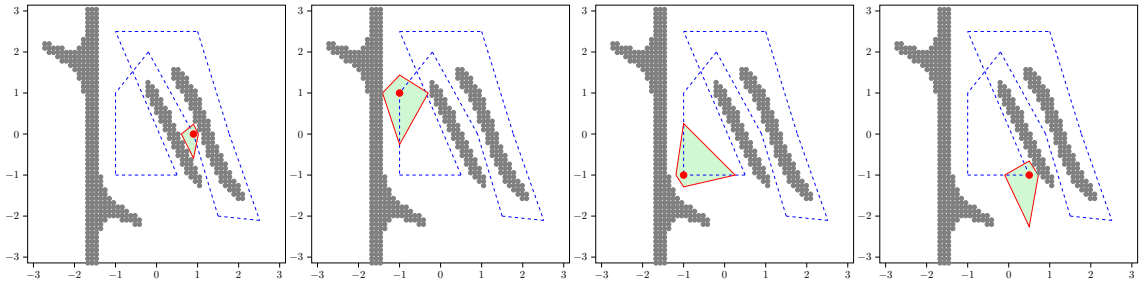
Input: v, $\mathcal{G}$, $\mathcal{T}_H$
for *each* v_n in getSuccessors($v, \mathcal{T}_H$) **do**
  if !isInTree(v_c, $\mathcal{T}_H$) **then**
    growHeuristicsTree(v_n, $\mathcal{G}$, $\mathcal{T}_H$)
  end
  $v_{\{nns\}} \leftarrow$ getNeighbours($v_n, \mathcal{G}, \mathcal{T}_H$)
  $n_{\{nns\}}, d_{\{nns\}} \leftarrow$ getHeuristics($\{v_{nns}\}, \mathcal{G}, \mathcal{T}_H$)
  $v_{best} \leftarrow \arg\min_{nn}(d_{\{nns\}})$ // Best heuristics
  v_n.setPredecessor(v_{best}, $\mathcal{T}_H$)
end

Algorithm 5: getSolution

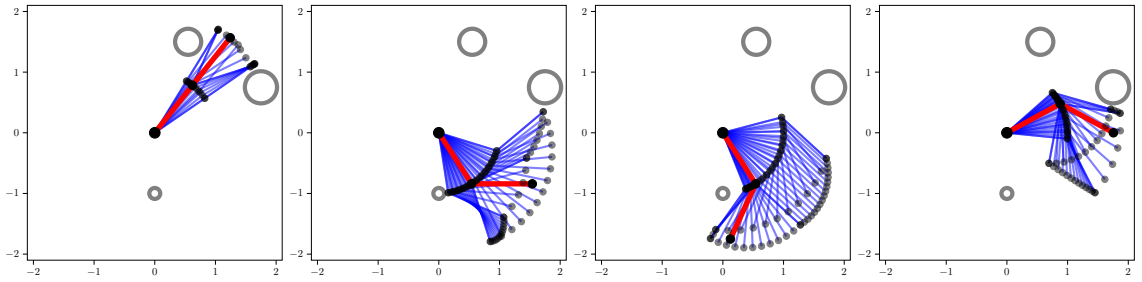
Input: v_s, v_g, $\mathcal{T}_S$
$path \leftarrow [\,]$, $v \leftarrow v_g$
while $v! = v_s \wedge v$.hasPredecessor($\mathcal{T}_S$) **do**
  $path$.append(v)
  $v \leftarrow v$.getPredecessor($\mathcal{T}_S$)
end
$path$.revert()
return $path$ // Return empty list if problem not solved

3.5 Edge Examination with Safe Zones

In sampling-based motion planning, samples are usually connected using straight lines in the configuration space. These straight lines are also termed edges. While constructing a solution, edges must be checked to ensure they are collision-free with respect to the



(a) Configuration space



(b) Task space

Figure 3.7: Safe zones of a 2-DoF robot with respect to three circular obstacles. The green polygons in (a) indicate the safe zones around specific configurations marked by red dots. The safe zones correspond to the blue swept volumes in (b). The red robots in (b) represent the specific configurations in (a).

current state of the environment. Edge examination involves numerous collision checks and distance computations, making it computationally expensive. The core idea of edge examination with safe zones is to outline the region where no collision checks are needed and allocate the computational resources elsewhere during the edge examination, e. g., the regions we are not sure whether they are collision-free or not.

While the standard collision-checking pipeline is sensitive to the choice of checking resolution, safe zones explicitly determine a collision region around the joint configuration of interest. Similar to collision checking, safe zones are computed using the distance information between the robot and the environment. The shape and size of the safe zones adapt according to the distance information. For a given robot joint configuration, the safe zone grows as the distance between the robot and the obstacle increases. The outcome is a flexible checking resolution. The following describes the mathematical details for deriving safe zones and their application to edge examination.

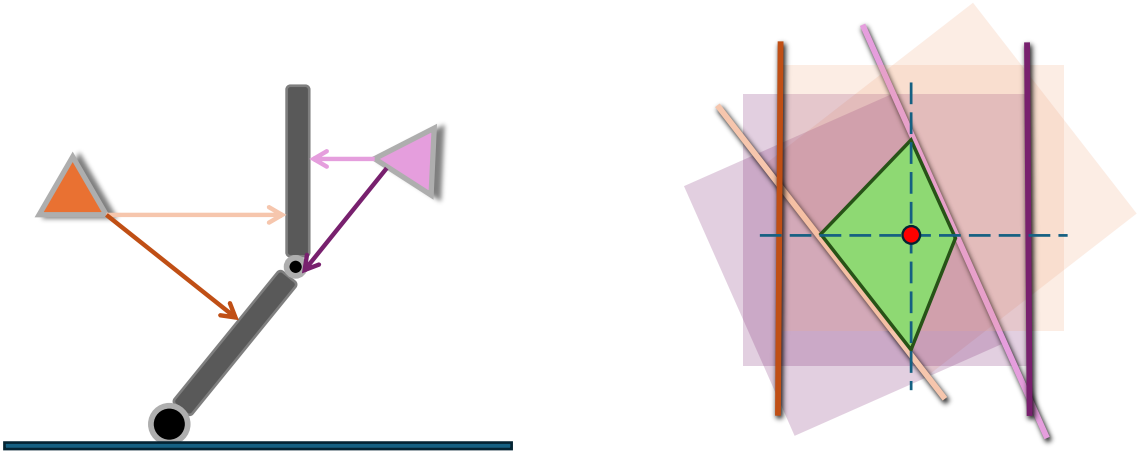
3.5.1 Safe Zones

Formally, a safe zone is a closed continuous volume constructed by multiple hyperplanes. Because of its continuity, no further collision checks are needed within the safe zone, which does not exhibit the false-positive problem of the pipeline with discrete resolutions.

Given a N-dimensional collision-free robot joint configuration $\boldsymbol{q}_0 \in \mathbb{R}^N$, its safe zone is a closed non-symmetric volume around this configuration. All configurations within the safe zone are collision-free regarding the dynamic environment[6]. In a two-dimensional

[6]The configurations of interest lie on the precomputed roadmap. Therefore, no consideration regarding the static environment is required.

configuration space, safe zones have a form of quadrilaterals as shown in Figure 3.7 [7]. The quadrilateral is defined by two sets of $\{q^-_{[\cdot]}, q^+_{[\cdot]}\}$, indicating the lower and upper bounds of the movement along the two joints, respectively. In Figure 3.8b, $\boldsymbol{q}_0$ is marked in red and $\{q^-_{[\cdot]}, q^+_{[\cdot]}\}$ refers to the four corner points of the quadrilateral, i. e., $[q^+_0, 0], [q^-_0, 0], [0, q^+_1]$ and $[0, q^-_1]$. As the degrees of freedom grow to N, the safe zone is an N-D cross-polytope defined by N sets of lower and upper bounds. The bounds are determined by constraints derived from the spatial relations between robot links and obstacles in the task space; see Figure 3.8a. Note that the safe zone is a subset of the collision-free configurations around $\boldsymbol{q}_0$, and this thesis does not try to identify all collision-free configurations. The following presents details on establishing the constraints.



(a) Task space **(b)** Configuration space

Figure 3.8: Illustration of computing safe zones for a 2-DoF robot with two triangular obstacles. The colored distance in (a) corresponds to the hyperplanes in the same color in (b). The green polygon in (b) represents the safe zone of this particular robot configuration.

Determining the safe zone for a configuration $\boldsymbol{q}_0$ starts by constraining the robot joints' motion in certain directions. The goal is to find a set of configurations $\boldsymbol{q}_{sz}$ around the joint configuration $\boldsymbol{q}_0$ that is collision-free, illustrated in Figure 3.8. Consider the closest point $C_{j,i}$ between the j-th link of the robot and the i-th obstacle in the environment. The closest distance vector is defined as $\boldsymbol{d}_{C_{j,i}} \in \mathbb{R}^3$. For a safe movement at $\boldsymbol{q}_0$ regarding the i-th obstacle, any point on the j-th link should not travel a distance larger than the shortest distance $||\boldsymbol{d}_{C_{j,i}}||_2$, which is the L2-norm of the closest distance vector $\boldsymbol{d}_{j,i}$, highlighted by the colored vector in Figure 3.8a.

Lemma 3.5.1. *Consider an articulated manipulator with revolute joints at a joint configuration $\boldsymbol{q}_0$. Suppose the distance between the j-th link and the i-th obstacle is given by $||\boldsymbol{d}_{C_{j,i}}||_2$. If a joint movement $\Delta\boldsymbol{q}$ driving the robot $\boldsymbol{q}_0$ to $\boldsymbol{q}_0 + \Delta\boldsymbol{q}$ results in the maximum displacement of any point on the j-th link in the task space to be strictly less than $||\boldsymbol{d}_{C_{j,i}}||_2$, then it is sufficient to conclude that the j-th link will remain collision-free with respect to the i-th obstacle after the movement.*

[7] An animation of safe zones in a two-dimensional configuration, see `https://xi-hhhm.github.io/roboX.huang/images/HIRO_2d_volume_animation_arm.gif`

Proof. Let the point on the j-th link closest to the i-th obstacle at configuration $\boldsymbol{q}_0$ be denoted as $C_{j,i}$, with distance $\|\boldsymbol{d}_{C_{j,i}}\|_2$. After the joint moves from $\boldsymbol{q}_0$ to $\boldsymbol{q}_0 + \Delta\boldsymbol{q}$, let the new closest point on the link to the obstacle be $C'_{j,i}$.

If a collision occurs, then $C'_{j,i}$ must lie on the surface of the i-th obstacle with $\|\boldsymbol{d}_{C'_{j,i}}\|_2 = 0$. In this case, the displacement of the point on the link that reaches the obstacle must be at least $\|\boldsymbol{d}_{C_{j,i}}\|_2$, since it started at that distance from the obstacle. However, by assumption, the maximum displacement of any point on the j-th link in task space is strictly less than $\|\boldsymbol{d}_{C_{j,i}}\|_2$. Therefore, no point on the j-th link can reach the surface of the obstacle, and a collision cannot occur. Hence, the j-th link remains collision-free with respect to the i-th obstacle after the joint movement. □

Another way to interpret Lemma 3.5.1 is to accumulate the instantaneous movement starting at $\boldsymbol{q}_0$ and establish the inequality

$$||\boldsymbol{d}_{C_{j,i}}||_2 \geq \int_{\boldsymbol{q}_0}^{\boldsymbol{q}_0+\Delta\boldsymbol{q}} ||\boldsymbol{J}_{C_{j,i}}(\boldsymbol{q})\,\mathrm{d}\boldsymbol{q}||_2, \tag{3.3}$$

where $\boldsymbol{J}_{C_{j,i}}(\boldsymbol{q}) \in \mathbb{R}^{3\times N}$ is a partial Jacobian matrix and $\boldsymbol{J}_{C_{j,i}}(\boldsymbol{q})\,\mathrm{d}\boldsymbol{q}$ represents the instantaneous movement at $C_{j,i}$ in task space. For a manipulator with revolute joints, the mapping from the joint velocity $\dot{\boldsymbol{q}} \in \mathbb{R}^N$ to the velocity at a point $\boldsymbol{x} \in \mathbb{R}^3$ on the robot is nonlinear. The Jacobian matrix $\boldsymbol{J}_{C_{j,i}}(\boldsymbol{q})$ in Eq. (3.3) is a linearization of the mapping at $\boldsymbol{q}$ with

$$\boldsymbol{v}_{C_{j,i}}(\boldsymbol{q}) = \boldsymbol{J}_{C_{j,i}}(\boldsymbol{q})\dot{\boldsymbol{q}}. \tag{3.4}$$

By finding the local maximum along the integral path $||\boldsymbol{J}_{C_{j,i},max}\,\mathrm{d}\boldsymbol{q}||_2 \geq ||\boldsymbol{J}_{C_{j,i}}(\boldsymbol{q})\,\mathrm{d}\boldsymbol{q}||_2$, $\forall\boldsymbol{q} \in [\boldsymbol{q}_0, \boldsymbol{q}_0 + \Delta\boldsymbol{q}]$, we can have

$$||\boldsymbol{J}_{C_{j,i},max}\Delta\boldsymbol{q}||_2 = \int_{\boldsymbol{q}_0}^{\boldsymbol{q}_0+\Delta\boldsymbol{q}} ||\boldsymbol{J}_{C_{j,i},max}\,\mathrm{d}\boldsymbol{q}||_2 \geq \int_{\boldsymbol{q}_0}^{\boldsymbol{q}_0+\Delta\boldsymbol{q}} ||\boldsymbol{J}_{C_{j,i}}(\boldsymbol{q})\,\mathrm{d}\boldsymbol{q}||_2. \tag{3.5}$$

In the remainder of this chapter, the Jacobian matrix $\boldsymbol{J}_{C_{j,i},max}$, which results in the local maximum $||\boldsymbol{J}_{C_{j,i},max}\,\mathrm{d}\boldsymbol{q}||_2 \geq ||\boldsymbol{J}_{C_{j,i}}(\boldsymbol{q})\,\mathrm{d}\boldsymbol{q}||_2$, $\forall\boldsymbol{q} \in [\boldsymbol{q}_0, \boldsymbol{q}_0 + \Delta\boldsymbol{q}]$, is termed as *maximum Jacobian*. The maximum Jacobian links the motion of each joint to the maximum possible motion in the task space. Combining Eq. (3.3) and Eq. (3.5), any movement $\Delta\boldsymbol{q}$ fulfilling

$$||\boldsymbol{d}_{j,i}||_2 \geq ||\boldsymbol{J}_{C_{j,i},max}\Delta\boldsymbol{q}||_2 \tag{3.6}$$

is safe according to Lemma 3.5.1, since the largest possible movement is less than the current closest distance $||\boldsymbol{d}_{j,i}||_2$. The inequality expressed in Eq. (3.6) is the basis for deriving the final expression for safe zones. The key is determining the local maximum $\boldsymbol{J}_{C_{j,i},max}$. Without losing the generality, the inequality Eq. (3.6) can be rewritten by multiplying a rotation matrix $\boldsymbol{R} \in \mathbb{R}^{3\times3}$,

$$||\boldsymbol{R}\boldsymbol{d}_{j,i}||_2 \geq ||\boldsymbol{R}\boldsymbol{J}_{C_{j,i},max}\Delta\boldsymbol{q}||_2, \tag{3.7}$$

with

$$\boldsymbol{R}\boldsymbol{J}_{C_{j,i},max}\Delta\boldsymbol{q} = \begin{bmatrix} 0 \\ 0 \\ ||\boldsymbol{J}_{C_{j,i},max}\Delta\boldsymbol{q}||_2 \end{bmatrix}. \tag{3.8}$$

Take an articulated manipulator with 4 DoFs in a 3-dimensional task space as an example, the rotation matrix $\boldsymbol{R}$ converts the inequality Eq. (3.7) into the following form

$$\boldsymbol{R}\boldsymbol{J}_{C_{j,i},max}\Delta\boldsymbol{q} = \begin{bmatrix} J_{R,x,0} & J_{R,x,1} & J_{R,x,2} & J_{R,x,3} \\ J_{R,y,0} & J_{R,y,1} & J_{R,y,2} & J_{R,y,3} \\ J_{RC_{j,i},max,0} & J_{RC_{j,i},max,1} & J_{RC_{j,i},max,2} & J_{RC_{j,i},max,3}, \end{bmatrix} \begin{bmatrix} \Delta q_0 \\ \Delta q_1 \\ \Delta q_2 \\ \Delta q_3 \end{bmatrix} \tag{3.9}$$

The subscript $[\cdot]_R$ indicates that the multiplication with a rotation matrix compared to Eq. (3.6). This rewritten form simplifies the matrix multiplication to $\sum_{k=0}^{3} J_{RC_{j,i},max,k}\Delta q_k$, with $\sum_{k=0}^{3} J_{R_{x,k}}\Delta q_k = 0$ and $\sum_{k=0}^{3} J_{R_{y,k}}\Delta q_k = 0$. The values of $J_{R_{x,k}}$ and $J_{R_{y,k}}$ do not contribute to the computation of safe zones and do not need to be calculated. The constraint in Eq. (3.6) becomes

$$||\boldsymbol{d}_{j,i}||_2 \geq |\sum_{k=0}^{N-1} J_{RC_{j,i},max,k}\Delta q_k|. \tag{3.10}$$

The L2-norm operation is replaced by the absolute operator $|\cdot|$ on the right-hand side of the inequality since the right-hand side indicates only a sum of scalar multiplications. Applying $|a||b| \geq |ab|$ and $|a| + |b| \geq |a + b|$ to the right-hand side, it has

$$\sum_{k=0}^{N-1} |J_{RC_{j,i},max,k}||\Delta q_k| \geq |\sum_{k=0}^{N-1} J_{RC_{j,i},max,k}\Delta q_k|, \tag{3.11}$$

and the constraint in Eq. (3.10) can be made to be stricter as

$$||\boldsymbol{d}_{j,i}||_2 \geq \sum_{k=0}^{N-1} |J_{RC_{j,i},max,k}||\Delta q_k|. \tag{3.12}$$

Eq. (3.12) defines a linear programming problem. The region specified by Eq. (3.12) is a closed volume formed by hyperplanes. It restricts the motion of all joints symmetrically in both $q_{[\cdot]}^{-}$ and $q_{[\cdot]}^{+}$ directions. As shown in Figure 3.8, the constraints are lines (1D) that separate the safe and non-safe regions in a 2-dimensional space. In three-dimensional space, constraints are planes (2D), whereas in an N-dimensional space, constraints are (N-1)-dimensional hyperplanes.

The maximum Jacobian links the upper bound of task-space movements to the joint-space bounds along the integral path. We can compute the derivative of each $J_{RC_{j,i},k}(\boldsymbol{q})$ and then find the nearest local maximum. It is usually impossible to have a single joint configuration $\boldsymbol{q}$ at which all Jacobian terms $J_{RC_{j,i},k}(\boldsymbol{q})$ for $k \in \{0, ..., N-1\}$ simultaneously attain their local maxima along the integration path. Gathering the local maximal values $J_{RC_{j,i},k}(\boldsymbol{q})$ of each joint from different joint configurations and substituting them into the constraint Eq. (3.12) leads to an overly stricter bound. In practice, an approximation of the maximum Jacobian is used. Given the current joint configuration, the Jacobian of the point farthest away from the rotating joint on the link is used as the approximation of the maximum Jacobian.

Lemma 3.5.2. *For a revolute joint of an articulated manipulator at a joint configuration $\boldsymbol{q}_0$, its maximum Jacobian is attained at the point on the downstream links that has the maximum perpendicular distance to the joint's rotation axis.*

Proof. The maximum Jacobian describes the relation between a joint and the maximum possible velocity by rotating this joint at a joint configuration $\boldsymbol{q_0}$. The velocity of each point on the manipulator from this joint onward is determined by $\boldsymbol{v} = l(\boldsymbol{q}_0)\dot{\boldsymbol{q}}$ where $l(\boldsymbol{q}_0)$ denotes the distance from the joint to the point of interest at the configuration $\boldsymbol{q}_0$. Therefore, the maximum value $\boldsymbol{v}$ on the robot is located at the point where $l(\boldsymbol{q}_0)$ reaches the maximum. □

Considerations in practice: For articulated manipulators, the links typically have shapes that can be approximated by cylinders or capsules. In this case, the farthest points away from the rotating joints are located at one or both ends of the robot link. For robots with tools, especially those with special finish forms, we can patch the robot with spheres to get the maximum Jacobian. These approximations can be automated based on the robot's mesh description.

So far, the focus has been on safe joint movements regarding a single link-obstacle pair. One pair of robot links and obstacles introduces a set of linear constraints that define hyperplanes outlining the safe movement area. The overlap area of all link-obstacle pairs is the desired safe zone, an illustration of two-dimensional space, see Figure 3.8. In two-dimensional space, the hyperplanes become lines. These lines intersect with the axes at $\Delta q_{[\cdot]} = 0$. A simpler and stricter approach is to use the intersection points and reformulate the linear program. This helps ease the complexity of later edge validation. An intersecting point of a link-obstacle pair is determined by

$$\Delta q_{k_{j,i}} = \frac{||\boldsymbol{d}_{j,i}||_2}{|J_{RC_{j,i},k,max}| + \epsilon}, \tag{3.13}$$

where ϵ is a small positive value to handle the singularity. For each joint k, we select the strictest intersecting point, which has the smallest value, to reformulate the linear program. In the example shown in Figure 3.8b, the light purple line intersects the axis at a smaller value than the dark purple line; this intersection point is then used in the reformulation.

The region determined by Eq. (3.12) and Eq. (3.13) restricts the motion of every joint symmetrically in both directions, which is unnecessary. Under the assumption that convex volumes can represent the obstacles and the robot, only one side of the movement should be constrained. The following explains how to determine which side to constrain the movement.

Consider applying force along the distance direction $\boldsymbol{d}_{j,i}$ at the closest point $C_{j,i}$ between this obstacle and the j-th link of the robot, where $i, j \in \mathbb{N}$, the relation between the resultant torque $\boldsymbol{\tau} \in \mathbb{R}^N$ applied to the joints and the Cartesian force $\boldsymbol{F}_{C_{j,i}} = f_{C_{j,i}} \frac{\boldsymbol{d}_{j,i}}{||\boldsymbol{d}_{j,i}||_2} \in \mathbb{R}^3$ can be expressed with a partial Jacobian matrix $\boldsymbol{J}_{C_{j,i}}(\boldsymbol{q}_0) \in \mathbb{R}^{3\times N}$:

$$\boldsymbol{\tau} = \boldsymbol{J}^T_{C_{j,i}}(\boldsymbol{q}_0)\boldsymbol{F}_{C_{j,i}}, \tag{3.14}$$

with N denoting the DoF of the robot.

An interpretation of Eq. (3.14) is that the robot is pulled along the distance direction to approach the obstacle. Applying $\boldsymbol{\tau}$ to the joints leads to the same effect. The sign of the applied torque $\boldsymbol{\tau}$ indicates the direction of the joint movement. To avoid the robot link approaching and coming closer to the obstacle, these directions of movement should be constrained. Normalizing this equation regarding the force $\boldsymbol{F}_{C_{j,i}}$, it becomes

$$\boldsymbol{s} = \boldsymbol{J}_{C_{j,i}}^{T}(\boldsymbol{q}_0)\frac{\boldsymbol{d}_{j,i}}{||\boldsymbol{d}_{j,i}||_2}. \tag{3.15}$$

In other words, for a configuration $\boldsymbol{q}_0$, the sign of s_k in the vector $\boldsymbol{s}$ indicates the direction to restrict movements of the k-th joint to avoid a collision [8]. Given the restrictive movement $\Delta q_{k_{j,i}}$ for the joint configuration q_k caused by the spatial relation between the j-th link and the i-th obstacle, the lower bound $\Delta q^{-}_{k_{j,i}}$ or the upper bound $\Delta q^{+}_{k_{j,i}}$ is assigned according to

$$\Delta q^{-}_{k_{j,i}} = -\frac{||\boldsymbol{d}_{j,i}||_2}{|J_{RC_{j,i},k,max}| + \epsilon} \text{ if } s_k < 0, \tag{3.16}$$

$$\Delta q^{+}_{k_{j,i}} = \frac{||\boldsymbol{d}_{j,i}||_2}{|J_{RC_{j,i},k,max}| + \epsilon} \text{ if } s_k \geq 0, \tag{3.17}$$

with $\Delta q^{-}_{k_{j,i}}$ and $\Delta q^{+}_{k_{j,i}}$ denoting the lower and upper intersection point regarding the j^{th}-link-i^{th}-obstacle pair. The strictest intersection point is used for reforming linear programming problem

$$\Delta q^{-}_{k} = \max_{j,i}\{\Delta q^{-}_{k_{j,i}}\} \tag{3.18}$$

$$\Delta q^{+}_{k} = \min_{j,i}\{\Delta q^{+}_{k_{j,i}}\}. \tag{3.19}$$

Considerations in practice: Every joint should have Δq^{-}_{k} and Δq^{+}_{k} defined. If one or both are not defined due to the current robot and environment settings, a constant such as $\frac{\pi}{2}$ is used to keep the safe zone conservative and less aggressive.

Using the strictest intersection points, a set of linear programming problems is defined by

$$\sum_{k=0}^{N-1} a_k^{[\cdot]}\Delta q_k^{[\cdot]} + b^{[\cdot]} = 0, \tag{3.20}$$

where $[\cdot]$ indicates the combination of the restrictive direction of the active hyperplane. For a 2-DoF robot, there are four possible combinations for $[\cdot]$, i.e., $\{+,+\}$, $\{+,-\}$, $\{-,+\}$, and $\{-,-\}$. The combination $\{+,+\}$ indicates that Δq_0^{+} and Δq_1^{+} are used to form the linear programming problem Eq. (3.20) to compute the coefficients $a_k^{[\cdot]}$ and $b^{[\cdot]}$ that defines the hyperplanes. The hyperplanes form a closed volume in the configuration space that can be used as a safe zone. After solving for $a_k^{[\cdot]}$ and $b^{[\cdot]}$ using

[8] Similar formalism can also be found in literature [36].

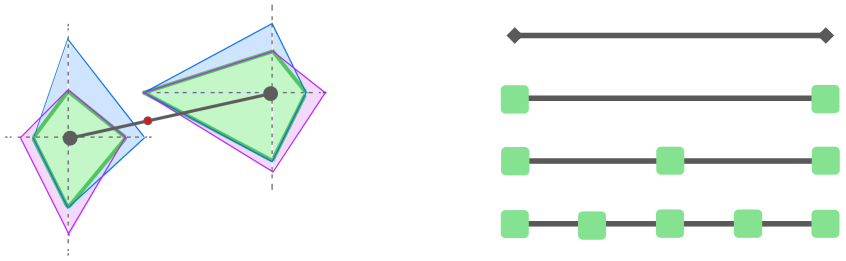
$\Delta \boldsymbol{q}^- = [\Delta q_0^-, \ldots, \Delta q_{N-1}^-]^\top$ and $\Delta \boldsymbol{q}^+ = [\Delta q_0^+, \ldots, \Delta q_{N-1}^+]^\top$, a test configuration $\boldsymbol{q}_{test}$ is in the safe zone of $\boldsymbol{q}_0$, if

$$\sum_{k=0}^{N-1} a_k^{[\cdot]}(q_{test,k} - q_{0,k}) + b^{[\cdot]} < 0 \tag{3.21}$$

holds for all combinations of restrictive directions. An example in a configuration space with two degrees of freedom is shown in Figure 3.7. In this example, the hyperplanes become lines of quadrilaterals.

3.5.2 Edge Examination

As noted in the preliminaries, the standard procedure begins the examination in the middle. Starting from the middle only changes the order of the sequence and can allow the colliding points, if any, to be identified earlier. If a collision is found, there is no need to check the rest of the sequence, and the examination is aborted.



(a) Edge examination with safe zones **(b)** Standard edge examination

Figure 3.9: Comparison of edge examinations with safe zones and standard edge examination. The red dot in (a) indicates the middle of the edge excluded by safe zones, which is the next point to build safe zones.

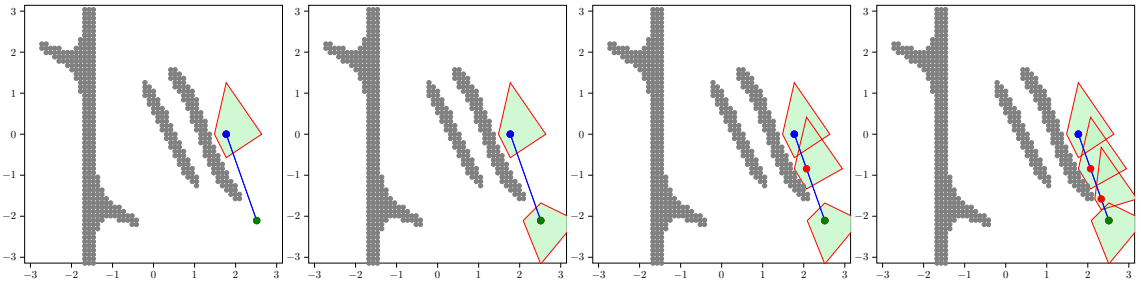


Figure 3.10: An example of examining edges with safe zones in the configuration space of a 2-DoF robot. Robots and obstacles are the same as in Figure 3.7.

The edge examination with safe zones is similar to the standard collision procedure, see Figure 3.9. The difference is the determination of the next point to examine. As shown in Figure 3.10, to verify the connection between two nodes, the safe zones of both nodes are first determined. The connection between two configurations, in this case, is a straight line, or in short, an edge. Since we are certain that the region covered by the safe zones is collision-free, we need only focus on the region excluded by both safe zones. The checking process iterates by computing safe zones of the middle points on the excluded

segments, e.g., the red one in Figure 3.7, until the edge is covered by safe zones or a collision is detected.

Assuming that we have an edge e defined by its two endpoints $\{\boldsymbol{q}_{ep}, \boldsymbol{q}_{ec}\}$, the direction of this edge is represented by the sign of $\boldsymbol{q}_{ep} - \boldsymbol{q}_{ec}$, which indicates which hyperplanes are active. By computing the intersection of the hyperplanes and the edge, the coverage of the safe zones on the edge is derived. To showcase the computation, an example using the 2-DoF robot in Figure 3.8 is given as follows. Consider an edge e defined by $\boldsymbol{q}_{ep}$ and $\boldsymbol{q}_{ec}$, whose direction is represented by $\{+, -\}$. The activated hyperplanes defined by

$$a_0^{+-}\Delta q_0 + a_1^{+-}\Delta q_1 + b^{+-} = 0, \tag{3.22}$$
$$a_0^{-+}\Delta q_0 + a_1^{-+}\Delta q_1 + b^{-+} = 0, \tag{3.23}$$

where Eq. (3.22) indicates the hyperplane in the positive direction of the edge and Eq. (3.23) in the negative direction. Specifically for a configuration $\boldsymbol{q}_e$ on edge e, the goal is to determine the intersection between the active hyperplanes Eq. (3.22) and Eq. (3.23) of its safe zone and the edge e. It leads to

$$a_0^{+-}\Delta q_{e,0}^{+} + a_1^{+-}\Delta q_{e,0}^{+}\frac{q_{ec,1} - q_{ep,1}}{q_{ec,0} - q_{ep,0}} + b^{+-} = 0, \tag{3.24}$$
$$a_0^{-+}\Delta q_{e,0}^{-} + a_1^{-+}\Delta q_{e,0}^{-}\frac{q_{ec,1} - q_{ep,1}}{q_{ec,0} - q_{ep,0}} + b^{-+} = 0, \tag{3.25}$$

where $q_{ec,[\cdot]}$ and $q_{ep,[\cdot]}$ denotes the $[\cdot]$-th joint of $\boldsymbol{q}_{ec}$ and $\boldsymbol{q}_{ep}$, respectively. The partial on e between $q_{e,0} + \Delta q_{e,0}^{-}$ and $q_{e,0} + \Delta q_{e,0}^{+}$ is covered by the safe zone [9].

3.6 Evaluation

The objective of HIRO is to reduce the accumulated number of distance computations and collision checks n_{cc}. Experiments in simulation and the real world with robots are conducted to evaluate the proposed method. With the experiments, the following questions should be answered:

Q3.1 How does the approach perform across environments of varying complexity?

Q3.2 How does each component contribute to improvement?

Q3.3 Is this approach fast enough for dynamic environments?

For the evaluation in simulation, we generate datasets of random planning scenes. A planning scene includes obstacles $\mathcal{S}_d$ in the reachable workspace and a planning query with valid start $\boldsymbol{q}_s$ and goal $\boldsymbol{q}_g$ configurations. At least one feasible path exists between the start and goal configurations in a planning scene. Figure 3.11 illustrates examples of planning scenes, where the green spheres indicate obstacles $\mathcal{S}_d$. The hardware experiment is conducted with a UR10e Robot. Throughout the robot experiment, the robot has to

[9] The edge e has a 1-D representation, defining the range of $q_{e,0}$ is sufficient to outline the safe region on edge.

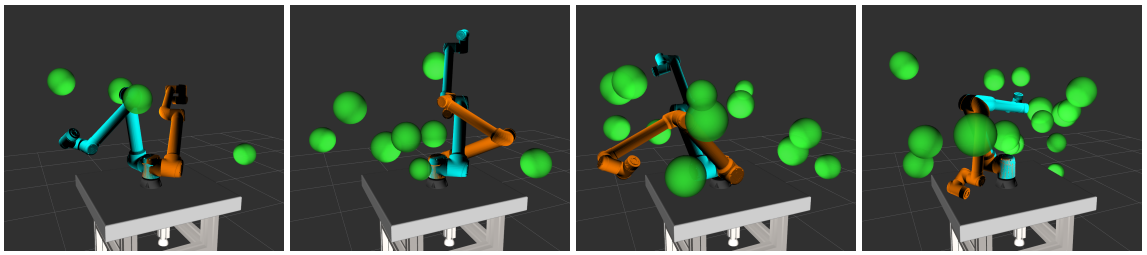


Figure 3.11: Examples of the planning scene datasets with 4, 8, 12, and 16 spherical random obstacles; start(orange) and goal(blue) poses are randomly generated.

	4 Spheres			8 Spheres			12 Spheres			16 Spheres		
	Planning Time [ms]		Mean / HIRO M.	Planning Time [ms]		Mean / HIRO M.	Planning Time [ms]		Mean / HIRO M.	Planning Time [ms]		Mean / HIRO M.
	Mean	Std		Mean	Std		Mean	Std		Mean	Std	
HIRO	**2.47**	**6.45**	-	**5.35**	**17.97**	-	**6.11**	**10.56**	-	**11.96**	**23.50**	-
RRT	61.36	258.83	24.84	83.25	291.08	15.56	147.90	465.86	24.2	230.09	566.33	19.23
RRTConnect	15.19	8.99	6.14	21.06	26.46	3.93	38.43	139.63	6.28	58.36	144.35	4.87
PRM	46.53	188.41	18.83	68.95	185.46	12.88	110.52	320.10	18.08	222.25	525.96	18.58
LazyPRM	27.83	209.81	11.26	33.73	104.28	6.3	66.97	239.57	10.96	177.71	838.46	14.85

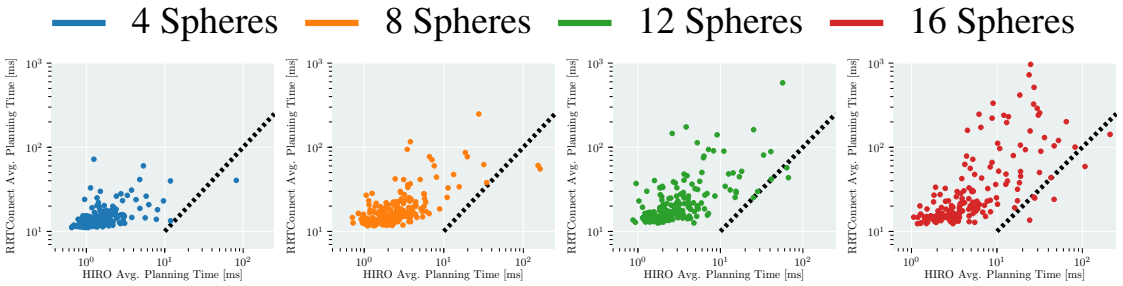
Table 3.1: Planning results of HIRO and baseline methods. The mean and standard deviation of the planning time are shown in milliseconds. The ratio between the mean of the baseline methods and HIRO is shown in green.

respond to environmental changes and adapt its motion accordingly, which is crucial for real-world applications in human-robot collaboration. The experiments are implemented using the Robot Operating System (ROS), the Open Motion Planning Library (OMPL) [158], and the Flexible Collision Library (FCL) [119]. In all experiments, the results do not include path smoothing.

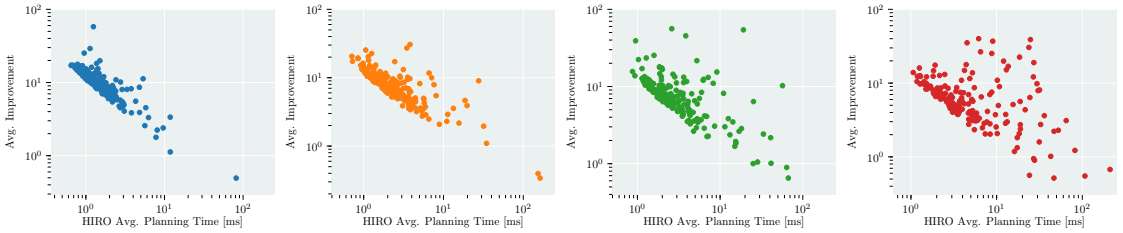
3.6.1 Evaluation in Simulation

In the simulation experiment, the proposed method is compared with baseline methods across various planning problems of varying complexity. Baseline methods are open-source implementations of RRT [92], RRTConnect [86], PRM [74], and LazyPRM [48] from OMPL. To be specific, the MoveIt! [23] wrapper of OMPL is used. To represent the different complexities of the problems, four groups of problems with 4, 8, 12, and 16 obstacles are created. Each problem group comprises 250 problems, in which the obstacles and the start-goal pair are randomly sampled. At least one feasible path exists between the start and goal configurations. For all problem groups, the same roadmap with 40,000 nodes is used. Each node is allowed to have 20 neighbors in a radius of $\frac{\pi}{2}$[rad]. The baseline methods use default parameters defined in MoveIt!. Each problem is run 20 times with different seeds.

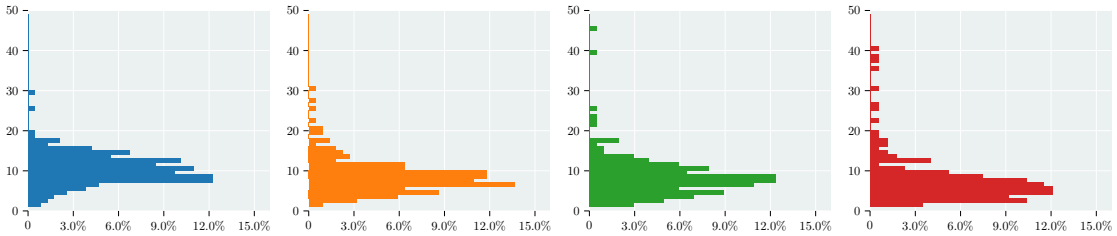
Table 3.1 lists the results of the mean and standard deviation of the planning time of the baseline methods and HIRO. All 5,000 runs of each problem group are used to compute the mean and standard deviation. HIRO exhibits a much shorter mean computation time while maintaining a small standard deviation within each problem group. This indicates that the proposed method is robust in solving a range of problems. This can be attributed to the low discrepancy of the precomputed roadmap. While the samples are less clustered in the high-dimensional spaces, it is easier to find a solution on the roadmap.



(a) Distribution of planning time using HIRO and RRTConnect. The dashed lines indicate equal performances. For the scatter points on the left of the dashed line, HIRO exhibits shorter planning time. Note that both axes are on a **logarithmic** scale and RRT-Connect starts with 10^1 ms.



(b) Planning time and improvements compared to RRTConnect. Data points above 10^0 in the vertical direction indicate an improvement. Note that both axes are on a **logarithmic** scale.



(c) Histogram of the improvement over various planning problems. The y-axis indicates the average improvements, same as in (b).

Figure 3.12: Results of various planning problems with different numbers of obstacles. Each point in the scatter plots indicates an individual problem.

From another perspective, the number of obstacles does not truly reflect the difficulty of a planning problem since difficulty depends on both the environment and the query. If the obstacles are clustered in a small region that does not block the robots, the planning problem becomes easy. To have a closer look at each problem and show the difference between HIRO and the baseline methods, Figure 3.12a and Figure 3.12b present the results of every planning scene as a dot. Table 3.1 exhibits that RRTConnect performs the best among all the baselines. Therefore, the results of RRTConnect are used for this comparison. Figure 3.12a visualizes the distribution of the average planning time of each problem over 20 runs. For the points that lie precisely on the dashed line, HIRO is identically fast as RRT-Connect in these planning problems. Figure 3.12b exhibits the improvements compared to RRTConnect. The improvement is measured as the ratio of the average computation time for RRTConnect to that for HIRO for each problem. An improvement equaling one indicates identical performance. The distribution of the improvements is summarized in histograms in Figure 3.12c. Data points with ratios exceeding 50 are omitted.

The results show clear improvements across most planning problems. Note that HIRO uses a deterministic roadmap, keeping the computational cost of a problem constant across

runs. The computation time varies only slightly due to CPU scheduling. Hence, the average planning time is a useful metric for assessing the planner's responsiveness. However, some outlier cases with worse performance than the baseline are also shown in Figure 3.12a. The number of these cases increases with the number of obstacles. At the same time, HIRO's performance appears to decline with increasing obstacle density. One reason is that the same 40,000-node roadmap is used for all four problem sets. As obstacles increase, more detours in the roadmap are required due to insufficient density. While increasing the number of nodes in the roadmap may increase search complexity and lead to worse performance, a hierarchical approach can be considered. The hierarchy will be discussed and explored at the end of this chapter. Nevertheless, the dataset with 16 obstacles shows a mean improvement of 7.62-fold.

Q3.1 How does the approach perform across environments of varying complexity?

A3.1 The proposed method shows significant improvements in all four tasks of different complexity. The improvements are evident in most of the problems tested. The proposed method also exhibits low variance across different planning problems. However, the approach performs worse in some outlier cases, and the number of these cases increases with the number of obstacles. This can be attributed to the insufficient density of the precomputed roadmap. To solve this problem, a hierarchy of the roadmap can be considered.

3.6.2 Ablation Study

Despite the significant speedup achieved, it is still necessary to investigate the contributions of each component. The first ablation study compares Halton sequences with uniform sampling methods. The second ablation study examines the improvement with and without safe zones in place. As the benefit of using informed search has been shown in Figure 3.5, no further investigation is conducted.

Table 3.2 shows the results collected with a single core on an Intel i9-9900K CPU. The results for HIRO differ slightly from those in Table 3.1 because the data in Table 3.1 were collected with different hardware. The variant using uniform sampling with safe-zone checks exhibits a longer average planning time than HIRO, suggesting that the search algorithm must explore more of the graph to find solutions. On the other hand, this variant shows shorter planning time than the one with a standard checking procedure. The advantage of safe zones becomes increasingly significant as problem complexity increases. The planning time associated with precomputed roadmaps is generally lower than that of the other baseline methods reported in Table 3.1. This indicates that the precomputed roadmap effectively eliminates infeasible configurations and reduces the search space.

Q3.2 How does each component contribute to improvement?

A3.2 Deterministic roadmaps offer general contributions to improvements, while safe zones provide significant improvement when the complexity of planning problems increases.

	4 Spheres		8 Spheres		12 Spheres		16 Spheres	
	Planning Time [ms]		Planning Time [ms]		Planning Time [ms]		Planning Time [ms]	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std
HIRO	**2.094**	5.369	**4.571**	15.204	**5.203**	9.134	**10.312**	20.474
Uniform + Safe Zone	3.511	2.019	5.459	10.859	7.176	13.205	12.781	28.005
Uniform + Standard Checking	3.685	2.130	5.562	12.192	7.903	17.226	14.104	36.015

Table 3.2: Ablation study. All variants have a planning budget of one second.

3.6.3 Evaluation in Robot Experiments

In the robot experiment, a human-size model, Jessica[10] represents the dynamic changes in the environment. The robot setup is shown in Figure 3.13. During the experiment, Jessica appears and disappears in the workspace, forcing the robot to adapt its motion frequently. Jessica is tracked using an ArUco marker [40] captured by an Intel RealSense D435 camera mounted above the workspace. An oriented bounding box (OBB) approximates the collision volume for planning.

Throughout the experiment, the robot traverses between both ends of the table. This scenario simulates that the robot primarily conducts a pick-and-place task while simultaneously sharing the workspace with Jessica. Figure 3.14 presents a complete workflow cycle. The arrows refer to Jessica's sudden displacement while the robot is moving. Upon detection of a specific movement, a planning request is triggered. To better visualize the planned path, we let the robot execute the adjusted path to the end. No path smoothing or shortening method is used for the presented path. The hardware experiment confirms that HIRO can quickly respond to environmental changes and adapt the path accordingly [11].

Figure 3.13: Robot experiment setup. The left figure shows a UR10e robot and a dynamic object with an ArUco marker attached for tracking purposes. Throughout the experiments, the robot travels between the two configurations of the middle and right figures.

Q3.3 Is the approach fast enough for dynamic environments?

A3.3 The approach can instantly replan in response to sensor events while Jessica constantly moves in the workspace. However, due to the limited number of nodes in the precomputed roadmap, the resulting trajectory may not be optimal and intuitive.

[10] The name arose during a late-night lab experiment.
[11] A supplementary video is available at `https://youtu.be/ITBz1W7Ecbg`

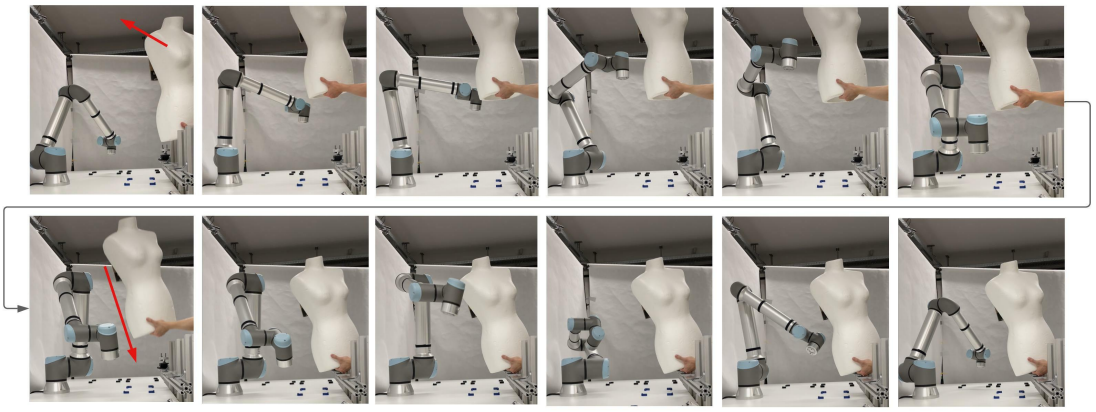


Figure 3.14: Robot experiment with Jessica moving within the workspace. The robot moves between the far end (top left) and the near end (bottom left) of the table. The red arrows indicate Jessica's sudden movement during the robot's motion. Snapshots are shown in chronological order from left to right.

3.7 Limitations and Discussion

The proposed method, HIRO, demonstrates an approximately $7\times$ reduction in computation time across diverse planning problems in the most challenging scenarios. Three components, i.e., precomputed roadmaps, heuristic-informed search, and edge examination with safe zones, contribute together to the speedup.

The precomputed roadmap is the foundation of such improvements. Using the roadmap, edge examination can be prioritized according to heuristics based on the approximated lower bound of the path cost. This results in fewer edges, n_{ee}, being examined. Furthermore, while searching the roadmap, self-collision and collision checks with respect to the static environment are unnecessary, thereby reducing computational burden. However, using fixed-sized roadmaps sacrifices the completeness of the sampling-based method. Completeness guarantees that, given unlimited time, the algorithm can return a solution if the planning problem has a feasible solution. An unlimited planning budget is neither realistic nor reasonable in dynamic environments. To have a balanced trade-off between completeness and speed, the following concepts can be considered: (1) a hierarchical roadmap whose resolution can be adjusted by partially activating or deactivating the nodes and edges, and (2) introducing intermediate subgoals, to which the algorithm can surely return a feasible solution. An observation from the evaluation is that more time is inevitably required for reasoning as complexity increases. The concept of subgoals avoids solving complex problems in dynamic environments by decomposing them into subproblems. This concept can reduce the complexity of each planning and ensure a short planning time for each subproblem. More details regarding subgoals are introduced in Chapter 4.

Figure 3.15 demonstrates another limitation. After certain changes in the environment are detected, the resulting path first guides the robot toward the object and then retreats to a similar position. The effective avoidance behavior occurs after this unnecessary back-and-forth motion. This sub-optimality can be attributed to the fixed size of the roadmap and the placement of the samples in the roadmap. A more advanced method can be

considered to place the samples more strategically instead of using Halton sequences, such as identifying the regions that require denser sampling.

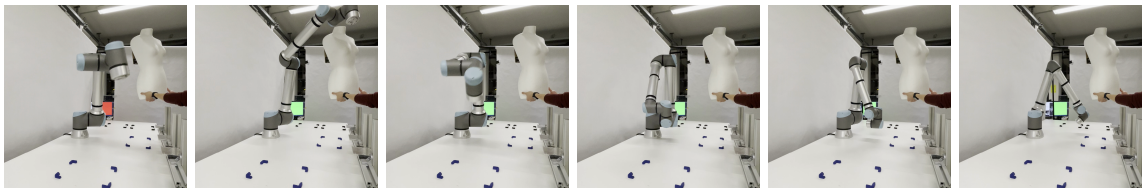


Figure 3.15: Snapshots of a sub-optimal avoidance trajectory, shown in chronological order from left to right. In the left figure, the red screen indicates that changes in the environment are detected and planning is triggered. The robot first approaches the object and then moves away from it, resulting in a suboptimal motion.

4 Planning with Learned Subgoals

When facing a generic robot motion planning problem, the HIRO method proposed in Chapter 3 addresses the bottleneck by reducing the number of collision checks and concentrating resources on uncertain regions. This planning problem can refer to either planning a complete problem from the current robot position to its final goal or planning a subproblem to incrementally progress toward the goal. Although HIRO demonstrates a significant speedup, its planning time remains unbounded and increases with the expansion of the search space and problem complexity.

Intuitively, if the start and goal configurations are closer to each other, a solution is more likely to be found in a shorter, bounded time. Therefore, breaking down a complete planning problem into smaller, more manageable subproblems can be a more practical and effective approach in dynamic environments. Each subproblem is defined by a specific subgoal, ideally located near the robot's current position. Additionally, closer proximity makes the motion less vulnerable to environmental changes over time, as it takes less time to reach the subgoal, and future events are less likely to affect execution. This chapter aims to investigate the following research question:

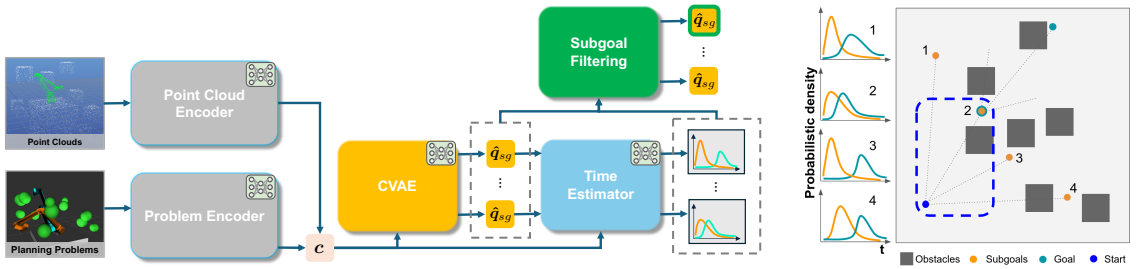
*Can identifying and reaching subgoals **indeed** be a more efficient strategy in dynamic environments?*

This question is equivalent to finding a set of subgoals that improve the planning performance with respect to desired metrics, such as planning and execution time. In the context of planning in dynamic environments, a good subgoal must have these two properties: (a) manageable planning time from the current robot position to the subgoal and (b) progressively leading the robot to the final goal. Therefore, the question above converts to

How to capture the distributions of the good subgoals?

Traditional reactive methods [37, 77] usually focus on the property (a). Despite the instant computation time, these methods can easily fall into local minima and fail to approach the final goal [84]. On the other hand, motion planning methods, including sampling-based [74, 86] and optimization-based methods [70, 140], put more weight on the progression towards the final goal, i.e., the property (b). They rely on the spatial domain alone to perform their computations, neglecting the fact that the resulting solution is time-sensitive[1], meaning that these solutions may only be effective for a brief period in dynamic environments. The method proposed in this chapter seeks to address both properties simultaneously.

[1] Time-sensitiveness indicates that the solution remains valid only in a limited and usually short time frame.



(a) Architecture of the method **(b)** Illustration of the method

Figure 4.1: Architecture and illustration of the method. Subgoal candidates $\hat{\boldsymbol{q}}_{sg}$ are shown within the dashed bounding box in (a) and indicated by numbered orange marks in (b). They are generated by a conditional variational autoencoder (CVAE) that takes encoded latent representations of point clouds and planning problems as input. By evaluating the planning time distributions from the start to these candidates and from the candidates to the final goal, the best candidate is selected as a subgoal, highlighted by a green outline in both (a) and (b).

Assuming that the dynamic environment can be seen as frozen for a short time frame, e.g., 50 ms, the method proposed in this chapter integrates temporal information into learning to be aware of the time budget while solving a planning problem. The outcome is a collision-free motion plan regarding the current status of the environment to an intermediate subgoal.

In this chapter, the focus is on determining the distribution of good subgoals whose temporal information satisfies constraints on real-world planning time. In this context, *temporal information* does **not** refer to how the environment evolves over time; rather, it refers to the time required to find a solution from the current robot state to this intermediate subgoal. To this end, a **P**lanning with **L**earned **S**ubgoal (PLS) method is proposed. Figure 4.1 visualizes this method. PLS consists of (1) a subgoal generation module conditioned on the final robot goal pose, current robot state, and the current state of the environment, and (2) a critic module that evaluates and selects the generated samples from the first module according to various metrics.

The temporal information is involved in two aspects. First, the problem can be viewed as finding the mapping from the original complete problems to subgoal distributions. Data-driven methods, especially generative models, are well-suited to learning such complex mappings. When all data points in the training dataset satisfy the constraints, temporal information is implicitly encoded in the trained model. Second, a critic trained on temporal information is used to assess whether the generated samples exhibit the aforementioned properties of good subgoals. Together with temporal information, methods learning solely from spatial data can be extended to scenarios where temporal constraints, such as planning time, are crucial.

Unlike methods that use a learned spatial distribution [61, 88, 96] to bias sampling, the subgoals offered by PLS are more akin to milestones. These milestones decompose a complex planning problem into small and easily solvable pieces. An example in 2D for illustration purposes is shown in Figure 4.1b. Given a planning problem with obstacles, the learned generative model generates a batch of candidate subgoals, marked in orange. Then, the critic examines each candidate by estimating the planning time from the cur-

rent robot position to the candidate and from the candidate to the goal. This examination assesses the likelihood of finding a solution within the constrained planning budgets and whether the candidates can contribute to achieving the final goal. In the illustrated case, the candidate with index 2 is the most suitable for the subgoal as it indicates a low planning time from start to candidate and from candidate to goal. The selection is detailed in Section 4.4.2. After selection, the range of the planning problem shrinks to the blue dashed box for better planning performance. Experiments show that integrating temporal information significantly improves the planning algorithm and can be applied to reactive planning scenarios in which the planner has limited time to find a solution. The contributions are mainly:

- Utilizing a generative model to predict spatial subgoals that decompose a complex planning problem.
- Employing learned time estimators as a critic to capture the temporal aspects regarding the planning time for given planning problems.
- Designing two metrics based on time estimations to identify suitable and goal-directed subgoal candidates.

In the remainder of this chapter, some preliminaries on variational inference and generative models are introduced in Section 4.1; Section 4.2 formulate the problem and introduce how the generative model and time estimator cooperate to generate and select subgoals; Later on, Section 4.3 depicts the training and inference of generative model in details; Section 4.4 gives details of how to select samples using a learned critic at the inference stage and Section 4.5 verifies the proposed method with simulative experiments and extensive ablation studies.

4.1 Preliminaries

One of the core components of the method proposed in this chapter is capturing the subgoal distribution from the data. This section introduces basic concepts of variational inference and variational auto-encoders.

4.1.1 Variational Inference

The core problem in this chapter is to approximate the distribution of the suitable subgoals. This problem aligns with the general objective of Variational Inference (VI) methods, which are widely used to approximate probability densities [13]. Consider a data point or an n-dimensional observation $\boldsymbol{x} \in \mathbb{R}^n$ in a dataset $\mathcal{X}$ and an m-dimensional latent variable $\boldsymbol{z}$, the goal of VI is to find a distribution $q^*(\boldsymbol{z})$ in a family of distributions Q such that

$$q^*(\boldsymbol{z}) = \arg\min_{q(\boldsymbol{z})\in Q} \mathcal{D}_{\mathrm{KL}}\left[q(\boldsymbol{z})||p(\boldsymbol{z}|\boldsymbol{x})\right], \tag{4.1}$$

where $p(\boldsymbol{z}|\boldsymbol{x})$ represents the true distribution of the latent variable $\boldsymbol{z}$ given the data $\boldsymbol{x}$ and $\mathcal{D}_{\mathrm{KL}}[\cdot]$ denotes the Kullback-Leibler (KL) divergence [87]. The KL-Divergence captures the divergence of two generic distributions $q(x)$ and $p(x)$ by

$$\mathcal{D}_{\mathrm{KL}}\left[q(x)||p(x)\right] = \mathbb{E}_{x\sim q}\left[\log\frac{q(x)}{p(x)}\right] = \int q(x)\log\frac{q(x)}{p(x)}\,\mathrm{d}x, \tag{4.2}$$

with x sampling from the distribution $q(x)$. The variables are changed in Eq. (4.2) to avoid confusion with the distributions in Eq. (4.1). Applying Bayes' rule

$$p(\boldsymbol{z}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{p(\boldsymbol{x})} \tag{4.3}$$

to Eq. (4.1), the KL-Divergence captures the divergence of two distributions by

$$\begin{aligned}
\mathcal{D}_{\mathrm{KL}}\left[q(\boldsymbol{z})||p(\boldsymbol{z}|\boldsymbol{x})\right] &= \mathbb{E}_{z\sim q}\left[\log\frac{q(\boldsymbol{z})}{p(\boldsymbol{z}|\boldsymbol{x})}\right] \\
&= \mathbb{E}_{z\sim q}\left[\log q(\boldsymbol{z}) - \log p(\boldsymbol{z}|\boldsymbol{x})\right] \\
&= \mathbb{E}_{z\sim q}\left[\log q(\boldsymbol{z}) - \log p(\boldsymbol{x}|\boldsymbol{z}) - \log p(\boldsymbol{z})\right] + \log p(\boldsymbol{x}) \\
&= -\underbrace{\mathbb{E}_{z\sim q}\left[\log p(\boldsymbol{x}|\boldsymbol{z}) + \log p(\boldsymbol{z}) - \log q(\boldsymbol{z})\right]}_{\mathrm{ELBO}} + \log p(\boldsymbol{x})
\end{aligned} \tag{4.4}$$

The term ELBO refers to the Evidence Lower Bound, which can also be derived via Jensen's Inequality. By maximizing the ELBO, the KL-divergence $\mathcal{D}_{\mathrm{KL}}\left[q(\boldsymbol{z})||p(\boldsymbol{z}|\boldsymbol{x})\right]$ is minimized. This turns an inference problem, i.e., computing the analytic posterior distribution, into an optimization problem. In practice, there are various methods to maximize the ELBO. However, introducing these methods is beyond the scope of this work. Eq. (4.4) will be used and discussed in the variational auto-encoder (VAE) and conditional variational auto-encoder (CVAE).

4.1.2 Variational Auto-Encoder

Different from variational inference, which uses a latent distribution $q(\boldsymbol{z})$ to approximate the posterior distribution $p(\boldsymbol{z}|\boldsymbol{x})$ given the data $\mathcal{X}$, Variational Auto-Encoder (VAE) is a type of generative model that learns the latent distribution $q(\boldsymbol{z}|\boldsymbol{x})$ given the data $\boldsymbol{x}$ and then generates samples similar to the data based on this latent distribution [30].

Put differently, the objective of the VAE is to find the samples of $\boldsymbol{z}$ that are likely to generate $\boldsymbol{x} \in \mathcal{X}$. Note that the representation of the latent variable is not unique. In the context of VAE, this representation can be customized to meet the needs. The objective of VAE is slightly different from VI

$$q^*(\boldsymbol{z}|\boldsymbol{x}) = \arg\min_{q(\boldsymbol{z}|\boldsymbol{x})\in Q} \mathcal{D}_{\mathrm{KL}}\left[q(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})\right], \tag{4.5}$$

where the approximated distribution $q(\cdot)$ also depends on $\boldsymbol{x}$. It indicates that the goal is to find a latent representation $q(\boldsymbol{z}|\boldsymbol{x})$ that is close to the real latent representation $p(\boldsymbol{z}|\boldsymbol{x})$ given the data $\boldsymbol{x}$. Eq. (4.4) becomes

$$\begin{aligned}
\mathcal{D}_{\mathrm{KL}}\left[q(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})\right] &= -\underbrace{\mathbb{E}_{z\sim q(\boldsymbol{z}|\boldsymbol{x})}\left[\log p(\boldsymbol{x}|\boldsymbol{z}) + \log p(\boldsymbol{z}) - \log q(\boldsymbol{z}|\boldsymbol{x})\right]}_{\mathrm{ELBO}} + \log p(\boldsymbol{x}) \\
&= \underbrace{\mathcal{D}_{\mathrm{KL}}\left[q(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})\right]}_{\mathrm{Encoder}} - \underbrace{\mathbb{E}_{z\sim q(\boldsymbol{z}|\boldsymbol{x})}\left[\log p(\boldsymbol{x}|\boldsymbol{z})\right]}_{\mathrm{Decoder}} + \log p(\boldsymbol{x})
\end{aligned} \tag{4.6}$$

The first term of the right-hand side in Eq. (4.6) encourages the approximated latent distribution $q(\boldsymbol{z}|\boldsymbol{x})$ to be close to the desired distribution $p(\boldsymbol{z})$. The second term denotes

the log-likelihood that a latent representation sampled from q can reconstruct $\boldsymbol{x}$ in the dataset. The optimization regarding Eq. (4.5) minimizes $\mathcal{D}_{\mathrm{KL}}\left[q(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})\right]$ and maximizes $\mathbb{E}_{z\sim q}\left[\log p(\boldsymbol{x}|\boldsymbol{z})\right]$.

Encoder The first term encodes the data distribution to a latent space and aligns the resulting distribution $q(\boldsymbol{z}|\boldsymbol{x})$ to the desired latent distribution $p(\boldsymbol{z})$. Hence, this term is used to refer to an encoder. The distributions $p(\boldsymbol{z})$ and $q(\boldsymbol{z}|\boldsymbol{x})$ of the latent variable $\boldsymbol{z}$ are usually assumed to be Gaussian, namely $p(\boldsymbol{z}) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and $q(\boldsymbol{z}|\boldsymbol{x}) \sim \mathcal{N}(\boldsymbol{\mu}_\theta(\boldsymbol{x}), \boldsymbol{\Sigma}_\theta(\boldsymbol{x}))$, where θ represents a neural network. The first term becomes

$$\mathcal{D}_{\mathrm{KL}}\left[q(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})\right] = \mathcal{D}_{\mathrm{KL}}\left[\mathcal{N}(\boldsymbol{\mu}_\theta(\boldsymbol{x}), \boldsymbol{\Sigma}_\theta(\boldsymbol{x}))||\mathcal{N}(\mathbf{0}, \boldsymbol{I})\right]. \tag{4.7}$$

Decoder Another assumption is that the distribution $p(\boldsymbol{x}|\boldsymbol{z})$ is Gaussian $\mathcal{N}(f_\phi(\boldsymbol{z}), \boldsymbol{\Sigma}_d)$, where $\boldsymbol{\Sigma}_d = \sigma_d^2 \boldsymbol{I}$ denotes the covariance matrix of the decoded samples and σ_d^2 will not be determined explicitly. Sampling from the distribution $p(\boldsymbol{z})$, VAE decodes the samples and projects them back to the data representation using the function $f_\phi(\cdot)$, where ϕ indicates the parameters of a neural network. According to the definition of Gaussian, the probabilistic density function of $\mathcal{N}(f_\phi(\boldsymbol{z}), \boldsymbol{\Sigma}_d)$ is proportional to $e^{-\frac{1}{2}(\boldsymbol{x}-f_\phi(\boldsymbol{z}))^\top \boldsymbol{\Sigma}_d^{-1}(\boldsymbol{x}-f_\phi(\boldsymbol{z}))}$. Therefore, with $\boldsymbol{\Sigma}_d = \sigma_d^2 \boldsymbol{I}$, the likelihood that the sample $\boldsymbol{z}$ reconstructs $\boldsymbol{x}$ can be written as

$$\log p(\boldsymbol{x}|\boldsymbol{z}) \propto -||\boldsymbol{x} - f_\phi(\boldsymbol{z})||_2^2 \tag{4.8}$$

With $\mathbb{E}_{z\sim q}\left[\log p(\boldsymbol{x}|\boldsymbol{z})\right] = \log p(\boldsymbol{x}|\boldsymbol{z})$, the expectation can be written as

$$\mathbb{E}_{\boldsymbol{z}\sim q(\boldsymbol{z}|\boldsymbol{x})}\left[\log p(\boldsymbol{x}|\boldsymbol{z})\right] \propto -||\boldsymbol{x} - f_\phi(\boldsymbol{z})||_2^2. \tag{4.9}$$

Training The training pipeline is visualized in Figure 4.2. Given the data from the dataset $\mathcal{X}$, the encoder outputs the mean $\boldsymbol{\mu}_\theta$ and the covariance matrix $\boldsymbol{\Sigma}_\theta$ of the corresponding multivariate Gaussian distribution of the latent variable $\boldsymbol{z}$. The decoder projects the samples from the latent distribution back to the high-dimensional space. The parameters of the neural networks θ and ϕ can be iteratively optimized using gradient descent. It is worth noting that the reparameterization trick [79] is employed to enable backpropagation through the encoder. The pipeline minimizes the loss function

$$J = \beta \mathcal{D}_{\mathrm{KL}}\left[\mathcal{N}(\boldsymbol{\mu}_\theta(\boldsymbol{x}), \boldsymbol{\Sigma}_\theta(\boldsymbol{x}))||\mathcal{N}(\mathbf{0}, \boldsymbol{I})\right] + ||\boldsymbol{x} - f_\phi(\boldsymbol{z})||_2^2 \tag{4.10}$$

where β denotes the weighting between the KL-divergence and the reconstruction error introduced by the encoder.

Inference During the training, the encoder learns to project the data to a distribution close to the standard multivariate Gaussian $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$. Therefore, a VAE can directly sample from $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and feed the samples into the decoder without using the encoder during inference. A question to ask is why an encoder is required if only the decoder works solely at the inference stage. In practice, it is usually the case that $q(\boldsymbol{z}|\boldsymbol{x})$ does not converge to $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ at the training stage. Sampling from the latent distribution $q(\boldsymbol{z}|\boldsymbol{x})$ does **not** imply a completely random variable but a perturbation or exploration from the latent feature so that the data can be reconstructed, reducing $||\boldsymbol{x} - f_\phi(\boldsymbol{z})||_2^2$. Forcing the $q(\boldsymbol{z}|\boldsymbol{x})$ close to $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ makes the model generalize. Put differently, β balances the generation and reconstruction. If the generation and reconstruction are well balanced, $q(\boldsymbol{z}|\boldsymbol{x})$ is not far from $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$, and hence similar data can be generated.

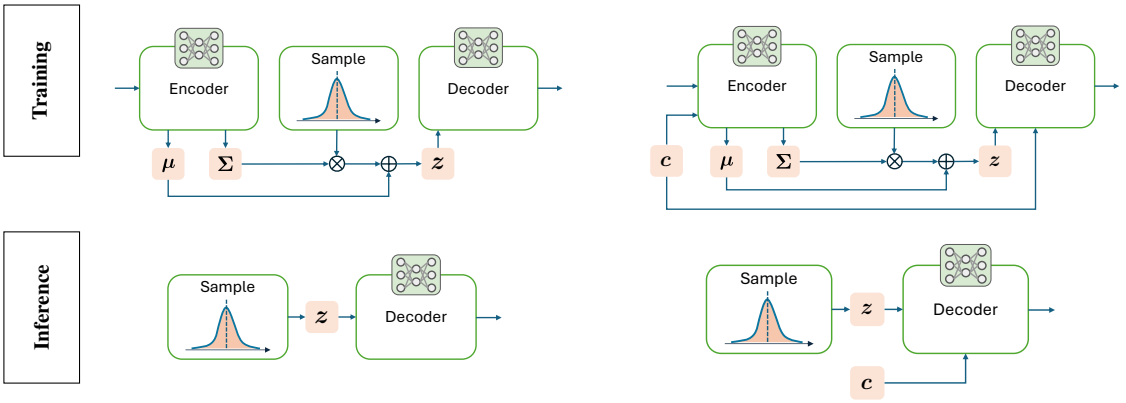


Figure 4.2: Training and inference pipeline of VAE (left) and CVAE (right). The encoders are not needed at the inference stage for both VAE and CVAE. Encoders and Decoders are usually neural networks.

4.1.3 Conditional Variational Auto-Encoder

The conditional variational auto-encoder (CVAE) is an extension of VAE. As shown in 4.2, the conditional variational auto-encoder takes an additional condition $\boldsymbol{c}$ to generate new synthetic data $f_\phi(\boldsymbol{z}, \boldsymbol{c})$ compared to the VAE. Consider the subgoal generation as an example. Subgoals are generated for specific planning problems and the surrounding environment. The condition $\boldsymbol{c}$ in this context refers to the planning problem and the surrounding environment.

The distribution of the latent variables $p(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{c})$ for generating the synthetic data should be similar to the distribution $q(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{c})$ represented by the dataset. The KL-divergence between these two distributions becomes

$$\mathcal{D}_{\mathrm{KL}}\left[q(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{c})||p(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{c})\right] = \mathbb{E}_{\boldsymbol{z}\sim q(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{c})}\left[\log q(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{c}) - \log p(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{c})\right]. \tag{4.11}$$

The loss for training CVAE becomes

$$J = \beta\mathcal{D}_{\mathrm{KL}}\left[\mathcal{N}(\boldsymbol{\mu}_\theta(\boldsymbol{x}, \boldsymbol{c}), \boldsymbol{\Sigma}_\theta(\boldsymbol{x}, \boldsymbol{c}))||\mathcal{N}(\mathbf{0}, \boldsymbol{I})\right] + ||\boldsymbol{x} - f_\phi(\boldsymbol{z}, \boldsymbol{c})||_2^2. \tag{4.12}$$

The training and inference of CVAE are the same as VAE. The condition $\boldsymbol{c}$ is usually an encoded latent representation of raw data.

4.2 Problem Description

Given a planning problem of a robot with N DoF defined by the current robot configuration $\boldsymbol{q}_s \in \mathbb{R}^N$, goal configuration $\boldsymbol{q}_g \in \mathbb{R}^N$, and the surrounding environment $\mathcal{S}$, the proposed method in this chapter aims to find a mapping $f_\phi(\boldsymbol{q}_s, \boldsymbol{q}_g, \mathcal{S})$ to the distribution of joint configuration $\boldsymbol{q}_{sg} \in \mathbb{R}^N$, where the maximum planning time t_s from $\boldsymbol{q}_s$ to $\boldsymbol{q}_{sg}$ using a sampling-based planner $\pi_s(\boldsymbol{q}_s, \boldsymbol{q}_{sg})$ is less than a desired bound t_b. The joint configuration $\boldsymbol{q}_{sg}$ is termed a subgoal. The surrounding environment $\mathcal{S}$ can be represented as a point cloud. Note that the objective is not to completely cover the actual distribution fulfilling $t_s \leq t_b$, but to find a subset of the distribution.

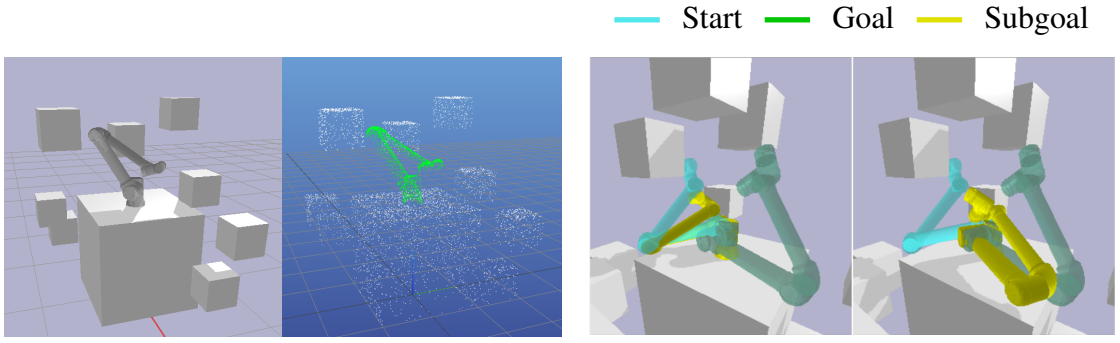
This can be understood as a three-part problem. The first step is to collect a dataset $\mathcal{X}$, which captures the correspondence between $\{\boldsymbol{q}_s, \boldsymbol{q}_g, \mathcal{S}\}$ and $\boldsymbol{q}_{sg}$. The second step focuses on training the model $f_\phi(\boldsymbol{q}_s, \boldsymbol{q}_g, \mathcal{S})$ to generate subgoals $\boldsymbol{q}_{sg}$ that closely resemble those in the dataset $\mathcal{X}$. The third step involves selecting the generated subgoal candidates during inference. A critic, which is a neural network $f_\psi(\boldsymbol{q}_s, \boldsymbol{q}_{sg}, \mathcal{S})$, is trained to help identify the best candidates according to specific desired metrics.

4.3 Generating Spatial Subgoals

The core idea behind generating subgoals is to first learn the distribution of desired subgoals from the data, then sample from this distribution during inference. The desired subgoals should consider both the spatial and temporal aspects of the planning. The spatial aspect requires that the subgoals progressively lead the robot to the final goal. The temporal aspect requires the planning time from the current robot position to the subgoal to be within the desired reaction time. This section first describes the details of collecting a dataset in Section 4.3.1. In Section 4.3.2, the dataset is used to train generative models to capture its latent representation and to generate similar subgoals conditioned on the planning problem.

4.3.1 Subgoal Dataset

A subgoal dataset consists of planning problems, corresponding point clouds, subgoals, and the planning time for each subgoal. An example is shown in Figure 4.3. Each component is described in detail in the following in Section 4.3.1.1. The data generation pipeline is introduced in Section 4.3.1.2.



(a) Cluttered environment and point clouds

(b) Planning problem and subgoals in dataset

Figure 4.3: Components in the subgoal dataset. The point cloud representation of a cluttered environment is shown in (a). A planning problem, including the start and goal configurations and the corresponding subgoals, is shown in (b).

4.3.1.1 Components in Dataset

A dataset of one million random planning problems is collected. Each planning problem includes the robot's start and goal configurations, the point cloud of the current workspace,

at least one subgoal, and the corresponding planning time from the start configuration to each subgoal. This section only depicts each component, and details of collecting will be presented in Section 4.3.1.2.

Planning Problem Description A planning problem is generally defined by the start configuration, i.e., the current robot configuration, the goal, and the surrounding environment. The goal can be specified as a joint configuration or as the Cartesian pose of the tool center point (TCP). If the goal is specified in TCP, an additional inverse kinematics (IK) module is required to convert the goal pose into joint configurations. Since IK often yields multiple solutions, the dataset directly represents the goal using joint configurations to eliminate ambiguity.

Point Clouds The surrounding environment is represented by point clouds in the dataset. The point cloud representation remains consistent regardless of the number of objects in the environment and is unaffected by changes in viewing angle. There are alternatives, such as state-based and image-based representations. A state-based representation provides structured information about objects in the environment, including their shape and position. However, when representing the state information by a vector $\boldsymbol{s}$, the length of the vector varies depending on the number of objects. It can be very challenging for current machine learning methods to capture a meaningful representation when the input state is inconsistent. One possible approach to encoding an arbitrary number of objects of different shapes into a unified state-based representation is to first approximate the objects with primitive shapes and then use a graph model to derive a latent representation [180]. Image-based representations do not suffer from variable dimensionality issues. However, they are strongly influenced by the camera's position. Changes in viewpoint can significantly affect the learned policy, potentially leading to execution failure.

Subgoals For each planning problem in the dataset, there is at least one subgoal. Subgoals are represented as joint configurations to avoid ambiguity. The planning time from the start to subgoal configurations must be less than 50 ms.

Planning Time to Subgoals Due to the stochastic process of sampling-based methods, the planning time of a planning problem is not deterministic but has a distribution. To represent this distribution, the dataset records the planning time for 20 runs between each subgoal and its corresponding start position.

4.3.1.2 Dataset Generation Pipeline

The dataset comprises over *one million* randomly generated planning requests for a UR10e robot, each specifying a unique start and goal configuration, with joint values constrained to $[-2\pi, 2\pi]$ across all joints. These requests are planned using OMPL [158] in diverse environments. The environment includes up to 12 randomly positioned boxes. There is at least one feasible path for each request. Feasible paths are represented as a list of waypoints. Suitable waypoints are extracted from these feasible paths and saved in the dataset. The data generation pipeline is visualized in Figure 4.4. The following presents an overview of generating subgoal datasets from a spatial and temporal perspective.

The spatial aspect of the subgoals concerns whether each subgoal can progressively guide the robot toward the final goal. The planned feasible paths are reliable sources for deriving subgoals regarding this aspect. Each waypoint in the resulting path is a candidate subgoal for the dataset. Sampling-based planners are stochastic and can output completely dif-

ferent results for the same problem. An asymptotically optimal sampling-based planner AIT* [157] is used to reduce subgoal variance. The AIT* planner refines the solution incrementally within the time budget, reducing unnecessary detours and making the waypoints in the resulting path more expressive and less redundant.

The subgoal's temporal aspect refers to the planning time from the start configuration to the subgoal configuration. For each waypoint along the feasible paths, the planning-time distribution from 20 runs is collected. The planner used for this computation should be consistent with the deployment planner. With this consistency, there are no other restrictions on choosing the planner. The planner can be HIRO, AIT*, RRTConnect, or something else. The data reported here were collected using RRTConnect. This choice is informed by the characteristics of AIT* as an optimal planner that continually refines its solutions until the given time budget is exhausted. Our observations indicate that AIT* generally takes a bit more time to return an initial solution than RRTConnect, consistent with the results reported in the original AIT* paper [157]. A maximum planning time of 50 ms is used as a threshold to filter out unsuitable subgoals. Theoretically, this constraint on planning time should be small to enable reactive planning in changing environments. Practically, however, the smaller the range, the less data it can accommodate, and thus training a model properly becomes harder.

The point cloud is captured from 4 viewpoints floating above the workspace in the PyBullet physics simulator. The original point clouds are downsampled to 4086 points to facilitate feature and representation learning; see Figure 4.3a.

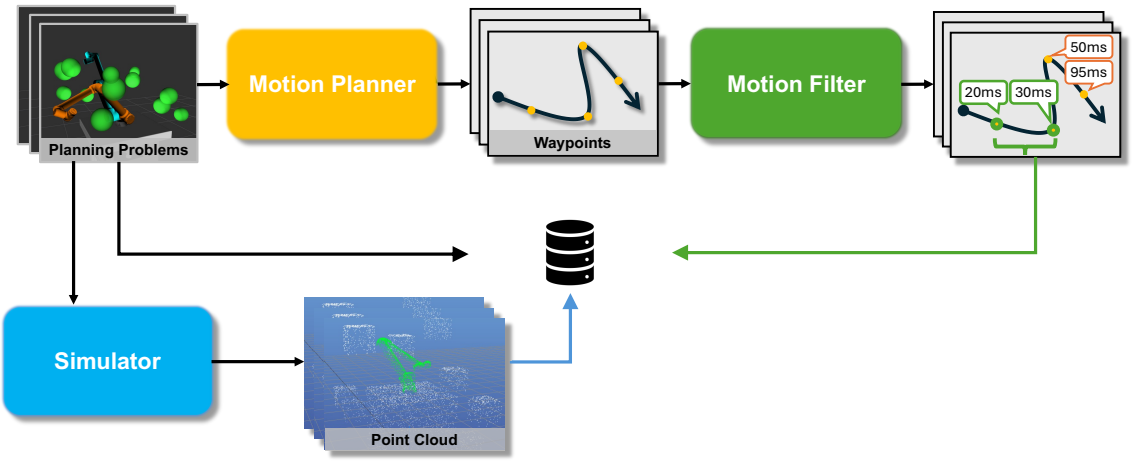


Figure 4.4: Pipeline of dataset collection. The motion planner receives planning problems as input and generates solutions as a list of waypoints. The motion filter computes the planning time from the planning problem's start configuration to each waypoint. It selects waypoints that meet the time constraints, designating them as valid subgoals marked in green. The collected information, including the planning problems, their point cloud representations, the filtered subgoals, and the associated planning times, is then saved in the dataset.

4.3.2 Learning Subgoal Distributions

A generative model is trained to capture the subgoal distribution for a given set of planning problems. This section first introduces the representation of the planning problems used by the generative model, then outlines the training and inference pipeline.

4.3.2.1 Planning Problem Representation

The generative model uses the start and goal configurations and the point clouds of the surrounding environment as inputs to generate subgoals. Both input sources are projected into latent representations using encoders based on neural networks. Then, the subgoals are generated, conditional on the latent representations. The following describes the encoder for joint configurations and the point cloud encoder in detail.

Planning Problem Encoder To achieve higher resolution for the reconstruction in joint space, the following positional encoding with L levels

$$\boldsymbol{x}_{\mathrm{PE}} = [\boldsymbol{x}, \cos \boldsymbol{x}, \sin \boldsymbol{x}, ..., \cos\left(2^L \boldsymbol{x}\right), \sin\left(2^L \boldsymbol{x}\right)]. \tag{4.13}$$

Positional encoding is used in recent methods like Neural Radiance Field (NeRF) [108] to capture the high-frequency part of the data. After projecting the joint configuration to higher dimensions, an additional multi-layer perceptron is used to process the data.

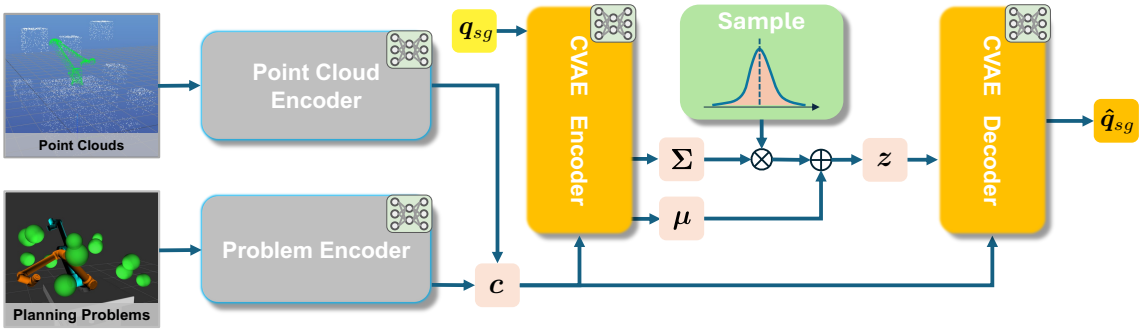


Figure 4.5: Pipeline of training CVAE to capture the spatial subgoals. Modules that use neural networks are marked with a symbol in the upper right corner.

Point Cloud Encoder As shown in Figure 4.3, the point cloud contains additional masks for the environmental obstacles (shown in white) and the current robot start state (shown in green). An architecture based on PointNet++ [133] is adopted. PointNet++ learns a hierarchical point cloud representation and is robust against various sampling densities of point clouds. Architectures based on PointNet++ have been used and proven effective in a wide range of point cloud processing tasks [133, 25, 110]. In addition, PointNet++ integrates PointNet [132] as a sub-component. Partially observed point clouds can be effectively processed using PointNet, even if trained only on fully observed scenes [132]. Specifically, the PointNet++ architecture uses three set abstraction groups followed by three fully connected layers. In the first set abstraction layer, an iterative farthest point sampling (FPS) [133] is performed to create a set of 512 points. FPS aims to find a representative subset of points that capture the essential characteristics of the entire point cloud. By selecting points as far apart as possible, farthest point sampling aims to preserve the overall structure and diversity of the point cloud data. A grouping query within 5cm

of at most 128 points is then performed on the sampled set of 512 points. At the end of the final abstraction, a local PointNet [132] consisting of layers of size [1, 64, 64, 64] is created. The second set of abstractions has a lower resolution. It samples 128 farthest points and then groups at most 128 points within a 30cm radius. The associated PointNet consists of layers of size [64, 128, 128, 256]. The third set of abstraction skips the farthest point sampling and groups all points together. The subsequent final PointNet consists of layers of size [256, 512, 512, 1024]. Finally, after the set abstraction layers come the three fully connected layers with dimensions [4096, 4096, 2048], respectively. Between these layers, group norm and Leaky ReLU is used. This point cloud encoder has 17.8 million parameters, and its output is an embedding of 2048 dimensions.

4.3.2.2 Training

The previous section introduced the dataset's components and generation pipeline. This section describes how to train a generative model to predict subgoals given the whole planning problem. Previous works [60, 88] showcase that a conditional variational autoencoder (CVAE) [154] is capable of generating samples along the optimal path or in the bottleneck region. The fundamental idea behind them is that CVAE can capture the latent representation and generate samples similar to the dataset, conditioning on the environment representation.

The subgoal generation problem aims to train a model to generate samples similar to the dataset mentioned in Section 4.3.1, fulfilling the spatial and temporal requirements. Since the dataset only includes subgoals with a planning time of less than 50 ms, the temporal requirement is implicitly included in the data distribution.

As introduced in the preliminaries Section 4.1, CVAE maps the data $\boldsymbol{x}$ to a latent distribution $\mathcal{N}(\boldsymbol{\mu}(\boldsymbol{x}, \boldsymbol{c}), \boldsymbol{\Sigma}(\boldsymbol{x}, \boldsymbol{c}))$ and then reconstructs the data as $\hat{\boldsymbol{x}} = f_\phi(\boldsymbol{z}, \boldsymbol{c})$ from samples $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}(\boldsymbol{x}, \boldsymbol{c}), \boldsymbol{\Sigma}(\boldsymbol{x}, \boldsymbol{c}))$, conditioning on $\boldsymbol{c}$.

In subgoal generation, the condition $\boldsymbol{c}$ refers to the planning problem description $\{\boldsymbol{q}_s, \boldsymbol{q}_g, \mathcal{S}\}$. The data example $\boldsymbol{x}$ refers to the corresponding subgoals $\boldsymbol{q}_{sg}$ of the planning problem in the joint configuration space. To adapt the loss to fit in this context, the loss described in Eq. (4.12) is modified as

$$J = \beta \mathcal{D}_{\mathrm{KL}} \left[\mathcal{N}(\boldsymbol{\mu}_\theta(\boldsymbol{x}, \boldsymbol{c}), \boldsymbol{\Sigma}_\theta(\boldsymbol{x}, \boldsymbol{c}))||\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})\right] + g(\boldsymbol{x}, f_\phi(\boldsymbol{z}, \boldsymbol{c})), \tag{4.14}$$

where $g(\boldsymbol{x}, f_\phi(\boldsymbol{z}, \boldsymbol{c}))$ is a non-linear function capturing the reconstruction error of subgoals in both joint configuration space $g_{\mathrm{joint}}(\cdot)$ and the task space $g_{\mathrm{FK}}(\cdot)$

$$g(\boldsymbol{x}, f_\phi(\boldsymbol{z}, \boldsymbol{c})) = \alpha g_{\mathrm{FK}}(\boldsymbol{x}, f_\phi(\boldsymbol{z}, \boldsymbol{c})) + (1 - \alpha) g_{\mathrm{joint}}(\boldsymbol{x}, f_\phi(\boldsymbol{z}, \boldsymbol{c})). \tag{4.15}$$

Although the model directly outputs joint configurations as subgoals, the loss function includes reconstruction errors in the task space and considers the robot's kinematics. Using the 2-DoF robot in Figure 3.8 as an example, predicting wrong values at the first joint and at the second joint can generally lead to significantly different effects at the robot's end-effector. The changes at the first joint usually result in a more significant robot displacement. Therefore, the g_{FK} is a weighting function that reflects the kinematics of the robot.

With the positional encoding introduced in Eq. 4.13, the loss g_{joint} is written as

$$g_{\text{joint}}(\boldsymbol{x}, f_\phi(\boldsymbol{z}, \boldsymbol{c})) = ||\boldsymbol{x} - f_\phi(\boldsymbol{z}, \boldsymbol{c})||_2^2 + \sum_{l=0}^{L} || \cos(2^l \boldsymbol{x}) - \cos(2^l f_\phi(\boldsymbol{z}, \boldsymbol{c}))||_2^2 + \sum_{l=0}^{L} || \sin(2^l \boldsymbol{x}) - \sin(2^l f_\phi(\boldsymbol{z}, \boldsymbol{c}))||_2^2 \tag{4.16}$$

Points are sampled from the surface of each robot link's corresponding oriented bounding boxes (OBBs) to represent the reconstruction errors in the task space. These points are sampled only once and then used throughout the training. The loss function g_{FK} is written as

$$g_{\text{FK}}(\boldsymbol{x}, f_\phi(\boldsymbol{z}, \boldsymbol{c})) = \sum_{i=1}^{n} ||T_i(\boldsymbol{x}) - T_i(f_\phi(\boldsymbol{z}, \boldsymbol{c}))||_2^2, \tag{4.17}$$

where T_i indicates the forward kinematics transforming the joint configurations to 3-D positions $[x, y, z]$ of the i^{th} point in task space and n denotes the number of points on the OBB surfaces. Orientation is not explicitly considered in the loss function since a group of points can also implicitly express orientation.

4.3.2.3 Inference

After training the CVAE model, it can be deployed to the dynamic environment. At the inference stage, the CVAE's data flow differs slightly from the training stage. The encoder is not involved at the inference stage, shown in Figure 4.2. At the inference stage, a latent variable $\boldsymbol{z}$ is directly sampled from a multivariate Gaussian $p(\boldsymbol{z}) \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. The decoder $f_\phi(\boldsymbol{z}, \boldsymbol{q}_s, \boldsymbol{q}_g, \mathcal{S})$ generates a subgoal configuration based on $\boldsymbol{z}$ and the encoded planning problem $\{\boldsymbol{q}_s, \boldsymbol{q}_g, \mathcal{S}\}$.

The timing of triggering the inference is also an essential factor. Two options can be considered: the inference can be triggered (1) at a fixed frequency or (2) when a new motion plan is needed. In the second option, a new motion plan is needed when a predicted collision between the robot and obstacles is detected or the robot has successfully reached the subgoal. The first option is used for the policy rollouts in Section 4.5.3.

The CVAE uses a Gaussian distribution to approximate the actual data distribution. This assumption results in a problem that the learned distribution $\mathcal{N}(f_\phi(\boldsymbol{z}, \boldsymbol{c}), \boldsymbol{\Sigma}_d)$ overapproximates the actual distribution. When sampling from this learned approximation, the samples may not satisfy the temporal or spatial requirements. To address this issue, an additional module is trained to serve as a critic for the samples.

4.4 Temporal Distributions as Critic

The model introduced in Section 4.3 can generate diverse subgoals for the same planning problem $\{\boldsymbol{q}_s, \boldsymbol{q}_g, \mathcal{S}\}$, see Figure 4.6. This section introduces a critic module that evaluates the generated subgoal candidates and selects the best-suited one for planning. The objective is to select the subgoal that can be planned within the desired bounds while guiding the robot to the final goal.

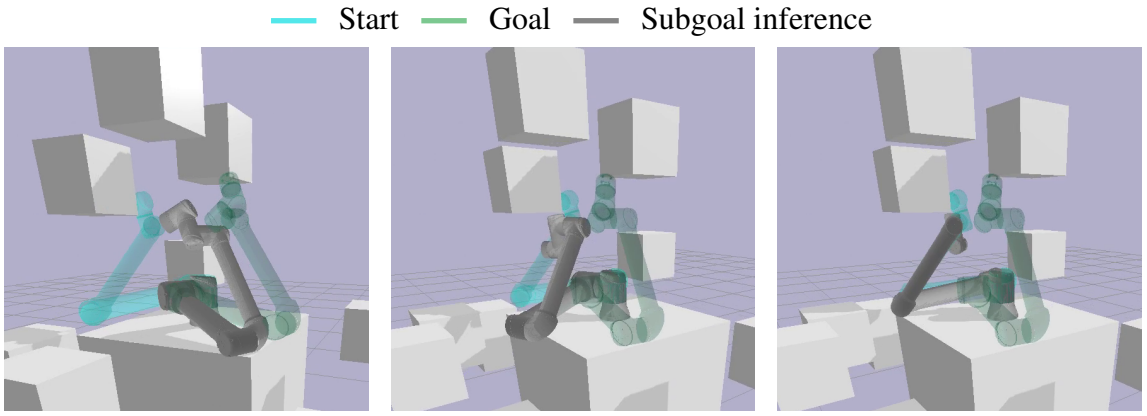


Figure 4.6: Generated subgoals for the problem shown in Figure 4.3. Due to the diversity of the subgoals, a critic is needed to evaluate them.

The critic module comprises a neural network-based time estimator and selection strategies informed by the time estimates. The time estimator captures the distribution of planning time given the start and goal configurations and the surrounding environments, as illustrated in Figure 4.7. The advantage of capturing the distribution rather than learning a classifier is that it can serve as a surrogate measure of planning effort. While a classifier provides a binary classification indicating whether a subgoal is good, multiple metrics can be defined based on the planning effort. Combining these metrics in different ways, diverse strategies can be used to define and evaluate the goodness of subgoal candidates. This section first details how to learn the distribution of temporal effort, and then develops two metrics based on the learned distribution.

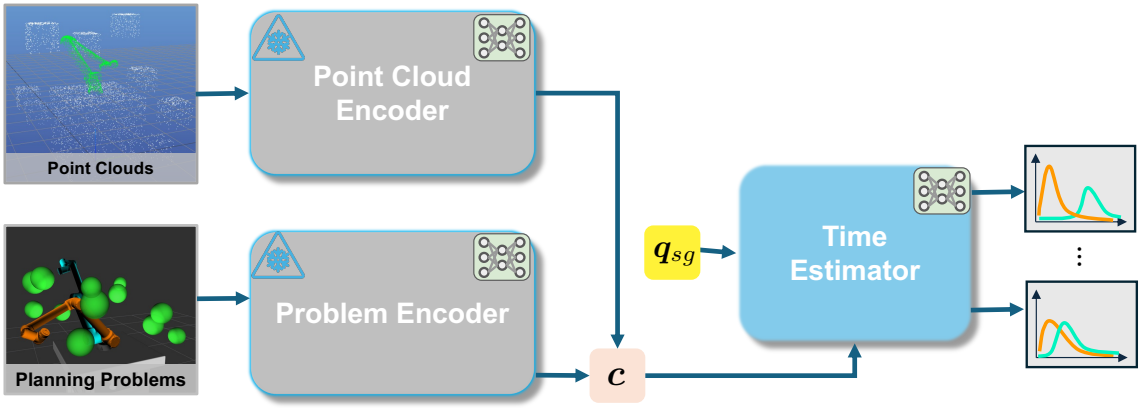


Figure 4.7: Pipeline of training the time estimator. Modules that use neural networks are marked with a symbol in the upper right corner. The parameters in the point cloud encoder and the problem encoder are frozen and don't change during training.

4.4.1 Capturing Temporal Distributions

The neural-network-based time estimator $f_\psi(\boldsymbol{q}_s, \boldsymbol{q}_{sg}, \mathcal{S})$ takes the start configuration $\boldsymbol{q}_s$, the subgoal candidate $\boldsymbol{q}_{sg}$ and the environment $\mathcal{S}$ as input to estimate the distribution of the planning time, where ψ denotes the parameters of the neural network.

The dataset described in Section 4.3.1 is used to train the time estimator. Each entry in the dataset includes a problem description and the planning time for 20 runs from the start configuration to each waypoint. The planning time in the dataset ranges from several milliseconds to seconds. The training includes all waypoints in the dataset, not just those with planning times under 50 milliseconds. Including all waypoints can broaden the prediction range and prevent the model from overfitting to a narrower range than 50 milliseconds. Ideally, the estimated distribution should maximize the accumulated likelihood of the data collected from the 20 runs. This distribution serves as a proxy for the planning effort for the given problem and is correlated with the number of collision checks performed. It is worth noting again that these predictions apply only to the planner used to collect the data. An example of the planning time distribution for a problem in the dataset is shown in Figure 4.8. Two statistical models are used to approximate the underlying data distribution, i.e., the normal and log-normal distributions.

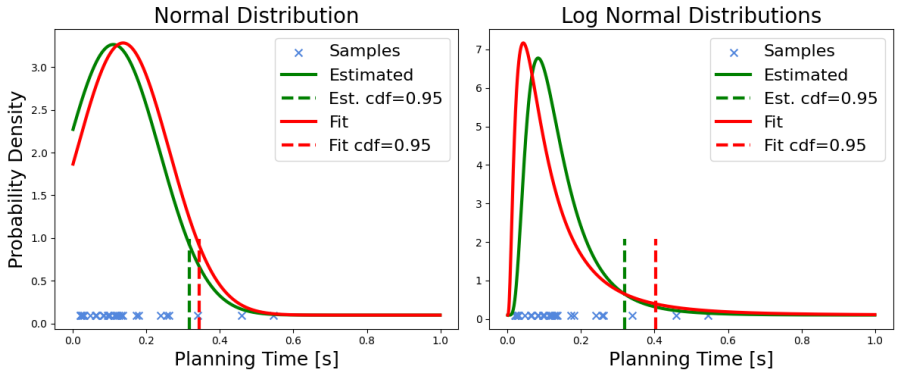


Figure 4.8: Samples of planning time described by normal and log-normal distributions. Red: statistically determined. Green: estimated by a model. The figure was adapted from a conference publication [54].

Normal distribution is also known as the Gaussian distribution. It is symmetric and can be determined by two parameters, mean μ and standard deviation σ. Since the planning time is a one-dimensional variable, a scalar expression is used to represent both parameters. Furthermore, because the planning time is always positive, the left tail of the distribution may get cut off, which means the model may not fully capture the data accurately.

Log-normal distribution describes the data whose logarithm is normally distributed. In other words, given a set of normally distributed data X, log-normal distribution describes the transformed data $Y = e^X$. The data from a log-normal distribution is always positive and skewed, which coincides with the planning time. As shown in Figure 4.8, the data has a long tail in the positive direction. Similar to the normal distribution, a one-dimensional log-normal distribution can be expressed by two parameters μ and σ.

Given a set of data points, the parameters μ and σ can be fit by maximizing the likelihood. This can be easily done using most modern machine learning libraries, such as PyTorch or scipy. Different from fitting, the goal for the time estimator is to estimate the parameters given $\{\boldsymbol{q}_s, \boldsymbol{q}_{sg}, \mathcal{S}\}$. Intuitively, the time distribution depends on the complexity of a planning problem defined by the start-goal query $\{\boldsymbol{q}_s, \boldsymbol{q}_{sg}\}$, and the environment $\mathcal{S}$. Specifically, the same encoded latent representation of $\{\boldsymbol{q}_s, \boldsymbol{q}_{sg}, \mathcal{S}\}$ in Section 4.3.2 are used to derive the distribution. A neural network parameterized by ψ takes this encoded representation as input and outputs the parameters μ, σ of a 1-D distribution. With the

same goal of maximizing the likelihood, the loss function to train the neural network is written as

$$J(T, \hat{\mu}, \hat{\sigma}) = \underbrace{-\frac{1}{M}\sum_{m=1}^{M} \log p(t_m|\hat{\mu}, \hat{\sigma})}_{\text{negative log-likelihood}} + w((\hat{\mu} - \mu)^2 + (\hat{\sigma} - \sigma)^2), \tag{4.18}$$

where T is the planning-time random variable, t_m is the planning time of one run collected by the dataset, and M refers to the number of total runs for one planning problem. Minimizing the negative log-likelihood is equivalent to maximizing the likelihood. In addition to the negative log-likelihood, we add a mean square error (MSE) loss between the parameters μ and σ empirically derived from the data t_m and the output of the time estimator $\hat{\mu}$ and $\hat{\sigma}$, weighted by w. This term serves a regularization purpose, balancing the mean and variance and avoiding mode collapse. As shown in Fig. 4.1a, the gradient of the time estimator does not backpropagate to the encoding blocks. Examples of learned distribution using normal and log-normal models are shown in Figure 4.8.

4.4.2 Metrics for Subgoal Selections

The time estimator estimates the surrogate planning effort between two joint configurations $\boldsymbol{q}_s$ and $\boldsymbol{q}_{sg}$ conditioning on the environment surroundings $\mathcal{S}$. Compared to simply learning a binary classifier, the surrogate planning effort enables the extension to design diverse metrics to quantitatively evaluate a subgoal candidate and finally make an informed selection.

Two aspects are considered for the evaluation: (a) whether it is likely to plan to the subgoal within the given time bounds and (b) whether the subgoal is goal-oriented, i.e., can progressively lead the robot to approach the final goal. Two extreme cases illustrate why it is important to consider both aspects simultaneously. In the first extreme case, a model copies the start configuration and uses it as output, making the generated subgoals cover aspect (a). Similarly, in the second extreme case, if a model outputs the goal configuration directly, aspect (b) is logically covered. However, neither of these models can contribute to planning in dynamic environments. Therefore, it is desirable that the subgoals address both aspects simultaneously.

To this end, a start-to-sample metric is designed to characterize aspect (a), and a goal-to-sample metric characterizes aspect (b). It is worth noting that the start and goal are interchangeable in planning time, and the learned model should produce similar results if the start and goal configurations are swapped.

Sampling-based motion planning is a stochastic process, and every planning problem can have an unusually long planning time. This is also known as tail risk. In the subgoal selection, the focus is not on tail-risk events but on the main part of the distribution. Thus, the start-to-sample metric uses the cumulative density function (CDF) $c(\cdot)$ of the estimated distribution to characterize the subgoal. The main part of the distribution can be represented by t_{95}, where $c(t_{95}) = 0.95$, indicating that 95% of the values drawn from this distribution are supposed to be smaller than t_{95}. The value of t_{95} is used to denote the *planning effort* of a problem since it represents 95% of the planning cases and gives a straightforward evaluation of how hard is the planning problem. t_{95} can be analytically

determined given the parameters μ and σ. Depending on the needs, one can set this confidence level to a higher value. The start-to-sample and goal-to-sample metrics are described in detail below.

Start-to-Sample Metric The start-to-sample metric estimates the planning effort between the start and the generated subgoal candidates. The time estimator uses the start and the subgoal configuration as input and predicts the distribution of the planning time $p(t|\hat{\mu}, \hat{\sigma})$. The variable t_{95} determined by $p(t|\hat{\mu}, \hat{\sigma})$ is used as a surrogate metric to represent the planning effort.

Goal-to-Sample Metric The goal-to-sample metric estimates the planning effort between the goal and generated subgoal candidates, addressing the aspect (b) mentioned earlier. In search algorithms, cost-to-go heuristics estimate the effort required to reach the goal and are used to prioritize exploration. Similar to this concept, the goal-to-sample metric uses the planning effort t_{95} between the goal and generated subgoal candidates to describe whether it is easy to plan from the subgoal to the final goal or vice versa. A lower t_{95} value indicates an easier planning problem, suggesting that the corresponding subgoal is goal-oriented.

4.4.2.1 Selection Strategies

Two selection strategies, termed best-effort and goal-oriented, are designed using the metrics mentioned above. These selection strategies are applied to the subgoals candidates generated by the CVAE.

The **best-effort** strategy uses only the start-to-sample metrics and selects the subgoal candidates whose t_{95} is less than the desired upper bound of the planning time t_d. If multiple subgoal candidates meet this requirement, then the subgoal will be randomly selected among them. If there are no qualified candidates, the one with the least t_{95} value will be selected. This strategy is designed to select the candidates that potentially need the least effort to plan, regardless of whether they can lead the robot to the goal or not.

The **goal-oriented** strategy builds on the best-effort strategy and uses the goal-to-sample metrics to select qualified candidates. Instead of randomly selecting, it ranks the samples by the goal-to-sample metric and selects the one with the lowest value, i.e., the most goal-oriented. If there are no qualified candidates, the goal-oriented and best-effort strategies are equivalent.

4.4.3 Planning Range Shaping

A larger planning range usually means a larger search space and, therefore, a longer planning time for sampling-based motion planners. For some complex planning problems, the planning range has to be large enough to avoid the infeasible region in the collision space. As subgoals break down complex problems into pieces, shaping the planning range accordingly for every small planning problem becomes possible and beneficial. This concentrates the samples in the region where the planner can find a solution. A similar concept can be found in batch-informed trees (BIT) [39]. The difference is that methods in the BIT family derive bounds defined by high-dimensional ellipsoids after an initial solution is found. In contrast, PLS directly sets the planning range based on the planning problems.

For an N-DoF robot arm, we set the lower bounds $\boldsymbol{b}_l \in \mathbb{R}^N$ and upper bounds $\boldsymbol{b}_u \in \mathbb{R}^N$ for joints depending on the mobility of each joint and the planning problem. In practice, we constrain the search space with greater paddings for the joints that can produce large motion, usually the ones close to the robot base, depending on the robot's kinematics.

4.5 Evaluation

As the method is proposed to generate subgoals that account for the spatial and temporal aspects, experiments are designed to examine the subgoals regarding these two aspects. With the experiments, the following questions should be answered:

Q4.1 Can generated samples be planned within a short period?

Q4.2 Can the generated samples guide the robot toward the final goal?

Q4.3 Can the learned model generalize to unseen scenarios?

The results in this section are collected with a model trained by the dataset mentioned in Section 4.3.1. The checkpoint with the lowest loss on the validation set is chosen as the best checkpoint and used to compute the results reported in this section.

4.5.1 Evaluation in Simulation

Two simulation experiments were conducted to answer Q4.1 and Q4.2. Both experiments use the same trained spatial subgoal generator model. The generator has a neural-network-based encoder with dimensions [1024, 512, 512, 256] that encodes the input to the mean and variance of a latent variable z with dimension 128. The sampled latent variable is then used as input to reconstruct the subgoals using a decoder with dimensions [1024, 1024, 512, 256]. The time estimator uses two residual blocks, each with 3 fully connected layers with a dimension of 128. For the loss functions, $\beta = 7$ in Eq. (4.14) and $w = 0.01$ in Eq. (4.18). During the training, the learning rate is $5e^{-4}$. The computation time of all planners is collected with a single core of an Intel i9-9900K CPU.

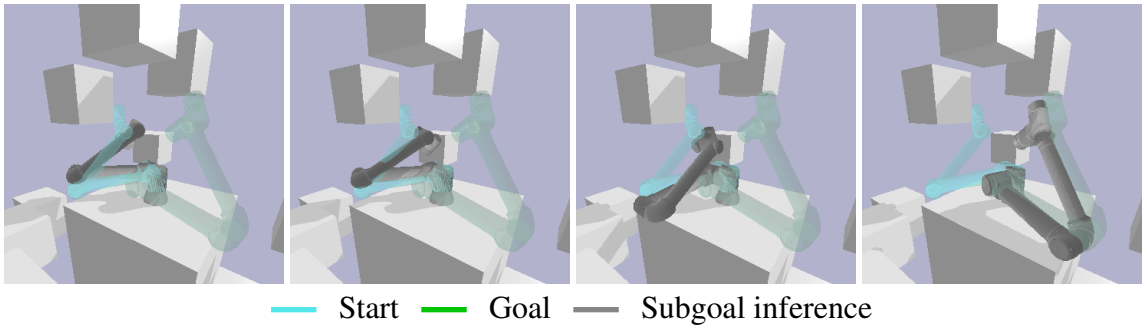


Figure 4.9: Rollouts of the learned model. The previously predicted subgoal is used to start the configuration for the current problem. The ground truth of the subgoals see Figure 4.3.

4.5.1.1 Planning Time Fulfillment

To answer the question **Q4.1**, the trained model is deployed to 1000 planning problems. The planning problems are first solved using the RRTConnect planner for 30 runs, all of which take at least 0.05 s. This guarantees the complexity of the test problems. The planning time for these runs serves as a baseline to highlight the improvements achieved by using subgoals.

At this stage of the experiment, the trained model outputs subgoals for each problem but does not work iteratively toward the final goal. The sole evaluation metric for this experiment is the planning time from the start configurations to the subgoals using RRT-Connect, the same planner used to generate the dataset. Throughout the evaluation, the trained model generates subgoal candidates in batches of 16 for each planning problem. Then, these subgoal candidates are selected either randomly or using the best-effort strategy. The results reflect the trained model's ability to generate samples while meeting the requirement for bounded planning time.

Since the test planning problems are selected with planning time greater than 0.05 s as a requirement, the baseline that solves the complete problem using RRTConnect naturally handles 0% of cases that meet the desired time bound t_d. Other planners can be used for baselines. However, the focus at this stage is on highlighting the distinction between planning subproblems and complete problems. If using other planners yields an improvement, the improvement attributable to the choice of planners can benefit both the subproblem and the complete problem. Using other planners does not affect the decomposition of complex problems into small, easy problems.

	Success Under Time Budget t_b [%]			Planning Time [s]	
	$t_b = 0.05$	$t_b = 0.1$	$t_b = 0.2$	Mean	Std.
RRTConnect (baseline)	0	3.8	12.9	1.153	1.86
Random selection	65.4	74.9	83.2	0.143	0.328
LogNormal	67.0	73.6	79.9	0.198	0.434
Normal	71.3	78.9	86.2	0.122	0.211
LogNormal + Shaping	85.1	90.0	94.1	0.056	0.203
Normal + Shaping	**89.2**	**93.8**	**96.7**	**0.035**	**0.137**

Table 4.1: Results of planning to subgoals once. The pipeline rolls out only once for each planning problem instead of iterating until the final goal is reached.

The results are reported in Table 4.1. For each problem, the RRTConnect algorithm plans the subproblem defined by the start configuration and the selected generated subgoal across 30 runs. The problem is considered successful only if all 30 runs yield a planning time below a specified threshold. The thresholds are set to be 0.05, 0.1, and 0.2 seconds.

While RRTConnect cannot manage to plan within the given planning budget of 0.05 s, 65.4% of the randomly selected generated samples can be planned within 0.05 s. Using the best-effort metric, the ratio of qualified subgoals increased up to 71% using the normal distribution. With the planning range shaping, up to 89.2% of the generated samples were planned within 0.05 s. As the threshold is extended to 0.1 and 0.2 seconds, we gain a big picture of how the planning time of the predicted subgoals is distributed. More than 93.8% of the predicted subgoals can be planned within 0.1 s, while only 3.8% of the complete

problem can be solved within this range. Therefore, the experimental results empirically show that the predicted samples can be planned in a much shorter period compared to the original complete problem. Thus, Q4.1 is answered:

Q4.1 Can generated samples be planned within a short period?

A4.1 While **none** of the original complete problems can be planned within 0.05 s, over 89% of the selected generated subgoals succeeded in being planned within this budget, proving the generated samples effective.

4.5.1.2 Goal Reaching

To answer question **Q4.2**, the trained model is used to sequentially roll out until the final goal is reached or the maximum number of iterations of 10 is reached. The same set of problems as in planning time fulfillment is used in this experiment. Each problem will be rolled out 10 times. Every rollout begins with the initial start configuration defined in the problem. After selecting a subgoal, the RRTConnect planner searches for a solution between the current start configuration and the subgoal, and the planning time is recorded. If no solution is found, the rollout will be aborted and reported as failed. If a solution is found, the predicted subgoal becomes the start configuration for the next iteration, and the point cloud representation gets updated. The same procedure is repeated for the next iteration until the termination condition mentioned above is met. A rollout is considered successful only if the final goal is reached. Using the same subgoal generation model, three subgoal selection strategies are evaluated: random selection, best-effort, and goal-oriented. For the best-effort and goal-oriented strategies, there are two distribution assumptions, i.e., normal and log-normal distribution, making them five different ways to select subgoals.

Planning time and path length between adjacent subgoals, as well as the accumulated planning time and path length, are the metrics used to evaluate performance. These metrics are computed using the RRTConnect planner. In addition, the AIT* planner is used to compute the *optimal path length* for baseline purposes.

			Reached Goal	All Iterations Under Budget t_b [%]			Planning Time [s]		Acc. Length
			[%]	$t_b = 0.05$	$t_b = 0.1$	$t_b = 0.2$	Path	Subgoal	[rad]
Baseline			0	0	3.8	12.9	1.153	-	**17.65** (AIT*)
Random Selection			61.7	82.6	87.4	91.9	0.517	0.061	50.67
Best-effort	+ LogNormal	+ Shaping	81.5	90.1	93.2	95.9	0.208	0.029	25.79
Best-effort	+ Normal	+ Shaping	69.0	**95.2**	**97.0**	**98.4**	**0.149**	**0.016**	23.55
Goal-oriented	+ LogNormal	+ Shaping	**85.2**	89.3	93.3	96.2	0.172	0.030	25.51
Goal-oriented	+ Normal	+ Shaping	82.3	88.5	92.2	96.2	0.214	0.039	25.56

Table 4.2: Planning to final goals. The pipeline rolls out iteratively until the final goal is reached. All results are computed using the RRTConnect planner unless stated otherwise. For the baseline statistics, the AIT* uses a planning budget of 5 seconds to compute the optimal path length.

Results of the baseline and the mentioned variants are shown in Table 4.2. The success rate reported in the table indicates that the method can plan directly or sequentially to

the final goal, with a maximum of 0.05 s per iteration. The maximum allowed number of iterations is set to 10. Since the baseline method, RRTConnect, cannot provide a solution within 0.05 s, its success rate is 0. Other than success rates, the accumulated planning time of the path from the initial start to the final goal as well as the planning time between subgoals are listed in the table. The results of the random selection indicate that a model relying solely on spatial information is ineffective at meeting time constraints or guiding the robot to its goal configuration. Using temporal information to select the candidates generally improves performance in both aspects. Compared to the baseline results regarding the path length provided by AIT*, other variants exhibit a longer path length. An obvious reason is that RRTConnect is used to plan between adjacent subgoals without path smoothing, resulting in a suboptimal path between the subgoals. Other metrics are needed to examine the optimality of the subgoals.

Q4.2 Can the generated samples guide the robot toward the final goal?

A4.2 Using a goal-oriented selection strategy based on log-normal distribution, the model succeeded in 85.2% of the test cases to sequentially guide the robot from the initial start position to the final goal, while the baseline method fails to achieve a single case. Results also show that a model relying solely on spatial information is not effective in either meeting time constraints or successfully guiding the robot to its goal configuration.

4.5.2 Ablation Study

In the previous section, the results for different combinations of selection strategies, statistical models for the planning-time distribution, and activation of planning range shaping are reported in Tables 4.1 and 4.2. This section analyzes the contribution of each component based on these results.

With and without time estimator In Table 4.1, the success rate of planning under the budget $t_b = 0.05$ using time estimator is higher than random selections. In addition to reducing planning time, the time estimator helps select candidates that progressively guide the robot toward the goal; see Table 4.2. When integrating the sample-to-goal metric, the percentage of goals reached is higher for both statistical models.

Statistical models In both Table 4.1 and Table 4.2, the time estimator using normal distribution shows better performance in planning time but worse performance in guiding the robot to the final goal. On the other hand, the log-normal distribution is robust in both experiments. In the goal-reaching experiment, the rollout is autoregressive, meaning that the current input $\{\boldsymbol{q}_s, \boldsymbol{q}_g, \mathcal{S}\}_t$ of the neural network is based on the previously selected subgoal $\boldsymbol{q}_{sg,t-1}$, i.e., $\boldsymbol{q}_{s,t} = \boldsymbol{q}_{sg,t-1}$. The generated subgoals do not belong to the dataset and are unseen by the model. The learned model based on the log-normal distribution demonstrates robustness to these unseen inputs. Therefore, this model is selected for the experiment in Section 4.5.3.

With and without planning range shaping The results show that using planning range shaping is beneficial in both experiments. The planning range shaping limits unnecessary sampling and checking in regions far away from the start and subgoal configuration. However, this cannot be applied to complex planning problems.

4.5.3 Generalization in Unseen Environments

To verify the model's ability to generalize to unseen environments, the model trained on the dataset described in Section 4.3.1 is applied to the environment shown in Figure 4.10. Two UR10e robot arms, marked in gray and pink, share a workspace with two static obstacles: a cube and a table.

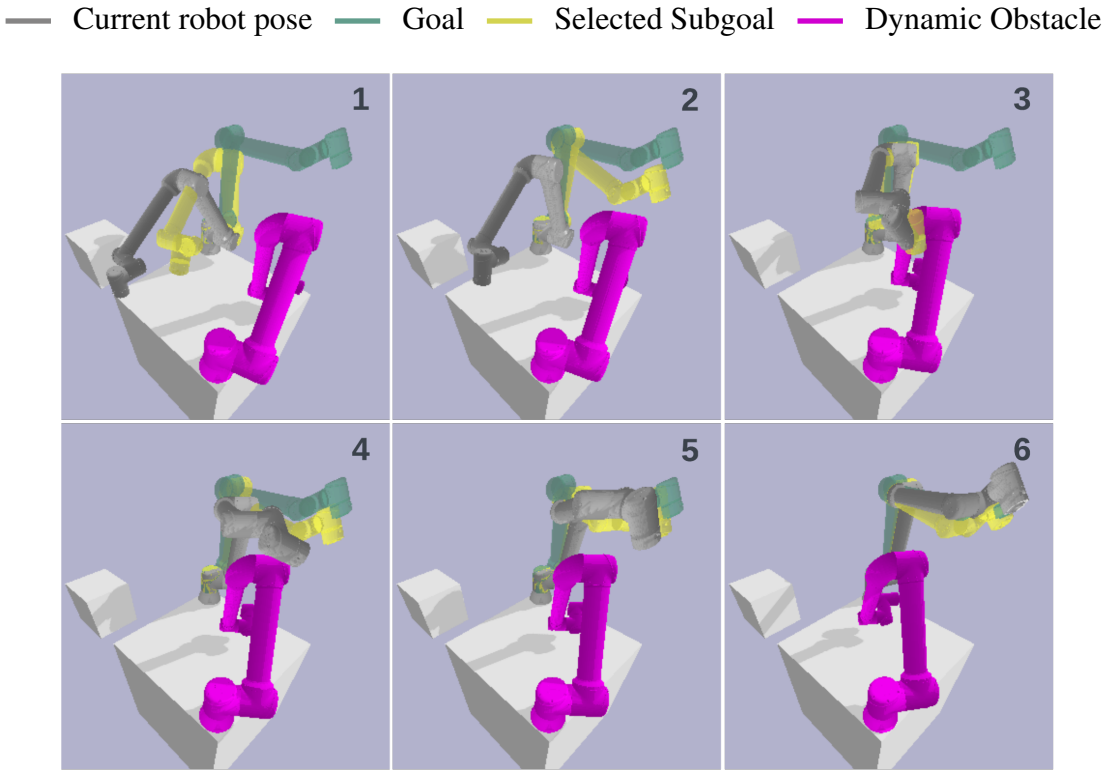


Figure 4.10: Generalization in an unseen environment with two robot arms, marked in gray and pink. The pink robot is the dynamic part of the environment. The generated subgoals iteratively lead the gray robot to the final goal.

In this setup, the pink robot strictly follows pre-defined trajectories and does not adapt to changes in the workspace. It serves as a dynamic part of the environment. The gray robot conducts goal-reaching tasks, assuming that the final goal configuration of the upstream manipulation task is given. Rather than planning directly to the final goal, subgoals are iteratively generated, and an RRTConnect planner plans a solution to each subgoal based on the current state of the environment. An online trajectory generator [126] converts the solution to the executable trajectory. Subgoal generation and planning are repeated at 20 Hz until the robot reaches the desired goal. For each iteration, 16 subgoal candidates are generated, and the log-normal time estimator and the goal-oriented strategy are used for the selection. The choice of subgoal selection strategy is an informed decision based on the results reported in 4.2. Compared with randomly selecting subgoals, this selection strategy yields less noisy motion.

Six snapshots of a rollout are shown in Figure 4.10 in chronological order. The snapshot labeled with "1" presents the initial setup. The first subgoal chosen generally steers the G-robot's end-effector toward its base link, as it appears to be the shortest path to reach the goal when the P-robot is still at a considerable distance. In snapshot "2", the model predicts a subgoal close to the final goal region right away. As the P-robot approaches

the center of the table in snapshot "3", the newly generated subgoal directs the G-robot to elevate to avoid a collision. Finally, it maneuvers over the P-robot and reaches the final goal region.

Q4.3 Can the learned model generalize to unseen scenarios?

A4.3 The model managed to avoid the moving robot in the workspace and reach the goal, although it is solely trained on the dataset with cubic static objects. The capability of generalization is attributed to the representation of the scene using point clouds.

4.6 Limitations and Discussions

The proposed method aims to generate subgoals that not only guide the robot toward its ultimate goal but also can be planned from the current robot configuration within the desired time period. Existing generative models that use only spatial information are extended by integrating temporal information to select appropriate subgoals from a batch of candidates. Although experiments show that the proposed method can achieve shorter planning time for both subgoals and the accumulated path, several limitations need to be addressed.

The first limitation is that the proposed method takes only the current state of the environment as input and neglects the history of the states. Neglecting the history of the states implies that the learned policy is not able to predict how the environment evolves in the upcoming time steps, making the predicted subgoals highly time-sensitive. This limitation comes from the dataset and the model architecture. Theoretically, episodic reinforcement learning or sequential models may be well-suited to solving this problem. This limitation is addressed in Chapter 5 using episodic reinforcement learning.

The second limitation is that the time estimator serves only as a critic. Ideally, the desired time constraints can be used as the condition of the generative model, and the time estimator can be used as a score function[2] whose gradients guide the subgoal candidates to the region where the time constraints are not violated. Therefore, replacing the generative model architecture and the time estimator with a diffusion-based method [21] is a viable option.

[2]The concept of score function is introduced in denoising diffusion probabilistic models [49].

5 Learning Motion Refinements for Spatiotemporal Awareness

Previous chapters introduced two methods, HIRO and PLS, that aim to reduce the computational time of a single planning process while keeping it bounded. The result is a collision-free, feasible path given the current state of the environment. However, these two methods consider only the current geometric state of the environment and neglect the temporal aspect. In this context, the *temporal* aspect refers to how the environment evolves along the time horizon. These two methods are categorized as geometric planning methods. In dynamic environments, geometric planning methods must constantly repeat the planning pipeline until the robot reaches the final goal. Due to their short-sightedness over the time horizon, they often become stuck in undesired local minima, for example, when avoiding obstacles by moving the robot along with them. This chapter investigates how to adapt the pre-planned solution, e.g., from HIRO and PLS, and gain foresight along the time horizon. The research question is formulated as

How can planned solutions be adjusted to changes in the environment along the time horizon?

To take into account the environment factor along the time horizon, a simple option is to extend the geometric planning problem with an additional time horizon and form a planning problem in $\mathbb{R}^{N+1}$ space [44, 153], where N denotes the DoFs of the system, and 1 indicates the additional temporal dimension. This option assumes that full knowledge of the environment over the time horizon is accessible and explicitly considers this knowledge. However, these methods suffer from extremely long planning times of tens of seconds [44], and the assumption of full environment knowledge is not reasonable in many cases. Another important perspective is that robots cannot only avoid but also interact with the environment in dynamic environments, such as catching a flying object. In this scenario, there are infinitely many possible goal configurations over the time horizon. Due to the undetermined goal selection, these methods are unsuitable for dynamic environments.

On the other hand, episodic reinforcement learning (ERL) methods treat planning problems as black-box optimization problems and implicitly encode how environments evolve in the policy. Given the initial observations, the ERL methods generate full trajectories that account for possible environmental changes and interactions over the horizon. This assumes that environments do not evolve completely stochastically and instead follow specific rules; for example, a flying tennis ball follows roughly a parabolic trajectory. This assumption is also valid in the real world. Generating a full trajectory requires the methods to understand the environment's dynamics and anticipate its states while interacting with it, especially for contact-rich tasks. When the system dynamics have evolved differently from the encoded anticipation, e.g., new objects enter the environment, the

policy can be triggered to adapt to the new observation [118]. This is particularly helpful for long-horizon tasks or for environments whose dynamics are difficult to distill. A typical example is multiple robots collaborating on the same task, where each robot is a moving object, and the environment changes continuously as the task progresses.

Usually, ERL methods use a single Gaussian to represent a policy, with a fixed number of movement-primitive bases. The learned policy may be insufficiently expressive for certain complex scenarios, even when these scenarios are part of the task-context distribution. Combining ERL with the planning methods proposed in the previous chapters is a good solution to address this problem. While these two geometric planning methods do not account for potential environmental changes over time, their solutions provide a good prior for ERL methods. Using these solutions as reference trajectories, the ERL methods need only account for potential environmental changes and refine the trajectories accordingly.

Compared to solely using ERL without references, two key advantages are that (a) it reduces the complexity of learning, thereby improving sample efficiency, and (b) it preserves dexterous robot behaviors that are challenging to learn from scratch. Furthermore, the inference, i. e., a forward pass of the neural network, takes a deterministic computation time, which ensures that the algorithm can rapidly respond in a dynamic environment.

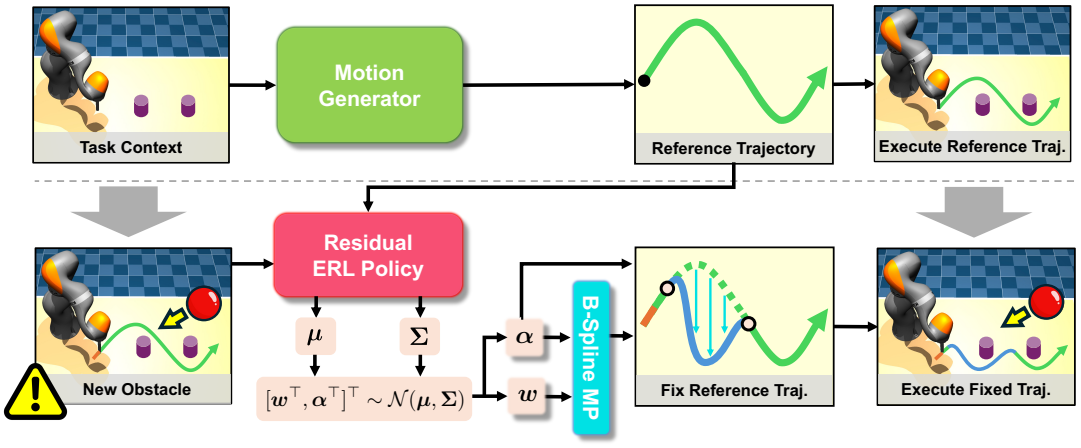


Figure 5.1: Illustration of the MoRe-ERL pipeline. The robot follows the reference trajectory provided by a motion generator, with the executed segment shown in – and the remainder in –, based on the task context. When the task context changes, such as the appearance of new obstacles, MoRe-ERL identifies critical segments on the remaining reference trajectory (–) using learned parameters $\boldsymbol{\alpha} = [\alpha_s, \alpha_e]^\top$ and parameterize residuals $f(\boldsymbol{w})$ for the selected segments using B-spline-based movement primitives. The adjusted trajectory, after applying these residuals, is shown by the solid blue-green curve (–).

In this chapter, to answer the research question above, a general framework for **Mo**tion **Re**siduals using **ERL** (MoRe-ERL) is proposed to generate motion residuals to refine previously planned task-related reference trajectories. This framework can seamlessly plug into arbitrary ERL methods and motion generators, such as HIRO and PLS, introduced in the previous chapters. Given a reference trajectory from these motion generators, MoRe-ERL learns residuals and refines the reference into safe, feasible, and efficient task-specific trajectories while implicitly accounting for system dynamics. As shown in Figure 5.1, MoRe-ERL (1) identifies the trajectory segments that require modification while pre-

serving critical task-related behaviors, and then (2) it generates trajectory refinements for these segments using B-Spline-based movement primitives to ensure smooth transitions.

Three refinement strategies are investigated, and their performance is evaluated in multiple simulation tasks. The segment identification and the trajectory refinements are jointly learned as a correlated policy. Having the reference trajectory as prior knowledge, MoRe-ERL significantly outperforms training from scratch using ERL methods, achieving superior sample efficiency and task performance. The main contributions are

- The first RL algorithm that combines ERL and residual learning to refine reference trajectories, offering spatiotemporal awareness and achieving superior sample efficiency and task performance.
- An end-to-end policy that identifies the segments of reference trajectories needing modification and parameterizes movement primitives as residuals.
- Three trajectory refinement strategies using B-spline-based movement primitives, which enforce smooth transitions between the reference and the fixed trajectories.

In the remainder of this chapter, Section 5.1 introduces the preliminaries, including episodic reinforcement learning and B-spline-based movement primitives; Section 5.2 defines the problem formally; Section 5.3 describes how motion refinements are learned using ERL; and Section 5.4 shows the experimental results and ablation studies in simulations as well as in the real world.

5.1 Preliminaries

5.1.1 Episodic Reinforcement Learning

Episodic Reinforcement Learning (ERL), see [174] and [82], predicts an entire sequence of actions to accomplish a task by optimizing cumulative rewards without explicitly modeling detailed state transitions within an episode. ERL methods usually predict a weight vector $\boldsymbol{w}$, given the task context. This weight vector is then used to parameterize a complete trajectory $\boldsymbol{q}(t) = f(\boldsymbol{w})$ for $\boldsymbol{q}(t) \in \mathbb{R}^N$ and $t \in [0, T]$, where N corresponds to the dimensionality of the trajectory space, such as the DoFs in a robotic system, T represents the trajectory duration, and $f(\cdot)$ indicates a generic function for trajectory parameterization using a motion generator. The predicted trajectory can be directly utilized as per-step actions or as input to a trajectory tracking controller for computing lower-level motor commands. Given the initial state $\boldsymbol{s}_0 \sim p(\boldsymbol{s}_0)$ specifying the starting configuration and task context, the goal of ERL is to find a weight vector $\boldsymbol{w}$ that maximizes the return $R(\boldsymbol{s}_0, f(\boldsymbol{w}))$ after executing the trajectory $\boldsymbol{q}(t) = f(\boldsymbol{w})$. The ERL learning objective is generally expressed as:

$$J = \mathbb{E}_{p(\boldsymbol{s}_0), \pi_\theta(\boldsymbol{w}|\boldsymbol{s}_0)} \left[R(\boldsymbol{s}_0, f(\boldsymbol{w})) - V_\phi(\boldsymbol{s}_0) \right], \tag{5.1}$$

where π_θ denotes the policy parameterized by θ, often implemented using a neural network. The return $R(\boldsymbol{s}_0, f(\boldsymbol{w})) = \sum_{t=0}^{T} \gamma^t r_t$ is the cumulative reward obtained by following the trajectory, where γ is the discount factor, and r_t is the reward at time step t. The term $V_\phi(\boldsymbol{s}_0)$ represents a value estimator of the state $\boldsymbol{s}_0$, parameterized by ϕ, and acts as a baseline to stabilize training [160].

Compared to traditional step-based RL (SRL) methods like PPO [148], ERL shifts the solution search from the per-step action space $\mathcal{A}$ to a parameterized trajectory space $\mathcal{W}$, predicting trajectory parameters as $\pi(\boldsymbol{w}|\boldsymbol{s})$. This often facilitates broader exploration and results in smooth, correlated motion trajectories [99]. Additionally, the learning objective in Eq. (5.1) relaxes the requirement for Markovian rewards [160], which enforces that the reward r_t at a given time step t depends only on the current state s_t and action a_t. Step-based RL methods such as SAC [45] rely on temporal difference (TD) learning. This requires Markovian rewards to assign value credits to per-step actions and states properly. In contrast, ERL assigns task credit to the entire trajectory episode parameterized by $\boldsymbol{w}$ by aggregating per-step rewards. This removes the requirement for rewards to be Markovian, allowing for the use of delayed or history-dependent rewards, referred to as *non-Markovian rewards* [160]. Intuitively, non-Markovian rewards offer greater flexibility and simplicity in task design [117], as they rely on fewer assumptions compared to their Markovian counterparts.

Usually, ERL methods use MPs as the motion generator. MPs can encapsulate trajectories from a lower-dimensional parameter space, thereby reducing the problem complexity.

5.1.2 Using Movement Primitives in ERL

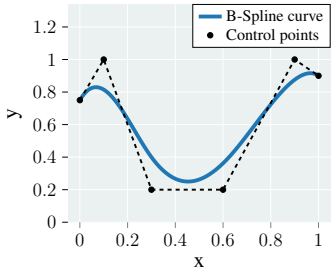
Parameterizing trajectories using MPs [144, 121, 98] is central to ERL methods. This section first describes Probabilistic Movement Primitives (ProMPs) and then introduces B-spline-based movement primitives (BMPs) using the same formalism as ProMPs. ProMPs [121] represent a trajectory $\boldsymbol{q}(t)$ using a linear basis function:

$$\boldsymbol{q}(t) = f(\boldsymbol{w}) = \boldsymbol{\Phi}(\frac{t}{T})^\top \boldsymbol{w} = \boldsymbol{\Phi}(u)^\top \boldsymbol{w}, \tag{5.2}$$

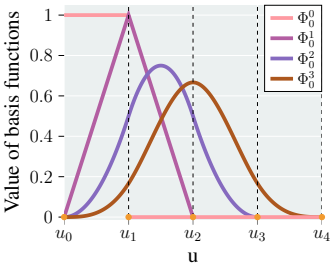
where $u = t/T \in [0, 1]$ denotes the normalized time, also called the phase variable. The term $\boldsymbol{\Phi}(u) = [\Phi_1(u), \Phi_2(u), ..., \Phi_{N_b}(u)]^\top$ represents N_b basis functions for each dimension in the trajectory space, evaluated at u. The weight vector $\boldsymbol{w} = [w_1, w_2, ..., w_{N_b}]^\top$ controls the trajectory shape by scaling the basis functions. Typically, a neural network is used to predict the mean $\boldsymbol{\mu}_{\boldsymbol{w}}$ and covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{w}}$ and the weight vector is sampled from the distribution $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}_{\boldsymbol{w}}, \boldsymbol{\Sigma}_{\boldsymbol{w}})$. For non-periodic trajectories, ProMPs often utilize radial basis functions (RBF) as the basis functions, with their centers uniformly distributed in the phase space $[0, 1]$.

ProMPs are advantageous due to their simple linear representation, which enables fast computation and probabilistic modeling (see [121] for details). However, ProMPs lack mathematical support for enforcing specific boundary conditions at the trajectory's start and end points. This limitation restricts their ability to generate new trajectories that seamlessly transition from an existing one. However, this is a critical requirement for real-world scenarios where frequent trajectory switching is necessary. Recent works [78, 102] address these inherent limitations by replacing the RBF functions in ProMPs with B-splines. The resulting model, B-spline-based movement primitives (BMPs), retains the linear basis function representation of ProMPs while supporting an arbitrary number and order of trajectory transition conditions. Mathematically, these conditions are known as boundary conditions.

Definition of BMP. The basis functions of BMP, $\boldsymbol{\Phi}^P(u) = [\Phi_1^P(u), \Phi_2^P(u), ..., \Phi_{N_b}^P(u)]^\top$, are defined as P-th order polynomial functions, where $0 \leq P < N_b$. These basis functions



(a) B-Spline in 2D



(b) Basis functions

Type	Condition	$w_1^{(i)}$ = ?
Position	$w_1^{(0)} = q(t_0)$	$w_1^{(0)}$
Velocity	$w_1^{(1)} = \dot{q}(t_0)$	$\frac{P}{\delta}(w_2^{(0)} - w_1^{(0)})$
Acceleration	$w_1^{(2)} = \ddot{q}(t_0)$	$\frac{P(P-1)}{\delta^2}(w_3^{(0)} - 2w_2^{(0)} + w_1^{(0)})$

(c) Mapping between boundary conditions and control points

Figure 5.2: Illustration of BMPs: (a) A clamped B-spline curve in 2D parameterized with six control points. (b) Basis function of different orders using a recursive formulation, where Φ_0^p denotes the basis function of p^{th} order for the 0^{th} control point. The knots u represent the change of time. (c) Mapping between boundary conditions and control points at t_0. Boundary conditions are defined by aligning the the control point $w_1^{(i)}, i = 0, 1, 2$ to given initial position $q(t_0)$, velocity $\dot{q}(t_0)$ and acceleration $\ddot{q}(t_0)$ conditions, respectively. The higher-order control point $w_1^{(i)}$ can be represented using the 0-th order control points.

are constructed over M definition intervals, equally divided by $M + 1$ knots $u_0, ..., u_M$, with $u_0 = 0$ and $u_M = 1$. Typically, $M = N_b + P$ [131] and the intervals between two adjacent knots have the same length δ. In B-splines, the weights $\boldsymbol{w}$ are also interpreted as *control points*, which define a convex hull that bounds the trajectory, see Figure 5.2a. Each basis function Φ_n^P, where $n \in [1, N_b]$, is defined recursively from order 0 to order P [131]. To illustrate this recursive process, we denote intermediate orders with the index p, where $p \in [0, P]$. For $p = 0$, the basis functions are piecewise constant:

$$\Phi_n^{p=0}(u) = \begin{cases} 1 & \text{if } u_n \leq u < u_{n+1}, \\ 0 & \text{otherwise.} \end{cases} \tag{5.3}$$

For $p > 0$, each basis function $\Phi_n^p(u)$ is computed by interpolating between two corresponding lower-order basis functions, $\Phi_n^{p-1}(u)$ and $\Phi_{n+1}^{p-1}(u)$, using coefficients j_n and j_{n+1}, respectively:

$$\Phi_n^p(u) = j_n \, \Phi_n^{p-1}(u) \; + \; j_{n+1} \, \Phi_{n+1}^{p-1}(u). \tag{5.4}$$

$$j_n = \frac{u - u_n}{p \, \delta}, \quad j_{n+1} = \frac{u_{n+p+1} - u}{p \, \delta} \tag{5.5}$$

By recursively[1] applying Eq. (5.4) until order P, the P-th order basis function is obtained, see Figure 5.2b. By substituting the resulting BMP basis functions into Eq. (5.2), the trajectory of BMP can be computed using the linear basis function representation of ProMPs.

Derivative of B-Splines. It is worth noting that the i-th order derivative of a P-th order B-spline remains a $(P - i)$-th order B-spline:

$$\boldsymbol{q}^{(i)}(t) = \boldsymbol{\Phi}^{P-i}(u)^\top \boldsymbol{w}^{(i)}, \tag{5.6}$$

[1]For illustrative visualization of this recursive process and more details, please refer to literature [102].

where $\boldsymbol{w}^{(i)} = [w_1^{(i)}, ..., w_{N_b-i}^{(i)}]^\top$ represents the control points of the B-spline's derivatives. These control points are computed recursively:

$$w_n^{(i)} = \frac{P-i}{\delta}[w_{n+1}^{(i-1)} - w_n^{(i-1)}]. \tag{5.7}$$

Enforcing boundary conditions of arbitrary orders. To ensure the trajectory passes through given starting and ending positions, BMP employs clamped B-splines [131] where the trajectory goes through the first and the last control points. Thus, we directly align these two control points with the given position values. Similarly, control points derived from Eq. (5.6) and (5.7) enforce higher-order conditions, such as velocity and acceleration. The mapping between boundary conditions at t_0 and the corresponding control points is summarized in Table 5.2c. Boundary conditions at the end follow similar mappings.

5.2 Problem Description

Consider a task context $\boldsymbol{c}_0$, and a robot trajectory $\boldsymbol{q}_{r,0:T} = \{\boldsymbol{q}_{r,0}, \ldots, \boldsymbol{q}_{r,T}\}$ is generated to solve this task. The task context describes how the problem is defined. This terminology follows the convention in episodic reinforcement learning literature [117, 118], and can be extended to include various objectives[2]. For a point-to-point motion planning problem, the task context is defined by $\boldsymbol{c}_0 = \{\boldsymbol{q}_0, \boldsymbol{q}_g, \mathcal{S}\}$, where $\boldsymbol{q}_0$ and $\boldsymbol{q}_g$ denote the current and goal robot joint configuration, respectively, and $\mathcal{S}$ represents the surrounding dynamic environment.

As the task context changes to $\boldsymbol{c}_\tau$ at time $\tau \in (0, T]$, the method proposed in this chapter aims to find a policy that generates trajectory-level refinements, $\Delta\boldsymbol{q}_{\tau:T}$, based on the current context $\boldsymbol{c}_\tau$ and original robot trajectory $\boldsymbol{q}_{r,\tau:T}$. A state encoder $s(\cdot)$ is used to encode $\boldsymbol{c}_\tau$ and $\boldsymbol{q}_{r,\tau:T}$ into a state vector $\boldsymbol{s}_\tau = s(\boldsymbol{c}_\tau, \boldsymbol{q}_{r,\tau:T})$. Following Section 5.1.1, an ERL problem is formulated to optimize the policy distribution $\pi(\boldsymbol{w}|\boldsymbol{s}_\tau)$ by maximizing the expected roll-out return $R(\boldsymbol{s}_\tau, f(\boldsymbol{w}))$ of an episode using the following objective function

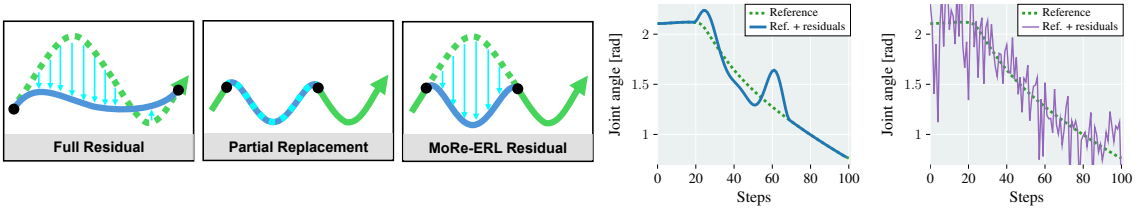
$$J = \mathbb{E}_{p(\boldsymbol{s}_\tau), \pi_\theta(\boldsymbol{w}|\boldsymbol{s}_\tau)} \left[R(\boldsymbol{s}_\tau, f(\boldsymbol{w})) - V_\phi(\boldsymbol{s}_\tau) \right], \tag{5.8}$$

where the vector $\boldsymbol{w}$ determines the start and the end of the residual action as well as the parameterization of the action sequence using BMPs. While MoRe-ERL adopts BBRL [117] for the optimization, the framework is modular enough to plug in any other ERL algorithm.

5.3 Learning Residuals for Reference Trajectories

Reference trajectories contain extensive information for completing the given tasks and can be accessed from various sources, such as optimization-based motion planning, sampling-based motion planning, or other learned policies. This information serves as a valuable prior in two key aspects: (a) it reduces the complexity of learning, thereby improving

[2]For example, reaching the desired goal while interacting with the environment at the same time



(a) MoRe-ERL residuals and two variants **(b)** MoRe-ERL res. **(c)** Step-based res.

Figure 5.3: Illustration of trajectory refinements using MoRe-ERL. (a) The reference trajectory is shown in green, with bold solid points indicating the timing variables α_s and α_e. Cyan sections show learned residuals or replacements, and the solid blue-green curve denotes the adjusted trajectory. Random roll-outs in (b) and (c) use MoRe-ERL and step-based residual methods. In the demonstrated case, the trajectory with MoRe-ERL residuals (–) deviates from the reference trajectory at $\alpha_s = 20$ and converges back at $\alpha_e = 70$.

sample efficiency, and (b) it preserves dexterous maneuvers that are challenging to learn from scratch.

When the task context $\boldsymbol{c}_\tau$ changes unexpectedly, e. g., the environment did not evolve as anticipated, we search for a policy that generates trajectory-level refinements $\Delta\boldsymbol{q}_{\tau:T}$ based on the current context $\boldsymbol{c}_\tau$ and the reference trajectory $\boldsymbol{q}_{r,\tau:T}$. The reference trajectory $\boldsymbol{q}_{r,\tau:T}$ can be a partially executed trajectory grounded on the previous context $\boldsymbol{c}_k$ at time k with $k < \tau$. The refinements are applied as residuals directly on the reference trajectory,

$$\boldsymbol{q}_{\tau:T} = \boldsymbol{q}_{r,\tau:T} + \Delta\boldsymbol{q}_{\tau:T}. \tag{5.9}$$

To allow for greater flexibility in modifying the reference trajectory, we introduce two additional timing variables, $\alpha_s, \alpha_e \in [\tau, T]$ with $\alpha_s \leq \alpha_e$, which denote the start and end point of refinements, respectively. Using these variables, the refinements at time $k \in [\tau, T]$ are defined as:

$$\Delta\boldsymbol{q}_{\tau:T} = \begin{cases} \Delta\boldsymbol{q}_k & \text{for } k \in [\alpha_s, \alpha_e] \\ 0 & \text{otherwise.} \end{cases} \tag{5.10}$$

Figure 5.3a visualizes the refinements described in Eq. (5.9) and (5.10), termed *MoRe-ERL residuals*. The reference trajectory is partially modified by the learned residuals $\Delta\boldsymbol{q}_k$ between α_s and α_e. These two timing variables are represented by bold solid points.

To ensure continuity and smoothness at α_s and α_e during trajectory switching, we use BMPs to parameterize the residual. BMPs enforce boundary conditions up to arbitrary orders. For a residual trajectory $\Delta\boldsymbol{q}_{\alpha_s:\alpha_e}$ using BMPs with N_b control points $\boldsymbol{w}_{1:N_b} = [w_1, ..., w_{N_b}]^\top$, the boundary conditions are set to be $\mathbf{0}$. This ensures that the position and velocity of the refined trajectory align with the reference trajectory at α_s and α_e, guaranteeing continuity and smoothness while the trajectory switches. To be more specific, to parameterize a residual trajectory $\Delta\boldsymbol{y}_{\alpha_s:\alpha_e}$ using N_b control points $\boldsymbol{w}_{1:N_b} = [w_1, ..., w_{N_b}]^\top$, BMPs use w_1, w_2 to enforce the boundary conditions at the start of the refinement and use w_{N_b-1}, w_{N_b} at the end. The remaining control points $\boldsymbol{w}_{3:N_b-2} = [w_3, ..., w_{N_b-2}]^\top$ parametrize the transition behavior between α_s and α_e. These control points for BMPs are jointly learned with α_s and α_e. During inference, given the encoded state $\boldsymbol{s}_\tau$, the policy $\pi(\boldsymbol{w}, \boldsymbol{\alpha}|\boldsymbol{s}_\tau)$ returns the mean $\boldsymbol{\mu}_{\boldsymbol{w},\boldsymbol{\alpha}}$ and the covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{w},\boldsymbol{\alpha}}$ of a single Gaussian distribution for $\boldsymbol{w} = [\boldsymbol{w}_{3:N_b-2}^\top, \alpha_s, \alpha_e]^\top$. Sampling from this Gaussian distribution

$[\boldsymbol{w}_{3:N_b-2}^\top, \alpha_s, \alpha_e]^\top \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{w},\boldsymbol{\alpha}}, \boldsymbol{\Sigma}_{\boldsymbol{w},\boldsymbol{\alpha}})$, the residual sequence can be expressed as

$$\Delta \boldsymbol{q}_{\alpha_s:\alpha_e} = \boldsymbol{\Phi}_{\alpha_s:\alpha_e}^\top \boldsymbol{w}_{3:N_b-2}, \tag{5.11}$$

Figure 5.3b illustrates how the MoRe-ERL residual is applied to the reference trajectory, marked in green. Different from the jerky motions produced by step-based methods in Figure 5.3c, BMPs ensure smooth transitions between the reference trajectory and the refinements.

In addition to partially applying residuals, we consider two alternative approaches: fully applying residuals and partially replacing segments between α_s and α_e. These variants are termed full residual and partial replacement, respectively. The full-residual approach is a special case of partial residuals with $\alpha_s = \tau$ and $\alpha_e = T$, applying residuals to the entire trajectory (Figure 5.3a, left). In contrast, partial replacement modifies the reference trajectory as:

$$\boldsymbol{q}_k = \begin{cases} \Delta \boldsymbol{q}_k & \text{for } k \in [\alpha_s, \alpha_e] \\ \boldsymbol{q}_{r,k} & \text{otherwise.} \end{cases} \tag{5.12}$$

When parameterizing trajectories with BMPs in partial replacement, boundary conditions are set to match the position and velocity of the reference trajectory at α_s and α_e (see Figure 5.3a, middle). Full replacement is excluded from consideration, as it would require learning the trajectory from scratch. Table 5.1 shows the difference among these variants.

Among MoRe-ERL residuals and these two variants, MoRe-ERL residuals demonstrate the best overall performance across various scenarios. Learning residuals leverages prior knowledge embedded in reference trajectories, preserving critical maneuvers and enhancing task completion. The identification of α_s and α_e helps retain essential behaviors. Further details are provided in Section 5.4.

Type	Learned Parameters	How to Refine
Full Residual	$\boldsymbol{w}$	Residuals on top of the reference
Partial Replacement	$\alpha_s, \alpha_e, \boldsymbol{w}$	Replace the reference
MoRe-ERL Residual	$\alpha_s, \alpha_e, \boldsymbol{w}$	Residuals on top of the reference

Table 5.1: Difference between MoRe-ERL residuals and the variants.

5.4 Evaluation

Three simulation experiments in MuJoCo [163] and an experiment with real-world hardware are designed to answer the following questions:

Q5.1 What benefits does MoRe-ERL have compared to sampling-based methods?

Q5.2 What benefits does MoRe-ERL have compared to other *RL* methods?

Q5.3 What are the benefits of spatiotemporal awareness?

Q5.4 Can the learned policy be directly deployed in the real world?

In simulation scenarios, we compare MoRe-ERL with baseline methods, including sampling-based motion planning, episode-based RL, and step-based RL methods. While the episode-based methods work perfectly with non-Markovian rewards, the step-based methods perform poorly in such settings [117]. For an *unfriendly* comparison against our method, we shape a Markovian reward based on the performance of step-based methods and evaluate MoRe-ERL on both Markovian and non-Markovian rewards. The Markovian return of an episode with n_e steps summarizes rewards from each step with a discount factor γ

$$R_M(\boldsymbol{w}, \boldsymbol{s}_\tau) = \sum_{t=\tau}^{n_e} \gamma^{t-\tau}(\beta_c r_{c,t} + \beta_g r_{g,t} + \beta_l r_{l,t}), \tag{5.13}$$

where $r_{c,t}$, $r_{g,t}$ and $r_{l,t}$ indicate the reward terms at t regarding collision, task completeness, and joint limit violation, respectively. The collision reward is assigned $r_{c,t} = -1$ when a collision occurs. The joint limits reward $r_{l,t}$ is computed using the L2-norm. The task completeness reward $r_{g,t}$ is described separately in each scenario. These reward terms are weighted by corresponding coefficients $\beta_{[\cdot]}$. The non-Markovian return $R_{NM}(\boldsymbol{w}, \boldsymbol{s}_\tau)$ does not collect rewards regarding collision and goal reaching at every step, but only at the end of the episode

$$R_{NM}(\boldsymbol{w}, \boldsymbol{s}_\tau) = \underbrace{\beta_c r_c + \beta_g r_g}_{\text{Non-Markovian}} + \beta_l \sum_{t=\tau}^{n_e} \gamma^{t-\tau} r_{l,t}. \tag{5.14}$$

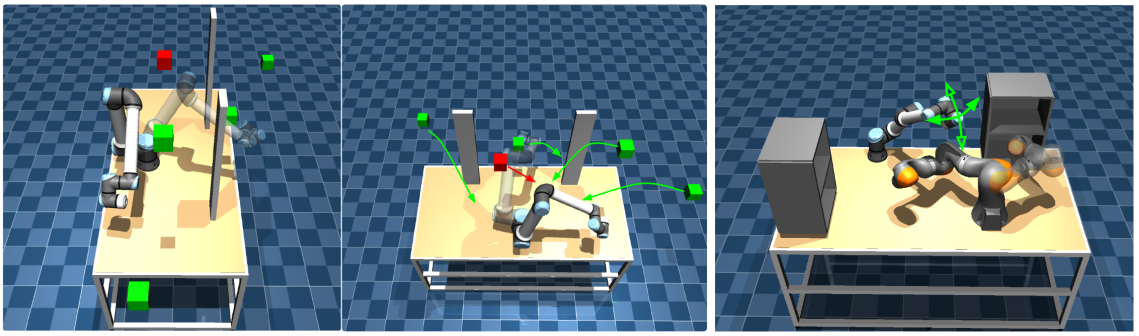
This setting links the reward closer to the definition of success and avoids potential reward hacking. The results of simulation scenarios show that MoRe-ERL with non-Markovian rewards achieved significantly higher performance than baselines, see Table 5.2. The results summarized in Table 5.2 are collected with a single core on an Intel i9-9900K CPU.

5.4.1 Evaluation in Simulation

5.4.1.1 Multi-Box Scenario

The multi-box scenario has a UR10e robot mounted on a table, which travels among three regions separated by two bars to complete arbitrary tasks. At the same time, dynamic obstacles enter the robot's working space, see Figure 5.4a. The obstacles either move with constant velocity or follow parabolic paths. Figure 5.5 shows an episode rollout.

In every episode, the initial joint configurations and goal are randomly selected. At the beginning of the episodes, the obstacles stand still. And then, the obstacles start to move and enter the workspace asynchronously during the episode. We design the trajectories of the obstacles to be *adversarial* to our method so that at least one of them will hit



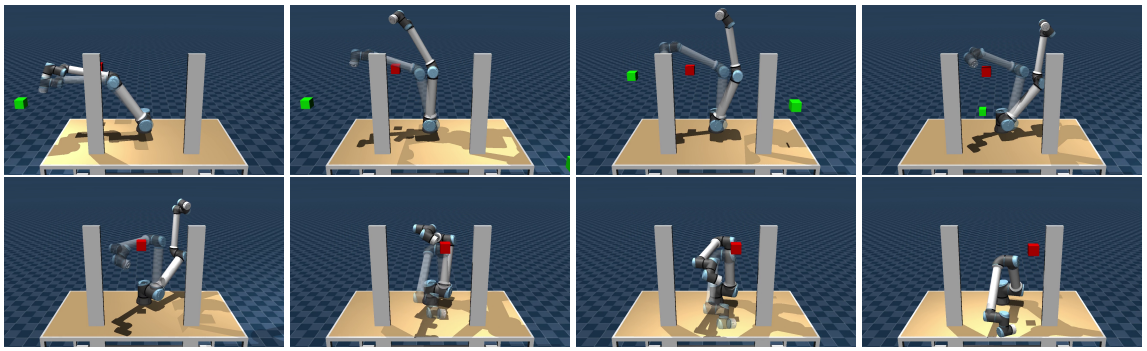
(a) Multi-box **(b)** Dual-arm

Figure 5.4: Simulation experiments in Mujoco. The robot with reduced opacity in simulation indicates the desired goal position. In the multi-box scenario, green boxes follow parabolic trajectories, while the red box moves at a constant velocity. Each box is released at a different timestamp. In the dual-arm scenario, the UR5 robot moves in the directions shown by the green arrows.

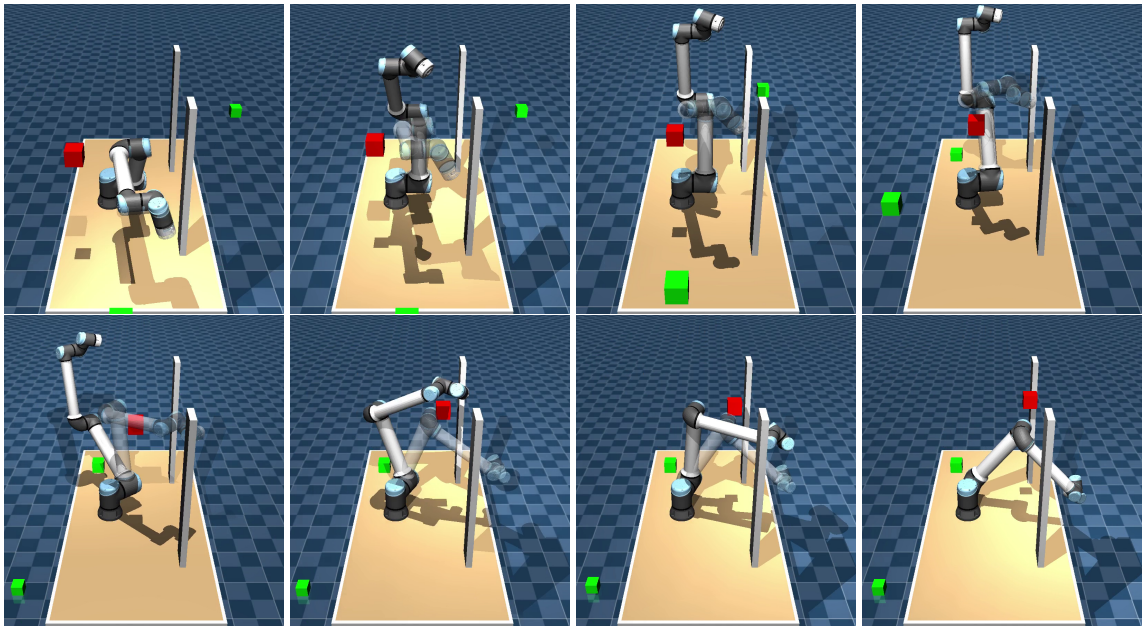
Methods	**Multi-box**		**Dual-arm**	
	PT [s] ↓	Success ↑	PT [s] ↓	Success ↑
ST-RRT*	1.0	**0.928**	5.0	0.392
RRT-Connect	0.609	0.587	4.229	0.204
ERL + DMPs	0.011	0.092	0.010	0.002
ERL + DMPs + Residuals		0.584		0.168
BBRL	**0.010**	0.731	**0.0098**	0.133
Full Residual		0.881		0.674
Partial Replacement		0.812		0.299
MoRe-ERL (ours)		0.889		**0.767**

Table 5.2: Results of MoRe-ERL and baseline methods in both environments, evaluated by planning time (PT) and success rate. The arrow ↓ indicates lower is better, and ↑ indicates higher is better. An episode is reported as a success only if the robot reaches the goal region (radius 0.1) without collisions. The results of sampling-based methods are averaged over 10 runs with different random seeds. The results of RL methods are average with 8 random seeds.

the robot if the robot follows the reference trajectory. The episode terminates when the goal or the pre-defined maximum duration of 3 seconds is reached. For episode-based methods, the observation includes a) the current robot configurations and velocity, b) the goal of the robot, and c) the parameters from which the agent can infer the trajectories of the dynamic obstacles, such as their initial position and velocity, and their end position. The end positions serve the purpose of distinguishing the parabolic trajectories from the ones with constant velocity. On the other hand, the step-based methods receive a new observation every simulation step and return the next joint position as action. The state for step-based methods additionally includes the current Cartesian position, the velocity of the boxes, and the current timestamp.



(a) Front view



(b) Side view

Figure 5.5: An episode rollout involving two boxes following parabolic trajectories (green) and one box moving at a constant speed (red). Snapshots from two perspectives are arranged chronologically from left to right and top to bottom. The robot with lower opacity represents the reference trajectory.

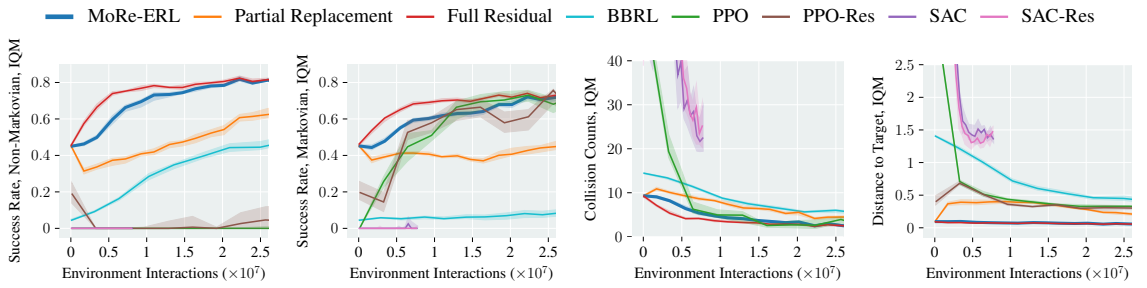


Figure 5.6: Learning curves of multi-box experiments. From left to right: (1) Success rate with non-Markovian reward; (2) Success rate with Markovian reward; (3) Collision counts; (4) Distance to target. Better results from two reward settings are shown in (3) and (4).

The residual version of both episode-based and step-based methods must be aware of the reference on which the residual acts. A representation of the reference is included

in the observation of the residual methods. We use five intermediate waypoints of the reference trajectory as the representation for ERL residual methods and the next reference action for step-based residual methods. The reference trajectories are generated using a sampling-based motion planner. Note that this approach is agnostic to the choice of planner, allowing alternative motion generators to be seamlessly integrated.

Sampling-based methods were allocated one second of planning time for each problem in the $\mathbb{R}^{6+1}$ space-time state space, with 6 DoFs for robot joints and 1 DoF for time. For ST-RRT*, the maximum arrival time was set to 3 seconds, while RRTConnect [86] used a fixed arrival time of 3 seconds due to its inability to handle unknown arrival times. For RL methods, the Markovian reward uses $\beta_c = 5$, $\beta_g = 20$ and $\beta_l = 0$, and the non-Markovian reward uses $\beta_c = 10$, $\beta_g = 40$ and $\beta_l = 1$.

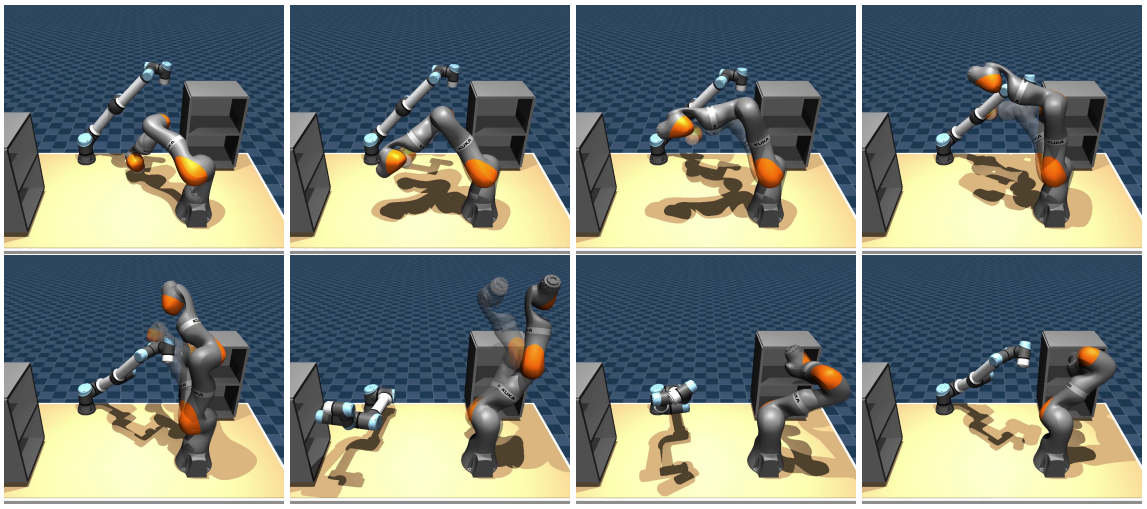
The learning curves in Figure 5.6 demonstrate that MoRe-ERL achieves higher sample efficiency and yields comparable or better performance compared to ERL trained from scratch and step-based residual RL approaches under both Markovian and non-Markovian reward settings. However, with a sufficient planning budget, ST-RRT* achieves the highest success rate of 92.8% in this task. When the planning time budget is reduced to 100 ms, the success rate of ST-RRT* drops to below 70%. In contrast, MoRe-ERL maintains a significantly faster inference time of 10.1 ms while achieving a competitive success rate of 88.9%. Note that the scenarios are initialized to be *adversarial* to residual methods. ST-RRT* does not suffer from such a disadvantage, which reduces the task complexity for ST-RRT*. A common solution is to wait for the boxes to settle and approach the goal. For a more complicated task where the environment is constantly moving, such as the dual-arm task, MoRe-ERL significantly outperforms ST-RRT*.

5.4.1.2 Dual-Arm Scenario

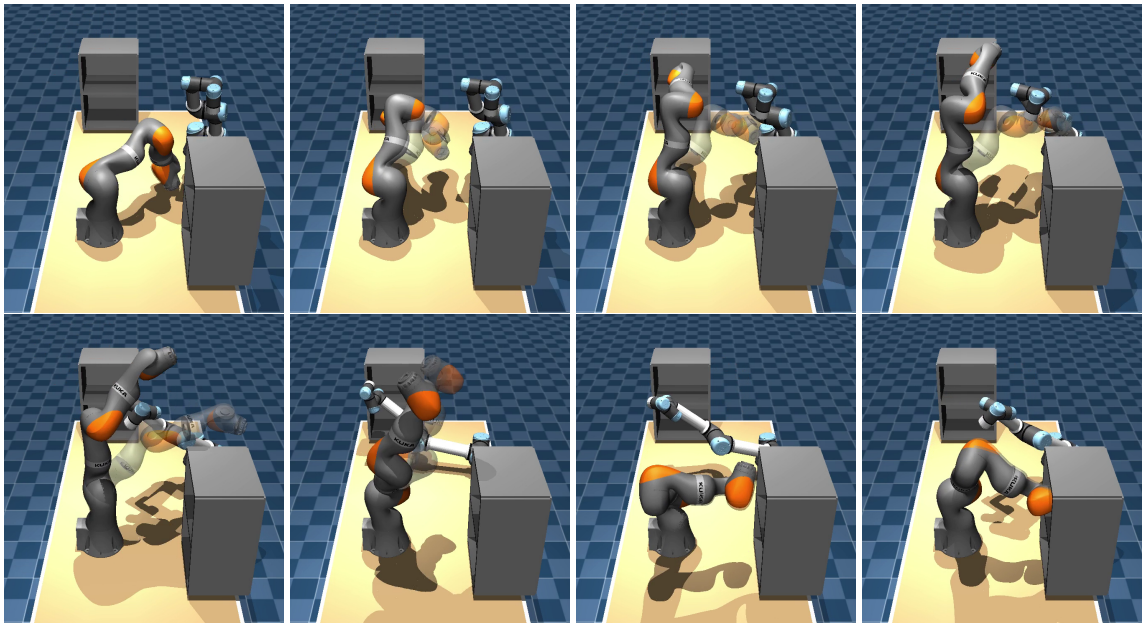
The dual-arm scenario involves a UR5 and a KUKA iiwa 14 robot mounted on a shared workspace, as shown in Figure 5.4b. The UR5 follows pre-defined trajectories, acting as a dynamic part of the environment, while the KUKA iiwa 14 actively avoids collisions and moves toward the goal pose. Figure 5.7 shows an episode rollout.

Similar to the multi-box environment, episodes are randomly initialized and terminate when the goal is reached or the pre-defined maximum duration of 5 seconds elapses. For step-based methods, the observation includes a) the position and velocity of both robots, b) the goal of the iiwa robot, and c) the current timestamp. As in Section 5.4.1.1, the step-based residual method includes the next reference action in the observation, while MoRe-ERL incorporates five intermediate waypoints from the reference trajectories. Sampling-based baselines reported in Table 5.2 are given 5 seconds planning time budgets for each problem in a space-time state space $\mathbb{R}^{7+1}$. The maximum arrival time of ST-RRT* is set to 5 seconds, and RRTConnect has a fixed arrival time. The Markovian reward is weighted by $\beta_c = 5$, $\beta_g = 20$ and $\beta_l = 0$ and the non-Markovian setting is same as Section 5.4.1.1.

In this scenario, MoRe-ERL achieves a 76.7% success rate under the non-Markovian reward, outperforming ST-RRT*, which succeeds in only 39.2% of cases after 5 seconds of planning. It is worth mentioning that the test cases are randomly generated, and some of them may not be solvable. To approximate the upper bound of success rate, we increased the planning budget to 120 seconds, where ST-RRT* achieved success in 86% of the test cases. No significant performance gain is shown if the planning budget increases



(a) Front view



(b) Side view

Figure 5.7: An episode rollout with a UR5 robot as dynamic environments. Snapshots from two perspectives are arranged chronologically from left to right and top to bottom. The robot with lower opacity represents the reference trajectory.

further. Figure 5.8 shows that step-based methods failed to learn reasonable policies in both Markovian and non-Markovian reward settings, while MoRe-ERL demonstrates superior performance under both reward settings. Two ablation variants of MoRe-ERL, full residual and partial replacement, report success rates of 67.4% and 29.9% under non-Markovian reward, respectively. MoRe-ERL's superior performance stems from its ability to identify the interval requiring residual while retaining critical maneuvers, such as retracting from a shelf. These behaviors are usually difficult to learn from scratch.

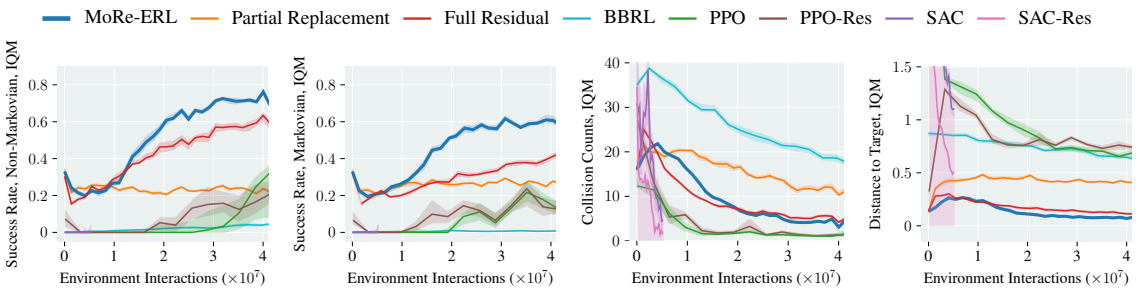


Figure 5.8: Learning curves of the dual-arm experiment. From left to right: (1) Success rate with non-Markovian reward; (2) Success rate with Markovian reward; (3) Collision counts; (4) Distance to target. Better results from two reward settings are shown in (3) and (4).

Q5.1 What benefits does MoRe-ERL have compared to sampling-based methods?

A5.1 Compared to sampling-based methods such as ST-RRT*, MoRe-ERL does not need full knowledge of how the environment evolves along the time horizon. It can account for potential changes and refine a whole trajectory solely based on the current state of the environment within a deterministic inference time of around 10 ms. Moreover, it yields a policy achieving nearly twice the success rate in the dual-arm experiment.

Q5.2 What benefits does MoRe-ERL have compared to other *RL* methods?

A5.2 Compared to **step-based RL** methods, an obvious advantage is that ERL methods can support non-Markovian rewards and converge to a better policy. While using a Markovian reward based on the performance of step-based methods, MoRe-ERL still shows higher success rates.
Compared to other **ERL** methods, MoRe-ERL can inherit some good priors from the reference trajectories, such as entering or pulling out of a shelf. Learning these behaviors from scratch requires more MP bases and interaction samples. This becomes more significant in the dual-arm experiment.

Advantages Compared to Geometric Planning

Three sampling-based planners are deployed in a dynamic environment as baselines to highlight the advantages of using MoRe-ERL compared to planning solely based on the current geometric state of the environment. Every planner is given a budget of 1, 2, and 3 seconds. Two receding horizon planning schemes are used: (1) replanning 200 ms earlier if a predicted collision between the robot and obstacles is detected, and (2) replanning at a fixed frequency of 20 Hz. Each planning iteration generates a trajectory to the final goal in both schemes. To simplify the comparison, computation time is ignored throughout the iterations, which implies that new trajectories can be seamlessly connected to previous ones, with the environment assumed to *remain static* during the planning interval. While this assumption benefits geometric planners, it does not hold in real-world scenarios. In practice, the planner must complete its computation within 200 ms in the first scheme and within 50 ms (20 Hz) in the second scheme to remain responsive. The results of the first

scheme are shown in Table 5.3 and Table 5.4. The number of replanning iterations n_{rc}, the planning time per iteration t_{it}, and the accumulated planning time per problem t_{acc} are listed as metrics. The results of planning at fixed frequency are not listed since all three planners have a zero success rate in the dual-arm scenario.

Methods	Budget [s]	n_{rc}	t_{it} [s]	t_{acc} [s]	Success
RRT	1.0	1.80	0.222	0.400	0.38
	2.0	1.95	0.272	0.531	0.395
	3.0	2.06	0.326	0.672	0.385
RRTConnect	1.0	2.11	0.025	0.055	0.563
	2.0	2.05	0.035	0.097	0.589
	3.0	2.07	0.048	0.072	0.543
AIT*	1.0	1.92	1.031	2.026	0.775
	2.0	1.91	2.049	3.915	0.770
	3.0	2.01	3.072	6.176	0.780

Table 5.3: Multi-box scenario with sampling-based planners using a receding horizon scheme, where the replanning is triggered when a potential collision is detected.

Methods	Budget [s]	n_{rc}	t_{it} [s]	t_{acc} [s]	Success
RRT	1.0	1.57	0.503	0.790	0.040
	2.0	1.65	0.909	1.495	0.045
	3.0	1.80	1.268	2.282	0.050
RRTConnect	1.0	2.99	0.129	0.385	0.175
	2.0	3.11	0.154	0.478	0.225
	3.0	3.21	0.158	0.507	0.180
AIT*	1.0	2.6	1.022	2.658	0.270
	2.0	2.93	2.047	6.007	0.345
	3.0	2.88	3.062	8.833	0.380

Table 5.4: Dual-arm scenario with sampling-based planners using a receding horizon scheme that the replanning is triggered when a potential collision is detected.

Three baseline methods, i. e., RRT[182], RRTConnect[86] and AIT*[157], are used as the geometric planner to form the comparison. Note that the planning is conducted in $\mathbb{R}^N$, while the planning in Table 5.2 is performed in the spatiotemporal space $\mathbb{R}^{N+1}$. Among these planners, AIT* has achieved the highest success rate in both scenarios, yet worse than the performance of MoRe-ERL. In the dual-arm scenario, MoRe-ERL successfully completed twice as many test cases. It is worth mentioning that the scene was frozen for RRT, RRTConnect, and AIT* during the planning for simplification purposes. Without this simplification, the fastest planner among these three, RRTConnect, demonstrates an average planning time of about 150 ms, which is not enough for dynamic environments. Comparing the results of geometric planners in Table 5.3 and Table 5.4 to the planners with spatiotemporal awareness in Table 5.2, such as MoRe-ERL and ST-RRT*, geometric planners generally demonstrate a worse performance.

Q5.3 What are the benefits of spatiotemporal awareness?

A5.3 MoRe-ERL achieved higher task performance compared to the geometric planners, which only consider the current geometric state of the environment and need to constantly replan at runtime.

5.4.2 Ablation Study

Ablation studies regarding different trajectory refinement strategies and trajectory encoding methods are conducted.

Trajectory refinement strategies Ablation studies on different trajectory refinement strategies, i. e., Full Residual and Partial Replacement, described in Section 5.3, are conducted on both simulation environments. These two strategies are illustrated in Figure 5.3a. Full Residual involves adding residual signals to the entire reference trajectory without learning the start α_s and end α_e points. Partial Replacement refers to replacing the reference trajectory segment between α_s and α_e with the trajectory generated by BMPs. The results in Figure 5.6 demonstrate that full residual achieves comparable performance with slightly better sample efficiency than MoRe-ERL in the multi-box task. This behavior can be attributed to the fact that full residual is a special case of MoRe-ERL with $\alpha_s = \tau$ and $\alpha_e = T$, leading to reduced search space dimension. However, in scenarios that require dexterous maneuvers, such as a dual-arm environment, MoRe-ERL residuals show clear advantages in both convergence speed and final performance, demonstrating the importance of identifying the interval requiring residual while retaining critical maneuvers. Meanwhile, partial replacement underperforms in both tasks compared to the other two variants, highlighting the effectiveness of learning motion residuals.

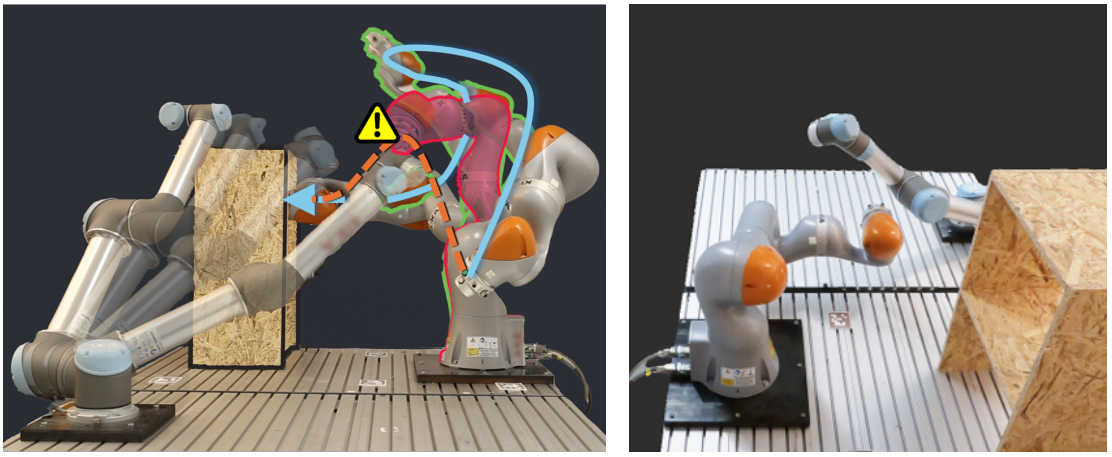
Movement Primitives. Prior work [124] has modeled obstacles in the task space using potential fields, which are explicitly incorporated into the DMP formulation as coupling terms. However, these methods are limited to simple obstacle geometries, such as a single moving sphere, and therefore do not scale to our experimental setup, which involves complex static and dynamic structures like bookshelves and moving robot arms. To demonstrate the benefits of BMPs, we replace them with DMPs in both the learning-from-scratch (ERL + DMPs) and residual settings. Table 5.2 shows that BMPs consistently outperform DMPs, as DMPs achieve high collision-free rates but often fail to reach the goal.

Encoding reference trajectory The reference trajectories can be encoded as input using different methods, such as intermediate waypoints, transformer-based encoder [173], and weight vector $\boldsymbol{w}$ of BMPs. With an extensive exploration of the architectural design of transformers, their performance is yet lower than simply using waypoints. Results are shown in Table 5.5. A possible reason is that training in such architecture requires more data, which reduces the sample efficiency.

	Waypoints	Transformer	BMP weights vector
Success	0.889	0.78	0.68

Table 5.5: Success rate of different trajectory representations in the multi-box scenario.

5.4.3 Evaluation in Robot Experiment



(a) Front view **(b)** Overview

Figure 5.9: Robot experiment setup. MoRe-ERL applies residuals to adjust the trajectory of a KUKA iiwa robot into a safe and feasible motion, effectively avoiding the moving UR5 robot. Snapshots of the KUKA iiwa executing the adjusted motion are outlined in green, while the red marker highlights the frame on the reference trajectory where the robots would have collided.

The hardware setup aligns with the dual-arm experiment in the simulation, with modifications for real-world calibration between two robots. Figure 5.9 provides two distinct views of the setup. Initially, the policy is trained in simulation using the procedure outlined in Section 5.4.1.2, achieving a success rate of approximately 95%. The trained policy is subsequently deployed on real-world hardware. Episodes are designed such that the goal of the current episode becomes the starting point of the next, allowing for seamless sequential rollouts. Applying the policy to the real world faces many challenges, especially for an end-to-end policy. The biggest problem during the real-world experiment is the path-following problem. This involves kinematic limits, i.e., joint position, velocity, and acceleration limits. During the training, the policy gets penalized if the resulting trajectory violates the joint limit constraints. This setting makes room for the policy to generate trajectories that are very close to the joint limits but do not violate them. These kinds of trajectories are prone to being rejected by the low-level controller of the real robots. The second problem is the real-world calibration. The relative position between the robots and objects in the real world cannot perfectly match the one in the simulation. This requires the learned policy to handle such uncertainty. Overall, two out of 80 trajectories in the real-world experiment are close to the joint limits [3].

Q5.4 Can the learned policy be directly deployed in the real world?

A5.4 The policy learned by MoRe-ERL has a strong capability in transferring from simulation to the real world. However, constraints such as joint limits and uncertainties due to the mismatches in the environments should be further addressed.

[3] Recorded episode sequences see `https://youtu.be/J9-r5mibG5o`.

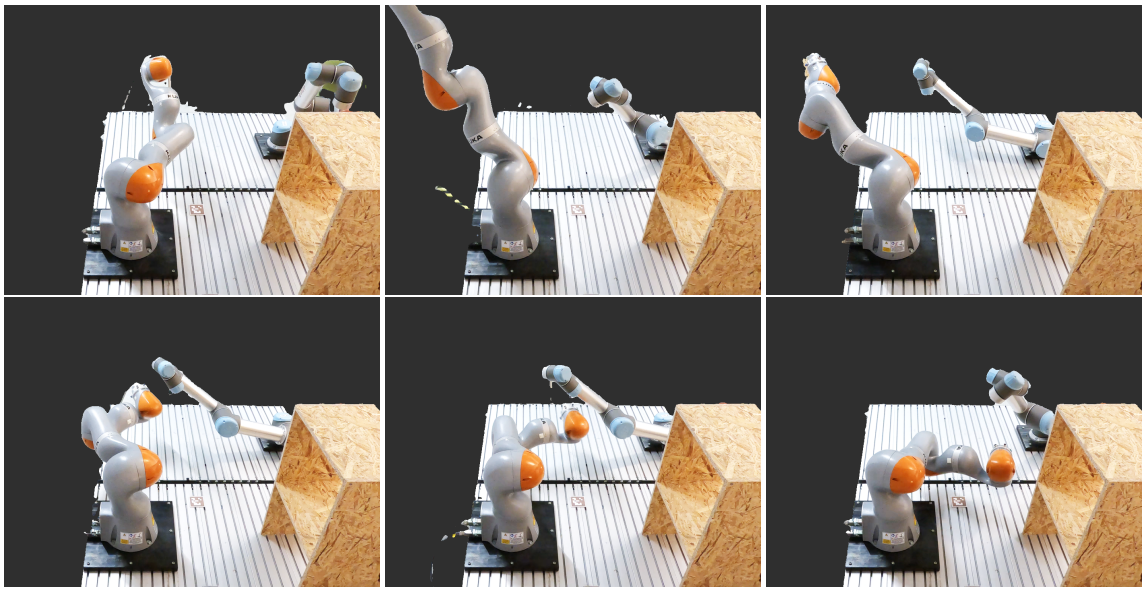


Figure 5.10: Rollout with robots in the real world. Snapshots are arranged chronologically from left to right and top to bottom. To avoid the UR5 robot, the KUKA iiwa has to first move away from its initial position (first row), and then it is able to reach the final goal (bottom right).

5.5 Limitations and Discussions

MoRe-ERL is the first residual learning method tailored for episodic reinforcement learning, which can be built on top of any method within the ERL category. By identifying the crucial intervals within the reference trajectory and applying residual learning to these segments, MoRe-ERL enhances both learning efficiency and task performance compared to learning from scratch. With the spatiotemporal awareness implicitly encoded in the policy, the trajectories refined by MoRe-ERL demonstrate a significant improvement in dynamic environments with a deterministic bounded computation time.

Despite the enhanced sample efficiency and task performance, several assumptions are made. The first one is that the reference trajectories are good priors. While this is usually true, the performance of the proposed method strongly relies on the quality of the reference trajectory. The current approach lacks the capability to deny the reference trajectory and choose a better one. The second assumption is an episode's finite horizon. The real world can be seen as an open-ended system with an infinite horizon. This mismatch makes the proposed method unsuitable in some cases, such as long-horizon planning. This is also a common problem of episodic reinforcement learning. The third limitation is the representation of the reference trajectory.

In the future, the following aspects can be explored: (a) more flexible modifications of reference trajectories, such as compressing, stretching specific segments, or completely rejecting the whole trajectory; (b) extending methods such as [118] to automatically identify triggers for replanning, and extending the formalism of ERL to support an undefined horizon length. (c) leveraging sequence encoding techniques for improved representation of both the reference [67] and the environment [156]. These directions promise to further expand the versatility and effectiveness of MoRe-ERL in addressing complex reinforcement learning challenges.

6 Conclusion and Future Work

This thesis investigated the problem of robot motion planning using a manipulator in dynamic environments. Planning in dynamic environments requires the algorithm to respond rapidly and be aware of the current state of the robot as well as how the environment evolves along the time horizon. These requirements form the following difficulties for the traditional motion planning methods, especially for sampling-based motion planning methods:

- First, the computation time for traditional motion planning methods ranges from hundreds of milliseconds to several seconds. The scale and variation in planning time make these planners unsuitable for online scenarios in dynamic environments.
- Second, even with a systematic and significant speedup, computation time naturally increases with problem complexity and eventually exceeds the threshold for online planning.
- Third, planning usually focuses only on the spatial aspect of the environment, showing a lack of awareness regarding potential changes in the environment over time.

In this thesis, these challenges are addressed by three standalone modules. These modules can be effectively combined to achieve further enhancements. In the following, Section 6.1 summarizes the research questions and the contributions of each module, while Section 6.2 outlines potential research directions for extending the contributions made in this thesis.

6.1 Conclusion

Three research questions have been formulated to address the challenges mentioned above. Thorough literature research reveals that most existing motion planning methods are not capable of performing online motion planning in dynamic environments. This leads to the formulation of the first research question, which aims to address the challenge of accelerating motion planning, specifically focusing on the large number of collision checks that are often required. While this is a common problem for motion planning methods, the first contribution focuses solely on sampling-based methods and consists of two components. The first component introduces a concept of heuristics-informed search based on a precomputed probabilistic roadmap. It significantly reduces the amount of exploration, leading to fewer edge examinations. The second component analyzes the spatial relation between the robot and obstacles and introduces the concept of safe zones. Safe zones outline the region surrounding a joint configuration that is surely collision-free. Consequently, the number of collision checks is further reduced by omitting these checks within safe zones during edge examinations. As a result, the proposed method, HIRO, achieved a sevenfold speedup in the most complex test environments in the evaluation.

A clear trend observed when evaluating HIRO is that the computation time increases with the complexity of the problems. To address this issue, the second research question is formulated to propose a new approach for planning in dynamic environments, termed PLS. PLS generates subgoals to decompose complex problems into small, manageable ones. The contribution to this research question includes a comprehensive pipeline for collecting datasets for subgoal learning, a generative model that captures the subgoal distribution in complex planning problems, and a critic module that selects subgoals based on various metrics. As a result, the proposed method can achieve the goal by sequentially planning to subgoals in 90% of test cases while keeping the planning time below 50 ms. When HIRO is integrated, the planning time can be theoretically further improved.

The last research question focuses on refining the planned reference trajectories to raise awareness of the system dynamics. Being aware of how the system evolves beforehand can improve the task success rates and reduce the need for constant replanning. The reference trajectories can be provided by HIRO or PLS. The contribution to this research question is a trajectory-refinement method based on episodic reinforcement learning (ERL) and B-spline movement primitives. The formalism of ERL ensures that the system dynamics are implicitly encoded in the policy without explicitly modeling it, and the B-spline enables a smooth transition while refining the trajectories. The learned policy can identify the segments of the reference trajectory that require refinements while preserving critical task-specific behaviors. Compared to geometric planning methods, this method achieves up to twice the task success rate while keeping a short computation time. This method accounts for the current robot state and potential environmental changes over time and has a deterministic inference time, fulfilling all three requirements listed at the beginning of this thesis.

In summary, the research carried out in this thesis led to three key contributions to the field of planning in dynamic environments:

- **Contribution 1: Speeding up geometric motion planning methods**
  This contribution describes a method consisting of (1) an offline step computing a deterministic roadmap regarding the static environment and (2) an online step conducting a heuristic-informed search over the precomputed roadmap. The deterministic roadmap must only be computed once for a static environment and saved locally. The heuristic-informed search prioritizes the edge with a good chance of being in the final solution and uses a novel concept of safe zones to examine their validity.
- **Contribution 2: Planning to geometric subgoals for bounded planning time**
  In this contribution, a generation module is trained to capture the subgoal distribution conditioned on the final robot goal pose, current robot state, and the current state of the environment. Sampling from the learned distribution, a critic module is designed to evaluate and select the generated samples from the generation module according to various metrics. Contribution 1 can be used as a planner here for further improvement.
- **Contribution 3: Trajectory refinements accounting for spatiotemporal environment variations**
  The proposed online trajectory refinement method combines sampling-based motion planning and episodic residual reinforcement learning. It can be built on top of any method within the ERL category. Identifying the crucial intervals within the reference trajectory and applying residual learning to these segments enhances

learning efficiency and task performance. It can be easily connected to Contribution 1 and Contribution 2 by using their planning outcomes as reference trajectories.

With these three contributions, the requirements for robot motion planning mentioned at the very beginning of this thesis are addressed.

6.2 Future Research Directions

The limitations of each contribution are fully discussed at the end of each chapter. At a high level, this thesis addressed the online motion planning problem mainly from two aspects: short and bounded motion planning time and spatiotemporal awareness. With the rise of task complexity and diversity, enabling a robot to operate flawlessly in a cluttered and dynamic environment requires more than these two aspects. The following lists two potential future research directions for robot motion planning in dynamic environments.

6.2.1 Interacting with Environments

Interacting with our environment is a common aspect of daily life. However, sampling-based motion planning methods, along with those proposed in this thesis, are designed solely to compute a trajectory that guides a robot from one pose to another without intentionally interacting with the environment. This limitation restricts the range of applications for these methods.

Contribution 3 demonstrates that combining reinforcement learning with trajectories originally computed for static environments can enhance both the success rate of task execution and sampling efficiency. While the scenarios evaluated in this contribution focus on point-to-point robot movements, the algorithm can be adapted to more complex situations that involve interactions with the environment, such as catching a flying ball or picking up a parcel from a conveyor belt. Additionally, various trajectory refinement techniques can be explored, including accelerating, holding, and even discarding the reference trajectories. It is important to note that the proposed methods utilize state-based inputs. Considering raw sensor data as input to address uncertainties may offer a more effective approach.

6.2.2 Explainable Safety Guarantees

Explainable behaviors are crucial when machine learning methods come into play [4]. When deploying a robot into a cluttered, dynamic environment, such as a warehouse, it shares the workspace with multiple human or robotic agents. In this scenario, its behavior is expected to be predictable, explainable, traceable, and safe, especially when an error occurs.

Contributions 1 and 2 use a spatial planner to compute the solution, indicating that the resulting solution is surely collision-free with respect to the state of the environment while the planning is triggered. However, reinforcement learning is used in Contribution 3, where the exploration of the policy is reward-guided and not controlled. Considering explainable reinforcement learning [9] and making the policy explainable and providing

safety guarantees is a step forward to deploying the proposed algorithms to real-world applications.

Bibliography

[1] Abbas Abdolmaleki, Rudolf Lioutikov, Jan R Peters, Nuno Lau, Luis Pualo Reis, and Gerhard Neumann. Model-based relative entropy stochastic search. *Advances in Neural Information Processing Systems*, 28, 2015.

[2] Abbas Abdolmaleki, Bob Price, Nuno Lau, Luis Paulo Reis, and Gerhard Neumann. Contextual covariance matrix adaptation evolutionary strategies. In *International Joint Conferences on Artificial Intelligence Organization (IJCAI)*, 2017.

[3] Tomoki Ando, Hiroto Iino, Hiroki Mori, Ryota Torishima, Kuniyuki Takahashi, Shoichiro Yamaguchi, Daisuke Okanohara, and Tetsuya Ogata. Learning-based collision-free planning on arbitrary optimization criteria in the latent space through cGANs. *Advanced Robotics*, 37(10):621–633, 2023.

[4] Alejandro Barredo Arrieta, Natalia Diaz-Rodrıguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, and others. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58:82–115, 2020.

[5] Tamim Asfour, Mirko Waechter, Lukas Kaul, Samuel Rader, Pascal Weiner, Simon Ottenhaus, Raphael Grimm, You Zhou, Markus Grotz, and Fabian Paus. Armar-6: A high-performance humanoid for human-robot collaboration in real-world scenarios. *IEEE Robotics & Automation Magazine*, 26(4):108–121, 2019.

[6] Jerome Barraquand and Jean-Claude Latombe. Robot motion planning: A distributed representation approach. *The International Journal of Robotics Research*, 10(6):628–649, 1991.

[7] Florian Beck, Minh Nhat Vu, Christian Hartl-Nesic, and Andreas Kugi. Model predictive trajectory optimization with dynamically changing waypoints for serial manipulators. *IEEE Robotics and Automation Letters*, 2024.

[8] Hadi Beik-Mohammadi, Søren Hauberg, Georgios Arvanitidis, Gerhard Neumann, and Leonel Rozo. Reactive motion generation on learned Riemannian manifolds. *The International Journal of Robotics Research*, 42(10):729–754, 2023.

[9] Yanzhe Bekkemoen. Explainable reinforcement learning (XRL): a systematic literature review and taxonomy. *Machine Learning*, 113(1):355–441, 2024.

[10] Mayur J Bency, Ahmed H Qureshi, and Michael C Yip. Neural path planning: Fixed time, near-optimal path generation via oracle imitation. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3965–3972, 2019.

[11] Dmitry Berenson, Pieter Abbeel, and Ken Goldberg. A robot path planning framework that learns from experience. In *2012 IEEE International Conference on Robotics and Automation*, pages 3671–3678. IEEE, 2012.

[12] Mohak Bhardwaj, Balakumar Sundaralingam, Arsalan Mousavian, Nathan D Ratliff, Dieter Fox, Fabio Ramos, and Byron Boots. Storm: An integrated framework for fast joint-space model-predictive control for reactive manipulation. In *Conference on Robot Learning*, pages 750–759. PMLR, 2022.

[13] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518): 859–877, 2017.

[14] Michael S Branicky, Steven M LaValle, Kari Olson, and Libo Yang. Quasi-randomized path planning. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (cat. No. 01CH37164)*, volume 2, pages 1481–1487, 2001.

[15] John Canny. *The complexity of robot motion planning*. MIT press, 1988.

[16] João Carvalho, Dorothea Koert, Marek Daniv, and Jan Peters. Adapting object-centric probabilistic movement primitives with residual reinforcement learning. In *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*, pages 405–412, 2022. doi: 10.1109/Humanoids53995.2022.10000148.

[17] Onur Celik, Dongzhuoran Zhou, Ge Li, Philipp Becker, and Gerhard Neumann. Specializing versatile skill libraries using local mixture of experts. In *Conference on Robot Learning*, pages 1423–1433. PMLR, 2022.

[18] Binghong Chen, Bo Dai, Qinjie Lin, Guo Ye, Han Liu, and Le Song. Learning to plan in high dimensions via neural exploration-exploitation trees. *arXiv preprint arXiv:1903.00070*, 2019.

[19] Ching-An Cheng, Mustafa Mukadam, Jan Issac, Stan Birchfield, Dieter Fox, Byron Boots, and Nathan Ratliff. Rmp flow: A computational graph for automatic motion policy generation. In *Algorithmic Foundations of Robotics XIII: Proceedings of the 13th Workshop on the Algorithmic Foundations of Robotics 13*, pages 441–457. Springer, 2020.

[20] Ching-An Cheng, Mustafa Mukadam, Jan Issac, Stan Birchfield, Dieter Fox, Byron Boots, and Nathan Ratliff. Rmpflow: A geometric framework for generation of multitask motion policies. *IEEE Transactions on Automation Science and Engineering*, 18(3):968–987, 2021.

[21] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 2024.

[22] Hao-Tien Lewis Chiang, Jasmine Hsu, Marek Fiser, Lydia Tapia, and Aleksandra Faust. RL-RRT: Kinodynamic motion planning via learning reachability estimators from RL policies. *IEEE Robotics and Automation Letters*, 4(4):4298–4305, 2019.

[23] David Coleman, Ioan Sucan, Sachin Chitta, and Nikolaus Correll. Reducing the barrier to entry of complex robotic software: a moveit! case study. *arXiv preprint arXiv:1404.3785*, 2014.

[24] David Coleman, Ioan A Şucan, Mark Moll, Kei Okada, and Nikolaus Correll. Experience-based planning with sparse roadmap spanners. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 900–905. IEEE, 2015.

[25] Michael Danielczuk, Arsalan Mousavian, Clemens Eppner, and Dieter Fox. Object rearrangement using learned implicit collision functions. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6010–6017. IEEE, 2021.

[26] Todor Davchev, Kevin Sebastian Luck, Michael Burke, Franziska Meier, Stefan Schaal, and Subramanian Ramamoorthy. Residual learning from demonstration: Adapting dmps for contact-rich manipulation. *IEEE Robotics and Automation Letters*, 7(2):4488–4495, 2022.

[27] Christopher Dellin and Siddhartha Srinivasa. A unifying formalism for shortest path problems with expensive edge evaluations via lazy best-first search over paths with edge selectors. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 26, 2016.

[28] Edsger Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[29] Andrew Dobson and Kostas E Bekris. Sparse roadmap spanners for asymptotically near-optimal motion planning. *The International Journal of Robotics Research*, 33 (1):18–47, 2014.

[30] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[31] Jing Dong, Mustafa Mukadam, Frank Dellaert, and Byron Boots. Motion planning as probabilistic inference using Gaussian processes and factor graphs. In *Proceedings of Robotics: Science and Systems (RSS)*, 2016.

[32] Kevin Duffy. Reaction times and sprint false starts. URL `https://condellpark.com/kd/reactiontime.htm`.

[33] Aleksandra Faust, Kenneth Oslund, Oscar Ramirez, Anthony Francis, Lydia Tapia, Marek Fiser, and James Davidson. Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5113–5120. IEEE, 2018.

[34] Bernardo Fichera and Aude Billard. Learning dynamical systems encoding nonlinearity within space curvature. *arXiv preprint arXiv:2403.11948*, 2024.

[35] Mark Nicholas Finean, Wolfgang Merkt, and Ioannis Havoutis. Predicted composite signed-distance fields for real-time motion planning in dynamic environments. *arXiv preprint arXiv:2008.00969*, 2020.

[36] Fabrizio Flacco, Torsten Kröger, Alessandro De Luca, and Oussama Khatib. A depth space approach to human-robot collision avoidance. In *2012 IEEE International Conference on Robotics and Automation*, pages 338–345, 2012.

[37] Fabrizio Flacco, Torsten Kroeger, Alessandro De Luca, and Oussama Khatib. A depth space approach for evaluating distance to objects. *Journal of Intelligent & Robotic Systems*, 80(1):7–22, 2015.

[38] Christopher K Fourie, Nadia Figueroa, and Julie A Shah. On-manifold strategies for reactive dynamical system modulation with non-convex obstacles. *IEEE Transactions on Robotics*, 2024.

[39] Jonathan Gammell, Siddhartha S Srinivasa, and Timothy Barfoot. Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3067–3074. IEEE, 2015.

[40] Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco Jose Madrid-Cuevas, and Manuel Jesus Marın-Jimenez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, 2014.

[41] Weizong Ge, Hongyu Chen, Hongtao Ma, Liuhe Li, Ming Bai, Xilun Ding, and Kun Xu. A dynamic obstacle avoidance method for collaborative robots based on trajectory optimization. *Cobot*, 2:6, 2023.

[42] Elmer G Gilbert, Daniel W Johnson, and S Sathiya Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2):193–203, 2002.

[43] Faustino Gomez, Jürgen Schmidhuber, Risto Miikkulainen, and Melanie Mitchell. Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research*, 9(5), 2008.

[44] Francesco Grothe, Valentin N Hartmann, Andreas Orthey, and Marc Toussaint. St-rrt*: Asymptotically-optimal bidirectional motion planning through space-time. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 3314–3320. IEEE, 2022.

[45] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

[46] John H Halton. Algorithm 247: Radical-inverse quasi-random point sequence. *Communications of the ACM*, 7(12):701–702, 1964.

[47] Peter Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi: 10.1109/tssc.1968.300136. URL `https://doi.org/10.1109/tssc.1968.300136`.

[48] Kris Hauser. Lazy collision checking in asymptotically-optimal motion planning. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2951–2957. IEEE, 2015.

[49] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[50] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[51] David Hsu, J-C Latombe, and Rajeev Motwani. Path planning in expansive configuration spaces. In *Proceedings of International Conference on Robotics and Automation*, volume 3, pages 2719–2726. IEEE, 1997.

[52] Huang Huang, Balakumar Sundaralingam, Arsalan Mousavian, Adithyavairavan Murali, Ken Goldberg, and Dieter Fox. Diffusionseeder: Seeding motion optimization with diffusion for rapid motion planning. *arXiv preprint arXiv:2410.16727*, 2024.

[53] Xi Huang, Gergely Sóti, Hongyi Zhou, Christoph Ledermann, Björn Hein, and Torsten Kröger. HIRO: Heuristics informed robot online path planning using precomputed deterministic roadmaps. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8109–8116. IEEE, 2022.

[54] Xi Huang, Gergely Sóti, Christoph Ledermann, Björn Hein, and Torsten Kröger. Planning with learned subgoals selected by temporal information. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9306–9312. IEEE, 2024.

[55] Lukas Huber, Jean-Jacques Slotine, and Aude Billard. Avoidance of concave obstacles through rotation of nonlinear dynamics. *IEEE Transactions on Robotics*, 40:1983–2002, 2023.

[56] Jinwook Huh and Daniel D Lee. Learning high-dimensional mixture models for fast collision detection in rapidly-exploring random trees. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 63–69. IEEE, 2016.

[57] Jinwook Huh, Bhoram Lee, and Daniel D Lee. Adaptive motion planning with high-dimensional mixture models. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3740–3747, 2017.

[58] Jinwook Huh, Galen Xing, Ziyun Wang, Volkan Isler, and Daniel D Lee. Learning to generate cost-to-go functions for efficient motion planning. In *International Symposium on Experimental Robotics*, pages 555–565. Springer, 2020.

[59] Brian Ichter and Marco Pavone. Robot motion planning in learned latent spaces. *IEEE Robotics and Automation Letters*, 4(3):2407–2414, 2019.

[60] Brian Ichter, James Harrison, and Marco Pavone. Learning sampling distributions for robot motion planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7087–7094. IEEE, 2018.

[61] Brian Ichter, James Harrison, and Marco Pavone. Learning sampling distributions for robot motion planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7087–7094. IEEE, 2018.

[62] Brian Ichter, Edward Schmerling, Tsang-Wei Edward Lee, and Aleksandra Faust. Learned critical probabilistic roadmaps for robotic motion planning. *arXiv preprint arXiv:1910.03701*, 2019.

[63] Christian Igel. Neuroevolution for reinforcement learning using evolution strategies. In *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.*, volume 4, pages 2588–2595. IEEE, 2003.

[64] Julius Jankowski, Lara Brudermüller, Nick Hawes, and Sylvain Calinon. Vp-sto: Via-point-based stochastic trajectory optimization for reactive robot behavior. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10125–10131. IEEE, 2023.

[65] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International Journal of Robotics Research*, 34(7):883–921, 2015.

[66] Lucas Janson, Brian Ichter, and Marco Pavone. Deterministic sampling-based motion planning: Optimality, complexity, and performance. *The International Journal of Robotics Research*, 37(1):46–61, 2018.

[67] Xiaogang Jia, Qian Wang, Atalay Donat, Bowen Xing, Ge Li, Hongyi Zhou, Onur Celik, Denis Blessing, Rudolf Lioutikov, and Gerhard Neumann. Mail: Improving imitation learning with mamba. *arXiv preprint arXiv:2406.08234*, 2024.

[68] Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6023–6029. IEEE, 2019.

[69] Tom Jurgenson and Aviv Tamar. Harnessing reinforcement learning for neural motion planning. *arXiv preprint arXiv:1906.00214*, 2019.

[70] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *2011 IEEE International Conference on Robotics and Automation*, pages 4569–4574. IEEE, 2011.

[71] Marcelo Kallman and Maja Mataric. Motion planning using dynamic roadmaps. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 5, pages 4399–4404, 2004.

[72] Jay Kamat, Joaquim Ortiz-Haro, Marc Toussaint, Florian T Pokorny, and Andreas Orthey. Bitkomo: Combining sampling and optimization for fast convergence in optimal motion planning. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4492–4497. IEEE, 2022.

[73] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[74] Lydia Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[75] J. Chase Kew, Brian Ichter, Maryam Bandari, Tsang-Wei Edward Lee, and Aleksandra Faust. Neural collision clearance estimator for batched motion planning. In *International Workshop on the Algorithmic Foundations of Robotics*, pages 73–89. Springer, 2020.

[76] Seyed Mohammad Khansari-Zadeh and Aude Billard. A dynamical system approach to realtime obstacle avoidance. *Autonomous Robots*, 32:433–454, 2012.

[77] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98, 1986.

[78] Piotr Kicki, Davide Tateo, Puze Liu, Jonas Günster, Jan Peters, and Krzysztof Walas. Bridging the gap between learning-to-plan, motion primitives and safe reinforcement learning. In *8th Annual Conference on Robot Learning*, 2024.

[79] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[80] Holle Kirchner and Simon J Thorpe. Ultra-rapid object detection with saccadic eye movements: Visual processing speed revisited. *Vision research*, 46(11):1762–1776, 2006.

[81] Holger Klein, Noémie Jaquier, Andre Meixner, and Tamim Asfour. On the design of region-avoiding metrics for collision-safe motion generation on riemannian manifolds. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2346–2353. IEEE, 2023.

[82] Jens Kober and Jan Peters. Policy search for motor primitives in robotics. *Advances in Neural Information Processing Systems*, 21, 2008.

[83] Mikhail Koptev, Nadia Figueroa, and Aude Billard. Neural Joint Space Implicit Signed Distance Functions for Reactive Robot Manipulator Control. *IEEE Robotics and Automation Letters*, 8(2):480–487, 2022.

[84] Mikhail Koptev, Nadia Figueroa, and Aude Billard. Reactive collision-free motion generation in joint space via dynamical systems and sampling-based MPC. *The International Journal of Robotics Research*, 43(13):2049–2069, 2024.

[85] Torsten Kröger and Friedrich M Wahl. Online trajectory generation: Basic concepts for instantaneous reactions to unforeseen events. *IEEE Transactions on Robotics*, 26(1):94–111, 2009.

[86] James J Kuffner and Steven M 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)*, volume 2, pages 995–1001. IEEE, 2000.

[87] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[88] Rahul Kumar, Aditya Mandalika, Sanjiban Choudhury, and Siddhartha S Srinivasa. Lego: Leveraging experience in roadmap generation for sampling-based planning. *arXiv preprint arXiv:1907.09574*, 2019.

[89] Tobias Kunz and Mike Stilman. Time-optimal trajectory generation for path following with bounded acceleration and velocity. *Robotics: Science and Systems VIII*, pages 1–8, 2012.

[90] Alexander Lambert, Brian Hou, Rosario Scalise, Siddhartha Srinivasa, and Byron Boots. Stein Variational Probabilistic Roadmaps. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 11094–11101. IEEE, 2022.

[91] Alexander Lambert, Brian Hou, Rosario Scalise, Siddhartha S Srinivasa, and Byron Boots. Stein Variational Probabilistic Roadmaps. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 11094–11101. IEEE, 2022.

[92] Steven LaValle. Rapidly-exploring random trees: A new tool for path planning. *Research Report 9811*, 1998.

[93] An T Le, Georgia Chalvatzaki, Armin Biess, and Jan R Peters. Accelerating motion planning via optimal transport. *Advances in Neural Information Processing Systems*, 36:78453–78482, 2023.

[94] John M Lee. Smooth manifolds. In *Introduction to smooth manifolds*, pages 1–29. Springer, 2003.

[95] Yiyuan Lee, Constantinos Chamzas, and Lydia E Kavraki. Adaptive experience sampling for motion planning using the generator-critic framework. *IEEE Robotics and Automation Letters*, 7(4):9437–9444, 2022.

[96] Peter Lehner and Alin Albu-Schäffer. Repetition sampling for efficiently planning similar constrained manipulation tasks. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2851–2856. IEEE, 2017.

[97] Anqi Li, Ching-An Cheng, M Asif Rana, Man Xie, Karl Van Wyk, Nathan Ratliff, and Byron Boots. RMP2: A structured composable policy class for robot learning. *arXiv preprint arXiv:2103.05922*, 2021.

[98] Ge Li, Zeqi Jin, Michael Volpp, Fabian Otto, Rudolf Lioutikov, and Gerhard Neumann. Prodmp: A unified perspective on dynamic and probabilistic movement primitives. *IEEE Robotics and Automation Letters*, 8(4):2325–2332, 2023.

[99] Ge Li, Hongyi Zhou, Dominik Roth, Serge Thilges, Fabian Otto, Rudolf Lioutikov, and Gerhard Neumann. Open the black box: Step-based policy updates for temporally-correlated episodic reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2024.

[100] Linjun Li, Yinglong Miao, Ahmed H Qureshi, and Michael C Yip. MPC-MPNet: Model-predictive motion planning networks for fast, near-optimal planning under kinodynamic constraints. *IEEE Robotics and Automation Letters*, 6(3):4496–4503, 2021.

[101] Yiming Li, Xuemin Chi, Amirreza Razmjoo, and Sylvain Calinon. Configuration space distance fields for manipulation planning. *arXiv preprint arXiv:2406.01137*, 2024.

[102] Weiran Liao, Ge Li, Hongyi Zhou, Rudolf Lioutikov, and Gerhard Neumann. Bmp: Bridging the gap between b-spline and movement primitives. *arXiv preprint arXiv:2411.10336*, 2024.

[103] Puze Liu, Kuo Zhang, Davide Tateo, Snehal Jauhri, Jan Peters, and Georgia Chalvatzaki. Regularized deep signed distance fields for reactive motion generation. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6673–6680, 2022.

[104] Qiang Liu and Dilin Wang. Stein variational gradient descent: A general purpose bayesian inference algorithm. *Advances in Neural Information Processing Systems*, 29, 2016.

[105] Wansong Liu, Kareem Eltouny, Sibo Tian, Xiao Liang, and Minghui Zheng. Kg-planner: Knowledge-informed graph neural planning for collaborative manipulators. *IEEE Transactions on Automation Science and Engineering*, 2024.

[106] Tomás Lozano-Pérez and Michael A Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

[107] Jonathan Michaux, Adam Li, Qingyi Chen, Che Chen, Bohao Zhang, and Ram Vasudevan. Safe planning for articulated robots using reachability-based obstacle avoidance with spheres. *Proceedings of Robotics: Science and Systems*, 2024. doi: 10.15607/RSS.2024.XX.035.

[108] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[109] Ieva Miseviciute. The speed of human visual perception. URL `https://www.tobii.com/resource-center/learn-articles/speed-of-human-visual-perception`.

[110] Arsalan Mousavian, Clemens Eppner, and Dieter Fox. 6-dof graspnet: Variational grasp generation for object manipulation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2901–2910, 2019.

[111] Mustafa Mukadam, Xinyan Yan, and Byron Boots. Gaussian process motion planning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9–15. IEEE, 2016.

[112] Adithyavairavan Murali, Arsalan Mousavian, Clemens Eppner, Adam Fishman, and Dieter Fox. Cabinet: Scaling neural collision detection for object rearrangement with procedural scene generation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1866–1874. IEEE, 2023.

[113] Allen Newell. *Unified theories of cognition*. Harvard University Press, 1994.

[114] Ruiqi Ni and Ahmed H Qureshi. NTFields: Neural time fields for physics-informed robot motion planning. *arXiv preprint arXiv:2210.00120*, 2022.

[115] Andreas Orthey, Constantinos Chamzas, and Lydia E Kavraki. Sampling-based motion planning: A comparative review. *Annual Review of Control, Robotics, and Autonomous Systems*, 7, 2023.

[116] Takayuki Osa. Multimodal trajectory optimization for motion planning. *The International Journal of Robotics Research*, 39(8):983–1001, 2020.

[117] Fabian Otto, Onur Celik, Hongyi Zhou, Hanna Ziesche, Vien Anh Ngo, and Gerhard Neumann. Deep black-box reinforcement learning with movement primitives. In *6th Annual Conference on Robot Learning (CoRL 2022)*, volume 205 of *Proceedings of Machine Learning Research*, pages 1244–1265. Machine Learning Research Press (ML Research Press), 2022.

[118] Fabian Otto, Hongyi Zhou, Onur Celik, Ge Li, Rudolf Lioutikov, and Gerhard Neumann. Mp3: Movement primitive-based (re-) planning policy. *arXiv preprint arXiv:2306.12729*, 2023.

[119] Jia Pan, Sachin Chitta, and Dinesh Manocha. FCL: A general purpose library for collision and proximity queries. In *2012 IEEE International Conference on Robotics and Automation*, pages 3859–3866. IEEE, 2012.

[120] Jia Pan, Sachin Chitta, Dinesh Manocha, Florent Lamiraux, Joseph Mirabel, Justin Carpentier, Louis Montaut, and others. Coal: an extension of the flexible collision library, 2015. URL `https://github.com/coal-library/coal`.

[121] Alexandros Paraschos, Christian Daniel, Jan Peters, and Gerhard Neumann. Probabilistic movement primitives. *Advances in Neural Information Processing Systems*, 26, 2013.

[122] Chonhyon Park, Jia Pan, and Dinesh Manocha. ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments. In *Proceedings of the*, volume 22, pages 207–215, 2012.

[123] Chonhyon Park, Jia Pan, and Dinesh Manocha. High-DOF robots in dynamic environments using incremental trajectory optimization. *Int. J. Humanoid Robotics*, 2014. doi: 10.1142/s0219843614410011.

[124] Dae-Hyung Park, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields. In *Humanoids 2008-8th IEEE-RAS international conference on humanoid robots*, pages 91–98. IEEE, 2008.

[125] Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

[126] Hung Pham and Quang-Cuong Pham. Time-optimal path tracking via reachability analysis. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3007–3012, 2018.

[127] Quang-Cuong Pham, Stéphane Caron, and Yoshihiko Nakamura. Kinodynamic planning in the configuration space via admissible velocity propagation. In *Robotics: Science*, volume 32, 2013.

[128] Jeff M Phillips, Nazareth Bedrossian, and Lydia E Kavraki. Guided expansive spaces trees: A search strategy for motion-and cost-constrained state spaces. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 4, pages 3968–3973. IEEE, 2004.

[129] Kai Ploeger, Michael Lutter, and Jan Peters. High acceleration reinforcement learning for real-world juggling with binary rewards. In *Conference on Robot Learning*, pages 642–653. PMLR, 2021.

[130] Thomas Power and Dmitry Berenson. Constrained stein variational trajectory optimization. *IEEE Transactions on Robotics*, 2024.

[131] Hartmut Prautzsch, Wolfgang Boehm, and Marco Paluszny. *Bézier and B-Spline Techniques*. Springer, Berlin, Heidelberg, 2002. ISBN 978-3-642-05240-8. doi: 10.1007/978-3-662-04947-3.

[132] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017.

[133] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in Neural Information Processing Systems*, 30, 2017.

[134] Ahmed Qureshi and Michael Yip. Deeply informed neural sampling for robot motion planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6582–6588. IEEE, 2018.

[135] Ahmed Qureshi, Anthony Simeonov, Mayur Bency, and Michael Yip. Motion planning networks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2118–2124, 2019.

[136] Ahmed H Qureshi, Jiangeng Dong, Austin Choe, and Michael C Yip. Neural manipulation planning on constraint manifolds. *IEEE Robotics and Automation Letters*, 5(4):6089–6096, 2020.

[137] Ahmed Hussain Qureshi, Yinglong Miao, Anthony Simeonov, and Michael Yip. Motion planning networks: Bridging the gap between learning-based and classical motion planners. *IEEE Transactions on Robotics*, 2020.

[138] Clayton W Ramsey, Zachary Kingston, Wil Thomason, and Lydia E Kavraki. Collision-affording point trees: SIMD-amenable nearest neighbors for fast collision checking. *arXiv preprint arXiv:2406.02807*, 2024.

[139] Alireza Ranjbar, Ngo Anh Vien, Hanna Ziesche, Joschka Boedecker, and Gerhard Neumann. Residual feedback learning for contact-rich manipulation tasks with uncertainty. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2383–2390. IEEE, 2021.

[140] Nathan Ratliff, Matt Zucker, Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *2009 IEEE International Conference on Robotics and Automation*, pages 489–494. IEEE, 2009.

[141] Nathan Ratliff, Marc Toussaint, and Stefan Schaal. Understanding the geometry of workspace obstacles in motion optimization. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4202–4209. IEEE, 2015.

[142] Nathan Ratliff, Marc Toussaint, Jeannette Bohg, and Stefan Schaal. On the fundamental importance of gauss-newton in motion optimization. *arXiv preprint arXiv:1605.09296*, 2016.

[143] Nathan D Ratliff, Jan Issac, Daniel Kappler, Stan Birchfield, and Dieter Fox. Riemannian motion policies. *arXiv preprint arXiv:1801.02854*, 2018.

[144] Stefan Schaal. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *Adaptive Motion of Animals and Machines*, pages 261–280. Springer, 2006.

[145] Peter H Schiller and Jennifer Kendall. Temporal factors in target selection with saccadic eye movements. *Experimental Brain Research*, 154:154–159, 2004.

[146] John Schulman, Jonathan Ho, Alex X Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: Science and Systems*, volume 9, pages 1–10. Berlin, Germany, 2013.

[147] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014.

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

[149] Philip Schömer, Mark Timon Hüneberg, and J Marius Zöllner. Optimization of sampling-based motion planning in dynamic environments using neural networks. In *2020 IEEE intelligent vehicles symposium (IV)*, pages 2110–2117. IEEE, 2020.

[150] Deval Shah, Ningfeng Yang, and Tor M Aamodt. Energy-efficient realtime motion planning. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pages 1–17, 2023.

[151] Bruno Siciliano, Oussama Khatib, and Torsten Kröger. *Springer Handbook of Robotics*, volume 200. Springer, 2008.

[152] Tom Silver, Kelsey Allen, Josh Tenenbaum, and Leslie Kaelbling. Residual policy learning. *arXiv preprint arXiv:1812.06298*, 2018.

[153] Avishai Sintov and Amir Shapiro. Time-based RRT algorithm for rendezvous planning of two dynamic systems. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6745–6750. IEEE, 2014.

[154] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. *Advances in Neural Information Processing Systems*, 28, 2015.

[155] Davood Soleymanzadeh, Xiao Liang, and Minghui Zheng. SIMPNet: Spatial-informed motion planning network. *IEEE Robotics and Automation Letters*, 2025.

[156] Gergely Sóti, Xi Huang, Christian Wurll, and Björn Hein. dgrasp: Nerf-informed implicit grasp policies with supervised optimization slopes. *arXiv preprint arXiv:2406.09939*, 2024.

[157] Marlin P Strub and Jonathan D Gammell. Adaptively Informed Trees (AIT*) and Effort Informed Trees (EIT*): Asymmetric bidirectional sampling-based path planning. *The International Journal of Robotics Research*, 41(4):390–417, 2022.

[158] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. doi: 10.1109/MRA.2012.2205651. `https://ompl.kavrakilab.org`.

[159] Balakumar Sundaralingam, Siva Kumar Sastry Hari, Adam Fishman, Caelan Garrett, Karl Van Wyk, Valts Blukis, Alexander Millane, Helen Oleynikova, Ankur Handa, Fabio Ramos, Nathan Ratliff, and Dieter Fox. curobo: Parallelized collision-free minimum-jerk robot motion generation, 2023.

[160] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[161] Gildardo Sánchez and Jean-Claude Latombe. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In *Robotics Research: The Tenth International Symposium*, pages 403–417. Springer, 2003.

[162] Wil Thomason, Zachary Kingston, and Lydia E Kavraki. Motions in microseconds via vectorized sampling-based planning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8749–8756. IEEE, 2024.

[163] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

[164] Marc Toussaint. Robot trajectory optimization using approximate inference. In *Proceedings of the 26th annual International Conference on Machine Learning*, pages 1049–1056, 2009.

[165] Marc Toussaint. Newton methods for k-order markov constrained motion problems. *arXiv preprint arXiv:1407.0414*, 2014.

[166] Marc Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1930–1936, 2015.

[167] Marc Toussaint. A tutorial on Newton methods for constrained trajectory optimization and relations to SLAM, Gaussian Process smoothing, optimal control, and probabilistic inference. *Geometric and numerical foundations of movements*, pages 361–392, 2017.

[168] Marc Toussaint and Christian Goerick. A bayesian view on motor control and planning. In *From Motor Learning to Interaction Learning in Robots*, pages 227–252. Springer, 2010.

[169] Marc Toussaint, Jason Harris, Jung-Su Ha, Danny Driess, and Wolfgang Hönig. Sequence-of-constraints mpc: Reactive timing-optimal control of sequential manipulation. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 13753–13760. IEEE, 2022.

[170] Julen Urain, Anqi Li, Puze Liu, Carlo D'Eramo, and Jan Peters. Composable energy policies for reactive motion generation and reinforcement learning. *The International Journal of Robotics Research*, 42(10):827–858, 2023.

[171] John Vannoy and Jing Xiao. Real-time adaptive motion planning (RAMP) of mobile manipulators in dynamic environments with unforeseen changes. *IEEE Transactions on Robotics*, 24(5):1199–1212, 2008.

[172] Vasileios Vasilopoulos, Suveer Garg, Pedro Piacenza, Jinwook Huh, and Volkan Isler. RAMP: Hierarchical reactive motion planning for manipulation tasks using implicit signed distance functions. *arXiv preprint arXiv:2305.10534*, 2023.

[173] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[174] Darrell Whitley, Stephen Dominic, Rajarshi Das, and Charles W Anderson. Genetic reinforcement learning for neurocontrol problems. *Machine Learning*, 13: 259–284, 1993.

[175] Grady Williams, Paul Drews, Brian Goldfain, James M Rehg, and Evangelos A Theodorou. Information-theoretic model predictive control: Theory and applications to autonomous driving. *IEEE Transactions on Robotics*, 34(6):1603–1622, 2018.

[176] Tyler S Wilson, Wil Thomason, Zachary Kingston, Lydia E Kavraki, and Jonathan D Gammell. Nearest-neighbourless asymptotically optimal motion planning with fully connected informed trees (FCIT*). *arXiv preprint arXiv:2411.17902*, 2024.

[177] Zhengtian Wu, Jinyu Dai, Baoping Jiang, and Hamid Reza Karimi. Robot path planning based on artificial potential field with deterministic annealing. *ISA Transactions*, 138:74–87, 2023.

[178] Yiming Yang, Wolfgang Merkt, Vladimir Ivan, Zhibin Li, and Sethu Vijayakumar. HDRM: A resolution complete dynamic roadmap for real-time motion planning in complex scenes. *IEEE Robotics and Automation Letters*, 3(1):551–558, 2017.

[179] Yuandong Yang and Oliver Brock. Elastic roadmaps—motion generation for autonomous mobile manipulation. *Autonomous Robots*, 28(1):113–130, 2010.

[180] Chenning Yu and Sicun Gao. Reducing collision checking for sampling-based motion planning using graph neural networks. *Advances in Neural Information Processing Systems*, 34:4274–4289, 2021.

[181] Hongzhe Yu and Yongxin Chen. Stochastic motion planning as gaussian variational inference: theory and algorithms. *arXiv preprint arXiv:2308.14985*, 2023.

[182] Xiao Zang, Miao Yin, Jinqi Xiao, Saman Zonouz, and Bo Yuan. Graphmp: Graph neural network-based motion planning with efficient graph search. *Advances in Neural Information Processing Systems*, 36:3131–3142, 2023.

[183] Clark Zhang, Jinwook Huh, and Daniel D Lee. Learning implicit sampling distributions for motion planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3654–3661, 2018.

[184] Xin Zhao, Huan Zhao, Shaohua Wan, and Han Ding. A Gaussian Mixture Models based Multi-RRTs method for high-dimensional path planning. In *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1659–1664. IEEE, 2018.

[185] You Zhou, Jianfeng Gao, and Tamim Asfour. Learning via-point movement primitives with inter-and extrapolation capabilities. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4301–4308. IEEE, 2019.

[186] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. Chomp: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 32(9-10):1164–1193, 2013.

List of Figures

List of Tables