Learning geometric reasoning for multi-robot task and motionplanning

Smail Ait Bouhsain

RÉPUBLIQUE FRANÇAISE
Liberté Égalité Fraternité

Université de Toulouse

Doctorat de l'Université de Toulouse

préparé à l'INSA Toulouse

---

méthodes d'apprentissage de raisonnement géometrique pour la planification de tâches et de mouvements multi-robots.

Thèse présentée et soutenue, le 16 avril 2025 par

Smail AIT BOUHSAIN

École doctorale
SYSTEMES

Spécialité
Robotique et Informatique

Unité de recherche
LAAS - Laboratoire d'Analyse et d'Architecture des Systèmes

Thèse dirigée par
Thierry SIMEON

Composition du jury
M. Olivier SIMONIN, Président, INSA Lyon
Mme Lydia KAVRAKI, Rapporteure, Rice University
M. Rachid ALAMI, Examinateur, CNRS Occitanie Ouest
M. Justin CARPENTIER, Examinateur, Inria de Paris
M. Oliver BROCK, Examinateur, TU Berlin
M. Thierry SIMEON, Directeur de thèse, CNRS Occitanie Ouest

Remerciements

Il y a des voyages qui marquent une vie, et cette thèse en est un. Elle a été faite de doutes, de découvertes, de nuits blanches, de rencontres, et surtout de beaucoup d'apprentissage.

Je tiens tout d'abord à exprimer ma profonde reconnaissance à mes deux encadrants, Nic et Rachid. Merci pour votre engagement constant, vos conseils avisés, votre soutien, et surtout pour la confiance que vous m'avez accordée. Merci pour votre patience, votre compréhension et votre motivation. Nos échanges, nos discussions, vos relectures attentives et vos corrections minutieuses ont largement contribué à la qualité de ce travail. Vous avez su, très tôt, comprendre et accepter mes méthodes de travail atypiques, et pour cela, je vous en suis particulièrement reconnaissant.

À ma mère Naima, il est difficile de trouver les mots justes. Merci d'avoir fait de moi l'homme que je suis aujourd'hui. Merci pour ton soutien inconditionnel, pour tes innombrables sacrifices, pour ton intérêt constant pour mon travail, même quand tu n'en comprenais pas tous les détails. Tu as toujours été là, fière de chaque petite victoire, et je te dédie une grande partie de ce travail.

À mon père Abdelfettah, merci pour ta sagesse, tes précieux conseils, pour avoir toujours répondu présent quand j'avais besoin de toi. Tu m'as constamment poussé à donner le meilleur de moi-même et tu as été, et restes encore, un modèle de rigueur et de persévérance.

À ma compagne Kenza, mon soutien indefectible tout au long de cette aventure, malgré les 700 kilomètres qui nous séparaient. Merci pour ta patience, ta compréhension et ton amour inconditionnel. Merci d'avoir été là dans les moments difficiles, de m'avoir encouragé à persévérer et d'avoir partagé les joies et les succès de cette thèse. Merci pour les massages, les repas, les câlins, les sorties, les voyages, les soirées séries, les soirées discussions, les soirées à ne rien faire, et pour tout le reste. Sans toi, cette thèse aurait sans doute perdu une grande partie de sa lumière et de son équilibre. Tu as été mon ancrage, mon réconfort, et la douceur dont j'avais besoin pour avancer, jour après jour.

À mon frère Badreddine et ma sœur Nour El Houda, merci pour votre soutien moral, vos encouragements et votre présence constante dans ma vie. Merci pour les moments de joie partagés, les rires, et pour me rappeler que la famille est toujours là, même à distance. Merci de me faire confiance comme grand frère.

À mes amis Bouds, Yas, Othy, Wiwi, Ismail, Anas, Laf, Malak, Lamia, Marouane, merci pour votre amitié, pour ces moments où, le temps d'une soirée, d'un dîner festif ou même d'un simple Coca Zéro en terrasse, j'oubliais le stress et les difficultés de la thèse.

Les collègues rencontrés pendant cette thèse ont été bien plus que des partenaires de bureau. Je tiens tout d'abord à remercier Simon, mon partenaire de thèse, mon compagnon de voyage, mon hébergeur officiel, mon conseiller administratif. Merci pour ta gentillesse, ta bonne humeur, ton soutien et ta rigueur. C'était un plaisir de partager cette aventure avec toi, et c'est un honneur de te compter parmi mes coauteurs, et surtout parmi mes amis. Et bien sûr merci aux Swifties, Guillaume, Philippe, Lou et Bastien. Merci d'avoir fait de notre carré, pendant les quelques fois où j'y étais, un endroit de travail à la fois agréable et chaleureux.

Je remercie également Valentin, Jeremy, Amandine, William, Ilinka, Phani, Antoine, Virgile, Anthony, Roland, Baptiste, Laure, Stephy, Adrien, Rebecca, Gianluca et Dario, d'avoir été des collègues formidables, de m'avoir accueilli parmi vous et de m'avoir permis de ne jamais me

sentir comme un outsider.

Je tiens à remercier Malik pour ses conseils et son feedback inestimables, Arthur pour m'avoir pris sous son aile lors de mes missions d'enseignement, Félix, Juan, Marco, Aurélie, pour avoir fait de notre équipe RIS un endroit où faire de la recherche est un plaisir. Une pensée particulière à Simon Lacroix, qui a dirigé cette équipe pendant ma thèse. Merci pour ta compréhension, ton écoute, tes qualités humaines et ton leadership. Je n'aurais pas pu rêver un meilleur cadre pour mener à bien cette thèse.

Un grand merci aux membres de mon jury, Lydia, Olivier et Oliver, pour leur temps, leur disponibilité, pour avoir accepté de juger ce travail, et pour leurs retours constructifs et leurs encouragements.

Enfin, merci au LAAS-CNRS de m'avoir accueilli et permis de réaliser cette thèse dans un environnement de recherche stimulant, bienveillant et enrichissant.

À toutes et tous, du fond du cœur : merci.

Abstract

Robotic Systems are increasingly deployed in diverse complex applications, where the ability to perform long-horizon manipulation tasks in cluttered environments is crucial. Task and Motion Planning (TAMP) offers a unified framework that integrates symbolic decision-making with geometric reasoning to achieve this goal. However, this hybrid discrete/continuous nature of TAMP generally leads to a combinatorial explosion of the search space, requiring a large number of costly geometric planning queries to ensure solution feasibility. This results in a significant computational burden, especially in highly constrained environments, with many TAMP algorithms struggling to find solutions in a reasonable time frame. These challenges are exacerbated in the context of multi-robot systems, where the complexity of the planning problem increases significantly. In this thesis, we identify key limitations of typical TAMP approaches and propose a novel framework for single and multi-robot task and motion planning using learned geometric reasoning, that is capable of tackling complex manipulation tasks in cluttered environments.

To this end, we first introduce two action and grasp feasibility prediction methods that leverage deep learning to provide fast geometric feedback to the TAMP planner. The first method, called **AGFPNet**, demonstrates how such learning approaches can be used to accelerate the TAMP process by reducing the dependency on costly geometric planning calls for action feasibility verification. The second method, called **GRN**, takes advantage of graph-based scene representations and graph neural networks to improve the generalizability, scalability and interpretability of feasibility predictions. By also predicting the causes for action infeasibility, GRN can be used to guide the TAMP planner towards more promising regions of the search space. Moreover, we propose extensions of these methods to multi-robot systems and mesh-shaped objects, enabling the prediction of the feasibility of collaborative actions involving objects with diverse shapes. We show that these models achieve state-of-the-art performance in action and grasp feasibility prediction tasks. We also demonstrate that the proposed **GRN** model outperforms existing methods in terms of generalization to unseen environments and scalability to larger scenes, while also providing interpretable predictions.

Using these methods, we propose a multi-robot feasibility-informed TAMP algorithm that integrates learned geometric reasoning into a search-based TAMP framework. This algorithm leverages efficient feasibility predictions as heuristics for exploring the search space more effectively by prioritizing promising actions, thus resulting in a significant speedup of the planning process. In parallel, we introduce a novel informed backtracking mechanism that allows the planner to use the predicted causes of infeasibility to construct planning constraints. These can then be used not only to prune the search space more effectively, but also to compute more accurate cost-to-goal heuristics. Through extensive experiments, we demonstrate that these contributions allow the proposed TAMP algorithm to solve complex manipulation tasks with very long planning horizons and in heavily cluttered environments, while achieving near-optimal performance in terms of planning time.

Keywords: Robotics, Task and Motion Planning, Machine Learning, Geometric Reasoning, Collaborative Manipulation, Multi-Robot Systems.

Résumé

Cette thèse, intitulée "Learning Geometric Reasoning for Multi-Robot Task and Motion Planning", se concentre sur l'amélioration de la planification des tâches et des mouvements (Task and Motion Planning - TAMP) en robotique en utilisant l'apprentissage automatique pour le raisonnement géométrique. La planification des tâches et des mouvements est une problématique clé en robotique, particulièrement pour la manipulation d'objets dans des environnements complexes et encombrés. Cette tâche est difficile en raison de la complexité combinatoire des problèmes de planification et du coût computationnel élevé des planificateurs géométriques. L'objectif principal de cette thèse est de réduire cette complexité et d'améliorer l'efficacité des algorithmes de planification en intégrant un raisonnement géométrique basé sur l'apprentissage automatique.

L'une des principales contributions de cette recherche est le développement d'AGFPNet, un réseau de neurones convolutif permettant de prédire la faisabilité d'actions et de prises dans des environnements 3D encombrés, à partir de représentations visuelles de ces derniers. Grâce à cette prédiction, il est possible de réduire le nombre d'appels aux planificateurs géométriques coûteux, en priorisant les actions les plus susceptibles d'être réalisables. Ce modèle améliore ainsi considérablement les performances en terme de taux de succès et d'efficacité computationnelle. Afin d'élargir l'applicabilité de cette approche, la thèse explore également son extension aux problèmes multi-robots ainsi qu'à la manipulation d'objets en maillage 3D.

En parallèle, la thèse introduit les Geometric Reasoning Networks (GRN), un modèle avancé basé sur des graphes modélisant les relations géométriques entre les objets dans un environnement. Ce modèle surmonte les limitations des approches basées sur l'image en résolvant les problèmes d'occlusion. GRN offre également des informations plus riches à travers la prédictions des raisons d'infaisabilité d'une action, notamment la prédiction de l'existence de solution de cinématique inverse et l'estimation des obstructions de prises par d'autres objects. Ces informations peuvent ainsi être utilisées afin de rectifier l'infaisabilité et mieux guider le planificateur vers une solution géométriquement faisable. En plus de sa précision accrue, ce modèle se distingue par son interprétabilité, scalabilité, sa capacité de généralisation à des environnements plus complexes et par son adaptabilité aux applications réelles sur des robots tels que Franka Emika Panda et PR2. En outre, cette thèse propose également une méthode novatrice pour intégrer ces prédictions de faisabilité et de raisons d'infaisabilité dans un algorithme de planification des tâches et des mouvements via un backtracking informé. Cette technique améliore considérablement la recherche de solutions en grâce à un pruning plus efficace de l'arbre de recherche et une estimation plus précise du coût pour atteindre l'objectif. Les expérimentations démontrent que cette approche permet de résoudre efficacement des problèmes de planification complexes, atteignant un taux de succès de 100% sur plusieurs benchmarks. De plus, cette méthode surpasse considérablement plusieurs approaches TAMP précédentes.

En résumé, cette thèse propose une approche innovante et performante pour la planification des tâches et des mouvements en robotique. En exploitant l'apprentissage profond pour prédire la faisabilité des actions et optimiser la recherche de solutions, elle ouvre de nouvelles perspectives pour la robotique autonome et collaborative, avec des applications potentielles dans des secteurs variés allant de l'industrie manufacturière à la robotique domestique.

Mots-clés : Robotique, Planification des tâches et des mouvements, Apprentissage automatique, Raisonnement géométrique, Manipulation collaborative, Systèmes multi-robots.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

In recent years, the field of robotics has witnessed unprecedented growth, with robots now playing essential roles in diverse applications spanning manufacturing, healthcare, logistics, and even home assistance. These advancements are driven by a combination of hardware improvements, sophisticated planning and reasoning algorithms, breakthroughs in artificial intelligence, and advanced control systems, which enable robots to interact with and adapt to their environments. From autonomous vehicles navigating city streets to robotic arms assembling intricate components on production lines, robots are increasingly deployed to tackle complex tasks, often in close proximity to humans and in unstructured, dynamic environments.

As the range and complexity of robotic applications expand, so does the need for robots to possess advanced capabilities in reasoning, decision-making, and planning. Among these, one of the most critical skills is the ability to plan and execute long-horizon, complex manipulation tasks. In environments where robots interact with a variety of objects and must respond to changing constraints, they must be capable of understanding intricate spatial relationships, predicting the outcomes of their actions, and adjusting their strategies as new information becomes available. Effective task and motion planning (TAMP) lies at the heart of this capability, enabling robots to autonomously plan long sequences of actions and manage physical interactions in ways that are safe, efficient, and adaptable. While task and motion planning may seem like an easy task for humans, it is a challenging problem for robots due to the high dimensionality of the state and action spaces, the need to reason about both symbolic and geometric aspects of the environment, and the requirement to generate long-horizon plans in a timely manner. This thesis focuses on developing algorithms and methods for task and motion planning that are capable of handling these challenges.

Before diving into the specifics of TAMP, we first provide an illustrative example to motivate the challenges and complexities of planning for robotic manipulation in a real-world scenario.

1.1 Task and Motion Planning: An Illustrative Example

Consider an autonomous robot worker in a grocery store tasked with restocking shelves with new goods, as shown in Figure 1.1. The robot is equipped with a robotic arm and a gripper, and is responsible for picking up items from a carton containing new products, and placing them on the shelves. A common strategy for waste reduction in grocery stores is to place older products at the front of the shelf so that customers are more likely to purchase them before the newer products. Therefore, the robot must place the new products behind the older ones, while ensuring that the products are neatly arranged and that the shelves are not overfilled. Additionally, the robot must be able to adapt to changes in the environment, such as the presence of obstacles in front of the shelves, or the need to move around customers or other workers in the store.

More formally, the robot's task is to generate a sequence of actions that transitions the environment from an initial state (e.g., the current state of the shelves) to a goal state (e.g., the desired arrangement of products on the shelves), as well as the corresponding motion trajectories that enable the robot to complete these actions. This is a typical task and motion planning problem, in which the robot must reason about both symbolic aspects of the task (e.g., the order in which products should be picked or placed on the shelves) and geometric aspects of the environment (e.g., kinematic constraints of the robot arm, collision avoidance with obstacles).

For humans, this task is relatively straightforward, as we can easily reason about the spatial relationships between objects and predict the outcomes of our actions. As a result, the solution to this problem might seem trivial: we start by removing the older products from the shelves, then we place the new products behind them, and finally, we put the older products back in front of the new ones. However, for a robot, this task is far from trivial, as it requires reasoning about a large number of variables, constraints, and uncertainties. The environment is highly cluttered, with many objects on the shelves, in the carton, and on the floor, making it difficult to plan collision-free trajectories for the robot arm. Additionally, the environment contains a large number of objects, making the space of possible actions and states very large. Moreover, the robot must reason about the spatial relationships between objects and understand what makes

Figure 1.1: An autonomous robot worker in a grocery store restocking shelves with new products.

an action feasible or not. While mitigating these difficulties, the robot must answer a number of questions, such as: Which products should be picked up first? How should each product grasped? Where should each product be placed ? What motion allows to avoid collisions with obstacles? How can access to a desired position be freed?

In this application, as in most task and motion planning problems, the symbolic and geometric aspects of the problem are deeply intertwined, and the robot must be able to reason about both simultaneously. However, this coupling of symbolic and geometric reasoning raises a number of challenges for task and motion planning.

1.2 The Challenges of Task and Motion Planning

The hybrid symbolic/geometric nature of task and motion planning presents a number of challenges. Below, we expand on these challenges in greater detail:

High-Dimensional Action and State Spaces: The action space in task and motion planning is extremely large. At any given step, there are numerous potential actions the robot could take. For example, any object can be picked up using different possible grasps, and each object can be placed in various positions, all of which need to be considered during planning. The intertwined nature of symbolic and geometric reasoning leads to a combinatorial explosion during planning. The planner must explore a vast number of potential combinations of actions and corresponding geometric paths to find a solution that is both symbolically correct and geometrically feasible. As the number of objects or possible actions increases, the number of potential task plans grows exponentially. This combinatorial explosion makes it computationally expensive to find feasible plans, especially when considering long-horizon tasks. The high dimensionality of the state space further complicates matters because it not only accounts for the robot's configuration, but also the arrangement and properties of all the objects in the environment. Managing these vast action and state spaces is a fundamental challenge for task and motion planning.

Symbolic-Geometric Coupling: Another significant challenge arises from the inherent coupling between task-level (symbolic) and motion-level (geometric) planning. Symbolic planning involves reasoning over discrete actions, such as deciding which object to pick or where to place it, whereas geometric planning involves finding the continuous paths required to execute these actions without collisions. Actions that appear feasible symbolically may turn out to be geometrically infeasible due to factors like collisions, kinematic limitations, or reachability constraints. For instance, while it may be symbolically possible to move an object to a certain location, the motion required to achieve this may be blocked by another object or require the robot to adopt an infeasible configuration. This means that symbolic planners must repeatedly verify their decisions through geometric checks, significantly increasing planning time and computational cost.

Dependency on Expensive Geometric Feasibility Checks: Task and motion planning strongly depends on geometric planners to verify the feasibility of considered actions. Geometric feasibility checks involve evaluating whether a given trajectory is collision-free and whether the robot can execute the action while respecting its kinematic constraints. These checks are computationally expensive, especially for infeasible actions that require multiple attempts before identifying that a valid solution does not exist. In TAMP, a significant amount of planning time is often wasted on evaluating actions that ultimately turn out to be infeasible. This dependency on repeated geometric feasibility checks, particularly when dealing with cluttered or constrained environments, becomes a major bottleneck. This issue is exacerbated as the number of objects and possible actions increases.

Generalization Across Tasks and Environments: Another major challenge in TAMP

is generalization. Ideally, a planner should be able to generalize across different tasks and environments without requiring extensive re-engineering or retraining. However, real-world environments vary widely in terms of object types, spatial layouts, and task requirements. Planners must adapt to new conditions and learn from previous experiences to improve efficiency. This generalization requires sophisticated learning-based approaches that can infer feasible plans from prior knowledge and apply them to novel scenarios. Such capabilities are crucial for building versatile robots that can operate in various environments, from factories to homes, without significant modifications to their planning systems.

Complexity of Multi-Robot settings: Multi-robot collaboration introduces an additional layer of complexity in task and motion planning. As the number of robots increases, the dimensionality of the action space grows significantly, leading to an even larger combinatorial explosion. Each robot must plan its own actions while collaborating with other robots to avoid collisions and contribute to the overall task. Checking the geometric feasibility of collaborative actions, such as handing off an object between two robots, requires considering the combined kinematics and workspaces of multiple robots. This adds significant computational complexity, as the system must ensure that both robots can execute their parts of the task without interference.

Overall, task and motion planning is a highly challenging problem that requires the development of efficient algorithms capable of dealing with high-dimensional action spaces, tightly coupled symbolic and geometric reasoning, and the ability to generalize across different environments. The contributions of this thesis aim to address these challenges by developing novel learning-based models, interpretability mechanisms, and extensions to multi-robot and complex object scenarios.

1.3 Geometric Reasoning for Task and Motion Planning

Geometric reasoning is a fundamental aspect of task and motion planning that addresses some the challenges of TAMP. By reasoning explicitly about the geometric properties of the environment, TAMP planners can reduce their heavy reliance on computationally expensive geometric planning components, while simultaneously gaining valuable insights to guide the planning process. Traditionally, a common approach to handle geometric infeasibility is through symbolic and geometric backtracking, where the planner revisits previous decisions and attempts alternative paths whenever infeasibility is encountered. Although effective in some cases, backtracking is generally not scalable, especially when considering the combinatorial explosion of potential actions and state configurations, leading to an exponential increase in planning time.

To move beyond basic backtracking, this thesis introduces more sophisticated forms of geometric reasoning through learning-based feasibility prediction models. Specifically, we leverage action and grasp feasibility prediction networks to predict the likelihood of action success based on the current state of the environment. This predictive capability allows the planner to prioritize actions that have a higher chance of success, effectively reducing the number of costly calls to geometric planners. As a result, the overall task and motion planning process is accelerated, making it more suitable for real-time applications. Another major benefit of integrating learned geometric reasoning into task and motion planning is its potential for enhancing interpretability. For a robot operating in a cluttered environment, the ability to identify and explain why a certain action or grasp is infeasible (e.g. collisions with other objects or kinematic limitations) can significantly enhance the planning process. This interpretability allows the planner to adapt its strategies, avoid repeated failures, and provide more robust solutions. For example, if an attempted grasp is obstructed by another object, the planner can utilize this feedback to reconfigure the environment or choose a different action to proceed.

In addition to predictive feasibility checking, **Geometric Reasoning Networks (GRN)** developed in this thesis provide a richer understanding of the spatial relationships between objects and help mitigate infeasibility before it occurs. These networks use graph-based scene representations and leverage geometric features, such as inverse kinematics (IK) feasibility and grasp obstructions, to predict not only the feasibility of actions but also the causes of infeasibility. This capability allows the planner to proactively adapt its strategies, further reducing the computational overhead typically associated with geometric feasibility checks. In summary, geometric reasoning plays a critical role in addressing some of the core challenges of task and motion planning. By integrating learning-based models for action and grasp feasibility, as well as developing interpretability mechanisms, this thesis aims to create a more efficient and reliable task and motion planning framework capable of tackling a wide range of robotic manipulation tasks, from simple object relocation to complex multi-robot collaboration.

1.4 Contributions

This thesis proposes a series of novel contributions aimed at advancing the state of task and motion planning in robotics. The main contributions include:

- **Action and Grasp Feasibility Prediction:** One of the key contributions of this thesis is the development of action and grasp feasibility prediction as an efficient way to replace expensive geometric feasibility checks. Instead of repeatedly relying on costly geometric planners to verify the feasibility of each potential action, we propose using learning-based predictions to provide a fast and reliable assessment of feasibility. We propose two neural networks, **AGFPNet** and **GRN**, capable of simultaneously predicting both action feasibility and grasp feasibility, significantly reducing the computational burden associated with geometric queries. These predictions provide valuable heuristics that can guide the task and motion planning process, focusing computational resources only on the most promising actions and grasps.
- **Feasibility-informed TAMP Algorithms:** The predicted feasibility scores are directly integrated into TAMP algorithms to effectively guide the exploration of the search space and accelerate the planning process. By using feasibility predictions as heuristics, the planner can prioritize actions with a higher likelihood of success, thereby avoiding the exploration of infeasible paths. This integration not only speeds up planning but also improves the overall success rate of task and motion planning in complex environments.
- **Interpretability Mechanisms for Planning Guidance:** Another important contribution is the incorporation of interpretability mechanisms into the feasibility prediction framework. By identifying and explaining the reasons for infeasibility, such as potential collisions or kinematic limitations, the planner can use this information to adapt its strategy and re-plan effectively. These interpretability mechanisms provide insights that help in modifying the symbolic plan or selecting alternative actions that are more likely to succeed, thus improving the robustness of the planning process.
- **Extensions to Multi-Robot settings and Complex Object Scenarios:** To push the boundaries of existing TAMP methods, we extend our approach to handle multi-robot task and motion planning scenarios, as well as environments that involve mesh-shaped and complex objects. By leveraging the action and grasp feasibility predictions in these challenging settings, we demonstrate the scalability and adaptability of our approach. The proposed methods enable efficient collaboration between multiple robots, making our approach suitable for diverse applications in real-world environments.

1.5 Outline

This thesis is organized as follows:

Chapter 2 provides a review of the literature on task and motion planning, starting with some background on task planning an motion planning, then covering both classical TAMP approaches and recent learning-based methods that seek to address the challenges discussed.

Chapter 3 defines the problem addressed by this thesis and introduces key concepts and definitions. Additionally, it presents a reference search-based Task and Motion Planning (TAMP) algorithm, which we use as a baseline and aim to enhance throughout this thesis using our various contributions.

Chapter 4 introduces the **Action and Grasp Feasibility Prediction Network (AGFPNet)**, a Convolutional Neural Network (CNN) developed for fast feasibility checking of actions and grasps to reduce the reliance on computationally expensive geometric planners. This chapter also presents the integration of AGFPNet into our reference TAMP algorithm, significantly improving its overall efficiency.

Chapter 5 presents **Geometric Reasoning Networks (GRN)**, an advanced model designed to overcome the limitations of **AGFPNet**. **GRN** provides richer information by identifying reasons for infeasibility at a fraction of the inference time of **AGFPNet**, which can be leveraged to further guide and accelerate task and motion planning.

Chapter 6 proposes a task and motion planning framework that leverages learned geometric reasoning. By representing reasons of infeasibility as planning constraints, the proposed method takes advantage of the interpretability mechanisms provided by **GRN** to better guide the search towards a geometrically feasible solution, considerably reducing planning time, and allowing larger and more complex problems to be solved. Finally, chapter 7 concludes the thesis and discusses potential directions for future work.

1.6 List of Publications

The following publications were produced during the course of this research:

Published:

- *Bouhsain, Smail Ait, Rachid Alami, and Thierry Simeon. "Learning to predict action feasibility for task and motion planning in 3d environments." IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2023.*
- *Bouhsain, Smail Ait, Rachid Alami, and Thierry Simeon. "Simultaneous Action and Grasp Feasibility Prediction for Task and Motion Planning through Multi-Task Learning." IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2023.*
- *Bouhsain, Smail Ait, Rachid Alami, and Thierry Simeon. "Extending Task and Motion Planning with Feasibility Prediction: Towards Multi-Robot Manipulation Planning of Realistic Objects." IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2024.*
- *Bouhsain, Smail Ait, Rachid Alami, and Thierry Simeon. "Learning Geometric Reasoning Networks for Robot Task and Motion Planning." The Thirteenth International Conference on Learning Representations (ICLR). 2025.*

As co-author:

- *Wasiela, Simon, Smail Ait Bouhsain, Marco Cognetti, Juan Cortés, and Thierry Simeon. "Learned Uncertainty Tubes via Recurrent Neural Networks for Planning Robust Robot Motions." 27th European Conference on Artificial Intelligence (ECAI). 2024.*

Chapter 2

Literature Review

Contents

2.1 Background

Task and motion planning (TAMP) takes its roots from two well-studied research fields: task planning and motion planning. Task planning entails reasoning about symbolic states and generating sequences of discrete actions to achieve specified goals. Motion planning, on the other hand, focuses on computing continuous, collision-free paths that satisfy the geometric and kinematic constraints of the robot. The interplay between these fields arises from the need to reconcile high-level symbolic reasoning with the low-level geometric realities of real-world scenarios. TAMP addresses challenges that emerge from this coupling By bridging these domains, TAMP offers a unified framework for solving intricate planning problems with both symbolic and geometric components. In this section, we review some of the well-established methods in task planning and motion planning.

2.1.1 Motion Planning

Motion planning forms the backbone of robotics, focusing on computing feasible paths for robots to move from an initial configuration to a goal configuration while avoiding collisions. Foundational methods in motion planning encompass a variety of approaches, each with its own strengths and limitations. The earliest works in motion planning were search-based methods, such as Dijkstra's algorithm (Dijkstra 1959) and A* (Hart et al. 1968), which discretize the

configuration space, systematically exploring paths to find the shortest or most cost-effective route to the goal. Although Dijkstra's algorithm ensures optimality, it is computationally expensive, making it unsuitable for high-dimensional configuration spaces such as those required for robotic arms. A* improves upon Dijkstra by incorporating heuristics to prioritize nodes closer to the goal, thereby reducing the number of explored nodes. However, A* remains limited to systems with low degrees of freedom and struggles in high-dimensional, continuous spaces. Artificial potential fields (APFs) (Khatib 1986) offer a different perspective on motion planning, using potential functions to generate a virtual field where the robot is attracted to the goal and repelled from obstacles. While APFs are simple to implement and computationally efficient for real-time applications, they suffer from significant limitations. These include the risk of local minima, where the robot can become stuck in non-goal states, and their inability to handle complex environments or high-dimensional systems.

Lozano-Pérez and Wesley 1979 formalized the motion planning problem as a search in the Configuration Space (C-Space) of the robot, which dimensions represent the controllable joints of the robot. The C-Space is a high-dimensional space where each point corresponds to a valid configuration of the robot. The motion planning problem is then to find a continuous path in the C-Space that connects the initial and goal configurations while avoiding collisions with obstacles. This representation of the motion planning problem has been widely adopted and serves as the foundation for many motion planning algorithms. Specifically, it gave rise to a class of approaches known as sampling-based methods, such as Probabilistic Roadmaps (PRM) (Kavraki et al. 1996) and Rapidly-exploring Random Trees (RRT) S. LaValle 1998. These methods overcome the limitations of search-based approaches by probabilistically sampling the configuration space. PRMs construct a roadmap by sampling random configurations and connecting them with linear interpolation or local planners, making them effective for multi-query problems. However, PRMs require dense sampling to ensure probabilistic completeness, which can be computationally expensive. RRTs grow a tree by incrementally exploring random configurations and prioritizing unexplored regions, offering efficiency for single-query problems. Nevertheless, RRTs often lack path optimality. Advanced variants such as RRT* and PRM* (Karaman and Frazzoli 2011) address these limitations by iteratively refining paths to ensure asymptotic optimality. Despite these advancements, sampling-based methods still face challenges in highly constrained environments, particularly when narrow passages or intricate constraints are present. Also, these methods can be computationally expensive for high-dimensional systems or complex environments, requiring significant planning time to generate feasible paths. Moreover, they are unable to detect motion infeasibility, as there is no way to guarantee that a motion planning problem is infeasible unless the configuration space is fully explored.

Optimization-based methods formulate motion planning as an optimization problem, aiming to minimize a cost function while adhering to constraints. Covariant Hamiltonian Optimization for Motion Planning (CHOMP) (Ratliff et al. 2009) uses trajectory optimization to smooth paths while accounting for collision avoidance and kinematic constraints. However, CHOMP is prone to convergence in local minima, limiting its ability to find globally optimal solutions. Stochastic Trajectory Optimization for Motion Planning (STOMP) (Kalakrishnan et al. 2011) introduces stochastic perturbations to improve trajectories, increasing robustness against local minima but at the cost of higher computational complexity. Trajectory Optimization (TrajOpt) (Schulman et al. 2014) employs sequential convex optimization, balancing efficiency and flexibility, but requires careful parameter tuning and struggles with convergence in highly non-convex spaces. These methods offer fine-grained control over trajectories but demand significant computational resources. They also face difficulties in scaling to complex, high-dimensional systems. Moreover, they are highly sensitive to the initial guess provided for the trajectory, as it significantly influences

convergence and solution quality. Furthermore, these methods lack global or probabilistic completeness.

Recent works have explored learning-based approaches to motion planning, leveraging neural networks to predict feasible paths or optimize trajectories. Learning-based methods offer the potential to generalize across different environments and robot configurations, learning from experience to improve planning efficiency. Deep Reinforcement Learning (DRL) (Everett et al. 2018) and Generative Adversarial Networks (GANs) (T. Zhang et al. 2021) have been used to generate collision-free paths, while Variational Autoencoders (VAEs) Hu et al. 2025 have been employed to optimize trajectories. These methods offer promising results in terms of generalization and efficiency, but they require large amounts of training data and can be difficult to train. Moreover, they lack guarantees on completeness and path optimality, making them unsuitable for safety-critical applications.

Sampling-based and Optimization-based methods have gained popularity in the robotics community due to their flexibility and scalability. Many variants and extensions have been proposed to address specific challenges (Bohlin and Kavraki 2000; J. J. Kuffner and S. M. LaValle 2000; Jaillet et al. 2010; Gammell et al. 2015; Strub and Gammell 2020). For a comprehensive review of motion planning methods, we refer the reader to Choset et al. 2005; S. M. LaValle 2006; Jean-Claude Latombe 2012; Orthey et al. 2023. In this thesis, we focus on sampling-based methods, as they offer a balance between efficiency and completeness, making them suitable for complex, high-dimensional systems. Nevertheless, optimization based methods can also be used with our proposed work.

2.1.2 Task Planning

Task planning focuses on reasoning over discrete actions to achieve desired goals, encompassing a variety of methods that address different aspects of the problem. Early methods, such as search-based approaches, employ algorithms like Breadth-First Search (BFS) (Moore 1959), Depth-First Search (DFS) (Shannon 1948), and A* (Hart et al. 1968). These methods operate by systematically exploring the state space to find an optimal sequence of actions. BFS guarantees finding the shortest path but is memory-intensive, while DFS is memory-efficient but may fail to find the optimal solution in infinite-depth spaces. A*, which combines cost-based evaluation with heuristic guidance, improves efficiency but is still limited in scalability to large or high-dimensional planning problems.

Logic-based planning introduced more structured frameworks such as STRIPS (Stanford Research Institute Problem Solver) (Fikes and Nilsson 1971) and PDDL (Planning Domain Definition Language) (McDermott et al. 1998). These approaches formalize task planning as a sequence of state transitions governed by preconditions and effects, creating a logical structure that ensures clarity and precision in defining tasks. STRIPS, in particular, offers a simple yet robust mechanism for action definition and goal specification. This mechanism operates by defining actions as a set of preconditions, effects. The preconditions specify the logical conditions that must be true for the action to be executed, while the effects describe how the world state will change after the action is performed. For example, a *pick* action may require a precondition that the robot's gripper is empty and the target object is within reach, and its effects would update the state to indicate that the object is now held by the gripper. This structure provides a declarative and systematic way to encode tasks, enabling planners to sequence actions logically from an initial state to a desired goal state. PDDL builds on this foundation by incorporating advanced features such as temporal and numerical constraints, making it suitable for more complex domains. However, these methods exhibit some limitations, especially in large and complex problems. The

mechanism assumes deterministic transitions and complete knowledge of the environment, which limits its adaptability to uncertain settings (Russell 2022). Additionally, as the complexity of tasks grows, the number of preconditions and effects can lead to scalability challenges (Geffner and Bonet 2013). This exponential growth not only increases the computational burden but also makes it challenging to find solutions in a reasonable amount of time. Indeed, reasoning over large state spaces with intricate interdependencies often leads to a combinatorial explosion, making these methods less practical without significant heuristics or approximations to reduce the computational load. Their applicability also heavily depends on the quality and completeness of the domain model, which can be challenging to construct in practical applications.

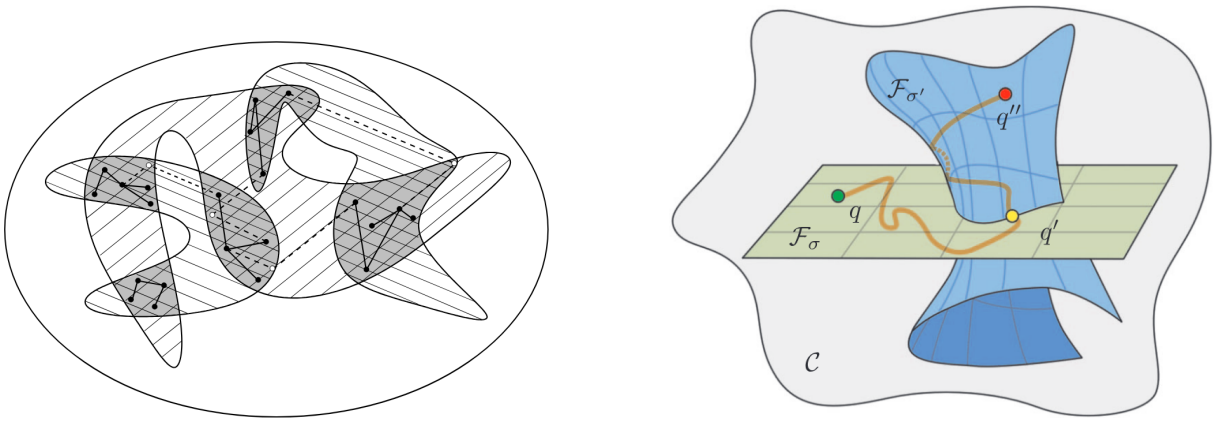
Hierarchical task planning, such as Hierarchical Task Networks (HTNs) (Erol et al. 1994), provides a structured framework for breaking down high-level goals into smaller, more manageable sub-tasks. This decomposition not only reduces the computational complexity of planning but also leverages domain-specific knowledge to guide the planning process efficiently. HTNs represent tasks as a hierarchy of methods, where each method defines how a high-level task can be decomposed into a sequence of lower-level tasks or primitive actions. For example, in a robot cooking scenario, a high-level goal such as *prepare-a-meal* can be broken down into sub-tasks like *chop-vegetables* and *cook-ingredients*. Each sub-task is further decomposed until primitive actions executable by the robot are identified. Systems like SHOP (Simple Hierarchical Ordered Planner) (D. Nau et al. 1999) and its successor SHOP2 (D. S. Nau et al. 2003) enhance HTNs by introducing features such as ordered task decomposition and dynamic goal reasoning, allowing planners to handle tasks with time-sensitive constraints or dynamically changing environments. Despite their strengths, HTNs face significant limitations. They heavily rely on accurate, detailed, and domain-specific knowledge to define the hierarchy of tasks and methods, which can be difficult to provide, especially in complex or poorly understood domains. Moreover, the rigid structure of task hierarchies may limit adaptability to unforeseen situations or novel tasks. These limitations pose challenges to scaling HTNs for real-world robotics applications, where environments are often unstructured and uncertainty is prevalent.

Probabilistic task planning methods, such as Markov Decision Processes (MDPs) (Bellman 1957) and Partially Observable Markov Decision Processes (POMDPs) (Kaelbling et al. 1998), address uncertainty by modeling probabilistic transitions and observations. MDPs excel in environments where the state is fully observable, while POMDPs extend this to partially observable settings. These approaches offer robustness to uncertainty but often require significant computational resources, limiting their scalability to large state spaces. In recent years, learning-based methods (Amir and Chang 2008; Arfaee et al. 2011; Aineto et al. 2018; Anderson et al. 2020) have gained prominence, leveraging machine learning to improve task planning efficiency and generalization. Techniques such as reinforcement learning (RL) (Sutton 2018) train agents to optimize policies through interaction with the environment, offering adaptability to complex and dynamic scenarios. Other methods utilize supervised learning to infer planning heuristics from expert demonstrations (Argall et al. 2009), accelerating the planning process. However, learning-based approaches can be data-intensive and may struggle with generalization to entirely novel tasks without extensive retraining. Despite their differences, these methods collectively contribute to the field of task planning by addressing various aspects of complexity, uncertainty, and efficiency, forming a robust foundation for hybrid task and motion planning approaches. A more comprehensive review of task planning methods can be found in Ghallab et al. 2004, 2016, 2025.

2.2 Task and Motion Planning

2.2.1 Manipulation Planning

The earliest works in task and motion planning (TAMP) (Lozano-Pérez et al. 1987) studied the problem of planning for a robot to manipulate objects in the environment, introducing concepts of grasp planning (Lynch and Mason 1996; Shimoga 1996; Bicchi and Kumar 2000) and object placement planning (Cosgun et al. 2011; Harada et al. 2012; Holladay et al. 2013; Haustein et al. 2019b) into motion planning. This work laid the foundation for manipulation planning, which extends motion planning by considering the interaction between robots and objects in the environment. Subsequent research (Alami et al. 1990; Koga and Latombe 1994; Ahuactzin et al. 1998; Siméon et al. 2004) formalized this perspective by representing the problem using manipulation graphs (Alami et al. 1990). These graphs represent the interplay between valid grasps CG, stable placements CP, and the motions that connect them. Nodes in the graph correspond to configurations where the robot holds an object at a valid grasp or where the object rests at a stable placement. Edges represent feasible motions, categorized as either *transit paths* (where the robot moves alone to a new configuration while the object remains stationary) or *transfer paths* (where the robot moves the object while maintaining a fixed grasp). The construction of manipulation graphs involves sampling grasp and placement configurations from the robot's configuration space, identifying regions where grasps and placements intersect, and computing collision-free motions between these regions. The goal is to ensure connectivity across these subspaces by identifying points of intersection, referred to as $\mathbf{CG} \cap \mathbf{CP}$, where the robot can transition between transit and transfer paths as shown in Figure 2.1a. The reduction property of manipulation planning ensures that any feasible manipulation sequence can be decomposed into a finite sequence of transit and transfer paths, greatly simplifying the search space (Siméon et al. 2004).



(a) Manipulation graph. Image source: Siméon et al. 2004

(b) Constraint manifolds. Image source: Hauser and Jean-Claude Latombe 2010

Figure 2.1: Illustration of multi-modal motion planning methods using manipulation graphs and constraint manifolds.

Planning on manipulation graphs involves searching for a path that alternates between these transit and transfer edges, effectively connecting an initial configuration to a goal configuration. However, as the number of objects in the scene increases, the size of the manipulation graph grows exponentially, leading to significant computational challenges. Despite these challenges, manipulation graphs remain a foundational approach for reasoning about object interactions and have inspired subsequent advancements in manipulation planning. Barry et al. 2013a extend

these method to consider non-prehensile actions such as pushing or pulling, while Yoshida et al. 2010 introduce pivoting actions, which expand the range of feasible solutions beyond direct pick-and-place strategies.

Hauser 2010; Hauser and Jean-Claude Latombe 2010; Hauser and Ng-Thow-Hing 2011; Barry et al. 2013b generalize these approaches under the umbrella of multi-modal motion planning.These methods represent interactions between the robot and movable objects as *mode* switches, where each mode corresponds to a kinematic state of the environment as shown in Figure 2.3. These modes represent manifolds of the configuration space of the robot . Task and motion planning problems are then solved by finding a geometrically feasible sequence of mode switches and motions through the corresponding manifolds that connect an initial state of the environment to a goal state (cf. Figure 2.1b). This paradigm allows for greater adaptability in complex scenarios, but it still faces challenges as the combinatorial complexity of TAMP problems grows.

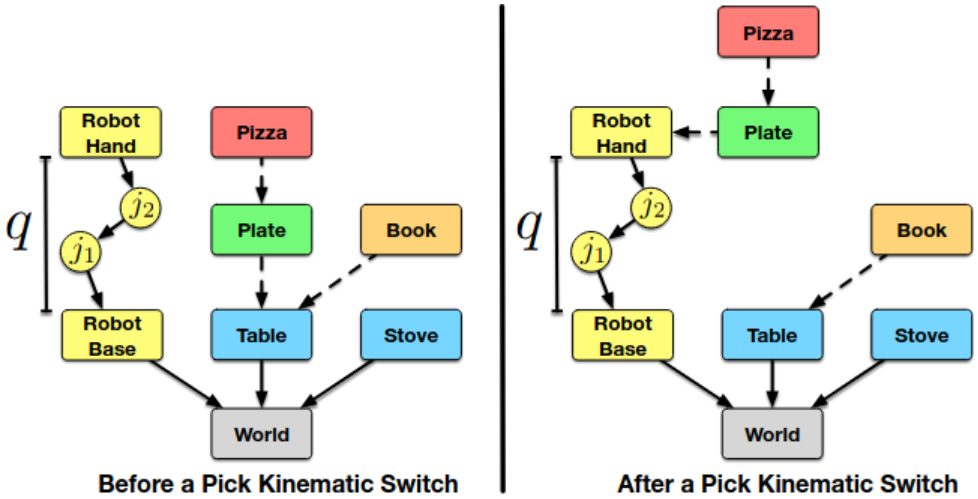


Figure 2.2: An illustration of a mode switch after a *Pick* action by the robot. Image source: Garrett et al. 2021.

Some recent methods (Mirabel et al. 2016; Englert et al. 2020; Lamiraux and Mirabel 2021) focus on representing actions using constraints that define manifolds. These constraints are comparable to the **CG** ∩ **CP** regions in manipulation graphs, and modes in multi-modal motion planning. These constrained motion planning methods aim at building a constraint graph, where nodes represent manifolds and edges represent transitions between them. The goal is then to find a sequence of manifolds that brings the environment from an initial to a goal state. However, if the number of objects increases, the constraint graph can be very complex to build. Also, a known issue with these approaches is the crossed foliation issue (Mirabel and Lamiraux 2017).

Optimization-based TAMP methods (Toussaint 2015; Phiquepal and Toussaint 2019; Zhao et al. 2024) formulate planning as a constrained optimization problem, integrating logical action sequencing with geometric reasoning to find feasible solutions. These methods treat the entire planning problem holistically as a mathematical program. One influential approach, Logic-Geometric Programming (LGP) (Toussaint 2015), introduces a first-order logic extension of a mathematical program, formulating the problem as a non-linear constrained optimization over the full trajectory of the world state, encoding symbolic action sequences as constraints. The optimization problem is then solved through three approximation levels: (i) optimizing only over the final geometric configuration, (ii) optimizing keyframe interactions where kinematic switches occur, and (iii) optimizing the full trajectory while respecting constraints such as collision avoidance and kinematic feasibility. LGP aims to provide optimal solutions in addition to ensuring

geometric feasibility. However, they face challenges in scalability due to the computational complexity of solving large non-linear constrained programs.

2.2.2 Combined Task and Motion Planning

Most of existing TAMP methods combine a discrete symbolic planner with a continuous geometric planner. The symbolic planner explores the symbolic states of the environment and possible actions to find a sequence of actions that achieves the desired goal. The geometric planner checks the feasibility of these actions and plans their corresponding motions. Cambon et al. 2009 pioneered this class of approaches through Asymov, a task and motion planning framework that generalizes the manipulation graph approach from Siméon et al. 2004 to multi-object and multi-robot TAMP problems. They introduce a task planner that maintains multiple roadmaps for each object and robot. The geometric planner then uses these roadmaps with heuristics provided from the task planner to find a feasible solution. Stilman and J. Kuffner 2008 generalize Navigation Among Movable Obstacles (NAMO) methods (Stilman and J. J. Kuffner 2005; Renault et al. 2024) to 3D manipulation, leveraging a reverse-order search and artificial constraints to solve monotonic TAMP problems. Srivastava et al. 2014 propose a method that uses a symbolic planner to generate a sequence of actions, which are then verified by a geometric planner, with an interface layer between the two. This interface layer provides feedback from the geometric planner to the symbolic planner, allowing the latter to refine its search based on the feasibility of actions. Lagriffoul et al. 2014 focus on geometric backtracking as a core issue in TAMP. They propose an informed backtracking strategy that uses revisits actions in failed symbolic plans while prioritizing objects that frequently cause collisions, and actions that led to kinematic failures. Dantam et al. 2016 leverage an incremental constraint-based approach to combine task and motion planning, allowing for the incremental addition of constraints to incorporate motion feasibility.

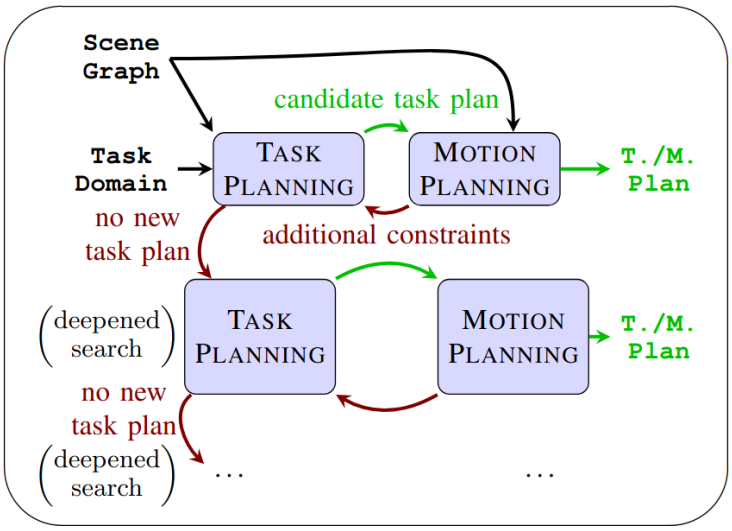


Figure 2.3: A diagram of typical combined task and motion planning methods. Image source: Dantam et al. 2016.

Garrett et al. 2018a propose to first discretize the geometric space, allowing the symbolic planner to plan in a *conditional reachability graph*. They also introduce a heuristic to guide the search process, which is computed by solving a relaxed planning problem. Ferrer-Mestres et al. 2017 propose a similar method, using *novelty* heuristics (Lipovetzky and Geffner 2012) which push the search towards unexplored regions of the state space. Garrett et al. 2018b build upon constrained

motion planning methods by modeling planning problems as factored transition systems, where state changes are defined by constraint intersections rather than full-space transformations. Garrett et al. 2020 introduce PDDLStream, a method that allows integrating symbolic task planning with blackbox samplers for motion planning in high-dimensional continuous spaces. Ren et al. 2024 propose a method that constructs an extended decision tree, combining Monte-Carlo Tree Search (MCTS) with top-k skeleton planning to efficiently search large, complex task and motion spaces. Other methods tackle task and motion planning under uncertainty. The latter can be due to sensing noise, state estimation, action execution errors, or incomplete knowledge of the environment. Şucan and Kavraki 2012 introduce a multi-graph based approach that uses probabilistic motion planning to compute feasibility probabilities for actions and propagate this information to the task planning level, allowing the task planner to make more informed decisions, favoring actions with higher probabilities of success. Kaelbling and Lozano-Pérez 2013 propose a multi-level hierarchical method that plans in belief space using probabilistic reasoning to decide when to take information-gathering actions, such as actively sensing an object before grasping. Ha et al. 2020 extend the LGP formulation (Toussaint 2015) to stochastic domains to handle uncertainty.

Previous works have also explored task and motion planning in multi-robot settings. T. Pan et al. 2021 propose a method that supports simultaneous discrete actions and uses multi-robot scheduling layer between the task planner and motion planner to explore different robot assignments dynamically. Umay et al. 2019 and Motes et al. 2023 propose graph-based frameworks to efficiently handle multi-robot task and motion planning. Other works explore task and motion planning from a human-robot interaction perspective. De Silva et al. 2014 propose a two-Layered control architecture for human-aware task and motion planning in Human-Robot Collaboration (HRC), integrating real-time human motion tracking and adaptive replanning to ensure safe and efficient task execution. Gharbi et al. 2015 introduce a bidirectional feedback mechanism where the symbolic planner provides constraints such as social norms and human comfort to guide geometric motion planning, while the geometric planner refines task feasibility and suggests alternative motion configurations to improve efficiency. Akbari et al. 2020 propose a human-aware planning model, where human operators assist in tasks that are difficult or infeasible for robots. The planner dynamically assigns actions to either the robot or a human, depending on task constraints.

These methods have significantly advanced the field of TAMP, offering a diverse range of approaches to tackle complex planning problems. However, they still face challenges, particularly in scalability, adaptability, and efficiency. The combinatorial complexity of TAMP problems can lead to significant computational burdens, especially in cluttered environments with multiple objects and robots. The need to reconcile symbolic reasoning with geometric planning poses challenges in ensuring that plans are both logically sound and geometrically feasible. These challenges motivate the need for more efficient, scalable, and adaptable TAMP methods that can handle complex, high-dimensional problems in real-world scenarios. For a more in-depth review of classical TAMP methods, we refer the reader to Garrett et al. 2021; Antonyshyn et al. 2023.

2.3 Learning for Task and Motion Planning

2.3.1 Learning Heuristics

Recent works in TAMP have explored a variety of learning-based methods to accelerate planning by providing heuristics or guidance to the task planner. Classical approaches to TAMP often rely on carefully crafted, domain-specific heuristics and manually encoded rules, which can

be inflexible and computationally expensive when scaled to complex tasks or unstructured environments. Learning-based methods seek to address these limitations by deriving heuristic functions, symbolic representations, or predictive models from collected data or prior experience.

A number of studies focus on predicting action affordances to inform both task- and motion-level decisions. For example, D. Xu et al. 2021 learn to predict future affordances, enabling planners to reason about the downstream implications of actions, while Curtis et al. 2022a propose a learned affordance estimation pipeline and integrate it with perception modules to handle previously unseen objects without relying on detailed geometric models. Other works aim at learning of geometric constraints and manipulation skills in order to streamline the search process. For instance, Ren et al. 2022 demonstrate how acquiring geometric constraints from previous experiences can drastically narrow down the solution space, while Abdo et al. 2013 show how the preconditions and effects of actions can be learned from a small set of demonstrations. X. Li and Brock 2022 learn to reproduce contact-rich manipulation skills from a single human demonstration by focusing on environmental constraints as the fundamental representation of a task rather than object-specific details. On the other hand, Ding et al. 2022 propose an object-centric TAMP algorithm that enables robots to plan and execute tasks without prior knowledge of object properties, by learning to ground objects dynamically using a physics engine. In parallel, Haustein et al. 2019a combine sampling-based motion planning with reinforcement learning (RL) and generative modeling to handle planar manipulations without the need for predefined manipulation primitives, and Zi Wang et al. 2018 learn sensorimotor primitives through Gaussian Process (GP)-based active model learning for actions such as pouring or stirring, these primitives can then be composed for longer-horizon tasks that require diverse skills. Similarly, Zi Wang et al. 2021 fuse learned and pre-existing sensorimotor skills for long-horizon tasks, and use them compositionally to solve long-horizon manipulation problems. Further work has explored learning state and action abstractions, such as Burbridge et al. 2012; Dearden and Burbridge 2014 which integrate learned symbolic abstractions with geometric reasoning to reduce the need for hand-crafted or domain-specific logic. Likewise, Curtis et al. 2022b propose a framework for learning state and action abstractions from few training examples. In the context, Silver et al. 2021b present a method for automatically learning symbolic operators directly from low-level robot interactions instead of relying on hand-coded symbolic models, enabling faster and more efficient planning for long-horizon robotic manipulation tasks. In parallel, Kim and Shimanuki 2020 introduce a relational state representation to express the state space using a small set of predicates such as which objects occlude the manipulation of others. Kim et al. 2022, on the other hand, present a hybrid approach to TAMP that combines learning and heuristic-based planning, that learns from planning experiences to improve task-level search heuristics and motion-level sampling. Other works aim at learning simple yet important heuristics such as identifying the smallest set of objects for solving a planning problem (Silver et al. 2021a).

Taken collectively, these works emphasize how learning-based heuristics for TAMP can reduce the computational overhead of planning by narrowing down the search space, learning new hard-to-model robot skills, and enhancing the robot's ability to adapt to previously unseen scenarios. These methods, however, still suffer from challenges related to generalizability, scalability, and interpretability, particularly when applied to complex, high-dimensional tasks or in uncertain environments.

2.3.2 Action Feasibility Prediction for Manipulation Planning

Action and grasp feasibility prediction is a fast-growing research topic, which aims at accelerating Task and Motion Planning (TAMP) algorithms by reducing their dependency on geometric planners. Wells et al. 2019 introduced the idea of using machine learning, particularly Support

Vector Machines (SVM), to classify action and grasp feasibility in tabletop environments. These feasibility predictions are then used as heuristics within a task planner, thereby reducing the number of expensive motion-planning calls. However, this method is limited to simple environments with a fixed number of objects. Indeed, the input to the SVM is a hand-crafted feature vector representing a pair of objects in the scene, making the method impractical for complex TAMP problems with many objects. Moreover, the method requires training a separate SVM for each (action, grasp) pair, further limiting its scalability.

To handle more complex scenes, some methods replaced hand-crafted features with top-view depth images. For instance, Driess et al. 2020a,b present a convolutional neural networks (CNNs) for predicting the feasibility of multiple (action, grasp) pairs, using a top-view depth image of the scene as input. This representation allows all objects in the environment to be encoded while keeping a fixed dimensionality. While this approach scales better to multiple objects, it has generally been restricted to tabletop settings, where occlusions and complex 3D arrangements do not pose as significant a challenge. L. Xu et al. 2022 made a first contribution towards a generalization to 3D environments, by providing the relative position of the support surface to the robot's base as input to the classifier. However, only a small area from the scene in represented. It is also not applicable to environments containing multiple support surfaces, especially when they act as obstacles. Addressing the need for 3D scene understanding, Yang et al. 2022 introduce an approach capable of processing multiple scene views combined with text descriptions of actions and predicates, leveraging a Transformer model to infer feasibility for entire action sequences. However, heavy reliance on image-based representations makes such methods susceptible to occlusions, adversely affecting their prediction accuracy.

Alternative representations have been explored to capture more nuanced 3D geometry. For instance, Park et al. 2022 use 3D voxel grids with variational autoencoders (VAE) to represent the scene and predict the feasibility of different grasp modes in multi-robot settings. H. Huang et al. 2024, on the other hand, leverage a pointcloud-based representation as input to a bi-equivariant policy learning model for robotic pick-and-place tasks in 3D manipulation. While these methods offer more detailed scene representations, they suffer from high computational overhead, limiting their applicability to environments containing many objects, as dense 3D data representations generally require significant inference time. A distinct line of work aims to reduce computational overhead in 3D scenes by encoding object relationships in scene graphs. For instance, Khodeir et al. 2023a use Graph Neural Networks (GNNs) to predict the success of geometric planning steps from prior search experiences. In these approaches, the connectivity of the scene graph encodes symbolic interactions (e.g., object on table) among objects and the robot, while node feature vectors capture object properties (e.g., shape, size, pose). The GNN then learns to prioritize object expansion in PDDLStream problems, guiding a best-first search. Finally, S. Li and Dantam 2021 propose a method for learning proofs of motion planning infeasibility, allowing early detection of infeasible actions.

While these methods offer an efficient way to provide fast geometric feedback to the task planner through feasibility predictions, they often lack interpretability and can not provide feedback on why actions are infeasible. This limitation motivates the need for more interpretable and generalizable methods that can handle complex 3D scenes with multiple objects and obstacles.

2.3.3 Learning to Plan

Recent works in TAMP have also focused on learning to plan in an end-to-end manner rather than predicting heuristics, bypassing the need for a separate task and motion planner. For instance, Qureshi et al. 2021 propose a deep-learning-based approach for multi-step object rearrangement

in environments with unknown object. This method uses a graph neural network (GNN)-based representation to process scene observations and determine an optimal sequence of pick-and-place actions to achieve a desired goal configuration. Similarly, A. Wang et al. 2019 learn a latent space for feasible object transformations, allowing robots to predict goal-directed sequences of actions for robotic manipulation of deformable objects. Other researchers focus on imitation learning (IL) to learn manipulation policies from demonstrations. Dalal et al. 2023 leverage a visuomotor Transformer-based imitation learning method that learns robot manipulation policies by imitating task and motion planning demonstrations. Jäkel et al. 2012 present a learning-based framework for acquiring planning models for dexterous robotic manipulation. The approach leverages Programming by Demonstration (PbD), where robots learn manipulation tasks directly from human demonstrations by automatically extracting relevant task models from multiple demonstrations instead of manually defining task constraints. McDonald and Hadfield-Menell 2022 introduce a guided imitation learning approach that trains a hierarchical policy to mimic TAMP-generated solutions, enabling efficient and scalable planning and execution in robotic manipulation tasks.

A key challenge in TAMP is the long planning horizon required for complex tasks. Y. Zhang et al. 2024 tackle this by automatically decomposing high-level goals into smaller, parallelizable subproblems through sequential pattern mining. By focusing on these manageable subproblems, the method enhances scalability and efficiency. Yuan et al. 2019 present an end-to-end approach for non-prehensile object rearrangement using deep reinforcement learning (RL) and simulation-to-reality transfer, where the learned policy maps raw pixel inputs directly to control actions. Beyond single-shot tasks, Kim et al. 2019 propose an actor-critic RL algorithm that improves TAMP efficiency by learning from past planning experience. Unlike standard RL approaches that require extensive exploration, this method leverages past search trees generated by TAMP solvers to learn a policy that selects continuous parameters for operators efficiently. This reduces the need for expensive motion planner calls, leading to significant speedup in task execution. Mendez-Mendez et al. 2023 introduce a lifelong learning paradigm using a mixture of generative models to learn continuous action parameters (e.g., grasps, poses, paths) over time, improving their planning efficiency through experience in real-world environments. This approach eliminates the traditional separation between training and testing, as robots are continuously evaluated and updated while solving problems in their environment.

Other works such as Bejjani et al. 2021 integrate IL and RL to enable efficient prehensile (grasp-based) and non-prehensile (pushing, sliding) manipulation from abstract color-labeled image representations of the scene, while Chitnis et al. 2016 use reinforcement learning (RL) to optimize continuous motion refinements and apply inverse reinforcement learning (IRL) to learn high-level plan selection heuristics from expert demonstrations. An emerging trend in TAMP is the use of Large Language Models (LLMs) Ding et al. 2023; K. Lin et al. 2023; Y. Chen et al. 2024; Y. Huang et al. 2024; S. Wang et al. 2024, bridging natural language instructions and manipulation planning. These approaches employ LLMs to parse user commands and generate symbolic or partially symbolic action sequences. Feedback from geometric planning either refines these instructions or flags infeasibility, prompting the model to propose alternative strategies. While LLM-based solutions demonstrate notable adaptability and ease of knowledge encoding, they require substantial computational resources, motivating research into more efficient models and algorithms for real-world deployments.

Collectively, these works highlight the growing trend of learning-based TAMP methods that aim to streamline planning by learning from experience, imitating human demonstrations or existing planners, or leveraging LLMs. However, end-to-end learning approaches often require extensive data collection and training, limiting their generalizability to novel tasks or environments.

Moreover, since they are prone to errors, they do not provide guarantees on completeness or safety, necessitating further research into hybrid methods that combine learning with traditional planning techniques to ensure robustness and efficiency in complex TAMP problems.

2.4 Discussion

Task and Motion Planning (TAMP) has motivated extensive research in the past decades, and has seen a significant rise in interest in recent years with the advent of learning-based methods. However, given the many challenges posed by TAMP problems, such as combinatorial complexity and high dimensionality, much work remains to be done to develop efficient, scalable, and robust planning methods. Prior works have made significant contributions to the field, although they often require some simplification of the problem, for instance through an initial discretization of the search space. Even with these simplifications, the computational complexity of task and motion planning can be prohibitive, especially in cluttered environments with many objects and robots, and intricate spatial relationships. Learning-based methods have shown promise in addressing some of these challenges, by providing fast geometric feedback to the task planner, learning new robot skills, or even bypassing the need for a separate task and motion planner. However, these methods often lack scalability, generalizability, and interpretability, limiting them to specific tasks or environments. In this thesis, we aim to take a step towards addressing these challenges by proposing a novel learning-based task and motion planning method that is capable of tackling various TAMP problems with a significant combinatorial complexity, while improving efficiency and scalability. By combining action and grasp feasibility prediction with interpretability mechanisms, our goal is to provide a generalizable and interpretable framework for TAMP, which can handle complex 3D scenes with multiple robots, movable objects and obstacles, as well as intricate spatial relationships and long planning horizons.

Chapter 3

Manipulation Planning Context

Contents

3.1 Introduction

In this thesis, we address offline manipulation planning problems, a subset of robotics challenges where one or more robotic manipulators must determine a sequence of actions and the corresponding motion trajectories to transition an environment with static obstacles and movable objects from an initial state to a desired goal state. These problems fall within the domain of task and motion planning (TAMP), which requires robots to integrate reasoning across two tightly coupled components: the symbolic aspects of the task, such as determining the logical sequence of object manipulations, and the geometric aspects, such as identifying feasible grasps or collision-free object placements and motions. This dual reasoning capability is critical in real-world scenarios such as object rearrangement, where robots are tasked with organizing, relocating, or assembling objects in three-dimensional cluttered environments.

In this chapter, we formally define the manipulation planning problem as a TAMP problem, highlighting its hybrid symbolic-geometric nature and the challenges it poses. We then present a

search-based multi-robot task and motion planning algorithm that serves as a reference planner for our contributions. We introduce a set of benchmarks that we use throughout the thesis to evaluate the performance of our proposed methods, and show that the reference planner fails on these benchmarks, especially in cluttered environments with multiple objects and robots. Particularly, we show how the lack of geometric reasoning capabilities in the reference planner leads to a high number of calls to the geometric planner, resulting in large computational costs. This further motivates our contributions, which aim to accelerate task and motion planning by providing learned heuristics to the task planner, reducing the number of geometric planning queries and improving the overall planning efficiency.

3.2 Manipulation Planning as a TAMP Problem

In this section, we provide a formal description of the manipulation planning problem. We decompose the problem into its core components: the symbolic planning level, which deals with the discrete decision-making process; the geometric planning level, which handles the continuous motion and feasibility considerations; and the interface that bridges the two, ensuring that symbolic plans translate into geometrically realizable motions.

3.2.1 Symbolic Planning Level

The manipulation environment E is composed of n_{robots} robots and $n_{objects}$ objects. These objects can be distinguished into $n_{movable}$ movable objects, $n_{support}$ stable support surfaces and n_{obs} fixed obstacles. Each object has initially known shape, dimensions and pose, and movable objects remain static unless moved by the robot. We define the configuration of a robot $\boldsymbol{q}$ as the set of all its joints' parameters, and the configuration of an object as its position and orientation in the environment. The state s of the planning scene is defined as the configuration of all movable objects as well as the configurations of the robots. The configuration of a robot $\mathcal{R}$ at state s is denoted $s(\mathcal{R})$, whilst the placement of an object $\mathcal{O}$ is denoted $s(\mathcal{O})$. We consider the workspace of a robot $\mathcal{R}$, denoted $W_{\mathcal{R}}$, as a sphere centered at the first joint of the robot, with a radius equal to its reach. Note that this simple model of $W_{\mathcal{R}}$ is generalizable to more precise workspace representations, provided that mapping the intersection between two workspaces is possible.

An action a is defined as the transition between two distinct states s and s'. In this thesis, we focus on Pick-Place actions, denoted $Move$, which involve a specific robot $\mathcal{R}$ picking an object $\mathcal{O}$ from its configuration in s, and placing it at a new placement $\mathcal{P}$, which becomes the placement of $\mathcal{O}$ in s'. The action a can be explicitly represented as:

$$a = Move(\mathcal{R}, \mathcal{O}, s(\mathcal{O}) \rightarrow \mathcal{P}) \tag{3.1}$$

The transition between states s and s' is thus defined as the application of an action a at state s, governed by the transition function T that updates the configuration of the moved object $\mathcal{O}$ and the robot $\mathcal{R}$ transporting it, while keeping the configurations of all other objects unchanged:

$$s' = T(s, a) \quad \text{such that} \quad s'(\mathcal{O}) = \mathcal{P} \quad \text{and} \quad s'(\mathcal{O}') = s(\mathcal{O}') \quad \forall \mathcal{O}' \neq \mathcal{O} \tag{3.2}$$

The goal state s_{goal} is a state in which all objects are at their desired configurations. The specification of these goal configurations can vary:

- **Fully specified:** Exact poses for objects (e.g., place object $\mathcal{O}$ at position (x, y, z) with orientation θ around the z axis).
- **Partially specified:** Placement regions (e.g., place object $\mathcal{O}$ on a support surface without specifying exact coordinates).

Figure 3.1: Visualization of the three types of goal specification. (a) Fully specified goal, (b) Partially specified goal, (c) Unspecified goal.

- **Unspecified:** Any valid placement is acceptable.

Figure 3.1 illustrates these three types of goal specification.

We define three types of object placements: (i) $\mathcal{P}_{goal}$ is the desired pose or one of the desired poses of the object, (ii) $\mathcal{P}_{temp}$ is a pose where the object can be placed temporarily, they are typically needed when an object must be regrasped to reach its goal pose, or an object must be moved to access or make space for other objects (iii) $\mathcal{P}_{pass}$ is a pose where the object can be passed from one robot to another. Following these definitions, we define three types of actions:

- ***Goal* actions a_{goal} :** which aim at bringing an object to its goal placement:

$$a_{goal} = Move(\mathcal{R}, \mathcal{O}, s(\mathcal{O}) \rightarrow \mathcal{P}_{goal}) \tag{3.3}$$

- ***Temporary* actions a_{temp} :** that move an object to a temporary placement:

$$a_{temp} = Move(\mathcal{R}, \mathcal{O}, s(\mathcal{O}) \rightarrow \mathcal{P}_{temp}) \tag{3.4}$$

- ***Pass* actions a_{pass} :** which aim at moving an object using a robot $\mathcal{R}_i$ to a placement in the intersection of $\mathcal{R}_i$'s workspace with another robot $\mathcal{R}_j$'s workspace, in order to enable an exchange between the two robots:

$$a_{pass} = Move(\mathcal{R}_i, \mathcal{R}_j, \mathcal{O}, s(\mathcal{O}) \rightarrow \mathcal{P}_{pass}) \tag{3.5}$$

The goal of the manipulation planning problem is to, given the initial and goal states of the movable objects, find a geometrically feasible sequence of actions $\mathcal{A} = \{a_0, a_1, ..., a_H\}$ that transitions the environment from the initial state s_0 to a goal state s_{goal}, such that:

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} ... \xrightarrow{a_H} s_{goal} \tag{3.6}$$

Additionally, for each action a_i, the associated motion plan m_i must be computed, forming the sequence of motion plans $\mathcal{M} = \{m_0, m_1, ..., m_H\}$, where H is the horizon of the plan.

The main challenge in offline manipulation planning is to find a feasible, collision-free sequence of actions. This requires geometric-level planning to check the feasibility of each action, and to plan the corresponding motion. The geometric planner must ensure that each motion plan m_i is

executable by the robot, considering its kinematic constraints, and that it avoids collisions with other objects and obstacles in the environment. This involves solving inverse kinematics to find valid robot configurations, performing collision checking, and using motion planning algorithms to generate paths. The integration of symbolic and geometric planning is crucial to ensure that the high-level task plan can be realized in the physical world.

3.2.2 Geometric Planning Level

The geometric planning level is responsible for checking the feasibility of each action in the task plan and planning the corresponding motion. In our setting, objects can be manipulated through pick and place actions. A pick action involves grasping a movable object from its current position, while a place action refers to placing the object at a desired location. Planning a pick action requires several steps, each one being a component of a geometric planner. First, a set of grasps is sampled. A grasp g is defined as an end-effector pose with respect to the object, represented as a 3D position and a quaternion. It can be viewed as a contact point between the robot's gripper and the object, along with the angle of approach. Grasp sampling/planning is a well-established research topic (Shimoga 1996; Caldera et al. 2018; Dong and J. Zhang 2023), with many works proposing traditional and learning-based methods to efficiently find grasps for different types of objects. Grasps can be sampled using different strategies such as:

- **Grid-based sampling:** Systematic exploration of grasp space at a fixed resolution.
- **Random sampling:** Probabilistic sampling of grasps across the object surface.
- **Dedicated Grasp Planners:** Methods using analytical, numerical or learning-based approaches.
- **Grasp Databases:** Pre-computed sets of grasps for common objects.

In this thesis, we use grid-based grasp sampling for box-shaped objects, and a grasp database for mesh-shaped objects. This process results in the set of grasps $\boldsymbol{G} = \{g_0, g_1, ..., g_{n_g}\}$, where n_g is the number of grasps sampled.

Once the grasps are sampled, an inverse kinematics (IK) solver is queried for each one to compute the corresponding robot configurations. The IK solver takes as input an end-effector pose g and returns a set of joint configurations $\mathcal{Q}_g = \{\boldsymbol{q}_0, \boldsymbol{q}_1, ..., \boldsymbol{q}_{n_{IK}}\}$ that satisfy the pose, where n_{IK} is the number of solutions found. An IK problem can have infinite solutions for redundant robots, which adds to the complexity of solving TAMP problems and maintaining completeness. Indeed, in heavily cluttered environments, most of the IK solutions will result in robot collisions, either with objects, with another robot, or with itself (i.e. self-collisions). Therefore, finding the right solution can be tedious and computationally expensive. In practice, the maximum number of IK solutions to consider n_{IK} is fixed by the user. The IK problem can also have no solution, meaning that the grasp is unreachable by the robot given its kinematic constraints. The IK solver can be analytical, numerical, or learning-based. In this work, we use numerical IK solvers provided by the MoveIt! framework (Coleman et al. 2014). After computing the robot configurations, a collision checker is used to verify their feasibility and ensure that no collisions occur between the robot and the environment. We refer to these potential collisions as Grasp Obstructions (GO). The collision checker can be a simple bounding box collision checker, a distance-based collision checker, or a learning-based collision checker. In this work, we use the distance-based FCL collision checker included in MoveIt!.

For each valid collision-free configuration found $\boldsymbol{q}_i$, a motion planner computes the motion m from the robot's current configuration to $\boldsymbol{q}_i$. The motion planner is responsible for finding a

Algorithm 1 GeometricPlanner

Input: $E, s, a, n_{IK}, \tau_{local}, \tau_{global}$ ▷ Environment, robot, object, initial and final placements, max. IK solutions, local and global timeouts

$\mathcal{R}, \mathcal{O}, s(\mathcal{O}), \mathcal{P} \leftarrow extractActionParams(a)$
$\boldsymbol{G} \leftarrow sampleGrasps(\mathcal{O})$
while τ_{global} is not reached and $\boldsymbol{G}$ is not empty **do**
  $g \leftarrow \boldsymbol{G}.pop()$
  $\mathcal{Q}_g^{Pick} \leftarrow solveIK(\mathcal{R}, g, s(\mathcal{O}), n_{IK})$
  for each $\boldsymbol{q}_{Pick}$ in $\mathcal{Q}_g^{Pick}$ s.t. $checkCollision(\mathcal{R}, \boldsymbol{q}_{Pick}, E)$ is False **do**
    $m_{Pick} \leftarrow planMotion(\mathcal{R}, s(\mathcal{R}), \boldsymbol{q}_{Pick}, E, \tau_{local})$
    if m_{Pick} is infeasible **then**
      continue
    end if
    $\mathcal{Q}_g^{Place} \leftarrow solveIK(\mathcal{R}, g, \mathcal{P}, n_{IK})$
    for each $\boldsymbol{q}_{Place}$ in $\mathcal{Q}_g^{Place}$ s.t. $checkCollision(\mathcal{R}, \boldsymbol{q}_{Place}, E)$ is False **do**
      $m_{Place} \leftarrow planMotion(\mathcal{R}, \boldsymbol{q}_{Pick}, \boldsymbol{q}_{Place}, E, \tau_{local})$
      if m_{Place} is infeasible **then**
        continue
      end if
      $m \leftarrow \{m_{Pick}, m_{Place}\}$
      return m
    end for
  end for
end while
return Action Infeasible

collision-free path that connects the two configurations, while satisfying the robot's kinematic constraints. It can be a sampling-based planner (e.g RRT, RRT*, PRM), optimization-based planner (e.g. CHOMP, TrajOpt), or a learning-based planner. In this thesis, we use the sampling-based Bidirectional Transition-based Rapid Random Tree (BiTRRT) planner (Devaurs et al. 2015) from the OMPL library (Sucan et al. 2012) to plan the motions. Given the probabilistic completeness of sampling-based motion planners, finding the robot trajectory corresponding to a feasible action is possible given enough time. However, detecting infeasibility is more challenging, as it requires the planner to explore the entire configuration space to ensure that no feasible path exists. This would require running the planner for an infinite amount of time. In practice, a user-defined local timeout τ_{local} is set to limit motion planning time. If the planner fails to find a solution within this timeout, the action is considered geometrically infeasible.

This entire process is repeated for each grasp in the set $\boldsymbol{G}$, until a feasible robot motion is found, all grasps are exhausted, or a global timeout τ_{global} is reached. The same process applies to *Place* actions in reverse, starting with finding IK solutions for the grasp chosen for the *Pick* action when the object is placed at the desired location, and planning a motion from the *Pick* to the *Place* configuration. Algorithm 1 details the geometric planning process for a single *Move* action. This multi-step process, which involves sampling grasps, solving inverse kinematics, collision checking, and planning motions, constitutes what we refer to in this thesis as geometric planning. It can be computationally expensive, particularly in cluttered environments with numerous objects and complex constraints, as each query to the geometric planner adds significant overhead. Indeed, the geometric planner iterates over grasps and IK solutions until a feasible motion is found, all potential grasps have been evaluated, or a timeout is reached. This process is repeated for each action considered, which can lead to a high number of calls to the geometric planner, most of which are for infeasible actions. This motivates the need for learning-based methods to accelerate task and motion planning by providing heuristics to the task planner, which can prioritize promising actions and reduce the number of geometric planning queries.

3.2.3 Symbolic-Geometric Planning Interface

The symbolic-geometric planning interface is a critical component in Task and Motion Planning (TAMP), enabling the coordination between high-level task planning and low-level motion feasibility checks. This interface ensures that symbolic actions are grounded to specific geometric parameters, in order to enable geometric planning. It also facilitates communication between the symbolic and geometric planners, allowing the task planner to query the geometric planner to check the feasibility of task plans and find the corresponding motions, and backtrack when infeasibility is encountered. In this section, we detail how actions are grounded, how communication occurs between planners, and the role of backtracking in resolving infeasibility.

Actions in the symbolic plan must be grounded to specific geometric parameters to enable geometric planning. A placement sampler is used to randomly sample collision-free poses for objects in the environment. This is particularly important for:

- **Placement Sampling for Semi-Specified Goals:** If the goal state specifies a placement region (e.g., object must be on a support surface without defining an exact pose), the placement sampler uniformly samples n_{goal} placements within the specified region. This allows the planner to explore different object placements that satisfy the goal constraints.
- **Temporary Placements:** For intermediate actions that require temporarily relocating objects to enable other actions, n_{temp} placements are sampled dynamically during planning. These placements serve as stepping stones in the overall manipulation sequence.
- **Pass Placements:** In multi-robot settings, n_{pass} pass placements are sampled at locations within the intersection of two robots' workspaces to facilitate object exchanges. These placements allow a robot to bring an object to the the workspace of another robot, enabling collaborative manipulation tasks.

In each one of these cases, object placements are sampled and checked for collisions and reachability until a user-defined number n_{goal}, n_{temp} or n_{pass} of collision-free poses is found. If none of the sampled placements lead to a feasible solution, the placement sampler is re-invoked to generate new placements. This grounding process converts abstract symbolic actions into concrete parameters that the geometric planner can evaluate and plan motions for.

The symbolic-geometric interface operates in a loop, where the symbolic planner generates a task plan, and the geometric planner verifies the feasibility of each action. Once a symbolic task plan is identified, the geometric planner is queried to check the feasibility of each action in the sequence. For each action, the geometric planner attempts to compute a motion plan, ensuring that it adheres to the kinematic constraints of the robot and avoids collisions with the environment. If all actions in the task plan are feasible, the combined task and motion plan is returned as a solution. This tight coupling ensures that high-level plans are geometrically feasible, aligning abstract symbolic action sequences with low-level geometric robot motions.

When an action is found to be geometrically infeasible, the planner must backtrack to an earlier state to explore alternative plans. This process, known as backtracking, allows the task planner to leverage feedback from the geometric planner and resume planning from a different state:

- **Symbolic Backtracking:** Involves reevaluating the symbolic plan by selecting a different action sequence to explore, e.g., choosing a different object to manipulate, a different robot to perform the action, or a temporary action when a goal action is infeasible.
- **Geometric Backtracking:** Involves revisiting an earlier state and modifying the geometric parameters of an action, such as considering alternative goal, temporary, or pass placements.

Moreover, backtracking involves updating the search tree by pruning branches emanating from infeasible actions. This prevents the planner from revisiting the same infeasible sequences multiple times, which can be computationally expensive. By systematically resolving infeasibility, backtracking helps reduce the search space and ensures that the planner converges faster to a feasible task and motion plan. When an action is found to be infeasible, the planner prunes states yielded by the infeasible action or actions similar to it. It then backtracks to the next-best state in the search tree and explores the action space from there. This iterative process continues until a feasible sequence of actions and motions is identified. Based on these concepts, and in order to further motivate our contributions, we propose a search-based multi-robot TAMP algorithm which uses simple heuristics to find feasible solutions, and show how the lack of geometric feasibility information can lead to a high number of calls to the geometric planner, resulting in a high computational cost.

3.3 Search-based Multi-Robot Task and Motion Planning

This section presents the reference task and motion planner used in this thesis. We first detail the planning algorithm, which combines symbolic task planning with geometric motion planning to find feasible solutions in multi-robot manipulation problems, based on the concepts defined in the previous section. We then introduce a set of benchmarks that will be used throughout this thesis to evaluate the performance of our proposed methods. Finally, we show that the reference planner fails on these benchmarks and discuss the limitations that motivate our contributions.

Algorithm 2 Task and motion planner

Input: $E, s_0, s_{goal}, n_{IK}, n_{goal}, n_{temp}, n_{pass}, \tau_{local}, \tau_{global}$ ▷ Environment, initial state, goal state, max. IK solutions, nb. goal, temporary and pass placements to sample, local and global timeouts

$\mathbf{\Gamma} \leftarrow buildRobotGraph(E)$
$Q \leftarrow \{s_0\}$ ▷ Set of nodes to expand
while Solution not found **do**
  $s \leftarrow argmin_{cost} Q$ ▷ Extract node with minimum cost
  if $s \in s_{goal}$ **then** ▷ s is a goal state
    $\mathcal{A} \leftarrow retrieveSolution(s)$ ▷ Retrieve the list of actions leading to s
    $\mathcal{M} \leftarrow \emptyset$ ▷ Sequence of motion plans
    for each action a in $\mathcal{A}$ **do**
      $m \leftarrow GeometricPlanner(E, s, a, n_{IK}, \tau_{local}, \tau_{global})$
      if m is feasible **then**
        $\mathcal{M} \leftarrow \mathcal{M} \cup m$
      else
        $Q \leftarrow updateSearchTree(Q, a)$ ▷ Backtracking to prune infeasible nodes
        break
      end if
    end for
    if $\mathcal{M}$ is feasible **then**
      return $\mathcal{A}$, $\mathcal{M}$ ▷ Solution found
    end if
  else
    $Q \leftarrow Q \cup findChildren(s, E, s_{goal}, \mathbf{\Gamma}, n_{goal}, n_{temp}, n_{pass})$ ▷ Expand node
  end if
end while

Algorithm 3 updateSearchTree

Input: $Q, a_{infeasible}$ ▷ List of open nodes, Infeasible action

for each s in Q **do**
  if s is a descendant of $a_{infeasible}$ **then**
    $Q \leftarrow Q \setminus s$
  end if
end for
return Q

3.3.1 Planning Algorithm

We first describe a search-based task and motion planning algorithm that combines symbolic task planning with geometric planning to find feasible solutions in multi-robot manipulation problems. The main algorithm, shown in Algorithm 2, is based on a best-first tree search, where nodes represent states and edges are actions allowing transitions between states, following the definitions of Section 3.2.1. First, we build a robot graph $\mathbf{\Gamma}$ (cf. line 1), which represents the connectivity between robots workspaces in the environment. Nodes in the graph are robots, and edges represent the existence of an intersection between two robots' workspaces. The graph is used to determine which robots can exchange objects between each other, and guide the search towards feasible solutions.

Node selection. A list Q of open nodes is maintained and sorted according to a defined cost. Initially, the list contains the initial state s_0 of the environment (cf. line 2). At each iteration, the node s with the minimum cost is extracted from Q (cf. line 4), and compared to the goal state s_{goal}. If s is a goal state, we retrieve the complete task plan leading to s by backtracking through the parent nodes. We then query the geometric planner to check the feasibility of every action in the task plan and plan their corresponding motions (cf. lines 5-9). If the action sequence is geometrically feasible, then a solution was found (cf. lines 17-18). Otherwise, we perform backtracking by pruning all children of the infeasible action in the task plan from the tree (cf. line 13), using the *updateSearchTree* function detailed in Algorithm 3.

Algorithm 4 findChildren

Input: $s, E, s_{goal}, \mathbf{\Gamma}, n_{goal}, n_{temp}, n_{pass}$ ▷ state, Environment, Goal state, Robot graph, Nb. goal, temporary and pass placements to sample

1: $children \leftarrow \emptyset$
2: $Actions \leftarrow findActions(s, E, s_{goal}, \mathbf{\Gamma}), n_{goal}, n_{temp}, n_{pass}$
3: **for** each a in $Actions$ **do**
4: $\quad s' \leftarrow applyAction(s, a)$
5: $\quad s'.cost \leftarrow computeCost(s', s_{goal}, \mathbf{\Gamma})$
6: $\quad children \leftarrow children \cup s'$
7: **end for**
8: **return** $children$

Algorithm 5 findActions

Input: $s, E, s_{goal}, \mathbf{\Gamma}, n_{goal}, n_{temp}, n_{pass}$

1: $Actions \leftarrow \emptyset$
2: **for** each $\mathcal{O}$ in movable objects **do**
3: $\quad$ **for** each $\mathcal{R}_i$ in $findReachingRobots(s(\mathcal{O}), E)$ **do**
4: $\quad\quad P \leftarrow \emptyset$
5: $\quad\quad$ **if** $s(\mathcal{O}) \notin s_{goal}(\mathcal{O})$ **then**
6: $\quad\quad\quad$ **if** $s_{goal}(\mathcal{O})$ is reachable by $\mathcal{R}_i$ **then**
7: $\quad\quad\quad\quad P \leftarrow P \cup sampleGoal(E, \mathcal{R}_i, s_{goal}(\mathcal{O}), n_{goal})$
8: $\quad\quad\quad$ **else**
9: $\quad\quad\quad\quad$ **for** $\mathcal{R}_g$ in $findReachingRobots(s_{goal}(\mathcal{O}), E)$ **do**
10: $\quad\quad\quad\quad\quad next\_robots \leftarrow BFS(\mathbf{\Gamma}, \mathcal{R}_i, \mathcal{R}_g)$
11: $\quad\quad\quad\quad\quad$ **for** $\mathcal{R}_j \in next\_robots$ **do**
12: $\quad\quad\quad\quad\quad\quad P \leftarrow P \cup samplePass(E, \mathcal{R}_i, \mathcal{R}_j, n_{pass})$
13: $\quad\quad\quad\quad\quad$ **end for**
14: $\quad\quad\quad\quad$ **end for**
15: $\quad\quad\quad$ **end if**
16: $\quad\quad$ **end if**
17: $\quad\quad P \leftarrow P \cup sampleTemp(E, \mathcal{R}_i, n_{temp})$
18: $\quad\quad$ **for** each $\mathcal{P} \in P$ **do**
19: $\quad\quad\quad a \leftarrow Move(\mathcal{R}_i, \mathcal{O}, s(\mathcal{O}) \rightarrow \mathcal{P})$
20: $\quad\quad\quad Actions \leftarrow Actions \cup a$
21: $\quad\quad$ **end for**
22: $\quad$ **end for**
23: **end for**
24: **return** A

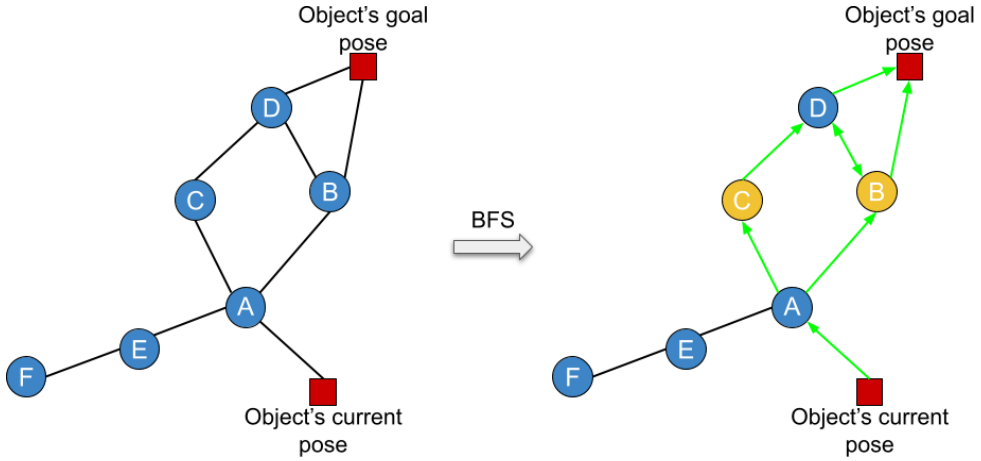


Figure 3.2: Visualization of the Breadth-First Graph Search used to generate *Pass* actions, and find the shortest sequence of robot exchanges to reach an object's goal state. The graph $\mathbf{\Gamma}$ is built using the connectivity between robots' workspaces. Blue nodes represent robots, black edges represent intersections between workspaces. The search starts from each robot capable of reaching the current object pose, and explores all possible paths to the goal state (green edges). The first robot in each path is selected as a candidate for the *Pass* action (yellow nodes).

Node Expansion. If s in not a goal state, we expand the node using the function $findChildren$, shown in Algorithm 4. It first samples applicable actions at state s, by calling the function $findActions$ which is detailed in Algorithm 5. For each movable object $\mathcal{O}$ in the environment, we find the set of robots capable of reaching $\mathcal{O}$ at its pose in s, using the function $findReachingRobots(s(\mathcal{O}), E)$ (cf. line 3). For each robot $\mathcal{R}_i$ in the obtained set, we generate *Goal*, *Pass* and *Temporary* actions. In the case where $\mathcal{O}$ is not already at its goal pose, if s_{goal} is reachable by $\mathcal{R}_i$, we sample a set of n_{goal} goal placements in the workspace of $\mathcal{R}_i$ using the function *sampleGoal* (cf. lines 6-7). Otherwise, if the goal pose of $\mathcal{O}$ is not reachable by $\mathcal{R}_i$, we try to generate *Pass* placements using *samplePass* (cf. lines 9-14). We first find the set of robots that are able to reach s_{goal}. The planner needs to figure out how to bring $\mathcal{O}$ to each one of these robots' workspaces. In order to do that, we leverage the robot graph $\mathbf{\Gamma}$. For each robot $\mathcal{R}_g$ reaching s_{goal}, we perform a Breadth-First Graph Search to find all possible sequences of robot exchanges from $\mathcal{R}_i$ to $\mathcal{R}_g$, and we extract the first robot from each path. These are all the robots $\mathcal{R}_i$ can pass object $\mathcal{O}$ to in order to bring it closer to its goal pose. Figure 3.2 shows a visualization of this process. For each robot $\mathcal{R}_j$ in the obtained set of robots, we sample n_{pass} *Pass* placements at the intersection between $W_{\mathcal{R}_i}$ and $W_{\mathcal{R}_j}$ (cf. line 12). Finally, we sample a set of n_{temp} *Temporary* placements in the workspace of $\mathcal{R}_i$ (cf. line 17). Then, we generate actions corresponding to each one of the *Goal*, *Pass* and *Temporary* placements sampled. Once these actions have been generated, we find the resulting children states obtained from applying the actions generated to the current state s (cf. Algorithm 4, line 4). In order to maintain probabilistic completeness, we ensure that the planner can revisit expanded states in case none of the sampled states lead to a feasible solution, retaining the possibility to sample new placements and generate new actions/states.

Cost Computation. We then compute the cost of each child state which is used to sort the set of open nodes, and defined as:

$$C_{Total} = C_{SoFar} + C_{ToGoal} \tag{3.7}$$

where C_{SoFar} represents the number of actions in the branch leading to the node and C_{ToGoal}

Algorithm 6 computeCost

Input: $s', s, s_{goal}, E, \mathbf{\Gamma}$
1: $C_{SoFar} \leftarrow C_{SoFar}(s) + C_{step}$
2: $C_{ToGoal} \leftarrow computeCostToGoal(s', s_{goal}, E, \mathbf{\Gamma})$
3: $C_{Total} \leftarrow C_{SoFar} + C_{ToGoal}$
4: **return** C_{Total}

Algorithm 7 computeCostToGoal

Input: $s, s_{goal}, E, \mathbf{\Gamma}$
1: $C_{ToGoal} \leftarrow 0$
2: **for** each movable object $\mathcal{O}$ in E **do**
3: **if** $s(\mathcal{O}) \notin s_{goal}(\mathcal{O})$ **then**
4: **if** $s(\mathcal{O})$ and $s_{goal}(\mathcal{O})$ are in the same robot's workspace **then**
5: $C_{ToGoal} \leftarrow C_{ToGoal} + 1$
6: **else**
7: $C_{ToGoal} \leftarrow C_{ToGoal} + BFS(\mathbf{\Gamma}, s(\mathcal{O}), s_{goal}(\mathcal{O}))$
8: **end if**
9: **end if**
10: **end for**
11: **return** C_{ToGoal}

is the minimum number of actions to reach the goal state. The cost-so-far is computed by incrementing the cost-so-far of the parent node by a user-defined step cost C_{step}. Typically, C_{step} is set to 1, meaning that each action has a cost of 1. Reducing the step cost increases the greediness of the search, leading the planner to prioritize depth over breadth. This allows the planner to expand recently expanded states rather than expanding older ones, which sacrifices optimality to reach a potential task plan faster. Throughout this thesis, we set C_{step} to 0.01, giving our planner a more greedy behavior. The cost-to-goal C_{ToGoal}, on the other hand, is computed using the function *computeCostToGoal* (cf. Algorithm 7), which takes advantage of the previously constructed graph $\mathbf{\Gamma}$ to compute for each object the shortest action sequence length to the goal using Breadth-First Graph Search (BFS). If an object's current configuration and goal configuration are in the same robot's workspace, there is no need for any robot exchange, the sequence length is hence equal to 1. Otherwise, we find the smallest number of *Pass* actions needed to bring the object to its goal configuration. In Figure 3.2, BFS finds multiple action sequences from the object's current pose to its goal pose, the shortest being $\{\mathcal{R}_A \rightarrow \mathcal{R}_B\}$. In this case, the minimum number of *Pass* actions needed is 1, and the total number of actions to reach the goal state is 2, e.g $\{Move(\mathcal{R}_A, \mathcal{O}, s(\mathcal{O}) \rightarrow \mathcal{P}_{pass}), Move(\mathcal{R}_B, \mathcal{O}, \mathcal{P}_{pass} \rightarrow s_{goal}(\mathcal{O}))\}$. Finally, we sum the obtained sequence lengths per object to compute the minimum number of actions to reach the goal state. The complete cost computation process is summarized in Algorithm 6.

Note that the C_{ToGoal} heuristic is too simple, and tends to significantly underestimate the number of actions necessary to reach the goal state, especially in cluttered and heavily constrained environments. Indeed, in most cases, moving an object to its goal placement requires multiple intermediate steps, such as moving other objects, or clearing a goal/exchange region. These geometric constraints are not accessible to the symbolic planner, hence there is no way to better estimate the number of actions needed to reach the goal state. Therefore, there is a need for more sophisticated heuristics to guide the search towards feasible solutions in a more efficient manner, which is the main motivation behind our contributions. In the following chapters, we propose learning-based methods that improve the cost estimation of a node. By adding a feasibility cost component that associates a large cost to infeasible nodes emanating from infeasible actions, we can guide the search towards feasible solutions. Moreover, by leveraging reasons of infeasibility, we can compute a more accurate cost-to-goal estimate, which will help the planner prioritize promising actions and accelerate the search process.

Planning rules. During the search, we set some planning rules that define the constraints and permissible actions governing node expansion. These rules encapsulate logical constraints and assumptions that allows the size of the search tree to remain tractable. They help reduce the search space, but they also reduce the number of feasible solutions that can be found. The rules are as follows:

- *Each object can be moved to a Temporary placement at most once in a task plan.*
- *If an object is moved to a Goal placement, it can not be moved again.*[1]
- *If an object is moved to a Pass placement for a specific receiving robot, the next action that manipulates the object must be performed by the receiving robot.*

3.3.2 Complexity Analysis

We analyze the computational complexity of the reference task and motion planning algorithm detailed in Section 2, focusing on both symbolic and geometric components.

Notations and Parameters: We remind the reader of the key parameters used in the algorithm:

- $n_{movable}$: Number of movable objects in the environment.
- n_{robots}: Number of robots in the environment.
- n_{goal}: Number of goal placements sampled for each object.
- n_{temp}: Number of temporary placements sampled for each object.
- n_{pass}: Number of pass placements sampled for each object.
- n_g: Number of grasps sampled for each object.
- n_{IK}: Maximum number of IK solutions computed per grasp.

Symbolic Search Complexity: The symbolic search explores a tree of states, where each state corresponds to a configuration of the environment. The size of the search space is determined by the branching factor B, and the maximum horizon H_{max}. In the worst case, the branching factor representing the number of children states added at each node expansions is given by:

$$B = n_{movable} \cdot n_{robots} \cdot (n_{goal} + n_{temp} + n_{robots} \cdot n_{pass}) \tag{3.8}$$

This is a loose upper bound because not all robots will necessarily be able to reach each object; also, once an object is definitively at its goal, it no longer contributes to branching. Given the planning rules defined, the maximum length of a solution involves each movable object being moved to a temporary placement, a pass placement for each robot, and a goal placement. Hence the maximum horizon is:

$$H_{max} = n_{movable} \cdot (n_{robots} + 2) \approx n_{movable} \cdot n_{robots} \tag{3.9}$$

The number of states explored in the symbolic search grows exponentially with the branching factor and the horizon:

$$\text{Symbolic States Explored} \propto B^{H_{max}} \tag{3.10}$$

Therefore, the complexity of the symbolic search is exponential in the number of movable objects and robots:

$$O(n_{movable} \cdot n_{robots} \cdot (n_{goal} + n_{temp} + n_{robots} \cdot n_{pass})^{n_{movable} \cdot n_{robots}})$$

[1] If an object is at its goal placement at the initial state, it can be moved to a temporary placement and then back to its goal placement.

Such exponential growth is typical in TAMP, and motivates the need for efficient search strategies and heuristics to guide the exploration.

Geometric Planning Complexity: The geometric planner is called whenever the planner finds a task plan. From Algorithm 1, the complexity of a single geometric planner call is:

$$O(n_g \cdot n_{IK}^2 \cdot C_{MP})$$

where C_{MP} is the complexity of the motion planner. In the extreme worst-case scenario where the last action in every task plan found is infeasible, the geometric planner is called for every action in the search tree. As a result, the total worst-case complexity of the proposed task and motion planning algorithm is:

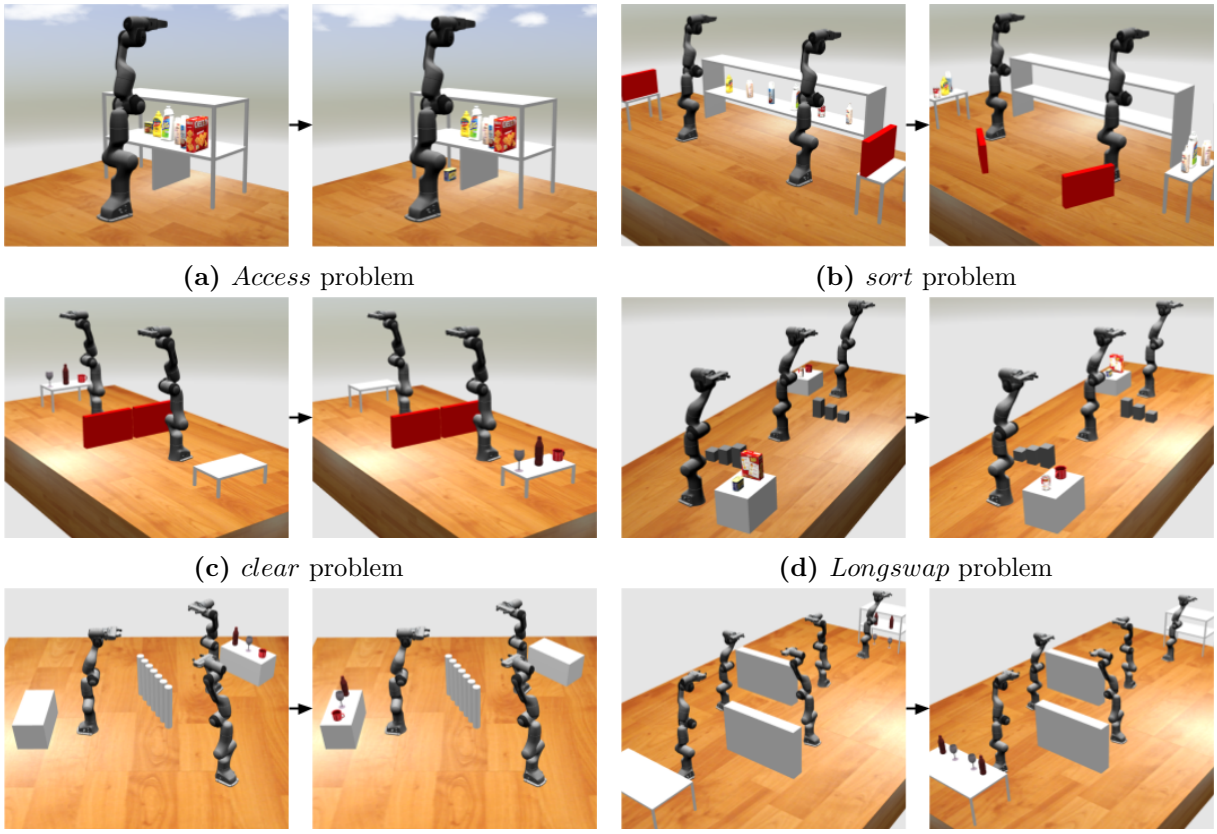
$$O(n_{movable} \cdot n_{robots} \cdot (n_{goal} + n_{temp} + n_{robots} \cdot n_{pass})^{n_{movable} \cdot n_{robots}} \cdot n_g \cdot n_{IK}^2 \cdot C_{MP})$$

Note that this complexity analysis is a worst-case scenario, which assumes that the planner explores the entire search space and calls the geometric planner for every action. In practice, the planning heuristics as well as the backtracking process help reduce the search space and the number of geometric planner calls. However, the resulting exponential complexity in the number of movable objects and robots, combined with the high number of geometric planner calls, highlights the difficulty of solving multi-robot TAMP problems. Thus, there is a need for efficient search strategies and heuristics to guide the exploration towards feasible solutions.

3.4 Benchmark TAMP problems

In order to evaluate the performance of the reference planner as well as our contributions throughout this thesis, we define a set of benchmark TAMP problems with various complexities. The goal is to test the planner's ability to handle different types of challenges commonly encountered in manipulation tasks, such as cluttered environments, multi-robot collaboration, and complex object interactions. These benchmarks also seek to measure the planner's ability to handle TAMP problem with a high combinatorial complexity, where the number of potential task plans is very large while only a few of them are feasible at the geometric level. Finally, we aim at evaluating the planner's generalizability to scenarios with different numbers of objects, various objects shapes, as well as multi-robot settings. We define 6 different problem domains, shown in Figure 6.3, that will be used throughout this thesis to evaluate our approach. Each problem contains a different number of robots, support surfaces, fixed obstacles, and movable objects, and requires different types of actions to be performed. The robotic platform used is the Franka Emika Panda robot, which is a 7-DOF robotic arm with a parallel-jaw gripper. Movable objects can be either box-shaped or mesh-shaped objects extracted from the YCB (Calli et al. 2015) and KIT (Kasper et al. 2012) objects databases. Each mesh-shaped object has a grasp database of up to 200 grasps to be used by the geometric planner. That is much more than previous works (e.g. Garrett et al. 2018a) which generally use a single to a few grasps only. The benchmark TAMP problems are defined as follows:

Access Problem. The **Access** problem, shown in Figure 3.3a, involves moving a single object (meat can) to a goal placement. A number of mesh objects are, however, blocking access to it, which requires removing all blocking objects to access the wanted one, then returning them to their initial pose. This problem is designed to test the planner's ability to handle complex object interactions and cluttered environments. The challenge comes from the fact that each object blocks access to the one next to it, requiring the robot to move obstructing objects one by one to temporary placements in a specific order, move the meat can to its goal placement, and



(a) *Access* problem **(b)** *sort* problem

(c) *clear* problem **(d)** *Longswap* problem

(e) *middleman*3 problem **(f)** *middleman*5 problem

Figure 3.3: Visualization of the initial and goal states of the six benchmark TAMP problems used in this thesis.

finally put back the previously moved objects at their initial placements in the reverse order. This problem is similar to the *non-monotonic* problem commonly found in TAMP literature (Garrett et al. 2018a; Lagriffoul et al. 2018; Khodeir et al. 2023a). However, where previous works generally use one to two obstructing objects, we use five obstructors, making the combinatorial complexity of the problem much higher. The space of potential symbolic plans is very large, while only one of them is feasible at the geometric level. Specifically, the number of potential task plans for the **Access** problem with $n_{movable}$ objects is given by:

$$n_{plans} = \sum_{k=1}^{n_{movable}} \binom{n_{movable}}{k} \cdot (k!)^2 \tag{3.11}$$

This number considers incrementally adding objects to move as well as all the permutations of these objects. It considers purely symbolic plans without accounting for geometric parameters such as different temporary placement locations. For $n_{movable} = 6$, the number of potential task plans is 614'226 (cf. Table 3.1). Only one of these plans is feasible at the geometric level.

Table 3.1: Exponential evolution of the number of potential task plans for the **Access** problem with increasing number of movable objects.

$n_{movable}$	2	3	4	5	6	7	8
n_{plans}	6	51	748	17'685	614'226	29'354'311	1'844'279'256

Sort Problem. The **Sort** domain, shown in Figure 3.3b aims at demonstrating the ability of the planning algorithm to handle semi-specified and unspecified goals. This is a dual-robot scenario where the objective is to sort object according to their type (food, cleaning products) on two pre-occupied tables. Here, the challenge comes from three main aspects:

- **Placement Sampling:** The planner must find a set of goal placements allowing all objects of the same type to be on a narrow support surface. The initial proximity of objects is also a challenge, since the objects have to be moved in a specific order.
- **Multi-Robot collaboration:** Each robot has access to only one table and a subset of the objects, requiring object exchanges and robots' collaboration to solve the problem. This introduces an additional layer of complexity as the robots pass objects between each other.
- **Obstructed Goal Placements:** The problem's complexity is further heightened by the addition of blocking objects (in red), which have unspecified goals and occupy the target tables. These blockers are initially positioned to allow the placement sampler to easily find collision-free goal placements, however, they obstruct place actions by causing collisions with the robot when trying to place objects. The planner must first move the blocking objects to clear the goal tables before placing the objects.

Clear Problem. In the **Clear** problem, illustrated in Figure 3.3c, two robots must collaborate in order to move three objects from one table to another. However, two large objects span over the intersection of the two robots' workspaces, rendering *Pass* actions infeasible and creating a bottleneck in the shared region. In order to reach the goal state, robots must first clear this intersection by removing the blocking objects. This also involves ensuring that temporary placements for blocking objects do not interfere with subsequent actions. Once the region is cleared, the robots can proceed to transfer the objects to their goal placements through Pass actions, before moving the initially blocking objects back to their initial placement. This scenario evaluates the ability of the planner to handle situation where Pass actions are initially infeasible, requiring intermediate actions to allow the resolution of the problem.

Longswap Problem. The **Longswap** problem (cf. Figure 3.3d) involves three sequential robots collaborating to swap the placements of four objects. Intersection regions between workspaces are partially obstructed by fixed obstacles, which necessitate careful planning to identify collision-free "Pass" placements. The interdependencies between objects further complicate the scenario, as each object's movement is contingent upon another being relocated first. In this problem, the main challenge comes from Pass placement sampling. Since intersection regions are partially obstructed, the regions where Pass placements are feasible are narrow. Furthermore, Objects must be exchanged in both directions, which limits the number of objects that can be in the same intersection region at the same time. In addition, the increased number of robots adds to the combinatorial complexity of the problem.

Middleman Problem. The **Middleman** problem involves a collaborative task in which multiple robots, configured in overlapping workspaces, must work together to transfer objects across shared regions to their goal placement. Multiple sequences of robots' exchanges are possible. The challenge comes, however, from the fact that some of these intersection regions are fully blocked by fixed objects. making only one sequence of robots' exchanges geometrically feasible. We developed two versions of this problem to test different levels of complexity:

- **Middleman-3:** This version (cf. Figure 3.3e) involves three robots arranged in a triangular configuration with overlapping workspaces and 3 objects to be moved from one side to the other. One intersection region is blocked by fixed obstacles, requiring the robots to perform two sequential "Pass" actions via a middleman robot. The objective is to transfer

three objects from one counter to another while ensuring that all intermediate placements and actions are collision-free.

- **Middleman-5:** This extended version (cf. Figure 3.3f) increases the complexity significantly by introducing five robots and four objects to be transferred. The higher number of robots and increased workspace overlaps exponentially increase the combinatorial complexity of the problem. This version is particularly representative of challenges in multi-robot manipulation problems where the number of robots is high.

3.5 Results

3.5.1 Experimental Setup

Throughout this thesis, we use the reference planner detailed in Section 3.3.1 and improve upon it. The planner is implemented in C++, using a maximum number of IK solutions $n_{IK} = 8$, a maximum number of *Goal*, *Temporary* and *Pass* placements to sample $n_{goal} = n_{temp} = n_{pass} = 3$, a local timeout $\tau_{local} = 3s$, and a global timeout $\tau_{global} = 12s$ for geometric planning. The cost step C_{step} is set to 0.01. The planner uses Moveit! Task Constructor (Görner et al. 2019) for geometric planning, which is a wrapper over the Moveit! framework (Coleman et al. 2014), using the KDL inverse kinematic solver, the FCL library for collision checking (J. Pan et al. 2012), and BiTRRT for motion planning (Devaurs et al. 2015). Experiments are conducted on an Intel i9-11950H @ 2.60GHz workstation, with 32GB of RAM, and an NVIDIA RTX A5000 GPU. The planner is tested on the six benchmark TAMP problems detailed in Section 3.4 over 10 trials, each trial using a different random seed. The planner is run with a timeout of 300s.

3.5.2 Evaluation Metrics

We evaluate the performance of the planner using a set of metrics that measure the planner's ability to find feasible solutions and the efficiency of the planning algorithm. First, we use the *Success Rate* metric, which measures the percentage of successful trials where the planner was able to find a geometrically feasible solution within the timeout. We also measure the *Total Planning Time*, which is the average time taken by the planner to find a solution over all trials. The total planning time is further broken down into the *Task Planning Time*, which is the average time taken by the symbolic planning component, *Collision Checking Time* which represent to time spent on checking whether sampled object placements are collision-free, and the *Geometric Planning Time* which is the average time taken by the geometric planning component. The latter is divided into the time taken to plan for feasible actions and the time taken to plan for infeasible ones. Furthermore, we also measure the number of *Infeasible Task Plans* generated by the planner before finding a feasible one, the *Number of Iterations*, the number of *Nodes Expanded* during the search, the number of *Collisions Checks*, and the number of *Geometric Planner Calls* on both feasible actions and infeasible ones. We also measure the number of *Nodes pruned using Backtracking* as well as the average *Solution Length*.

3.5.3 Planning Performance

Table 3.2 shows the planning performance of the reference planner on the six benchmark TAMP problems, namely the success rate. total planning time, the number of infeasible task plans generated before a feasible one was found, and the number of expanded nodes during the search. For cases where the planner failed to find a solution within the timeout, we still report the information recorded before the timeout. Table 3.3 shows the planning statistics for each benchmark TAMP problem, including the number of queries to geometric planner, the number of

Table 3.2: Planning time of our search-based multi-robot Task and Motion Planning algorithm on the benchmark TAMP problems averaged over 10 trials, using a timeout of 300s.

Problem	Success Rate	Total Planning Time (s)	Infeasible Task Plans	Expanded Nodes
Access	0%	> 300	> 204.4 (+/-18.4)	> 6848.0 (+/-832.3)
Clear	30%	190.0 (+/-51.9)	124.7 (+/-20.2)	5873.3 (+/-869.2)
Sort	0%	> 300	> 49.0 (+/-5.8)	> 7452.6 (+/-704.9)
Longswap	0%	> 300	> 200.7 (+/-34.2)	> 4674.4 (+/-2590.8)
Middleman-3	0%	> 300	> 155.8 (+/-18.4)	> 2394.2 (+/-372.0)
Middleman-5	0%	> 300	> 116.1 (+/-40.1)	> 8649.3 (+/-2339.3)

collision checks, and the number of pruned nodes using backtracking. Finally, Figure 3.4 shows the decomposition of the total planning time for each problem.

Results show that the proposed reference planner completely fails to find a solution for five out of the six benchmark problems within the specified timeout. A closer look at the the number of infeasible task plans generated shows that planner struggles to find a geometrically feasible solution due to the high combinatorial complexity of the problems. The latter is highlighted by the number of expanded nodes, which exceeds 7000 nodes on the **Sort** problem, and 8000 nodes on the **Middleman-5** problem. On the **Access** problem, the planner generates a high number of infeasible task plans, with an average of 204 geometrically infeasible task plans generated. Since the planner is unable to assess the feasibility of considered actions, it greedily attempts to access the desired object while incrementally moving more obstructing objects to temporary placements. This results in a high number of infeasible task plans, thus a high number of costly queries to the geometric planner as shown in Table 3.3. Furthermore, note that the planner reaches the timeout after exploring just 1% of the complete plan space (cf. eq 3.11), which highlights the difficulty of the problem.

The same observations can be made for the **Sort** problem, whose combinatorial complexity is highlighted by the high number of expanded nodes. In addition to a high number of geometric planner calls, the planner also performs a high number of collision checks exceeding 25′000 queries to the collision checker. This is caused by the small area available for placing objects on the support surfaces, making the task of finding a set of collision-free goal placements which allow all objects of the same type to be on the same support surface very challenging. The challenges of the **Longswap** problem are also reflected by the obtained results. The planner fails to find a solution within the timeout, even after generating 200 infeasible task plans and querying the geometric planner more than 300 times. This shows that partially obstructed intersection regions between robots' workspaces makes the problem heavily constrained, with only a small set of *Pass* placements allowing a collision-free exchange of objects. Comparing results on the **Middleman-3**

Table 3.3: Planning statistics on the benchmark TAMP problems averaged over 10 trials.

Problem	Geometric Planner Calls	Collision Checks	Pruned Branches
Access	> 315.1 (+/-28.3)	> 12355.7 (+/-1718.9)	> 6068.4 (+/-556.0)
Clear	172.3 (+/-27.4)	13297.7 (+/-1926.6)	5279.7 (+/-782.7)
Sort	> 52.1 (+/-6.1)	> 25393.1 (+/-2909.4)	> 7439.7 (+/-712.4)
Longswap	> 319.3 (+/-39.0)	> 9775.0 (+/-5704.2)	> 4402.4 (+/-2622.2)
Middleman-3	> 225.2 (+/-26.1)	> 4984.3 (+/-814.2)	> 2259.7 (+/-362.7)
Middleman-5	> 154.9 (+/-54.6)	> 18456.2 (+/-4591.4)	> 8362.9 (+/-2326.1)

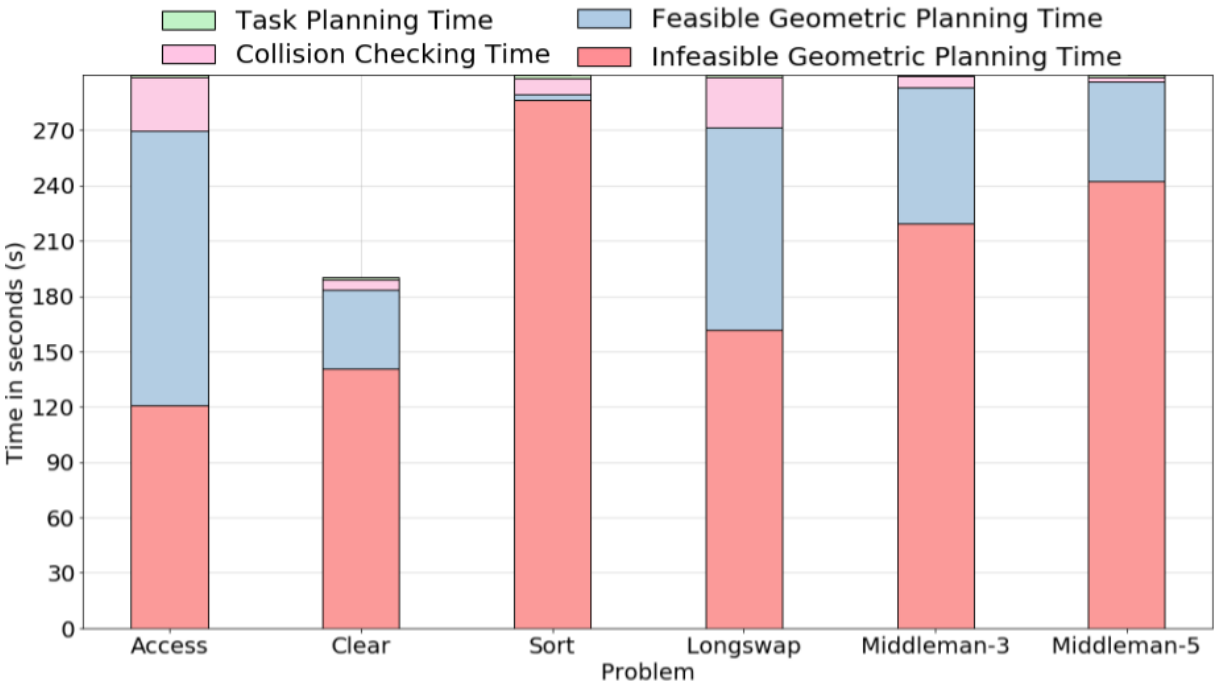
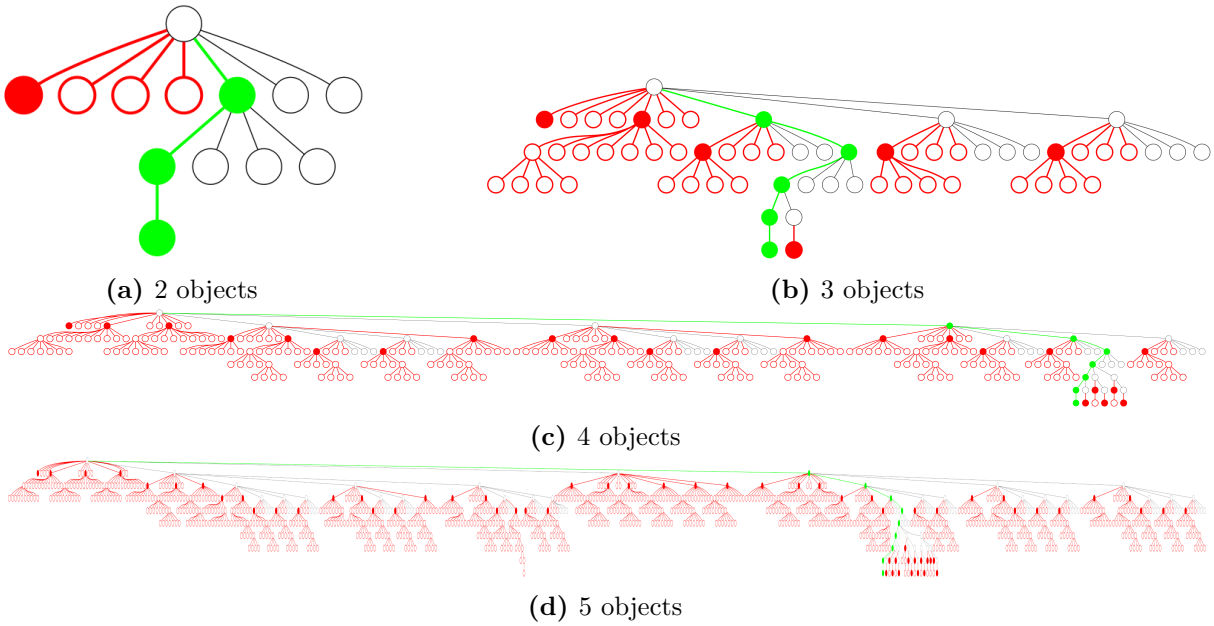


Figure 3.4: Planning time decomposition for each benchmark TAMP problem. The total planning time is broken down into the task planning time (green), collision checking time (purple), geometric planning time on feasible actions (blue), and geometric planning time on infeasible actions (red).

and **Middleman-5** problems shows how increasing the number of robots significantly increases the combinatorial complexity of the problem. This is particularly reflected by the number of expanded nodes, which grows from 2394 nodes on the **Middleman-3** problem to 8649 nodes on the **Middleman-5** problem. Similarly, the number of collision checks increases from 4984 on the **Middleman-3** problem to 18456 on the **Middleman-5** problem. This high number of collision checks is justified by the fact that exchange regions between robots are fully obstructed by fixed obstacles, making only one sequence of robots' exchanges geometrically feasible. Since the task planner does not have access to this information, it naively tries to sample *Pass* placements in the obstructed regions, leading to a high number of collision checks.

On the **Clear** problems, the planner yields a low performance with a success rate of 30%. Although the planner succeeds in some cases, the total planning time is still high, with an average of 190s. A closer look at the number of infeasible task plans generated and geometric planner calls shows that the planner requires, for these problems as well, a high number of queries to the geometric planner before finding a geometrically feasible solution. Note that, for all benchmark problems, a high number of nodes are pruned using backtracking. However, the naive pruning strategy detailed in Algorithm 3 is not sufficient to effectively reduce the search space and find a feasible solution within a reasonable amount of time. As expected, Figure 3.4 shows that planning time is heavily dominated by the geometric planning component for all benchmarks, mainly the queries on infeasible actions since they require more time to verify. The task planning time is negligible in most problems, while the collision checking time can also be high in some cases, such as the **Longswap** problem. These results further highlight the fact that geometric planning is the main bottleneck in task and motion planning problems, both in single and multi-robot settings.



(a) 2 objects **(b)** 3 objects

(c) 4 objects

(d) 5 objects

Figure 3.5: Visualization of the evolution of the search tree for the **Access** problem, depending on the number of objects in the environment. Black nodes represent nodes that have been expanded, filled red nodes/edges represent the actions for which geometric planning failed, while non-filled ones represent node/edges that have been pruned, and green nodes/edges represent nodes/edges that are part of the solution.

3.5.4 Qualitative Analysis

In order to better understand the planner's behavior, and showcase the combinatorial complexity induced by the high number of objects and robots, we visualize the evolution of the search tree for different versions of the **Access** problem, where the number of objects in the environment varies from 2 to 5. Figure 3.5 shows the search tree for each version of the problem, where black nodes represent nodes that have been expanded, red nodes/edges represent nodes/edges that have been pruned during backtracking after an unsuccessful geometric planning call, and green nodes/edges represent nodes/edges that are part of the geometrically feasible solution. Comparing the obtained search trees highlights the exponential growth of the search space with the number of objects, and the high number of infeasible task plans generated by the planner. For a low number of objects (e.g. 2 or 3), the size of the search tree remains tractable, and the number of calls to the geometric planner is reasonable (cf. Figure 3.5a and 3.5b). However, as the number of objects increases, the search space grows exponentially, requiring a high number of computationally expensive geometric planner calls to find a feasible solution (cf. Figure 3.5c and 3.5d). This further highlights the combinatorial complexity of the problem. The same observations can be made for the other benchmarks. In fact, the large size of the search tree is further exacerbated by the increase in the number of robots and the complexity of the interactions between objects.

3.6 Discussion

In this chapter, we have defined offline manipulation planning as a task and motion planning problem, where two planning components are combined: a symbolic planning level that reasons over symbolic states and actions to generate a task plan, and a geometric planning level that

validates the geometric feasibility of each action in the generated task plan and computes the corresponding motion plan. A symbolic-geometric interface bridges the gap between the two planning levels, by grounding symbolic actions and backtracking through the search tree to prune infeasible branches after a failed geometric planning call. We have presented the reference task and motion planning algorithm used throughout this thesis, and introduced a set of benchmark TAMP problems with various challenges used to evaluate the performance of the planner and the impact of our contributions. Results show that the reference planner struggles to find geometrically feasible solutions on the benchmark problems, due to their high combinatorial complexity. The planner generates a high number of infeasible task plans, and requires a high number of geometric planner calls to find a feasible solution. The geometric planning component is the main bottleneck in the planning process, and the size of the search tree grows exponentially with the number of objects and robots in the environment.

Given these results, there is a need for more efficient search strategies and heuristics to guide the exploration towards feasible solutions in a more efficient manner. Particularly, geometric planning must be replaced by a fast and reliable method to assess the feasibility of actions. This would allow the planner to prioritize the exploration of feasible actions, thus greatly reducing the number of infeasible task plans as well as the number of geometric planner calls. In the next chapter, we present a learning-based approach to predict the feasibility of actions and grasps in multi-robot manipulation tasks, and show how it can be integrated into the task and motion planning algorithm to improve its performance and solve complex TAMP problems efficiently.

Chapter 4

Action and Grasp Feasibility Prediction

Contents

4.1 Introduction

As illustrated in Chapter 3, task and motion planning (TAMP) is a challenging problem that requires the combination of discrete symbolic planning with continuous geometric reasoning. This combination results in a high combinatorial complexity making the search for a geometrically feasible task plan tedious, time consuming and, in some cases, unsolvable in a reasonable amount of time. Moreover, geometric planning is a major bottleneck to TAMP. Since symbolic planners do not have any geometric reasoning capabilities, they generate task plans without any guarantees of geometric feasibility. A geometric planner has to be queried in order to verify the feasibility

of each task plan generated and construct the corresponding motions. However, in complex problems, symbolic planners generate a high number of geometrically infeasible solutions before finding a feasible one, each plan requiring a call to the geometric planner which results in long planning time. Moreover, geometric planning might be time consuming even in the case of feasible tasks. Since not all grasps are feasible or lead to a feasible motion, finding the right one might require a lot of time.

Recent works in TAMP leverage learning methods in order to tackle this shortcoming of geometric planners. They propose to learn heuristics that efficiently guide the task planner towards feasible solutions, hence reducing the number of calls to the geometric planner and accelerating the planning process. Particularly, a class of methods (Wells et al. 2019; Driess et al. 2020a,b; L. Xu et al. 2022) use learning models to predict the feasibility of actions without the need for querying geometric planners. These feasibility predictions can then be used as heuristics during task and motion planning to prioritize promising actions, increase the probability of finding a feasible solution. However, these methods are limited to tabletop environments, and are not able to handle 3D environments with multiple objects, support surfaces and obstacles.

In this chapter, we propose a novel learning-based approach to predict the feasibility of actions and grasps in 3D environments. We introduce the **Action and Grasp Feasibility Prediction Network (AGFPNet)**, a convolutional neural network that simultaneously predicts the feasibility of actions and grasps in 3D scenes. The network takes as input an image-based representation of the environment and the action to be tested, and outputs the probability of feasibility of the action as well as the feasibility of different grasp types. We show that the proposed model achieves a high accuracy, and is able to generalize to unseen environments and objects. We also demonstrate that the predictions obtained by **AGFPNet** can be used to notably accelerate task and motion planning, by reducing the number of calls to the geometric planner, and by guiding the task planner towards feasible solutions.

4.2 Problem Definition

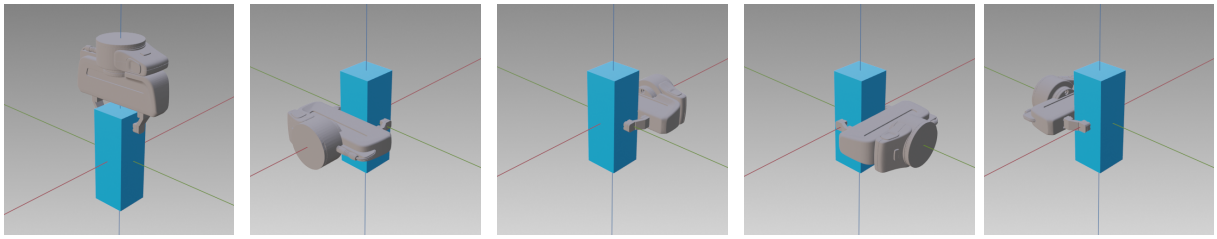
Given a state of the 3D environment s, an action a and an object $\mathcal{O}$ to manipulate, the goal is to predict the feasibility of performing the action on the object of interest at the given state s. The action a can be decomposed into a *Pick* and a *Place* sub-actions such that:

$$a(\mathcal{R}, \mathcal{O}, s(\mathcal{O}), \mathcal{P}) = Pick(\mathcal{R}, \mathcal{O}, s(\mathcal{O})) \rightarrow Place(\mathcal{R}, \mathcal{O}, \mathcal{P}) \tag{4.1}$$

where *Pick* corresponds to picking object $\mathcal{O}$ using robot $\mathcal{R}$ from its current placement $s(\mathcal{O})$, and *Place* refers to placing $\mathcal{O}$ using robot $\mathcal{R}$ at a new placement $\mathcal{P}$. From a geometric planning perspective, these actions are symmetrical (Wells et al. 2019). Assuming they start and end at the same home configuration of the robot[1], a place action at a specific pose is the reversed pick action from that pose. This assumption reduces the number of inputs to the neural network, and allows to speedup data generation and annotation.

Let $\mathcal{G} = \{top, front, rear, right, left\}$ be a set of grasp types, where each grasp type $\gamma \in \mathcal{G}$ corresponds to a continuous set of grasps, related to the specific side from which the object is approached in the perspective of the robot, as shown in Figure 4.1. It is important to note that these are types of grasps and not grasp poses. They correspond to the side from which the object is grasped, but the height, depth and orientation of the grasp are free parameters chosen by the geometric planner. This representation is similar to those introduced in previous works (Wells et al. 2019; Driess et al. 2020a,b).

[1]Assumed to be in the same connected component of the configuration space as the Pick/Place configurations



(a) *top* **(b)** *rear* **(c)** *front* **(d)** *right* **(e)** *left*

Figure 4.1: Visualization of example grasps for each of the 5 grasp types corresponding to the specific side from which the object is approached.

We define the feasibility of an action a as the probability that the geometric planner will find a feasible motion for the action. Let $p_F(s, a)$ be the probability of feasibility of the action a at state s, such that:

$$p_F(s,a) = \begin{cases} 0, & \text{if } a \text{ is geometrically infeasible} \\ 1, & \text{if } a \text{ is geometrically feasible} \end{cases} \tag{4.2}$$

Similarly, let $p_\gamma(s, a)$ be the probability of feasibility of the action a using grasps of type γ at state s such that:

$$p_\gamma(s,a) = \begin{cases} 0, & \text{if all grasps of type } \gamma \text{ are infeasible} \\ 1, & \text{if at least one grasp of type } \gamma \text{ is feasible} \end{cases} \tag{4.3}$$

Also, let $\boldsymbol{p}_{\mathcal{G}}(s, a)$ be the vector grouping the probabilities of feasibility of all grasp types in $\mathcal{G}$, i.e:

$$\boldsymbol{p}_{\mathcal{G}}(s,a) = \begin{bmatrix} p_{top}(s,a) \\ p_{front}(s,a) \\ p_{rear}(s,a) \\ p_{right}(s,a) \\ p_{left}(s,a) \end{bmatrix} \tag{4.4}$$

Here, infeasibility can be due to collisions, kinematic constraints of the robot or the lack of a feasible motion. We define the probability of feasibility of a grasp g as the probability of feasibility of its type $\gamma(g)$. For simplicity, we denote the feasibility of an action a as p_F^a, the feasibility of a grasp type γ as p_γ^a, and the vector of probabilities of feasibility of all grasp types in $\mathcal{G}$ as $\boldsymbol{p}_{\mathcal{G}}^a$. Given the 3D scene E, the state s, the object to manipulate $\mathcal{O}$ and the action a, the goal is to learn a single classifier capable of predicting the probabilities p_F^a and $\boldsymbol{p}_{\mathcal{G}}^a$.

4.3 Problem Representation

In order to predict the feasibility of actions and grasps in 3D environments, which can contain an arbitrary number of objects, we need a compact and efficient scene representation which can encode information about all objects while maintaining a fixed dimensionality. We extend the method proposed by Driess et al. 2020a,b for tabletop scene representation to 3D environments. Their idea is to use fixed-size, top view depth images that can represent scenes with different numbers of objects and a single support surface. Although this representation is efficient for tabletop environments, it has a major issue in the case of 3D environments. Indeed, contrarily to the tabletop case, 3D scenes can have support surfaces at different heights, which might create

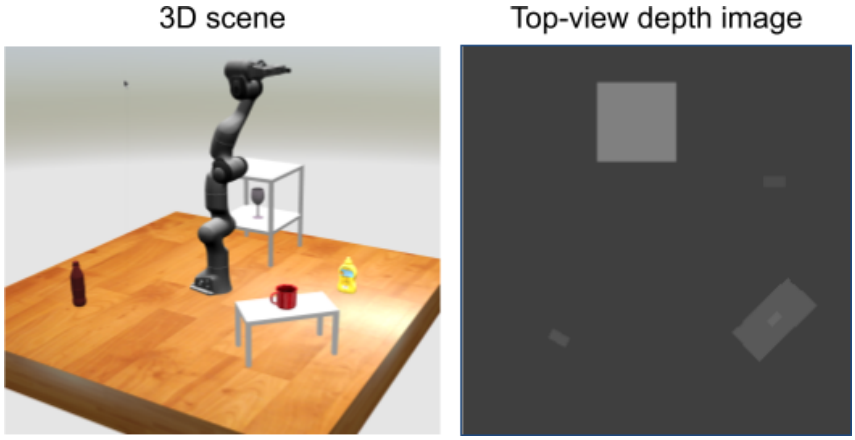


Figure 4.2: Visualization of the limitation of the single top-view depth image representation for 3D scenes. Although some objects are visible in the top view, the wine glass as well as the shelf it is placed on are not visible. This representation does not encode any information about the shape, size and pose of the object.

occlusions. For example, if a scene contains a table, a shelf, and one movable object placed underneath the shelf, the object will not be visible in a top view depth image of the scene. As a result this representation does not encode any information about the shape, size and pose of the object. Figure 4.2 shows an example of this limitation.

In order to solve this issue, we propose to represent the 3D scene as 5 depth images, each one corresponding to a specific view of the scene (top, front, rear, left, right) as shown in Figure 4.3. Each view corresponds to a specific plane in the 3D space. The top view shows a projection of the environment on the (x, y) plane, the front and rear views correspond to the (y, z) plane, and the right and left views correspond to the (x, z) plane. Note that these depth images are generated internally from the robot's model of the scene, and are not obtained from depth cameras. They show all the objects in the scene, fixed and movable, except the object to manipulate. This representation reduces greatly the effect of occlusions, because unless the object is hidden from all sides of the scene, it will be visible in at least one view, and information about that object is encoded. Using this approach, the environment E and the state s which dimensionality is clearly dependent on $n_{objects}$, becomes $\{I_{xy} \in \mathbb{R}^{1\times w\times h}, I_{xz} \in \mathbb{R}^{2\times w\times h}, I_{yz} \in \mathbb{R}^{2\times w\times h}\}$, which have a fixed dimensionality, w and h being the dimensions of the images. These images represent not only information about movable objects, but also the support surfaces and fixed obstacles in the scene (cf. Figure 4.3).

When generating these depth images, we express all objects poses, fixed or movable, in the frame of the robot performing the tested action. The scene projections are constructed such that the center of the robot's workspace is at the center of the depth images, and each projection covers the whole workspace of the robot. As a result, the scene projections only show the objects inside the robot's workspace, hence feasibility prediction can be performed on each one of these objects. This allows our approach to be robot-centric by design, meaning that it is dependent on the kinematics of the robot only, and does not depend on any other factors such as where the robot is placed in the environment. This design choice allows a smooth generalization of our method to multi-robot problems.

Regarding action representation, we consider *Pick* and *Place* tasks as equivalent. Using this assumption, we always consider the action to be tested as a *Pick* task, which reduces the number of inputs to the neural network, and allows to speedup data generation and annotation (cf.

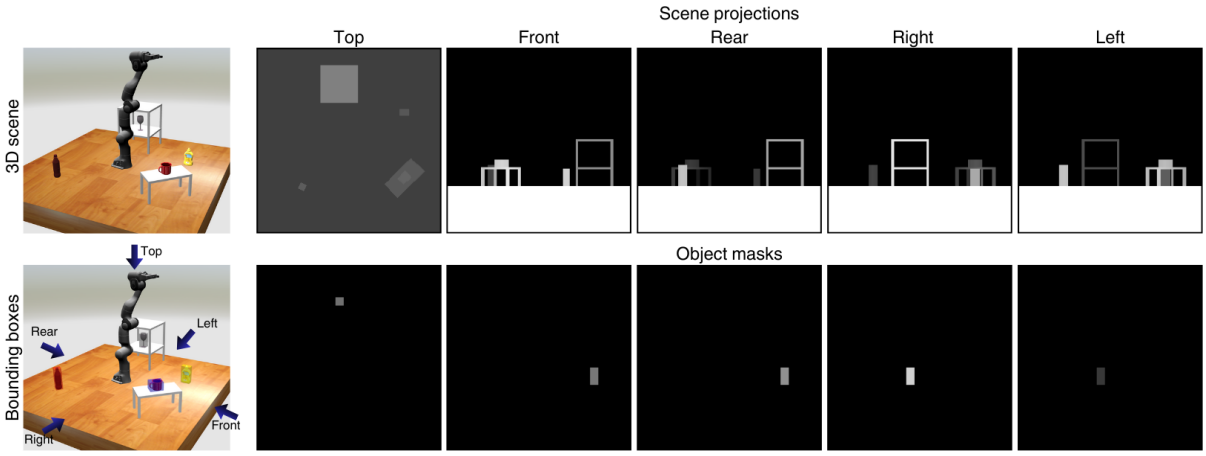


Figure 4.3: Visualization of the proposed 3D scene representation. The scene is represented using 5 depth images, each one corresponding to a specific view of the scene (top, front, rear, left, right). The object of interest (i.e the wine glass in the shelf) is represented using 5 masks over the scene views, showing only the object at the pose it should be picked or placed at.

Section 4.4.5).Note that this assumption is used only at the feasibility prediction level. The geometric planner used in our TAMP approach considers *Pick* and *Place* as different tasks that might start and end at different configurations of the robot. Given this representation, we only need to encode the object of interest and its configuration in the scene, which is done using 5 masks over the scene views, showing only the considered object at the configuration it should be picked or placed at. Note that the object of interest is not visible in the scene projections, and is only represented using these masks. They are added as additional channels to the scene depth images, resulting in a set of three input images $\{I_{xy} \in \mathbb{R}^{2\times w\times h}, I_{xz} \in \mathbb{R}^{4\times w\times h}, I_{yz} \in \mathbb{R}^{4\times w\times h}\}$.

This representation is highly compact compared to existing 3D environment representations such as point clouds, voxel-based or cell decomposition-based representations. It allows to encode all information related to the objects that is relevant to the classifier, i.e. their shape, size, and pose, as well as spatial proximity between objects.

4.4 Action and Grasp Feasibility Prediction Network (AGFPNet)

4.4.1 Neural Network Architecture

The proposed model **AGFPNet** is a convolutional neural network (CNN). It takes as input the 3D scene and the action to be tested represented using the 3D scene and action representations described in the previous section. The network outputs the probability of feasibility of the action, as well as the feasibility of 5 grasp types. The complete architecture of **AGFPNet** is shown in Figure 4.4. The input depth images I_{xy}, I_{xz} and I_{yz} are encoded using three ResNet-based convolutional neural networks (He et al. 2016), resulting in three embeddings h_{xy}, h_{xz} and h_{yz}:

$$h_{xy} = \mathbf{ResNet}_{\boldsymbol{xy}}(I_{xy}), \quad h_{xz} = \mathbf{ResNet}_{\boldsymbol{xz}}(I_{xz}), \quad h_{yz} = \mathbf{ResNet}_{\boldsymbol{yz}}(I_{yz}) \tag{4.5}$$

The resulting embeddings are concatenated, then fed to a fully connected linear layer followed by a ReLU activation function to obtain an encoding of the 3D scene and the action:

$$h = \mathbf{ReLU}(\mathbf{Linear}([h_{xy} \parallel h_{xz} \parallel h_{yz}])) \tag{4.6}$$

where $\parallel$ denotes the concatenation operation.

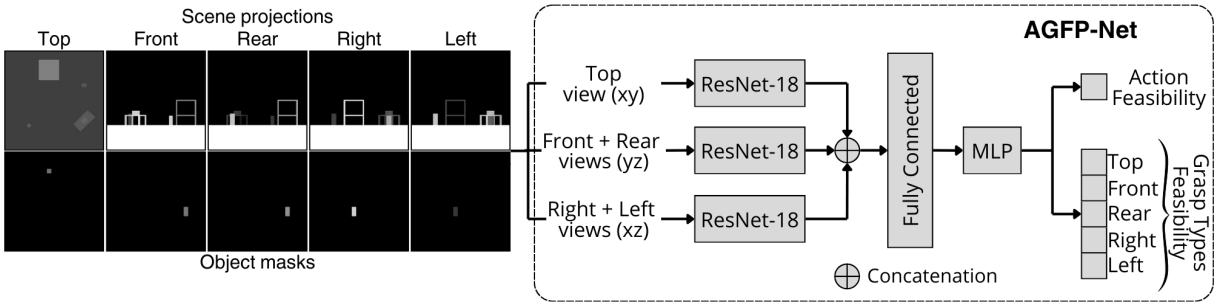


Figure 4.4: The architecture of the proposed neural network **AGFPNet**.

This encoding is finally given as input to a multi-layer perceptron (MLP) followed by a Sigmoid activation function to obtain two predictions. The first one is the probability of feasibility of the action p_F. The second is a vector p_G composed of 6 values, each one being the probability of feasibility p^a_γ of one of the grasp types $\gamma \in \mathcal{G}$:

$$\begin{bmatrix} p_F \\ \boldsymbol{p_\mathcal{G}} \end{bmatrix} = \textbf{Sigmoid}(\textbf{MLP}(h)) \tag{4.7}$$

The MLP is composed of 3 fully connected linear layers, each followed by a ReLU activation function. It acts as a classifier that maps the scene-action encoding to the feasibility of the action and the grasp types. The Sigmoid activation function, on the other hand, maps the output of the MLP to the interval $[0, 1]$, which is the range of the probabilities of feasibility.

4.4.2 Combining Feasibility Predictions

AGFPNet predicts the feasibility of *Pick* and *Place* actions independently. In order to compute the probability of feasibility of complete *Move* actions a, we need a strategy for combining the predicted probabilities of feasibility of the *Pick* and *Place* sub-actions, as well as the feasibility of the grasp types. Given a state s of the environment, the tested action $a = Move(\mathcal{R}, \mathcal{O}, s(\mathcal{O}) \rightarrow \mathcal{P})$, and the object of interest $\mathcal{O}$, we first decompose the action into its *Pick* and *Place* sub-actions following equation 4.1. For each sub-action we generate scene projections and object masks by projecting each object in the scene on the xy, xz and yz planes on each side of the robot, using grayscale as a color gradient representing depth. These projections are then given as input to **AGFPNet** to obtain predictions for each sub-action, namely p_F^{pick}, $\boldsymbol{p}_\mathcal{G}^{pick}$, p_F^{place} and $\boldsymbol{p}_\mathcal{G}^{place}$.

Using these predictions, we first compute the vector of combined grasp type probabilities $\boldsymbol{p}^a_G$. For a single grasp type γ, we define the probability p^a_γ that the complete action a is feasible using γ as follows:

$$p^a_\gamma = p^{Pick}_\gamma \cdot p^{Place}_\gamma \tag{4.8}$$

with p^{Pick}_γ and p^{Place}_γ being the probability that the *Pick* and the *Place* tasks, respectively, are feasible using grasps of type γ. Using this definition, we can obtain the complete vector of grasp types probabilities $\boldsymbol{p}^a_G$ as:

$$\mathbf{p}^a_\mathcal{G} = \mathbf{p}^{Pick}_\mathcal{G} \otimes \mathbf{p}^{Place}_\mathcal{G} \tag{4.9}$$

$\otimes$ being the element-wise product. Each probability in $\boldsymbol{p}^a_G$ can be viewed as the feasibility of picking the object and placing it using the same grasp type.

Regarding the probability of feasibility of the action $p_F(a)$, multiplying the probabilities of the *Pick* and *Place* actions is not sufficient. The reason is that even if both these tasks are feasible, the grasps leading to a feasible *Pick* might not be the same as the ones leading to a feasible

Place. Indeed, the complete $Move$ action a is feasible if and only if both the *Pick* and the *Place* sub-actions are feasible, and have at least one feasible grasp type in common. Therefore, we compute the probability of feasibility of the complete action p_F^a as:

$$p_F^a = p_F^{Pick} \times p_F^{Place} \times max(\mathbf{p}_\mathcal{G}^a) \tag{4.10}$$

where $max(\mathbf{p}_\mathcal{G}^a)$ is the maximum probability of feasibility of the grasp types in $\boldsymbol{p}_G^a$. This definition ensures that the complete action a is considered feasible only if both the *Pick* and the *Place* actions are feasible, and have at least one feasible grasp type in common. Indeed, if no common grasp type is feasible for both sub-actions, the maximum combined grasp type feasibility would tend to zero, and the complete action would be considered infeasible, even if the *Pick* and the *Place* tasks are feasible independently.

The probability of feasibility of the action p_F^a is used at the task planning level to prioritize feasible actions during the search, while the feasibility of grasp types $\boldsymbol{p}_G^a$ is also used during geometric planning to prioritize feasible grasps.

4.4.3 Extension to Mesh-shaped Objects

In order to maintain our model's generalizability to objects with complex shapes, we need a strategy for representing these objects at the feasibility prediction level. One possibility could be to represent mesh-shaped objects using their true shapes during depth image generation. However, this method has several drawbacks. First, building depth images of mesh objects can be time consuming, which defeats the purpose of accelerating TAMP. Second, this method does not guarantee the generalizability to unseen shapes during training. For these reasons, we propose to represent mesh-shaped objects at the feasibility prediction level using their bounding boxes. This representation is expressive enough to encode information about the shape, size and pose of the object, while maintaining efficiency. Moreover, it allows the neural network to generalize to unseen objects of various shapes, without the need for any additional data generation/annotation and training effort. Note that this simplification is limited to the feasibility prediction module, the geometric planner uses the real shapes of mesh objects, which does not hurt ability of TAMP planner to grasp and manipulate complex mesh-shaped objects.

When querying the neural network, the input scene projections are generated using the bounding boxes of mesh objects as shown in Figure 4.4. **AGFPNet** outputs the probability of feasibility of performing an action on mesh-shaped objects, as well as the feasibility of each grasp type $\gamma \in \mathcal{G}$. Since these grasp types represent continuous subsets of grasps, we need to associate the mesh object's grasps (obtained using an off-the-shelf grasp planner or from a grasp database) to one of these grasp types. More formally, γ is a function that maps a grasp g to a grasp type $\gamma(g)$ depending on the angle of approach of the robot's gripper with respect to the object's axes. Given an object $\mathcal{O}$ which axes are $(\boldsymbol{x}_\mathcal{O}, \boldsymbol{y}_\mathcal{O}, \boldsymbol{z}_\mathcal{O})$, and a grasp g of $\mathcal{O}$ which direction of approach is defined by the axes $(\boldsymbol{x}_g, \boldsymbol{y}_g, \boldsymbol{z}_g)$, we define the angles Θ, Φ, Ψ as the angles between $\boldsymbol{x}_g$ and $\boldsymbol{z}_\mathcal{O}$, $\boldsymbol{x}_g$ and $\boldsymbol{x}_\mathcal{O}$, $\boldsymbol{y}_g$ and $\boldsymbol{x}_\mathcal{O}$, respectively (cf. Figure 4.5). The grasp type of g can be obtained as follows:

$$\gamma(g) = \begin{cases} Top & |\Theta| \leq \frac{\pi}{4} \\ Front & |\Theta| > \frac{\pi}{4} \text{ and } |\Phi| > \frac{3\pi}{4} \\ Rear & |\Theta| > \frac{\pi}{4} \text{ and } |\Phi| \leq \frac{\pi}{4} \\ Right & |\Theta| > \frac{\pi}{4} \text{ and } \frac{-3\pi}{4} \leq \Phi < \frac{-\pi}{4} \\ Left & |\Theta| > \frac{\pi}{4} \text{ and } \frac{\pi}{4} < \Phi \leq \frac{3\pi}{4} \end{cases} \tag{4.11}$$

Using this formulation, we define the probability of feasibility of g as the probability of feasibility of its type.

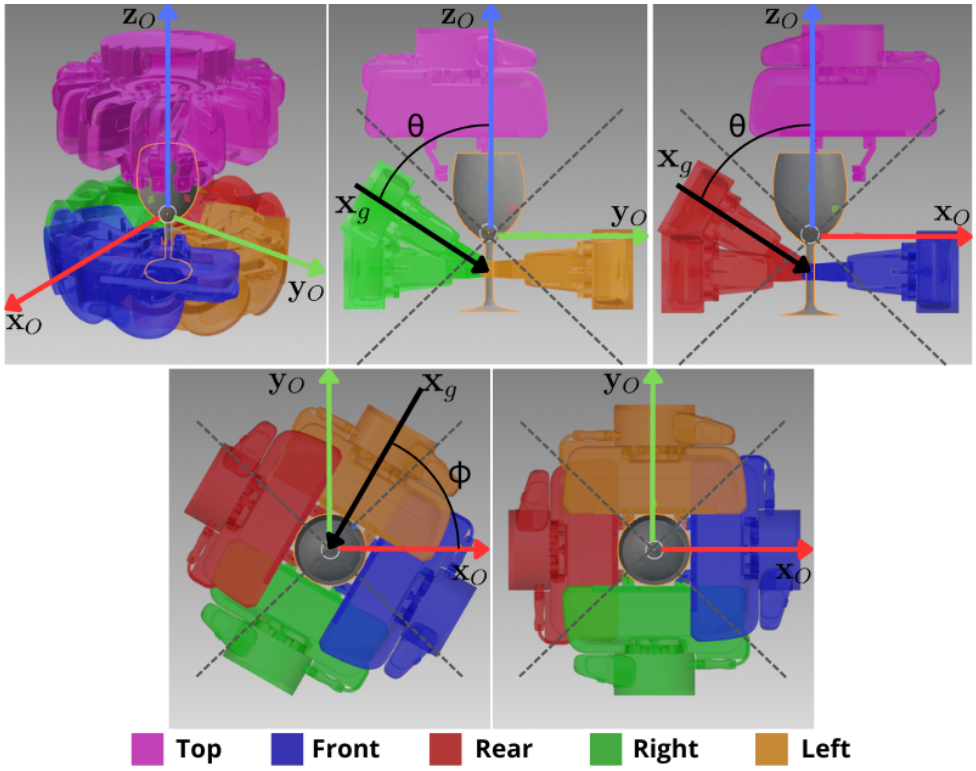


Figure 4.5: Visualization of the 5 grasp types depending on the angle of approach of the robot's gripper.

4.4.4 Extension to Multi-Robot Settings

AGFPNet is by design robot-centric, meaning that it is dependent on the geometry and kinematics of the robot only, and does not depend on any other factors such as where the robot is placed in the environment. Also, the model covers the whole workspace the robot. These properties are achieved by expressing all objects poses, fixed or movable, in the frame of the robot. The scene projections are constructed such that the center of the robot's workspace is at the center of the depth images, and each projection covers the whole workspace of the robot. As a result, the scene projections only show the objects inside the robot's workspace, hence feasibility prediction can be performed on each one of these objects. These capabilities allow a smooth generalization of **AGFPNet** to multi-robot problems. Indeed, the neural network can be queried for each robot individually by making sure the input images show the objects (or parts of objects) in the workspace of the said robot only, and that the shown poses of these objects are in the frame of the latter, as shown in Figure 4.6. As a result, the model can predict the feasibility of actions performed by each robot independently, allowing a straightforward extension to multi-robot problems while being trained on single-robot scenes only.

Moreover, **AGFPNet** can be used to predict the feasibility of collaborative actions involving two robots, such as *Pass* actions or *Handover* actions. In this thesis, we focus on *Pass* actions. The challenge, however, lies in estimating the combined feasibility of these collaborative actions. A *Pass* action aims at moving an object to the intersection region between two robots' workspaces using one robot, so that the other is able to manipulate it. Therefore, a *Pass* action is feasible only if one robot is able to pick the object, place it at the intersection region, and if the other robot is able to pick it from this region. In order to be reliable, feasibility prediction must take into account both robots when evaluating collaborative actions.

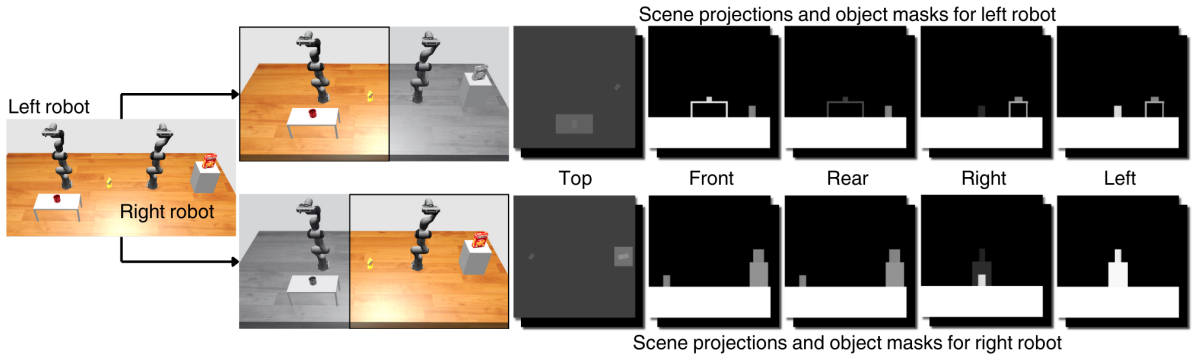


Figure 4.6: Visualization of **AGFPNet**'s input on an example 2-robot scene. To predict the feasibility of a *Pick* or *Place* action, the neural network is queried for each robot individually, using depth images showing the objects inside their workspaces, respectively. Objects in the intersection of both workspaces can be seen in both scene projections (i.e mustard bottle).

We propose a new method for estimating collaborative actions feasibility (CF). Let's consider an environment containing two robots $\mathcal{R}_i$ and $\mathcal{R}_j$, the state s of the environment, an object $\mathcal{O}$ to manipulate, and a *Pass* action a_{pass} that aims at exchanging $\mathcal{O}$ between $\mathcal{R}_i$ and $\mathcal{R}_j$. The goal is to estimate the probability of feasibility $p_{CF}^{a_{pass}}$ of the *Pass* action. At the feasibility prediction level, these actions can be decomposed into three sub-actions: a *Pick* action by the first robot, a *Place* action at the intersection region, and a *Pick* action by the second robot:

$$a_{pass}(\mathcal{R}_i, \mathcal{R}_j, \mathcal{O}, s(\mathcal{O}) \rightarrow \mathcal{P}) = Move(\mathcal{R}_i, \mathcal{O}, s(\mathcal{O}) \rightarrow \mathcal{P}) \rightarrow Pick(\mathcal{R}_j, \mathcal{O}, \mathcal{P}) \tag{4.12}$$

where $\mathcal{P}$ is an object placement sampled in the intersection region between the workspaces of $\mathcal{R}_i$ and $\mathcal{R}_j$. The object's initial configuration $s(\mathcal{O})$ and the placement pose $\mathcal{P}$ are initially expressed in the world frame. In order to estimate the feasibility of the *Pass* action, we first transform the object's initial configuration and the *Pass* placement to the frame of each robot. Given the robot $\mathcal{R}_i$'s frame transform $\mathbf{T}_{\mathcal{R}_i}$, the pose of $\mathcal{O}$ expressed in the frame of the robot can be computed as:

$$s(\mathcal{O})_{\mathcal{R}_i} = \mathbf{T}_{\mathcal{R}_i}^{-1} \cdot s(\mathcal{O}) \tag{4.13}$$

Similarly, the *Pass* placement pose $\mathcal{P}$ is transformed to the frame of each robot to obtain $\mathcal{P}_{\mathcal{R}_i}$ and $\mathcal{P}_{\mathcal{R}_j}$. Using these poses, we construct the input images for each robot and sub-action, and query the neural network to predict the probabilities of feasibility $p_F^{pick}(\mathcal{R}_i, \mathcal{O}, s(\mathcal{O})_{\mathcal{R}_i})$, $p_F^{place}(\mathcal{R}_i, \mathcal{O}, \mathcal{P}_{\mathcal{R}_i})$ and $p_F^{pick}(\mathcal{R}_j, \mathcal{O}, \mathcal{P}_{\mathcal{R}_j})$. We first compute the probability of feasibility of the *Move* sub-action $p_F^{a_{pass}}$ using equation 4.10. Then, we compute the collaborative feasibility $p_{CF}^{a_{pass}}$ of the *Pass* action between $\mathcal{R}_i$ and $\mathcal{R}_j$ as:

$$p_{CF}^{a_{pass}} = p_F^{a_{pass}}(\mathcal{R}_i, \mathcal{R}_j, \mathcal{O}, s(\mathcal{O})_{\mathcal{R}_i}, \mathcal{P}_{\mathcal{R}_i}) \times p_F^{Pick}(\mathcal{R}_j, \mathcal{O}, \mathcal{P}_{\mathcal{R}_j}) \tag{4.14}$$

This formulation ensures that our feasibility prediction method takes into account both robots when evaluating collaborative actions, and can be used to predict the feasibility of any collaborative action involving two robots. This is particularly useful in problems where intersection regions between robots' workspaces are obstructed by obstacles, or when the robots have different kinematic constraints, making only a subset of exchange regions feasible.

Note that for non-collaborative actions a_{goal} of type *goal*, and a_{temp} of type *Temp* using a robot $\mathcal{R}$ (as introduced in Section 3.2.1), the probability of feasibility is computed as in eq. (4.10):

$$p_F^{a_{goal}} = p_F^{a_{temp}} = p_F^{a}(\mathcal{R}, \mathcal{O}, s(\mathcal{O})_{\mathcal{R}}, \mathcal{P}_{\mathcal{R}}) \tag{4.15}$$

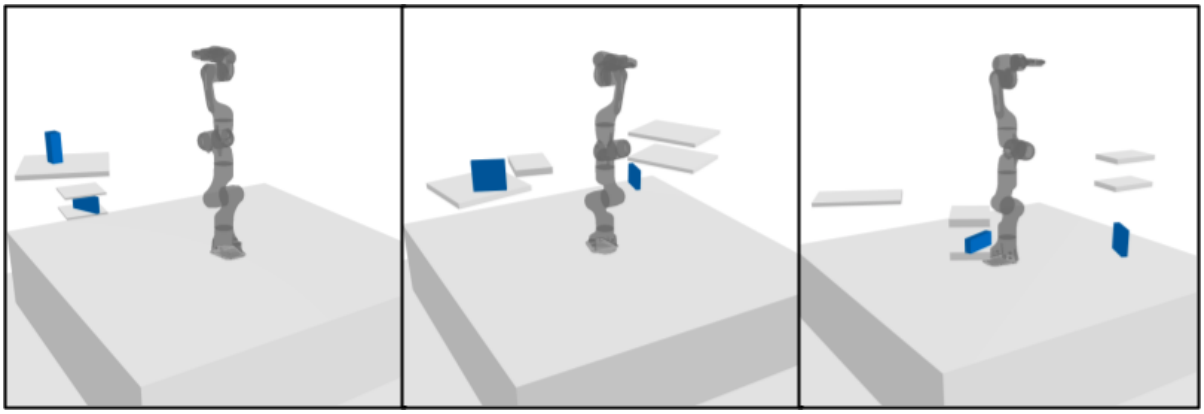


Figure 4.7: Visualization of environments generated using the scene generation method.

4.4.5 Data Generation and Annotation

The proposed model is trained in a supervised manner on a fully synthetic dataset. The latter is generated using a data generation method that can be divided into three steps, scene generation, dataset construction, and data annotation:

Scene Generation: This step consists of generating 3D scenes containing a large table, $n_{support}$ elevated support surfaces, and $n_{movable}$ movable objects. First, a large table is generated at the center of the scene, with the robot's base fixed at the center of the table. After that, we sample a number $n_{support} \in [0, N_{supp}]$ of support surfaces to add to the scene, where N_{supp} is the maximum number of support surfaces allowed. We generate each support surface as a box with a randomly sampled size and pose within a predefined range. These support surfaces are place at different heights, and might overlap. This process is repeated until the number of support surfaces to add is reached. Then, we generate $n_{movable}$ box-shaped objects with sizes sampled randomly. Since **AGFPNet** operates on objects' bounding boxes, it is sufficient to include box-shaped objects only in the dataset. In order to incorporate the different challenges of *Pick* and *Place* actions, we define 3 placement sampling types:

- **Random placement:** The object is placed at a random pose in the environment. We first randomly choose a support surface, then we uniformly sample a pose within the chosen support surface's boundaries.
- **Proximity placement:** We place the object in the neighborhood of another object. First, we randomly choose a movable object, then we uniformly sample a pose in a predefined radius around the chosen object.
- **Underneath placement:** The object is placed underneath a support surface. We randomly choose an elevated support surface, then we uniformly sample a pose underneath the chosen support surface.

We choose one of these placement types at random for each object, then we sample a pose before adding the object to the scene. As a final step, we perform a collision checking ensuring that the generated scenes are valid. This process is repeated until the desired number of scenes is reached. The scene generation procedure is detailed in Algorithm 8. We generate a dataset that consists of 20'000 scenes for training, 5'000 scenes for validation, and 5'000 scenes for testing, with $n_{movable} = 2$ and $N_{supp} = 4$. Examples of generated scenes are shown in Figure 4.7.

Dataset Construction: In this step, we generate the actions to be tested in each scene. As

Algorithm 8 GenerateScene

Input: $n_{movable}, n_{support}$ ▷ Number of movable objects and support surfaces
$E \leftarrow \emptyset$
$E \leftarrow E \cup generateTable()$
while $n_{support}$ is not reached **do**
  $D \leftarrow sampleSupportDimensions()$
  $\mathcal{P} \leftarrow sampleSupportPose()$
  $\mathcal{O}_{support} \leftarrow generateSupportSurface(D, \mathcal{P})$
  if $checkCollision(E, \mathcal{O}_{support})$ is False **then**
    $E \leftarrow E \cup \mathcal{O}_{support}$
  else
    continue
  end if
end while
while $n_{movable}$ is not reached **do**
  $D \leftarrow sampleMovableDimensions()$
  $placementType \leftarrow choice([Random, Proximity, Underneath])$
  $\mathcal{P} \leftarrow sampleMovablePose(placementType, E)$
  $\mathcal{O}_{movable} \leftarrow generateObject(D, \mathcal{P})$
  if $checkCollision(E, \mathcal{O}_{movable})$ is False **then**
    $E \leftarrow E \cup \mathcal{O}_{movable}$
  else
    continue
  end if
end while
return E

Algorithm 9 GenerateActions

Input: *Scenes* ▷ All generated scenes
Dataset $\leftarrow \emptyset$
for each E in *Scenes* **do**
  for each movable object $\mathcal{O}$ in E **do**
    $Dataset \leftarrow Dataset \cup generateAction(\mathcal{O}, E)$
  end for
end for
return *Dataset*

mentioned earlier, we consider *Pick* and *Place* actions as equivalent. This allows us to generate and annotate *Pick* actions only, while allowing the neural network to predict the feasibility of both *Pick* and *Place* actions. For each scene generated, we add an action for every movable object in the scene as a datapoint. In order to incorporate cases where the action is infeasible due to collisions between objects (e.g placing an object at a preoccupied location), we sample for each movable object a placement that causes a collision with another object or a support surface, and add an action for this placement to the dataset. This process results in a dataset containing 90'879 training actions, 22'679 validation actions, and 22'684 testing actions. The dataset construction procedure is shown in Algorithm 9.

Data Annotation: After generating scenes and actions, we use the geometric planner described in Section 3.2.2 to annotate the dataset. For each action in the dataset, we query the geometric planner to plan its corresponding motion. If the geometric planner finds a feasible motion, the action is annotated as feasible. Otherwise, it is annotated as infeasible. During geometric planning, we also record the feasibility of each grasp type. After grasp sampling, we first determine the grasp type of the grasp using equation 4.11. A grasp is feasible if the IK solver finds a corresponding collision-free robot configuration. Hence, we annotate a grasp type as feasible if at least one grasp of this type is feasible. If no grasp of a certain type yields a collision-free configuration, the grasp type is annotated as infeasible. The data annotation procedure is detailed in Algorithm 10.

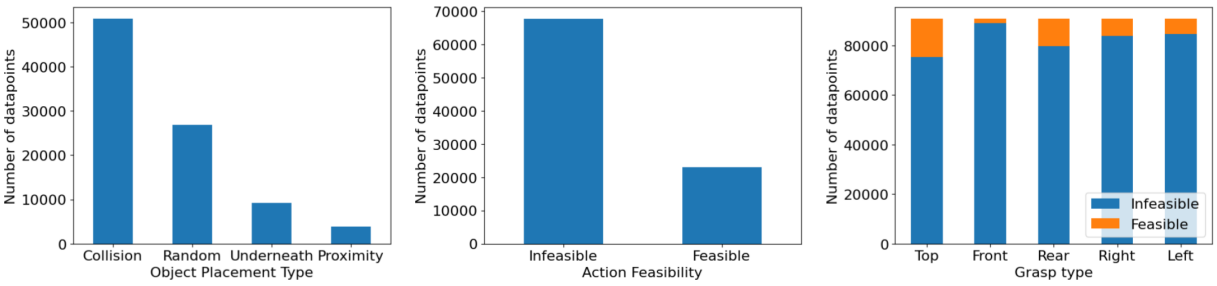
Figure 4.8 shows some statistics about the generated dataset. The dataset is heavily imbalanced towards infeasibility. Indeed, the number of infeasible actions is 3 times higher than the number

Algorithm 10 AnnotateData

Input: $E, \mathcal{R}, \mathcal{O}, K, \tau_{local}, \tau_{global}$ ▷ Environment, robot, object, max. IK solutions, local and global timeouts

$p_F \leftarrow 0$
$\boldsymbol{p_G} \leftarrow \mathbf{0}$
while τ_{global} is not reached **do**
  $G \leftarrow sampleAllGrasps(\mathcal{O})$
  $Q \leftarrow \emptyset$
  for each grasp $g \in G$ **do**
    $\gamma \leftarrow getGraspType(g)$
    $\mathcal{Q}_g \leftarrow solveIK(E, \mathcal{R}, \mathcal{O}, g, K)$
    for each $\boldsymbol{q} \in \mathcal{Q}_g$ **do**
      if $checkCollision(\mathcal{R}, \boldsymbol{q}, E)$ is False **then**
        $\boldsymbol{p_G}(\gamma) \leftarrow 1$
        $Q \leftarrow Q \cup \boldsymbol{q}$
      end if
    end for
  end for
  for each $\boldsymbol{q} \in Q$ **do**
    $m \leftarrow planMotion(E, \mathcal{R}, \boldsymbol{q}, \tau_{local})$
    if m is feasible **then**
      $p_F \leftarrow 1$
      break
    end if
  end for
end while
return $p_F, p_{\mathcal{G}}$

of feasible actions. This imbalance is more pronounced for grasp type feasibility with an average ratio of 10 infeasible cases per feasible case. This highlights the fact that, in TAMP problems, most actions considered are more likely to be infeasible, resulting in numerous computationally expensive queries to the geometric planner. This imbalance in the dataset is addressed during training by using a weighted loss function that accounts for the imbalance between feasible and infeasible datapoints, as explained in the next section.



(a) Number of datapoints per placement generation type

(b) Number of feasible and infeasible actions

(c) Number of feasible and infeasible cases per grasp type

Figure 4.8: Dataset generation and annotation statistics.

4.4.6 Training Method

The model is implemented on Pytorch (Paszke et al. 2019) and trained using a binary cross entropy loss to account for the imbalance between feasible and infeasible datapoints. The loss function is defined as:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} w_i \left[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right] \tag{4.16}$$

where N is the number of datapoints, y_i is the ground truth label, $\hat{y}_i$ is the predicted probability, and w_i is the weight of the datapoint. We set w_i to 1 for feasible actions, and to $\frac{N_{infeasible}}{N_{feasible}}$

for infeasible actions, where $N_{infeasible}$ and $N_{feasible}$ are the number of infeasible and feasible datapoints, respectively. This weighting scheme ensures that the model does not favor predicting one class over the other.

Moreover, in order to increase the size of our training set, we use online data augmentation. Since the robot can turn 360° around its base, rotating the scene around the z axis does not affect the feasibility of the action or the grasp types. This allows to obtain new labeled datapoints from already annotated ones. During training, scenes are either kept as they are, or rotated by a multiple of $\frac{\pi}{2}$ around the z axis. This data augmentation method improves training by allowing the neural network to learn more scenarios, and ensuring that the predictions are consistent between equivalent scenes. It also allows for a better generalization to new environments. The neural network is trained for 200 epochs, which takes approximately 24 hours.

4.4.7 Feasibility-Informed Task and Motion Planning

We propose a feasibility-informed multi-robot TAMP algorithm which leverages feasibility predictions to accelerate the search process, and is capable of solving a wide range of single and multi-robot manipulation problems. The main algorithm is the same as Algorithm 2 described in Section 3.3.1. The only difference is that the feasibility of actions and grasp types is predicted using **AGFPNet** and used to prioritize feasible actions during the search.

During node expansion, the neural network is queried for each considered action to predict its feasibility. Algorithm 11 shows the updated *findChildren* function. For each applicable action found, the feasibility of the action and the grasp types are computed using the *computeFeasibility* function. Following the definitions of Sections 4.4.2 and 4.4.4, the latter first decomposes the action into *Pick* and *Place* sub-actions, then queries the neural network to predict the feasibility

Algorithm 11 findChildren (Updated Alg. 4)

Input: $s, E, s_{goal}, \mathbf{\Gamma}$
$children \leftarrow \emptyset$
$Actions \leftarrow findActions(s, E, s_{goal}, \mathbf{\Gamma})$
for each a in $Actions$ **do**
  $p^a_F, \boldsymbol{p}^a_{\mathcal{G}} \leftarrow computeFeasibility(E, s, a)$
  $s' \leftarrow T(s, a)$
  $s'.cost \leftarrow computeCost(s', s_{goal}, \mathbf{\Gamma}, p^a_F)$
  $children \leftarrow children \cup s'$
end for
return $children$

Algorithm 12 computeFeasibility

Input: E, s, a
$\mathcal{R}, \mathcal{O}, s(\mathcal{O}), \mathcal{P} \leftarrow extractActionParams(a)$
$[p^{Pick}_F, \boldsymbol{p}^{Pick}_{\mathcal{G}}] \leftarrow$ **AGFPNet**$(\mathcal{R}, \mathcal{O}, s(\mathcal{O}), E)$
$[p^{Place}_F, \boldsymbol{p}^{Place}_{\mathcal{G}}] \leftarrow$ **AGFPNet**$(\mathcal{R}, \mathcal{O}, \mathcal{P}, E)$
$\boldsymbol{p}^a_{\mathcal{G}} \leftarrow \boldsymbol{p}^{Pick}_{\mathcal{G}} \otimes \boldsymbol{p}^{Place}_{\mathcal{G}}$
$p^a_F \leftarrow p^{Pick}_F \times p^{Place}_F \times max(\boldsymbol{p}^a_{\mathcal{G}})$
if a is *Pass* **then**
  $\mathcal{R}' \leftarrow extractReceivingRobot(a)$
  $[p^{Pick'}_F, \boldsymbol{p}^{Pick'}_{\mathcal{G}}] \leftarrow$ **AGFPNet**$(\mathcal{R}', \mathcal{O}, \mathcal{P}, E)$
  $p^a_{CF} \leftarrow p^a_F \times p^{Pick'}_F$
  return $p^a_{CF}, \boldsymbol{p}^a_{\mathcal{G}}$
else
  return $p^a_F, \boldsymbol{p}^a_{\mathcal{G}}$
end if

Algorithm 13 computeCost (Updated Alg. 6)

Input: $s', s, s_{goal}, E, \mathbf{\Gamma}, p_F^a, C_{max}$
1: $C_{SoFar} \leftarrow C_{SoFar}(s) + 1$
2: $C_{ToGoal} \leftarrow computeCostToGoal(s', s_{goal}, E, \mathbf{\Gamma})$
3: $C_{Feasibility} \leftarrow \frac{1}{p_F^a} - 1$
4: **if** $C_{feasibility} > C_{max}$ **then**
5: $\quad C_{feasibility} \leftarrow C_{max}$
6: **end if**
7: $C_{Total} \leftarrow C_{SoFar} + C_{ToGoal} + C_{feasibility}$
8: **return** C_{Total}

of each sub-action and the corresponding grasp types. The feasibility of the complete action is then computed using equations 4.10, 4.9 and 4.14. This process is detailed in Algorithm 12.

After the feasibility prediction step, we construct a new child as the result of applying action a to state s. The cost of node in the search tree, defined in equation 3.7, is the sum of the number of actions in the branch leading to the child node, the number of objects that are not at their goal placement, and the minimum number of robot exchanges necessary to reach the goal state. This formulation does not take into account the feasibility of the actions leading to the child node. Therefore, we propose a new cost component $C_{Feasibility}$ that incorporates the feasibility predictions in the state cost, defined as:

$$C_{Feasibility} = \frac{1}{p_F^a} - 1 \in]0, \infty[\tag{4.17}$$

where p_F^a is the probability of feasibility of the action a computed using equation 4.10 or 4.14 depending on the type of action (cf. Algorithm 12).

This cost component is added to the state cost, and is inversely proportional to the feasibility of the action. This means that the higher the probability of feasibility of an action is, the lower its cost will be. The new cost function is thus defined as:

$$C_{Total} = C_{SoFar} + C_{ToGoal} + C_{Feasibility} \tag{4.18}$$

This formulation allows to prioritize feasible actions during the search, and to avoid exploring infeasible branches. Thanks to the inverse proportionality of the feasibility cost, states issued from actions with a low probability of feasibility tend to have an infinite cost, while states emanating from feasible ones have a low cost. As a result, during node selection (cf. Algorithm 2, line 4)., the search algorithm will favor expanding nodes that are more likely to be feasible first, which increases the likelihood that the task plans found are geometrically feasible. Thus, the number of calls to the geometric planner can be greatly reduced, and the search process accelerated. To account for misclassification errors, in practice, the feasibility cost is clipped to a user-defined maximum value C_{max}.

An additional novelty introduced thanks to action and grasp feasibility predictions is the ability to leverage predicted grasp type feasibility to accelerate geometric planning. As shown in Algorithm 14, when the geometric planner is called to find the motion of an action a, we use the previously computed grasp type probabilities $\mathbf{p}_G(a)$ in order to prioritize feasible grasp types during grasp sampling. This is done by sorting the sampled grasps in decreasing order according to the probability of feasibility of their corresponding grasp type. Hence, the grasps predicted to be feasible are evaluated first. This heuristic allows to reduce the geometric planning time on feasible actions, and to avoid exploring infeasible grasps.

Algorithm 14 GeometricPlanner (Updated Alg. 1)

Input: $E, s, a, n_{IK}, \tau_{local}, \tau_{global}$, $\boldsymbol{p}^a_{\mathcal{G}}$ ▷ Environment, robot, object, initial and final placements, max. IK solutions, local and global timeouts, grasp type probabilities

$\mathcal{R}, \mathcal{O}, s(\mathcal{O}), \mathcal{P} \leftarrow extractActionParams(a)$
$\boldsymbol{G} \leftarrow sampleGrasps(\mathcal{O})$
$\boldsymbol{G} \leftarrow sortGrasps(\boldsymbol{G}, \boldsymbol{p}^a_{\mathcal{G}})$
while τ_{global} is not reached and $\boldsymbol{G}$ is not empty **do**
  $g \leftarrow \boldsymbol{G}.pop()$
  $\mathcal{Q}_g^{Pick} \leftarrow solveIK(\mathcal{R}, g, s(\mathcal{O}), n_{IK})$
  for each $\boldsymbol{q}_{Pick}$ in $\mathcal{Q}_g^{Pick}$ s.t. $checkCollision(\mathcal{R}, \boldsymbol{q}_{Pick}, E)$ is False **do**
    $m_{Pick} \leftarrow planMotion(\mathcal{R}, s(\mathcal{R}), \boldsymbol{q}_{Pick}, E, \tau_{local})$
    if m_{Pick} is infeasible **then**
      continue
    end if
    $\mathcal{Q}_g^{Place} \leftarrow solveIK(\mathcal{R}, g, \mathcal{P}, n_{IK})$
    for each $\boldsymbol{q}_{Place}$ in $\mathcal{Q}_g^{Place}$ s.t. $checkCollision(\mathcal{R}, \boldsymbol{q}_{Place}, E)$ is False **do**
      $m_{Place} \leftarrow planMotion(\mathcal{R}, \boldsymbol{q}_{Pick}, \boldsymbol{q}_{Place}, E, \tau_{local})$
      if m_{Place} is infeasible **then**
        continue
      end if
      $m \leftarrow \{m_{Pick}, m_{Place}\}$
      return m
    end for
  end for
end while
return Action Infeasible

4.5 Results

4.5.1 Experimental Setup

We use the same experimental setup detailed in Section 3.5.1. The neural network is trained on a single NVIDIA RTX A5000 GPU, using the ADAM optimizer (Kingma 2014) with a learning rate of 0.0001 and a batch size of 128. To prevent overfitting, we use dropout with a probability of 0.1, and weight decay of 0.0001. During planning, the planner is run on an Intel i9-11950H @ 2.60GHz workstation, with 32GB of RAM, while the GPU is used for the neural network inference. In this section, we first evaluate the prediction performance of **AGFPNet** on the generated test set (cf. Section 4.4.5), using the **F1-Score**, the **True Positive Rate (TPR)**, and the **True Negative Rate (TNR)** as evaluation metrics. Then, we evaluate the planning performance of the proposed feasibility-informed planning algorithm on the benchmark TAMP problems described in Section 3.4, in comparison to the baseline planner detailed in Section 3.3.1.

4.5.2 AGFPNet Prediction Performance

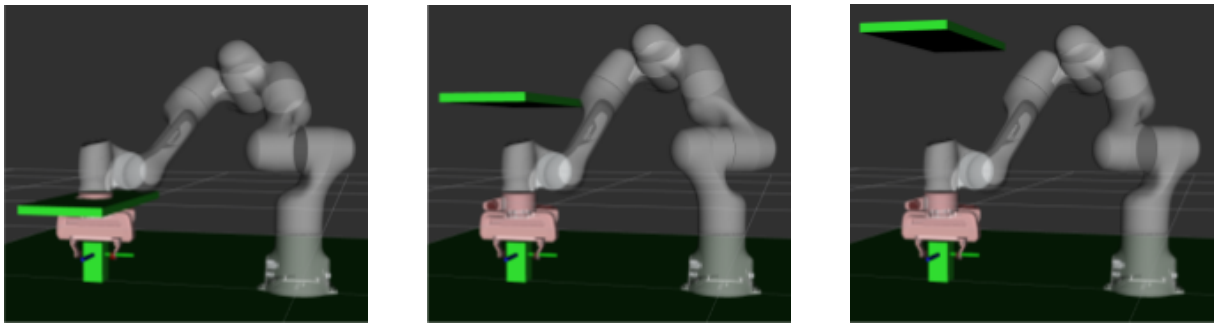
Table 4.1 shows the performance of the proposed neural network. Results show that the model is able to accurately predict the feasibility of actions with an F1-Score of 92.4%. Moreover, the yielded true positive rate (TPR) of 93.7% and true negative rate (TNR) of 97% show that **AGFPNet** is as good at predicting feasibility as it is at predicting infeasibility. Regarding grasp feasibility prediction, the results show that the model is able to accurately predict the feasibility of each grasp type with an F1-Score of 86.6%, a TPR of 93.3% and a TNR of 98.1%. These results demonstrate the reliability of the proposed neural network in predicting the feasibility of actions as well as grasp types. Also, the balance between the TPR and TNR shows the effectiveness of ou proposed training strategy in preventing the model from favoring one class over the other. Furthermore, it is important to note that **AGFPNet**'s inference time is in average $30ms$ per action, which is significantly faster than a typical geometric planning call, which can take up to several seconds. During planning, this inference time can further be reduced to $10ms$ by caching the predictions of the neural network for each action, and reusing them when the same action is

Table 4.1: Prediction performances of **AGFPNet** on the generated test set.

Model	Prediction Task	F1-Score	TPR	TNR
AGFPNet	Action Feasibility	92.4%	93.7%	97%
	Grasp Feasibility	86.6%	93.3%	98.1%

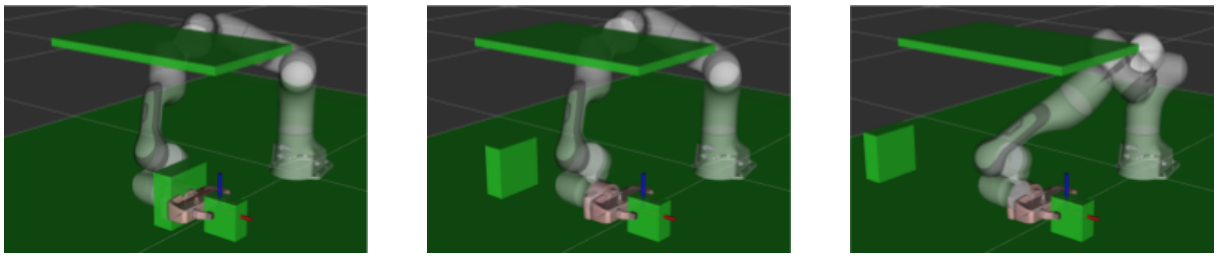
encountered again.

We also qualitatively analyze the predictions of **AGFPNet** on example scenes. Figure 4.9 shows the variation of the predicted feasibility of a *top* grasp type depending on the height of an obstacle above the object. The results show that the probability of feasibility predicted by the model increases with the height of the obstacle, with a near-zero probability when the obstacle causes a collision with the robot. The same can be observed for other grasp types as shown in Figure 4.10, which shows the variation of the predicted feasibility of the *right* grasp type depending on the distance to a nearby obstacle. As this distance increases, the probability of feasibility predicted by the model increases as well. These results show that the predicted probability of feasibility are based on geometric properties of the scene and are not random. Moreover, they demonstrate the ability of **AGFPNet** to capture the influence of the environment on the feasibility of actions and grasp types, and that the proposed 3D scene representation is capable of encoding enough geometric information about the objects and support surfaces in the environment.



(a) $p_{top} = 0.01$ **(b)** $p_{top} = 0.62$ **(c)** $p_{top} = 0.97$

Figure 4.9: Variation of the predicted feasibility of a *top* grasp type depending on the height of an obstacle above the object.



(a) $p_{right} = 0.26$ **(b)** $p_{right} = 0.54$ **(c)** $p_{right} = 0.62$

Figure 4.10: Variation of the predicted feasibility of a *right* grasp type depending on the distance to a nearby obstacle.

Table 4.2: Planning performances with and without using feasibility prediction averaged over 10 trials, using a timeout of 300s.

Problem	Method	Success Rate	Total Planning Time (s)	Infeasible Task Plans	Expanded Nodes
Access	Baseline	0%	> 300	> 204.4 (+/-18.4)	> 6848.0 (+/-832.3)
	AGFPNet	90%	50.1 (+/-49.3)	2.6 (+/-5.6)	2067.2 (+/-2978.9)
Clear	Baseline	30%	190.0 (+/-51.9)	124.7 (+/-20.2)	5873.3 (+/-869.2)
	AGFPNet	100%	70.5 (+/-27.9)	3.9 (+/-6.6)	1278.4 (+/-604.7)
Sort	Baseline	0%	> 300	> 49.0 (+/-5.8)	> 7452.6 (+/-704.9)
	AGFPNet	90%	102.1 (+/-47.3)	2.6 (+/-1.5)	2443.0 (+/-2396.6)
Longswap	Baseline	0%	> 300	> 200.7 (+/-34.2)	> 4674.4 (+/-2590.8)
	AGFPNet	90%	29.3 (+/-8.1)	0.7 (+/-0.7)	321.2 (+/-103.0)
Middleman-3	Baseline	0%	> 300	> 155.8 (+/-18.4)	> 2394.2 (+/-372.0)
	AGFPNet	100%	14.3 (+/-1.4)	0.0 (+/-0.0)	166.2 (+/-31.1)
Middleman-5	Baseline	0%	> 300	> 116.1 (+/-40.1)	> 8649.3 (+/-2339.3)
	AGFPNet	80%	92.2 (+/-29.8)	1.5 (+/-0.9)	1162.5 (+/-577.5)

4.5.3 Planning Performance

Table 4.2 and 4.3 show the planning performances of the proposed feasibility-informed planner on the benchmark TAMP problems compared to the baseline planner described in Chapter 3. The results are averaged over 10 trials, and the standard deviation is shown in parentheses. Results show that our feasibility-informed planner is able to solve all problems with almost perfect success rates, compared to the baseline which fails to solve 5 of the 6 benchmarks. The total planning time is also significantly reduced. On the *Access* problem, our approach improves the success rate from 0% to 90%, averaging a planning time of 50.1s. Thanks to efficient feasibility prediction, the planner is able to greatly reduce the number of infeasible task plans generated from 204.4 to 2.6. This results in much less geometric planning calls, with an average of 16.7 compared to 315.1 without feasibility prediction. This reduction in the number of geometric planner calls allows a 6-times speedup in total planning time.

Similarly, on the dual-robot *Clear* problem, our informed planner improves the success rate from 30% to 100%. Although the neural network makes some misclassifications resulting in an average of 3.9 infeasible task plans generated before a feasible one is found, the feasibility-informed planner is able to reduce the total planning time by at least 62%. **AGFPNet** predicts in advance that *Pass* actions between the two robots are initially infeasible due to the objects blocking the exchange region. This allows the planner to prioritize clearing this region first before attempting accomplishing the task, resulting in a low number of calls to the geometric planner as shown in Table 4.3. On the *Sort* problem, leveraging feasibility predictions increases the success rate from 0% to 90%, with an average planning time of 102.1s. In fact, using **AGFPNet**, the planner is able to reduce the number of infeasible task plans by a factor of 19 compared to the baseline planner. The model allows the planner to avoid infeasible actions, particularly by giving low feasibility probabilities to *Goal* placements that are obstructed by other objects.

Regarding the three-robot *Longswap* problem, the feasibility-informed planner improves the success rate by solving 9 of the 10 trials compared to the baseline planner which fails to solve any. The proposed approach also reduces the total planning time by at least 90%, with an average planning time of 29.3s. Accurate feasibility predictions allow the planner to find a feasible task plan without generating any infeasible ones in most trials, compared to an average of 200.7 infeasible task plans without feasibility prediction. **AGFPNet**'s accurate predictions

Table 4.3: Planning statistics with and without using feasibility prediction, averaged over 10 trials.

Problem	Method	Geometric Planner Calls	Feasibility Checks	Collision Checks	Pruned Branches
Access	Baseline	> 315.1 (+/-28.3)	-	> 12355.7 (+/-1718.9)	> 6068.4 (+/-556.0)
	AGFPNet	16.7 (+/-12.0)	4134.4 (+/-5957.9)	2541.7 (+/-3404.3)	1229.0 (+/-3115.7)
Clear	Baseline	172.3 (+/-27.4)	-	13297.7 (+/-1926.6)	5279.7 (+/-782.7)
	AGFPNet	19.4 (+/-12.2)	2996.0 (+/-1333.2)	3131.1 (+/-1373.0)	420.3 (+/-707.9)
Sort	Baseline	> 52.1 (+/-6.1)	-	> 25393.1 (+/-2909.4)	> 7439.7 (+/-712.4)
	AGFPNet	15.9 (+/-2.3)	4951.0 (+/-4780.7)	6930.4 (+/-6184.6)	756.0 (+/-649.9)
Longswap	Baseline	> 319.3 (+/-39.0)	-	> 9775.0 (+/-5704.2)	> 4402.4 (+/-2622.2)
	AGFPNet	13.9 (+/-1.1)	805.1 (+/-260.4)	760.3 (+/-253.9)	65.9 (+/-85.0)
Middleman-3	Baseline	> 225.2 (+/-26.1)	-	> 4984.3 (+/-814.2)	> 2259.7 (+/-362.7)
	AGFPNet	9.7 (+/-0.8)	437.1 (+/-88.9)	429.2 (+/-78.4)	0.0 (+/-0.0)
Middleman-5	Baseline	> 154.9 (+/-54.6)	-	> 18456.2 (+/-4591.4)	> 8362.9 (+/-2326.1)
	AGFPNet	22.6 (+/-1.7)	3107.6 (+/-1572.1)	2907.4 (+/-1438.8)	402.6 (+/-230.1)

are further highlighted in the $Middleman-3$ problem, which presents a similar scenario to the *Clear* benchmark, except that the objects blocking the intersection region are fixed obstacles. On this problem, the first task plan generated by the feasibility-informed planner is always a geometrically feasible one in all trials, leading to an increase in success rate from 0% to 100% success rate. In contrast, the baseline planner generates an average of 155.8 infeasible task plans before finding a feasible one. Furthermore, the average planning time is at least 21 times lower thanks to the use of **AGFPNet**. This shows that our feasibility-informed planner avoids spending extensive effort on trying to perform a single *Pass* action directly to the goal robot, and prefers the use of a middleman robot with two *Pass* actions for each object.

Results on the $Middleman-5$ problem demonstrate the ability of our approach to generalize to different multi-robot settings, and handle the combinatorial complexity induced by a higher number of robots. Table 4.2 shows that the success rate is increased from 0% to 80% using the proposed feasibility-informed planner. The total planning time is also reduced by at least 69%, with an average planning time of $92.2s$. Despite the higher complexity of the problem, our proposed approach is able to considerably reduce the number of infeasible task plans generated, with an average of 1.5 compared to 116.1 without feasibility prediction. The number of calls to the geometric planner is thus 7 times lower using **AGFPNet**.

Note that for all problems, the number of expanded nodes is significantly reduced when **AGFPNet** is used. This is due to the fact that the planner is able to avoid exploring infeasible branches, and to focus on feasible ones. This is particularly visible on the $Middleman-5$ problem, where the number of expanded nodes is reduced by a factor of 21 compared to the baseline planner. This also translates into considerably less collision checks. Moreover, since the planner is able to avoid infeasible calls to the geometric planner, the number of branches pruned from the search tree is also reduced. The overall reduction in the number of geometric planner calls, thus the gain in success rate and planning time, is made possible thanks to the fast geometric feedback provided by **AGFPNet**. Table 4.3 shows a high number of feasibility checks, which allow the planner to quickly assess the feasibility of actions and grasp types, and prioritize those predicted as feasible. In general, each expanded node requires two to three feasibility checks depending on the action type (i.e, *Goal*, *Temporary* or *Pass*). However, the overhead introduced by these efficient feasibility checks is largely compensated by the reduction in the number of computationally expensive geometric planner calls. This is shown in Figure 4.11, which compares the planning time decomposition between the baseline planner and the proposed

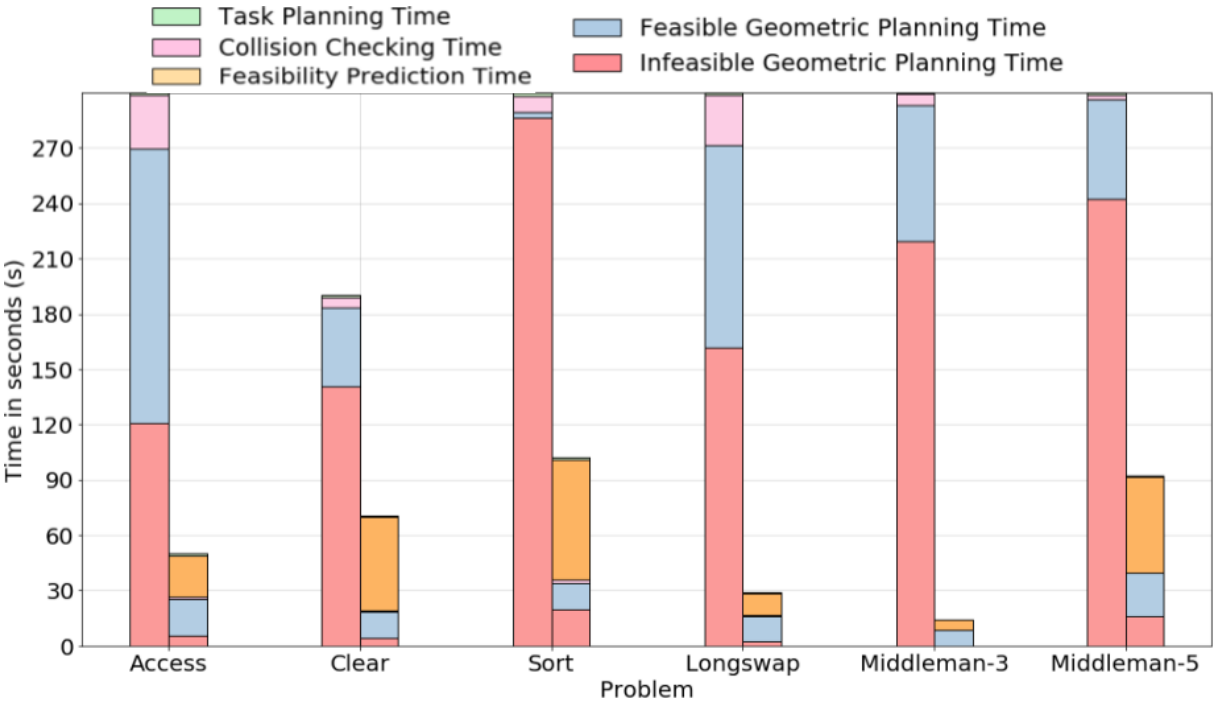


Figure 4.11: Comparison of the planning time decomposition between the baseline planner (left bars) and the proposed feasibility-informed planner (right bars) on each benchmark TAMP problem. The total planning time is broken down into the task planning time (green), feasibility checking time (orange), collision checking time (purple), geometric planning time on feasible actions (blue), and geometric planning time on infeasible actions (red).

feasibility-informed planner on each benchmark TAMP problem. As expected, the time spent on planning motions for infeasible actions is considerably lower when using **AGFPNet**. The time added by the feasibility checks, on the other hand, remains low for most benchmarks. In addition, thanks to the reduction in the number of expanded nodes and collision checks, the collision checking time become negligible. Furthermore, thanks to the use of predicted grasp type feasibility during geometric planning, the time spent on geometric planning on feasible actions is also reduced.

These results also showcase the generalizability of **AGFPNet** to unseen environments, mesh objects, and multi-robot setting. As a reminder, the neural network is trained on a fully synthetic dataset composed of 3D scenes containing a single robot, two movable objects and up to 4 support surfaces only. The environments used in the benchmarks, on the other hand, contain a higher number of fixed and movable objects. They also involve a higher number of robots with different configurations. Moreover, the neural network is trained on box-shaped objects only, while the benchmarks use a mix of box-shaped and mesh objects. Despite this domain shift, the proposed model is able to accurately predict the feasibility of actions and grasp types, and provide efficient feedback to the planner. This demonstrates the robustness of the proposed approach to unseen environments and objects, and its ability to generalize to different multi-robot settings.

4.5.4 Ablation Study

We conduct an ablation study in order to evaluate the benefit of using the collaborative feasibility introduced in Section 4.4.4. Table 4.4 shows a comparison of success rates, total planning time, number of expanded nodes and number of feasibility checks, with and without using collaborative

Table 4.4: Comparison of the planning performances obtained with (+) and without (-) collaborative feasibility (CF).

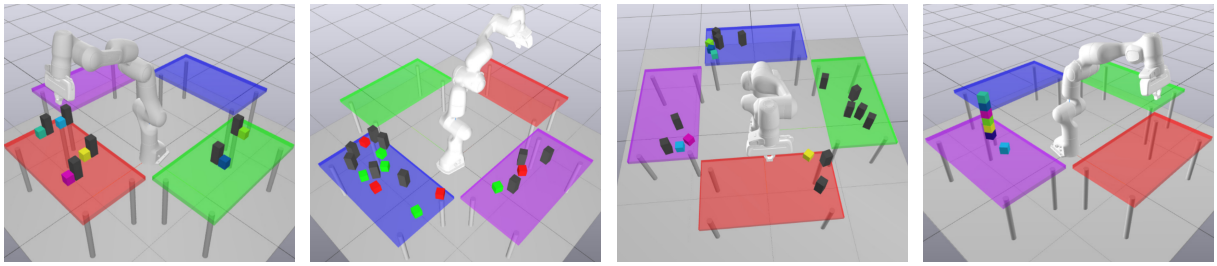
Problem	CF	Success Rate	Planning Time (s)	Expanded Nodes	Feasibility Checks
Clear	-	90%	88.0	3387	6773
	+	**100%**	**70.5**	**1278**	**2996**
Sort	-	80%	112.2	2720	5439
	+	**90%**	**102.1**	**2443**	**4951**
Longswap	-	10%	116.4	5848	11696
	+	**90%**	**29.3**	**321**	**805**
Middleman-3	-	30%	19.0	644	1289
	+	**100%**	**14.3**	**166**	**437**
Middleman-5	-	0%	> 300	> 17121	> 34243
	+	**80%**	**92.2**	**1163**	**3108**

feasibility (CF). Results show a clear performance improvement on all multi-robot problems when collaborative feasibility is used. Indeed, both the *Pick* and the *Place* actions composing a *Pass* action can be feasible. However, the following *Pick* action using the receiving robot might not be. Taking into account the feasibility of this second *Pick* action allows the planner to prioritize *Pass* actions that are more likely to succeed, or other actions that aim at clearing the exchange region if necessary. This translates into less expanded nodes as well as less feasibility checks, which can be observed in Table 4.4 on all multi-robot problems. Particularly, results obtained on the harder *Longswap*, $Middleman-3$ and $Middleman-5$ problems show CF not only reduces the number of expanded nodes and feasibility checks, it also improves success rate and planning time. On the *Longswap* problem, in addition to an 80% improvement in success rate, planning time is 4 times faster. Moreover, the number of expanded nodes is 18 times lower, while the number of feasibility checks is reduced by 93%. Results also show a 70% improvement in success rate for the $Middleman-3$ problem thanks to the use of CF, with 25% faster planning time, 3.9 times less expanded nodes and 3 times less feasibility checks. On the $Middleman-5$ problem, results show that the planner completely fails to solve the problem without collaborative feasibility, compared to a 80% success rate using CF. This demonstrates the necessity of using CF for the more complex problems.

4.5.5 Comparison to Prior Work

In order to compare the performance of our proposed approach to previous works, we test the feasibility-informed planner on the benchmarks proposed by Khodeir et al. 2023a, and compare the results to the **Adaptive** algorithm proposed by Garrett et al. 2020, the **Informed** algorithm introduced by Khodeir et al. 2023a, and the **PLOI** algorithm proposed by Silver et al. 2021a. Garrett et al. 2020 propose a framework that integrates symbolic planners with blackbox samplers (streams) using optimistic adaptive planning, by dynamically balancing the exploration of new task plans and the exploitation of already found ones. Khodeir et al. 2023a build upon this method by leveraging a graph neural network (GNN) to score the relevance of streams during planning in order to guide the search towards promising task plans. **PLOI** (Silver et al. 2021a) is a method that uses a GNN to predict the importance of individual objects in the planning problem, excluding the ones that are predicted to be irrelevant.

The benchmark problems used in this comparison are the same as the ones used in Khodeir et al. 2023a, and are shown in Figure 4.12. These TAMP problems are different from the



(a) Non-Monotonic **(b)** Sorting **(c)** Clutter **(d)** Stacking

Figure 4.12: Visualization of TAMP problems introduced by Khodeir et al. 2023a, and used for comparison with previous works. Image source: Khodeir et al. 2023a.

ones introduced in this thesis, as they present a different set of challenges. In contrast to the benchmarks described in Section 3.4, the problems introduced by Khodeir et al. 2023a are single-robot tabletop scenarios, where movable objects are box-shaped and relatively small in size, with a small set of grasps. Also, the interactions between objects are less complex (e.g., one to two objects only must be moved to access a desired one). However, these problems involve a higher number of objects (up to 20), with many subproblems that must be solved independently. The first problem, *Non-Monotonic* (cf. Figure 4.12a), is similar to the **Access** problem. It involves moving a set of objects to their target location, while a set blockers initially obstruct access to them. The second problem, *Sorting* (cf. Figure 4.12b), involves sorting a set of objects by color in a cluttered environment. The *Clutter* problem, shown in Figure 4.12c, is similar to the latter, but involves more complex goal placements. Finally, the *Stacking* problem, shown in Figure 4.12d, involves stacking a set of objects in a specific order.

We compare our feasibility-informed task and motion planner to the three methods described above following the same evaluation protocol as in Khodeir et al. 2023a. In the latter, two sets of instances are proposed for each problems, one for training and another for testing. Learning-based methods are trained for each problem individually, then evaluated on the corresponding set of test instances. This is different than our experimental setup, where the neural network is trained once on an independent dataset which is unrelated to the TAMP benchmarks used during evaluation. We believe that this setup is more realistic, as it allows a true evaluation of the neural network's generalizability to unseen problems. However, in order to ensure a fair comparison, we finetune **AGFPNet** on a training set composed of the combined training instances of all problems. Finetuning is done over 100 epochs using a learning rate of 10^{-5}. We then run the feasibility-informed planner on the test instances of each problem.

Results are summarized in Table 4.5. Our proposed approach outperforms the three methods on three of the four problems. Thanks to feasibility prediction, our planner is able to improve success rate on the *Non-Monotonic* problem by 30% compared to the **Informed** algorithm,

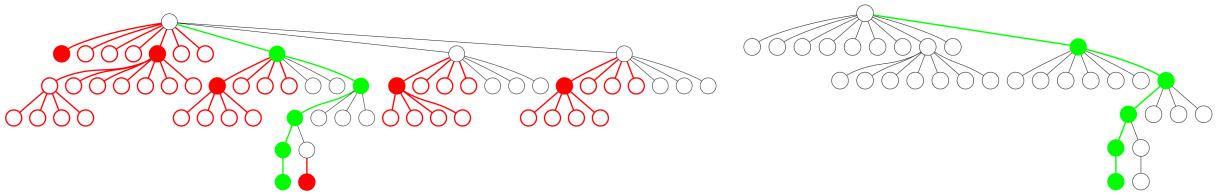
Table 4.5: Comparison of the planning performance with previous works. For the **Adaptive**, **Informed** and **PLOI** algorithms, results are taken from Khodeir et al. 2023a.

Problem	Adaptive		Informed		PLOI		Ours	
	Solved	Time	Solved	Time	Solved	Time	Solved	Time
Non-Monotonic	27	31.45	58	33.88	25	29.71	**88**	24.27
Sorting	66	16.95	**77**	21.07	68	20.47	68	21.11
Clutter	54	30.99	54	12.15	49	16.11	**61**	22.5
Stacking	47	16.95	54	7.94	47	9.20	**84**	4.75

and more than 60% compared to the **Adaptive** and **PLOI** algorithms. The total planning time is also reduced by at least 9s compared to the three methods. On the *Clutter* problem, although our approach has a slightly higher planning time, it achieves a success rate of 61% which is at least 12% higher than prior methods. The proposed feasibility-informed planner also outperforms the three methods on the *Stacking* problem, with an increase in success rate of 30% compared to the **Informed** algorithm, and more than 30% compared to the **Adaptive** and **PLOI** algorithms. This improvement in success rate is accompanied by a reduction in planning time of at least 40%. On the *Sorting* problem, our approach result in a success rate of 68% and a planning time of 21.11s, outperforming the **Adaptive** algorithm and matching the performance of the **PLOI** algorithm, but slight below the performance of the **Informed** algorithm. It is important to note that the three baseline methods are retrained for each problem individually. In contrast, our proposed approach is trained once on a combined dataset, and is able to generalize to all problems. These results demonstrate the effectiveness of our proposed feasibility-informed planner on a different set of TAMP problems, and show that it is able to outperform previous methods in terms of success rate and planning time.

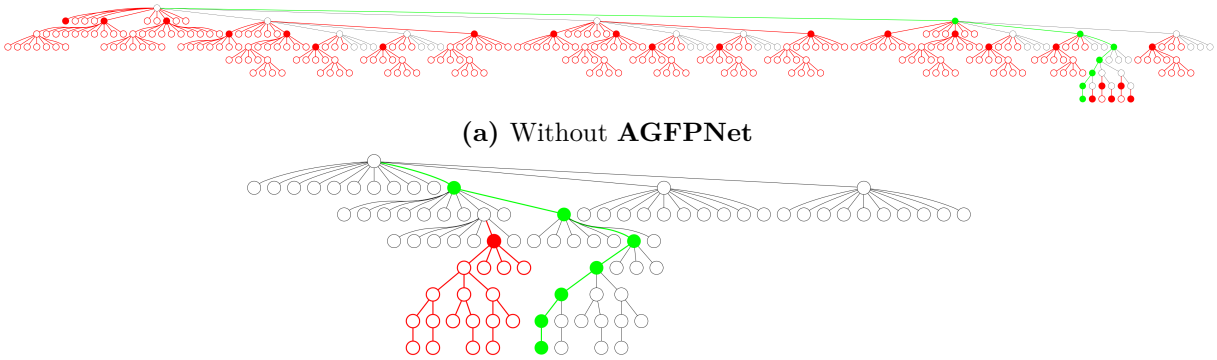
4.5.6 Qualitative Results

Last but not least, we analyze qualitatively the performance gain yielded by the proposed feasibility-informed planner. Figure 4.13 and 4.14 compare the evolution of the search tree on a 3-object and 4-object **Access** problem, with and without using **AGFPNet**. On the 3-object problem, using feasibility prediction not only reduces the size of the search tree, it also allows the planner to find a geometrically feasible task plan in the first trial. In contrast, the baseline planner generates a large number of infeasible task plans before finding a feasible one, which is highlighted by the difference in the search tree size and the number of infeasible nodes (red nodes). These observations are more pronounced on the 4-object problem, where the search tree of the baseline planner is much larger than the one obtained using **AGFPNet**. Indeed, while an additional object in the environment causes an exponential increase in the size of the search tree for the baseline planner, the proposed feasibility-informed planner keeps the search tree size relatively small. This is due to the fact that the planner is able to avoid exploring infeasible branches, and to focus on feasible ones. This reduction is guaranteed even if the neural network makes some misclassifications that lead to an infeasible call to the geometric planner (cf. Figure 4.14b), since the occurrence of such cases is rare. These results demonstrate the effectiveness of the proposed feasibility-informed planner in reducing the search space and the planning time, and in improving the success rate on complex TAMP problems.



(a) Without **AGFPNet** **(b)** With **AGFPNet**

Figure 4.13: Comparison of the evolution of the search tree on a 3-object **Access** problem, with and without using **AGFPNet**. Black nodes represent nodes that have been expanded, filled red nodes/edges represent the actions for which geometric planning failed, while non-filled ones represent node/edges that have been pruned, and green nodes/edges represent nodes/edges that are part of the solution.



(a) Without **AGFPNet**

(b) With **AGFPNet**

Figure 4.14: Comparison of the evolution of the search tree on a 4-object **Access** problem, with and without using **AGFPNet**. Black nodes represent nodes that have been expanded, filled red nodes/edges represent the actions for which geometric planning failed, while non-filled ones represent node/edges that have been pruned, and green nodes/edges represent nodes/edges that are part of the solution.

4.6 Discussion and Limitations

In this chapter, we have presented a method for predicting action and grasp type feasibility in TAMP problems, without the need for computationally expensive geometric planning. By representing the 3D environment using multi-view depth images, we are able to encode geometric information for accurate action and grasp feasibility prediction. We have shown that the proposed model, **AGFPNet**, achieves high accuracy and is able to generalize to unseen environments, objects shapes as well as multi-robot settings, while being trained on a fully synthetic dataset. Moreover, we proposed a feasibility-informed task and motion planner that leverages feasibility predictions to accelerate the search for a geometrically feasible task plan. By incorporating the predicted feasibility into the cost of nodes, the planner is able to avoid exploring infeasible branches and prioritize promising actions. We also introduced a collaborative feasibility mechanism that allows the planner to estimate the feasibility of actions involving two robots. Results show that the proposed feasibility-informed planner is able to significantly improve the success rate and reduce the planning time on the benchmark TAMP problems. Thanks to the use of **AGFPNet**, the planner is able to reduce the number of infeasible task plans generated, hence the time spent on geometric planning while adding a relatively low overhead in terms of feasibility checks. The proposed approach also outperforms previous methods on multiple TAMP problems, demonstrating its effectiveness in reducing the search space and planning time.

However, the method described in this chapter still presents some limitations. First, although **AGFPNet**'s multi-view input images reduces occlusions compared to the single image scene representation used in prior methods (Driess et al. 2020a,b), occlusions can still occur as the number of objects increases. This can lead to misclassification of actions and grasp types, and can result in infeasible plans. Therefore, we need a more sophisticated scene representation method and a more complex neural network architecture to handle more complex TAMP problems. Furthermore, the proposed model does not provide insights into the reasons behind the infeasibility of an action, which limits the planner's ability to prioritize actions that affect the feasibility of others. More specifically, even with the use of feasibility prediction, the cost-to-goal estimation remains inaccurate, tending to largely underestimate the true number of actions necessary to reach the goal. This results in the expansion of more nodes than necessary, thus larger overheads in planning time due to the high number of feasibility checks. In order to

address this issue, the cost-to-go estimation should consider reasons of infeasibility in order to prioritize actions that affect the feasibility of others, such as freeing up access to a desired object, clearing a preoccupied goal placement, or clearing the exchange region between robots. This would allow to further reduce the number of expanded nodes and feasibility checks, and to improve the overall planning time. In the following chapters, we propose a novel method that addresses these limitations by leveraging a more expressive scene representation and a graph neural network (GNN) to predict the feasibility of actions and grasp types, but also the reason of infeasibility. We show that this method is able to provide more accurate feedback to the planner, and to further reduce the planning time on complex TAMP problems.

Chapter 5

Learning Geometric Reasoning for Manipulation Planning

Contents

5.1 Introduction

In the previous chapter, in accordance with prior work (Wells et al. 2019; Driess et al. 2020b; Khodeir et al. 2023a), we have shown that learning methods can help accelerate Task and Motion Planning (TAMP) by providing fast geometric feedback to the task planner. In particular, action and grasp feasibility prediction offers an efficient alternative to geometric planning during the TAMP process. In offline manipulation planning, it can answer critical questions such as which object can be picked, how to grasp it and where to place it. However, action feasibility prediction presents several challenges. A suitable representation of 3D environments is needed, as these may contain an arbitrary number of objects, along with action representations that capture varying parameters such as grasps and placements. Furthermore, predictions must not only be fast and accurate but also interpretable to understand why an action is infeasible and how to rectify it (e.g. another object is blocking access to the desired one). Finally, action feasibility prediction must generalize to environments with numerous objects, objects of varying shapes, and multi-robot settings.

Existing approaches to action feasibility prediction often struggle with interpretability, scalability, and generalization across diverse environments. The previously presented **AGFPNet** suffers from occlusions as the number of objects in the environment increases, which can significantly impact performance. Moreover, it does not provide insights into why a specific action fails, making it challenging to improve planning efficiency on large and complex TAMP problems. To address these limitations, we propose a novel approach that leverages a GNN-based model for robot action and grasp feasibility prediction. Our method constructs a graph representation of 3D environments, where fixed and movable objects are represented as nodes, and edges capture spatial relationships and interaction constraints. Through this graph-based structure, we leverage an Edge-Enhanced Graph Attention Network (EGAT) to predict action and grasp feasibility for each movable object. A unique aspect of our approach is the introduction of two interpretability mechanisms: inverse kinematics (IK) feasibility predictions, which determine whether the robot can feasibly manipulate an object from different sides, and Grasp Obstruction (GO) predictions, which quantify how neighboring objects restrict access to grasps from different sides of the object. These interpretable features not only allow us to predict action infeasibility more accurately, they also explain why a specific action fails, enabling more efficient planning. We demonstrate that our approach outperforms state-of-the-art methods in terms of accuracy, interpretability, scalability, and generalization across diverse environments and robotic platforms, while providing additional valuable insights into the causes of infeasibility.

5.2 Problem Description

We address the problem of action and grasp feasibility prediction for offline manipulation planning in 3D environments. The goal is to determine whether a robot can successfully plan *Pick* or *Place* actions, and which grasps allow their collision-free motions, while accounting for inverse kinematics constraints and potential grasp obstructions. Our approach is tailored for offline manipulation planning tasks in 3D environments, as described in Chapter 3. Similarly to **AGFPNet**, we define $\mathcal{G} = \{Top, Front, Rear, Right, Left\}$ as a set of 5 grasp types, each one representing a subspace of grasps related to the side from which the object is grasped.

We aim to reduce the dependency on the complex geometric planning process involved in offline manipulation planning. Given an environment E and an object of interest $\mathcal{O} \in E$, the goal is to predict both the feasibility $p_F \in \mathbb{R}$ of picking or placing $\mathcal{O}$ at its pose in E, and the feasibility of each grasp type, $\boldsymbol{p_G} = [p_\gamma, \forall \gamma \in \mathcal{G}] \in \mathbb{R}^5$. Moreover, we aim to estimate the cause of infeasibility

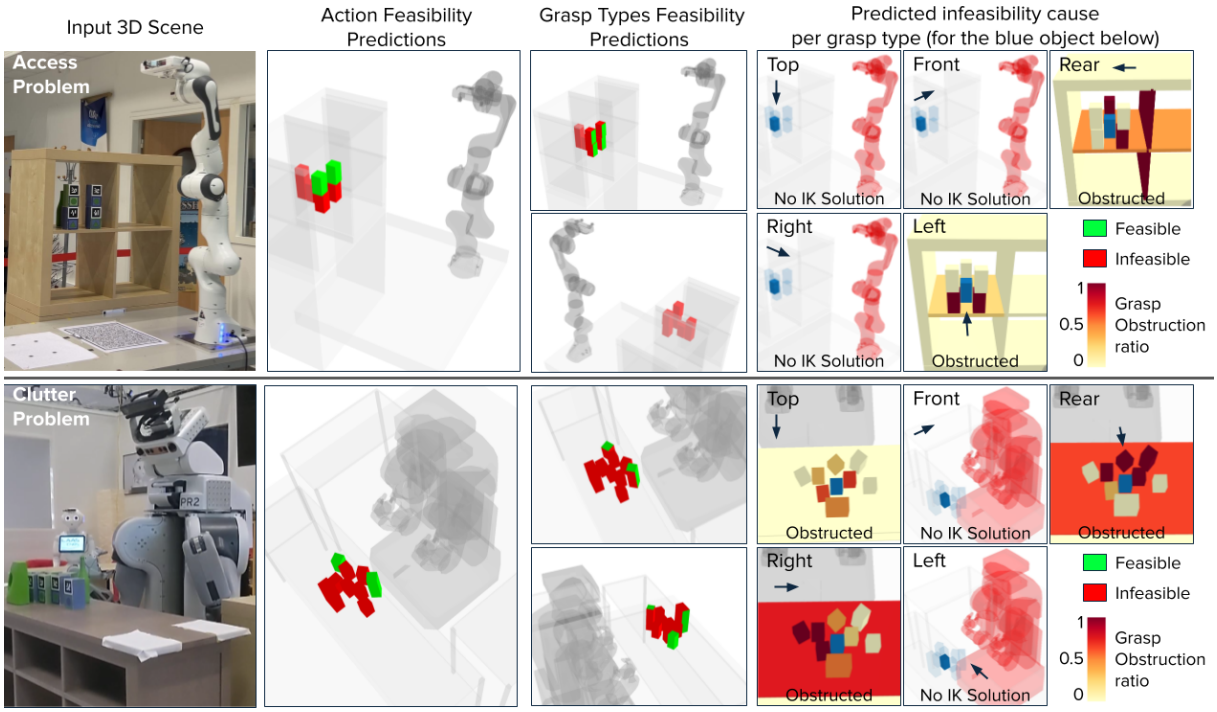


Figure 5.1: Visualization of **GRN** predictions on two manipulation problems, **Access-monotonic** (Panda arm) and **Clutter** (PR2 Robot, predictions shown for its right arm). A single query to **GRN** outputs 3 predictions for each movable object in the environment: Action feasibility, grasp types feasibility (two views), and the predicted infeasibility causes for each grasp type. For clarity, we show the predicted infeasibility cause for one object only (shown in blue), and distinguish two cases: (1) *No IK solution*: robot shown in red, (2) *Grasp type obstructed*: we show all obstructing objects in a color gradient representing the obstructions ratio. Arrows show approach directions of grasp types.

for each grasp type. We focus on two primary factors contributing to infeasibility:

1. **Inverse Kinematics (IK) Feasibility:** The absence of a valid inverse kinematics solution for all grasps within a grasp type $\gamma \in \mathcal{G}$, denoted $\boldsymbol{\kappa_{\mathcal{G}}}$, such that:

$$\boldsymbol{\kappa_{\mathcal{G}}} = [\kappa_\gamma(\mathcal{O}, E), \forall \gamma \in \mathcal{G}] \in \mathbb{R}^5 \tag{5.1}$$

2. **Grasp Obstruction (GO):** The obstruction of grasps by each neighboring object, represented as the ratio $\boldsymbol{\rho_{\mathcal{G}}}$ of obstructed grasps:

$$\boldsymbol{\rho_{\mathcal{G}}} = [\rho_\gamma(\mathcal{O}, \mathcal{O}'), \quad \forall \gamma \in \mathcal{G}, \quad \forall \mathcal{O}' \in \mathcal{N}(\mathcal{O})] \quad \in \mathbb{R}^{5 \times |\mathcal{N}(O)|} \tag{5.2}$$

   where $\mathcal{N}(\mathcal{O})$ denotes the distance-based neighborhood of $\mathcal{O}$ and $|\mathcal{N}(O)|$ is its cardinality.

In addition to helping explain infeasibility and providing insights into the constraints imposed by the environment, these predictions are also used in the feasibility prediction process. In summary, the task at hand is to learn two classification functions f_F, f_κ, and a regression function f_ρ s.t.:

$$\begin{bmatrix} F_a \\ \boldsymbol{F}_{\mathcal{G}} \end{bmatrix} = f_F(\mathcal{O}, E, \boldsymbol{\kappa_{\mathcal{G}}}, \boldsymbol{\rho_{\mathcal{G}}}) \qquad \text{where} \qquad \boldsymbol{\kappa_{\mathcal{G}}} = f_\kappa(O, E) \quad \text{and} \quad \boldsymbol{\rho_{\mathcal{G}}} = f_\rho(O, E) \tag{5.3}$$

5.3 Geometric Reasoning Networks

We propose **Geometric Reasoning Networks (GRN)**, a three-module GNN-based neural network which takes as input a graph representation of the environment, and outputs the action and grasp types feasibility for each movable object, as well as inverse kinematics feasibility (IK) and grasp obstruction (GO) predictions (cf. Figure 5.1). **GRN** is designed to provide interpretable insights into the causes of infeasibility, enabling more efficient task planning. In this section, we present the architecture of our proposed model, the graph representation of 3D environments, and the training strategy used to learn action and grasp feasibility predictions.

5.3.1 Graph Representation of 3D Scenes

The main challenge of learning methods in a manipulation planning context is finding an appropriate representation of the 3D environment which can contain an arbitrary number of objects. Previously proposed image-based representations suffer from occlusions as the number of objects in the environment increases, which can significantly impact performance. In this chapter, we tackle this issue by representing 3D scenes as graphs where nodes represent fixed and movable objects and edges represent geometric relationships between objects as shown in Figure 5.2. Given an environment E, we construct a directed graph $(\mathcal{V}, \mathcal{E})$ where each node corresponds to an object in E, fixed or movable. Nodes have a feature vector $\boldsymbol{x} = [l, w, h, x, y, z, \theta]^T$ where (l, w, h) are the length, width and height of the object's bounding box, (x, y, z) represent the position of the object in the environment, and θ is the object's orientation w.r.t its z axis. The position and orientation of the object are expressed in the reference frame of the base of the robot. This allows a straightforward generalization to multi-robot settings simply by switching the reference frame to the base of the robot of interest.

For each node u corresponding to a movable object $\mathcal{O}$, we add a self-loop edge $(u \to u) \in \mathcal{E}$, as well as a directed edge $(v \to u) \in \mathcal{E}$ from each node v corresponding to a neighboring (fixed or movable) object $\mathcal{O}' \in \mathcal{N}(\mathcal{O})$. Two objects are considered neighbors if the euclidean distance between their closest points along the (x, y) plane is lower than a user-defined threshold. Since

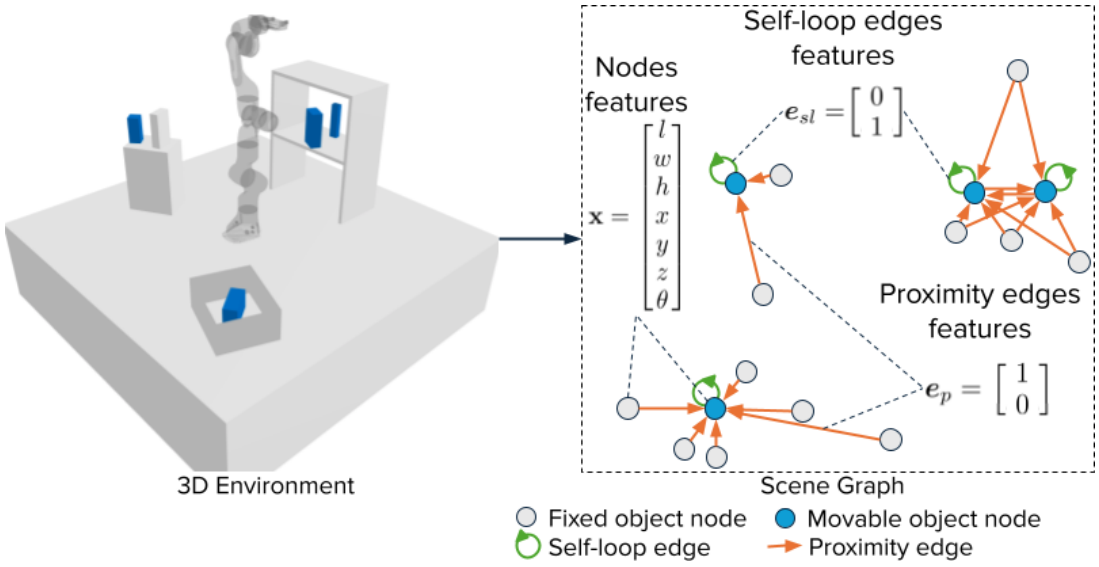


Figure 5.2: The proposed scene graph representation. Blue nodes represent movable objects, while gray nodes represent fixed objects. Self-loop edges are shown in green and proximity edges in orange.

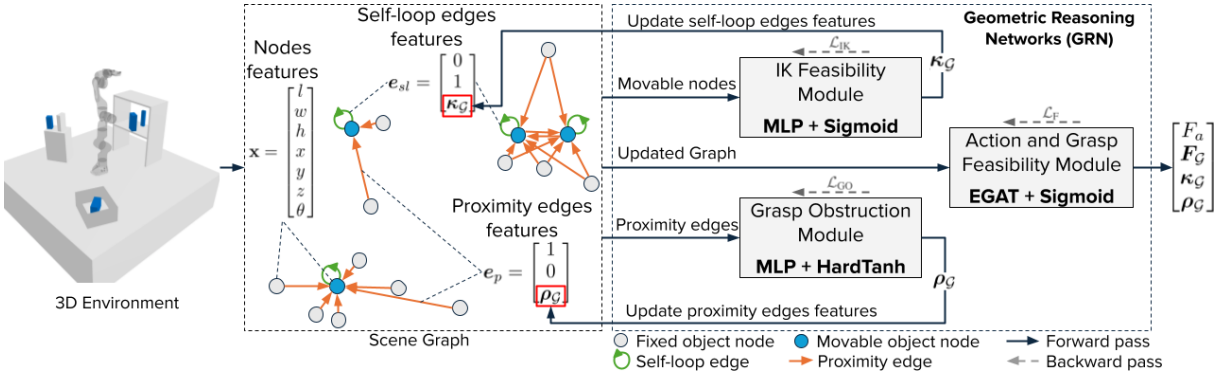


Figure 5.3: Complete GRN architecture. A scene graph is constructed from the input 3D environment. Node features of movable objects are given to IK feasibility module which outputs are used to update self-loop edge features. The concatenated features of nodes linked through a proximity edge are fed to the GO module to get grasp obstruction estimations, which are appended to proximity edge features. Finally, the updated graph is given to the AGF module to predict the action and grasp types feasibility for each movable object in the environment.

our approach is GNN-based, where edges are used to determine which features to aggregate, the more edges are present in the input graph, the higher the number of computations is. Hence, the permissiveness of this threshold creates a tradeoff between accuracy and inference speed. Finally, self-loop edges and proximity edges are differentiated using a one-hot encoded feature vector such that $\boldsymbol{e}_{sl} = [0,1]^T$ and $\boldsymbol{e}_p = [1,0]^T$, where $\boldsymbol{e}_{sl}$ and $\boldsymbol{e}_p$ represent the feature vectors for a self-loop edge and a proximity edge respectively. The obtained graph is given as input to **GRN**, shown in Figure 5.3, which is comprised of three modules.

5.3.2 Inverse Kinematics Feasibility Prediction

The first module is the Inverse Kinematics (IK) feasibility prediction module. This submodel is a binary classifier, composed of a 4-layer Multi-Layer Perceptron (MLP) with ReLU activation functions, followed by a Sigmoid activation function. It takes as input the feature vector $\boldsymbol{x}_u$ of each node $u \in \mathcal{V}$ corresponding to a movable object, and simultaneously outputs the predicted inverse kinematics feasibility for each of the five grasp types in $\mathcal{G}$:

$$\boldsymbol{\kappa_{\mathcal{G}}}(u) = \text{Sigmoid}(\mathbf{MLP}_{\text{IK}}(\boldsymbol{x}_u)) \quad \in [0,1]^5 \tag{5.4}$$

Each prediction κ_γ where $\gamma \in \mathcal{G}$ corresponds to the presence of a valid inverse kinematics solution for at least one grasp of type γ. This can be viewed as predicting whether the robot can reach a specific side of the object at its pose in the environment, without taking into account any other object. Since this information is valuable for action and grasp types feasibility prediction, we incorporate it to the previously constructed graph by concatenating the obtained predictions $\boldsymbol{\kappa_{\mathcal{G}}}$ to the features of self-loop edges resulting in $\boldsymbol{e}^+_{sl} = [\boldsymbol{e}_{sl} \parallel \boldsymbol{\kappa_{\mathcal{G}}}] \in \mathbb{R}^7$, where $\parallel$ denotes the concatenation operator.

5.3.3 Grasp Obstruction Estimation

The second stage is Grasp Obstruction (GO) estimation, which is a regression module consisting of 4-layer MLP with ReLu activation functions, followed by a HardTanh activation function which bounds the output between 0 and 1. It takes as input the concatenated features of each pair of nodes $(v \to u)$ linked through a proximity edge. For each grasp type $\gamma \in \mathcal{G}$, it outputs

the estimated ratio of grasps of u that are obstructed by the object corresponding to v, which is defined as the number of obstructed grasps divided by the total number of grasps of a specific type:

$$\boldsymbol{\rho}_{\boldsymbol{\mathcal{G}}}(u, v) = \text{HardTanh}(\mathbf{MLP}_{\text{GO}}([\boldsymbol{x}_u \parallel \boldsymbol{x}_v])) \quad \in [0, 1]^5 \tag{5.5}$$

This submodel not only predicts whether an object obstructs grasps of another, it also gives insight into how much it blocks access to it. In a manipulation planning context, this allows to rank obstructing objects and plan accordingly. Similarly to IK feasibility predictions, this information impacts directly the feasibility of actions and grasp types. Thus, we concatenate the grasp obstruction estimations $\boldsymbol{\rho}_{\boldsymbol{\mathcal{G}}}$ to the edge features of proximity edges to obtain $\boldsymbol{e}_p^+ = [\boldsymbol{e}_p \parallel \boldsymbol{\rho}_{\boldsymbol{\mathcal{G}}}] \in \mathbb{R}^7$.

5.3.4 Action and Grasp Feasibility Prediction

Once our graph is constructed and enriched through IK feasibility and GO predictions, it is given to the action and grasp feasibility (AGF) prediction module. As mentioned previously, IK feasibility and grasp obstructions, which constitute our edge features, have a direct impact on action and grasp type feasibility. However, the classic graph attention network (GAT) (Veličković et al. 2018) uses edge features simply for computing attention weights, and does not take them into account during the aggregation process. This causes the model to lose valuable information when applied to tasks where edge features are as important as node features. In this work, we propose to use the Edge-Featured Graph Attention Network (EGAT) (Ziming Wang et al. 2021) which, in addition to using edge features to compute attention, leverages both node and edge features during the aggregation process.

The third submodel first computes embeddings $\boldsymbol{h}_u$ and $\boldsymbol{e}^*_{\boldsymbol{uv}}$ of node and edge features using two fully-connected layers such that $\boldsymbol{h}_u = \boldsymbol{W}_{\boldsymbol{h}}.\boldsymbol{x}_u$ and $\boldsymbol{e}^*_{\boldsymbol{uv}} = \boldsymbol{W}_{\boldsymbol{e}}.\boldsymbol{e}^+_{uv}$, where $\boldsymbol{W}_{\boldsymbol{h}}$, $\boldsymbol{W}_{\boldsymbol{e}}$ are weight matrices. Then, EGAT computes the multi-head attention of each edge as:

$$\alpha_{uv} = \frac{\exp(\boldsymbol{a}^T \text{LeakyReLU}([\boldsymbol{h}_u \parallel \boldsymbol{h}_v \parallel \boldsymbol{e}^*_{uv}]))}{\sum_{k \in \mathcal{N}(u) \cup \{u\}} \exp(\boldsymbol{a}^T \text{LeakyReLU}([\boldsymbol{h}_u \parallel \boldsymbol{h}_k \parallel \boldsymbol{e}^*_{uk}]))} \tag{5.6}$$

Note that this formulation is slightly different from the one proposed by Ziming Wang et al. 2021. We adapt the attention computation introduced by Brody et al. 2022, which fixes the static attention problem of the standard GAT. Once the multi-attention weights computed, our model computes a weighted average of the concatenated node and edge embeddings, followed by a LeakyReLU activation:

$$\boldsymbol{h}'_u = \text{LeakyReLU}\left(\sum_{v \in \mathcal{N}(u) \cup \{u\}} \alpha_{uv} [\boldsymbol{h}_v \parallel \boldsymbol{e}^*_{uv}]\right) \tag{5.7}$$

Finally, the obtained vector is passed to a 2-layer MLP followed by a sigmoid activation function that outputs the action and grasp types feasibility predictions for movable object node u:

$$\begin{bmatrix} F_a(u) \\ \boldsymbol{F}_{\mathcal{G}}(u) \end{bmatrix} = \text{Sigmoid}(\mathbf{MLP}_F(\boldsymbol{h}'_u)) \tag{5.8}$$

5.3.5 Training Strategy

Our proposed **GRN** model is trained in a supervised manner by first pre-training each module separately. The IK feasibility prediction submodel is trained using a binary cross entropy loss denoted $\mathcal{L}_{\text{IK}}$ while the GO estimation module is trained using the mean square error loss $\mathcal{L}_{\text{GO}}$.

Regarding the AGF prediction module, it is first pre-trained using a binary cross entropy loss $\mathcal{L}_{\text{F}}$, using the ground truth of IK feasibility and grasp obstructions as edge features. The complete **GRN** model is then fine-tuned using a weighted sum of the previously defined losses such that:

$$\mathcal{L} = \mathcal{L}_{\text{F}} + \mathcal{L}_{\text{IK}} + \eta.\mathcal{L}_{\text{GO}} \tag{5.9}$$

where η is a weighting factor allowing a balanced order of magnitude across classification and regression. This training strategy is inspired by the one proposed by D. Chen et al. 2020.

During training, our 3D scene representation allows two data augmentation methods: dimensions switch and rotation. They both take advantage of the symmetry of bounding boxes. Indeed, switching the length and width of a bounding box, then applying a $\frac{\pi}{2}$ rotation around the z axis, keeps the geometric properties of the environment unchanged, resulting in a new node feature vector $\boldsymbol{x} = [w, l, h, x, y, z, \theta \pm \frac{\pi}{2}]^T$. The same goes for applying a π rotation without any dimensions switch such that $\boldsymbol{x} = [l, w, h, x, y, z, \theta \pm \pi]^T$. These can be applied to fixed objects as well as movable objects. For the latter case, the labels associated with grasp types need to be interchanged, since switching the dimensions and/or rotating the object changes which side is considered as the front for example.

5.3.6 Data Generation and Annotation Method

Data Generation. We propose an improved data generation process that builds upon the one detailed in Section 4.4.5. It consists of generating a number of 3D environments containing a random number of support surfaces, movable objects and obstacles within a specified range. We define 4 types of structures as shown in Figure 5.4: (1) a rack which is a structure containing a varying number of shelves separated by a gap, and a holder on each side or each corner, (2) a bar which is a L-shaped structure with one support surface and one holder, (3) a basket which consists of a support surface and 4 sides surrounding it, and (4) a counter which is a large block. When generating an environment, we first randomly sample a number of structures within the specified range. We then choose one of the predefined structures, before randomly sampling its dimensions and pose. After collision checking using the FCL library (J. Pan et al. 2012), all objects composing the structure are added to the environment as fixed objects. These can be either support surfaces or obstacles (e.g holders). This process is repeated until the number of structures sampled is reached.

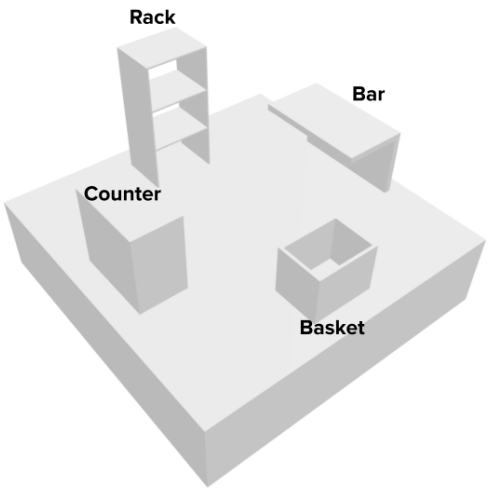


Figure 5.4: Visualization of the different types of structures used during data generation.

Given the fact that our approach represents objects using their bounding boxes, it is sufficient to generate a dataset consisting of box-shaped objects only. After sampling a number of movable objects to add, we randomly sample the dimensions of the object with a specified range. In order to incorporate difficulties encountered in real-world scenarios, we define three methods for object placement sampling: (1) the first is random placement in which a support surface and a pose within its bounds are randomly sampled, (2) proximity placement in which the object is placed in the neighborhood of a randomly chosen object, (3) the third is the underneath placement in which the object is placed underneath a randomly chosen support surface. We check the sampled object placement for collisions before adding it to the environment. We then repeat the process until the sampled number of movable objects is reached. Finally, a number of fixed box-shaped obstacles is randomly added to the scene, before storing it.

Data Annotation. Data annotation can be done using any off-the-shelf geometric planner. In this work, we use an adapted version of Moveit Task Constructor (Görner et al. 2019), with the KDL plugin for IK computation, FCL (J. Pan et al. 2012) for collision checking and, Bidirectional Transition-Based Rapid Random Tree (Devaurs et al. 2015) for motion planning. Since pick and

Algorithm 15 AnnotateData (Updated Alg. 10)

Input: $E, \mathcal{R}, \mathcal{O}, K, \tau_{local}, \tau_{global}$ ▷ Environment, robot, object, max. IK solutions, local and global timeouts
$p_F \leftarrow 0$, $\boldsymbol{p_G} \leftarrow \mathbf{0}$, $\boldsymbol{\kappa_G} \leftarrow \mathbf{0}$, $\boldsymbol{\rho_G} \leftarrow \mathbf{0}$
$\boldsymbol{n_G} \leftarrow 0$ ▷ Number of grasps per grasp type
while τ_{global} is not reached **do**
  $G \leftarrow sampleAllGrasps(\mathcal{O})$
  $Q \leftarrow \emptyset$
  for each grasp $g \in G$ **do**
    $\gamma \leftarrow getGraspType(g)$
    $n_G(\gamma) \leftarrow n_G(\gamma) + 1$
    $\mathcal{Q}_g \leftarrow solveIK(E, \mathcal{R}, \mathcal{O}, g, K)$
    if $\mathcal{Q}_g$ is empty **then**
      continue
    end if
    $\boldsymbol{\kappa_G}(\gamma) \leftarrow \boldsymbol{\kappa_G}(\gamma) + 1$
    for each $\boldsymbol{q} \in \mathcal{Q}_g$ **do**
      $collision, obstructors \leftarrow checkCollision(\mathcal{R}, \boldsymbol{q}, E)$
      if *collision* is True **then**
        for $\mathcal{O}' \in obstructors$ **do**
          $\boldsymbol{\rho_G}(\gamma, \mathcal{O}') \leftarrow \boldsymbol{\rho_G}(\gamma, \mathcal{O}') + 1$
        end for
        $\boldsymbol{p_G}(\gamma) \leftarrow 0$
      else
        $\boldsymbol{p_G}(\gamma) \leftarrow 1$
        $Q \leftarrow Q \cup \boldsymbol{q}$
      end if
    end for
  end for
  for each $\gamma \in \mathcal{G}$ **do**
    if $\boldsymbol{\kappa_G}(\gamma) > 0$ **then**
      $\boldsymbol{\kappa_G}(\gamma) \leftarrow 1$
    end if
    for each obstructing object $\mathcal{O}'$ **do**
      $\boldsymbol{\rho_G}(\gamma, \mathcal{O}') \leftarrow \boldsymbol{\rho_G}(\gamma, \mathcal{O}') / n_G(\gamma)$
    end for
  end for
  for each $\boldsymbol{q} \in Q$ **do**
    $m \leftarrow planMotion(E, \mathcal{R}, \boldsymbol{q}, \tau_{local})$
    if m is feasible **then**
      $p_F \leftarrow 1$
      break
    end if
  end for
end while
return $p_F, p_G, \boldsymbol{\kappa_G}, \boldsymbol{\rho_G}$

place actions are symmetrical, they can be considered as equivalent. Thus, we only annotate pick actions. Also, we do not focus on trivial infeasibility cases due to the object being larger than the gripper. Hence, during annotation, we allow collisions between the object to pick and the gripper fingers. This also allows the handling of mesh objects such as a mug or a wine glass, for which feasible grasps exist even if the bounding box is larger than the gripper's maximum width.

As detailed in Algorithm 15, for each movable object in a generated environment, we query the geometric planner to plan a pick action from its placement in the environment. First, we uniformly sample a number of grasps depending on the size of the object. For each sampled grasp, we compute up to 8 IK solution, which are then checked for collisions. Finally, the motion planner is queried for each collision-free IK solution until a feasible collision-free trajectory is found. If a solution is found, the action is annotated as feasible. In parallel, we set the feasibility label of each grasp type $\gamma \in \mathcal{G}$ to 1 if at least one grasp in γ is feasible. Similarly, we set the IK feasibility label of a grasp type to 1 if at least one grasp belonging to it has a valid inverse kinematics solution. Finally, we record all grasp-obstructing objects as well as the ratio of grasps obstructed per grasp type.

5.4 Experiments

We conduct a series of experiments in order to evaluate the performance of our proposed method compared to existing approaches, and showcase the generalization capabilities of our approach. We conduct our experiments using mainly the Franka Emika Panda, which is a 7 degrees-of-freedom (DOF) robotic arm with a parallel jaw gripper. Since our model is robot-centric, meaning that it is specific to the robot it is trained for, we showcase the applicability of our method to other robotic manipulators by running experiments on the Willow Garage PR2 robot as well. The latter is a dual-arm robot where each arm has 7 DOFs with a parallel jaw gripper, and a telescoping spine.

5.4.1 Datasets

Following the method described in Section 5.3.6, we generate a number of datasets to train and evaluate our method. Each one is characterized by a number of movable objects as well as a minimum and maximum numbers of structures (e.g, rack, counter, basket) and obstacles. They are also characterized by the robot used during data annotation. Figure 5.5 shows environments from different datasets.

Panda-3D-4: This is dataset is composed of 3D environments containing 4 movable objects, 1 to 4 structures and 0 to 4 obstacles and is annotated using a Panda robot. It is defined into 3 sets, a training set containing 70'000 scenes, a validation set of 10'000 scenes, and a test set of 20'000 scenes. Each set is generated using a different random seed to ensure that the environments are different across all three sets.

Panda-Tabletop-4: In order to conduct a fair comparison to tabletop methods (Wells et al. 2019; Driess et al. 2020b), we generate a dataset consisting of tabletop environments with 0 structures, 4 movable objects and up to 4 obstacles, all placed on the same support surface as the robot's base. It is composed of 25'000 training scene, 5'000 validation scenes and 10'000 test scenes.

PR2-3D-4: We generate this dataset to showcase the applicability of our approach to other robotic arms as well as multi-robot settings. It is generated using the same parameters as the **Panda-3D** dataset, and annotated for the right arm of PR2 robot. As explained is Section 5.3.1,

our model can be applied to different arms of the same type simply by expressing the objects' poses in the frame of reference of the considered arm. We consider that the base of robot is fixed to the ground, and that the telescopic spine is one of the DOFs of the arms. The dataset contains 25'000 scenes for training, 5'000 scenes for validation and 10'000 scenes for testing.

In order to quantitatively measure the generalizability of our approach to environments containing a higher number of objects than training environments, we generate three additional test sets denoted **Panda-3D-10**, **Panda-3D-15**, **Panda-3D-20**, each composed of 1'000 environments containing 10, 15 and 20 movable objects respectively. These environments also contain a higher number of fixed objects, with a number of structures ranging from 4 to 8, as well as 2 to 4 obstacles. Moreover, we increase the dimensions' range used during data generation resulting in larger objects.

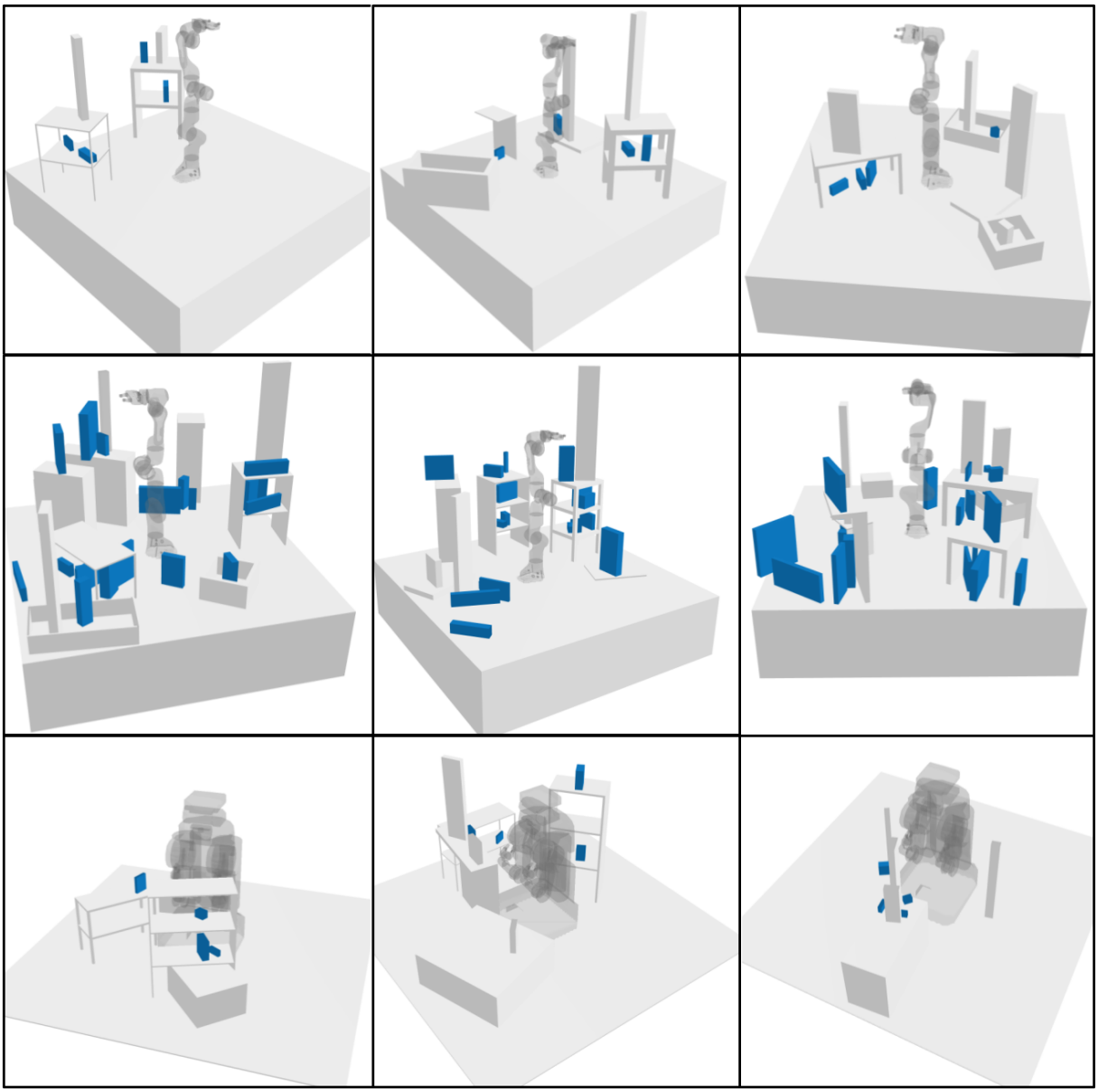
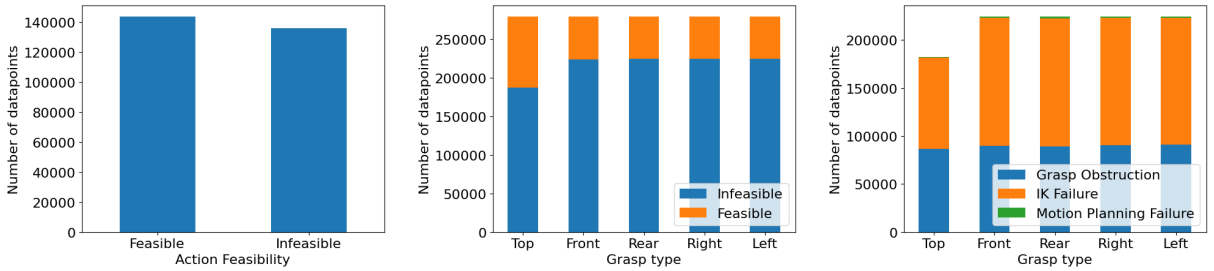


Figure 5.5: Visualization of generated environments from the (top) **Panda-3D-4**, (middle) **Panda-3D-20**, and (bottom) **PR2-3D-4** test sets

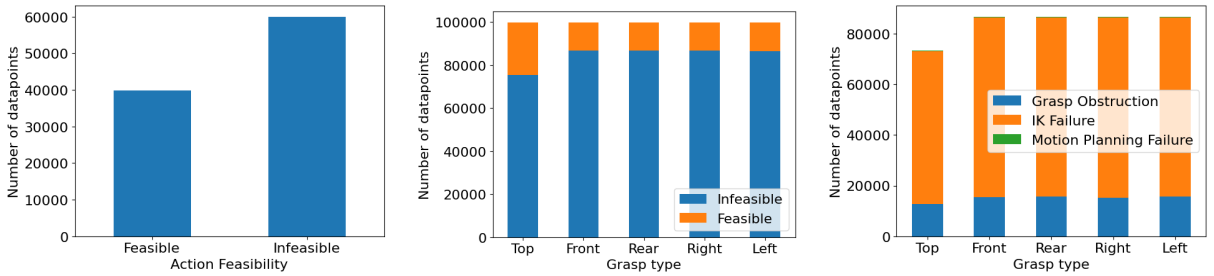


(a) Number of feasible and infeasible actions

(b) Number of feasible and infeasible cases per grasp type

(c) Distribution of failure causes per grasp type

Figure 5.6: Annotations statistics for the **Panda-3D-4** training set.



(a) Number of feasible and infeasible actions

(b) Number of feasible and infeasible cases per grasp type

(c) Distribution of failure causes per grasp type

Figure 5.7: Annotations statistics for the **PR2-3D-4** training set.

Figures 5.6 and 5.7 show the distributions of obtained labels for the **Panda-3D-4** and **PR2-3D-4** training sets. For the former, the action feasibility annotations are balanced, while grasp types feasibility is imbalanced with more infeasible cases. Regarding reasons of infeasibility, Figure 5.6c shows a balanced distribution between failures due to IK feasibility and grasp obstructions. Cases where infeasibility is due to motion planning failure, on the other hand, appear rarely in the dataset. The **PR2-3D-4** dataset has a more pronounced imbalance towards infeasibility, the latter being caused more often by the absence of inverse kinematics solutions than grasp obstructions, showcasing the more complex kinematics of the PR2 robot.

5.4.2 Implementation Details

Scene Graph Construction. Scene graphs are built using the graph representation from Pytorch Geometric (Fey and Lenssen 2019). Nodes correspond to fixed and movable objects from the environment. Self-edge loops are added for nodes corresponding to movable objects. When adding proximity edges, since evaluating the distance between two objects closest points can be computationally expensive, we consider two objects $\mathcal{O}$ and $\mathcal{O}'$ as neighbors if the euclidean distance between their centers is lower than a threshold $\epsilon = r_{\mathcal{O}} + r_{\mathcal{O}'} + K$, where $r_{\mathcal{O}}$ and $r_{\mathcal{O}'}$ are the bounding cylinder radii of $\mathcal{O}$ and $\mathcal{O}'$ respectively, and K is constant set to 0.6. This constant is chosen as twice the length of the last link of the robot. Using this definition, the neighborhood of an object is easy to compute and captures enough of its surroundings for accurate action/grasp feasibility prediction.

Model Architecture. The IK feasibility prediction and GO estimation modules are 4-Layer MLPs with a hidden size of 512. The AGF module, on the other hand, has a hidden size of 256, 4 attention heads and one message-passing step. Exceptionally, when training on the **PR2-3D-4**

dataset, we use a hidden size of 256 for the GO module as it yields better results.

Training Details. The three modules are implemented in Pytorch Geometric (Fey and Lenssen 2019) and trained using the Adam optimizer (Kingma 2014). During the pre-training stage, each module is trained for 100 epochs. We use a batch size of 8192 and a learning rate of 0.001 for the IK feasibility classifier and the GO estimator. For AGF classifier, we set the batch size to 2048 and the learning rate to 0.0001. During the fine-tuning stage, the complete **GRN** model is trained for 100 epochs with a batch size of 2048, a learning rate of 0.0001 and $\eta = 10$. The model is trained on an Intel(R) Xeon(R) W-2223 CPU @ 3.60GHz workstation, with an NVIDIA RTX A5000 GPU. The full training process takes approximately 15 hours.

5.4.3 Baselines

We compare our proposed approach to multiple baselines and methods proposed or adapted from previous works on action and grasp feasibility prediction for manipulation planning.

MLP: This is a simple baseline which uses a 4-layer MLP that takes as input the feature vector $\boldsymbol{x}$ of an object to predict action and grasp feasibility, without considering the rest of the environment.

Feasibility-SVM (F-SVM): Introduced by Wells et al. 2019, this method uses multiple SVMs to predict the action and grasp types feasibility prediction in tabletop environments containing 2 movable objects represented using hand-crafted feature vectors. This method is hard to adapt for 3D environments. For environments, containing more objects, SVMs are queried for each pair of objects, an action or grasp type is predicted as infeasible if at least one query returns infeasible.

Deep Visual Heuristics (DVH): This method, proposed by Driess et al. 2020b, represents environments using top-view depth images, then uses a CNN to predict the feasibility of an action using a grasp type. We adapt **DVH** to output the feasibility of the action and all grasp types simultaneously.

Action and Grasp Feasibility Prediction Network (AGFP-Net): The method detailed in Chapter 4.

Feasibility-GAT (F-GAT): This baseline is an adapted version of the methods proposed by Silver et al. 2021a and Khodeir et al. 2023a,b, initially developed for learning objects importance and geometric planning steps' relevance in manipulation planning problems, respectively. It represents environments as graphs where nodes represent objects, and edges represent symbolic relationships (e.g object on table). Nodes features are the dimensions and pose of objects and edge features are one-hot-encodings of the different relationship types. GAT is then used to predict action and grasp feasibility.

Feasibility-GCN (F-GCN): This baseline uses the same scene representation as **F-GAT**, except that GAT is replaced with a Graph Convolution Network (GCN), which does not use edge features.

5.4.4 Evaluation Metrics

We use the **F1** score as the evaluation metric for classification tasks, namely action, grasp types and IK feasibility predictions. For Grasp Obstruction (GO) predictions, we use the **Mean Absolute Error** to measure performance. For clarity, we report the mean and standard deviation of predictions across the different grasp types.

5.5 Results

5.5.1 Comparison to Prior Methods

Table 5.1 shows that our proposed model outperforms all prior works on both action feasibility and grasp types feasibility predictions, and on all datasets. First, comparing results obtained on the **Panda-Tabletop-4** and **Panda-3D-4** show that action and grasp feasibility prediction is harder in 3D environments than in tabletop environments. Particularly, the poor performance of the **MLP** baseline on the **Panda-3D-4** dataset shows how important capturing the structure of the environment is important to the task at hand. CNN-based methods, **DVH** and **AGFP-Net**, fall short compared to our approach, with a difference in F1 score on the **Panda-3D-4** of 10% (resp. 5.7%) for action feasibility prediction, and 22.6% (resp. 13.7%), for grasp type feasibility prediction. Indeed, image-based scene representation suffers from occlusions due to the 3D nature of the environment, resulting in inaccurate predictions for occluded objects. GNN-based methods, on the other hand, represent 3D environments using scene graphs. However, the performance of our approach compared to **F-GCN** and **F-GAT** shows that a careful design of the graph and its connectivity is needed. Silver et al. 2021a and Khodeir et al. 2023a,b connect nodes using symbolic facts. Our method uses geometric relationships between objects to connect the graph. This allows our model to achieve an F1 score up to 10.3% higher than other GNN-based baselines on action feasibility prediction, and up to 21.8% higher on grasp types feasibility prediction on **Panda-3D-4**.

Comparing the standard deviations across F1 scores of each grasp type shows that our method has a more consistent performance across the different grasp types than other models. This is due to our data augmentation method and the two interpretation mechanisms. Switching the dimensions then/or rotating an object implies interchanging grasp types annotations, which ensures balanced labels across grasp types. This method is not applicable to CNN-based methods, since switching the dimensions then rotating an object results in the same input images. Additionally, the use of IK feasibility predictions and GO estimations as edge features allows each grasp type feasibility prediction to be informed, which is not the case for other GNN-based methods.

In robotic manipulation planning, feasibility prediction must not only be accurate, it must also have a low inference time and memory footprint. Table 5.2 reports the number of parameters in previous models compared to **GRN**, as well as the inference time of each model on a 3D scene containing a Panda robot, 4 movable objects and 15 fixed object. The inference time incorporates the complete prediction process from the model's input construction to the output, for each movable object in the environment. For reference, we also report the planning time of an off-the-shelf geometric planner.

Table 5.1: Comparison with SOTA methods trained and tested on different datasets. For grasp types feasibility prediction, the mean (± standard deviation) of F1 scores of the 5 grasp types are reported.

Dataset	Panda-3D-4		Panda-Tabletop-4		PR2-3D-4	
Task	Action (F1)	Grasp (F1)	Action (F1)	Grasp (F1)	Action (F1)	Grasp (F1)
F-SVM	-	-	0.884	0.415 (± 0.220)	-	-
MLP	0.784	0.558 (± 0.089)	0.911	0.696 (± 0.121)	0.750	0.574 (± 0.104)
DVH	0.840	0.718 (± 0.108)	0.961	0.865 (± 0.073)	0.808	0.622 (± 0.179)
AGFPNet	0.882	0.806 (± 0.065)	0.964	0.916 (± 0.032)	0.836	0.655 (± 0.238)
F-GCN	0.836	0.721 (± 0.057)	0.955	0.879 (± 0.042)	0.791	0.680 (± 0.075)
F-GAT	0.867	0.796 (± 0.052)	0.961	0.904 (± 0.034)	0.827	0.764 (± 0.061)
GRN (Ours)	**0.939**	**0.940 (± 0.009)**	**0.976**	**0.976 (± 0.004)**	**0.908**	**0.903 (± 0.013)**

Table 5.2: Comparison of the number of parameters and inference time on a 3D environment with 4 movable objects and 15 fixed objects (4 queries).

Model	Inference time (ms)	Nb Parameters
Geometric planner	1500	-
MLP	0.6	269'830
DVH	25	11'225'282
AGFPNet	150	34'585'350
F-GCN	4.25	1'057'798
F-GAT	5.0	2'636'806
GRN (Ours)	5.5	2'259'472

Results show that, compared to previous works, our method yields the most accurate predictions while being one the models with the lowest inference times and memory footprints. Furthermore, **GRN** has a 99.6% lower inference time than traditional geometric planning. In an offline manipulation planning context, where the number of feasibility checks can reach tens of thousands of queries, this difference in computational cost can significantly reduce planning time.

5.5.2 Ablation Study

In order to justify the choices behind our neural network architecture and training strategy, we conduct an ablation study on the **Panda-3D-4** dataset. Results reported in Table 5.3 showcase the importance of the proposed interpretation modules. Our full model shows a 7.1% gain in performance compared to the one without IK feasibility and GO predictions. A more in-depth analysis shows that the grasp obstruction estimation module is the most important, while IK feasibility prediction yields a slight improvement in performance. This observed limited performance gain from the IK feasibility submodel is due to the AGF module's ability to implicitly learn and address cases where infeasibility is due to kinematic constraints, effectively replacing the IK module's role during feasibility prediction if the latter is omitted. The IK infeasibility module is, however, essential for interpretability and planning efficiency, providing explicit reasoning about infeasibility causes. For instance, if an object is both obstructed and unreachable due to IK constraints, GRN without the IK module will attribute infeasibility solely to the obstruction. The IK module, however, identifies unreachability as the root cause, distinguishing cases where clearing the obstruction resolves infeasibility from those where the object remains unreachable. This clarity benefits TAMP planners by avoiding unnecessary computations, such as attempting to clear obstructions for unreachable objects.

The improved performance obtained using EGAT instead of classical GAT (Veličković et al. 2018) shows that incorporating edge features in the attention computation and the aggregation

Table 5.3: Ablation Study on the **Panda-3D-4** dataset. For each task related to grasp types, we report the mean (± standard deviation) across all grasp types.

Task	Action (F1) ↑	Grasp (F1) ↑	IK (F1) ↑	GO (MAE) ↓
w/o IK, w/o GO	0.868	0.811 (± 0.043)	-	-
w/o GO	0.872	0.811 (± 0.046)	**0.995 (± 0.001)**	-
w/o IK	0.937	0.933 (± 0.011)	-	0.029 (± 0.003)
Full model w/ GAT	0.928	0.924 (± 0.013)	**0.995 (± 0.001)**	0.029 (± 0.003)
Full model w/o data aug.	0.915	0.903 (± 0.013)	0.990 (± 0.001)	0.044 (± 0.002)
Full model (Trained from scratch)	0.932	0.925 (± 0.011)	0.994 (± 0.001)	0.038 (± 0.002)
Full model (Ours)	**0.939**	**0.939 (± 0.009)**	**0.995 (± 0.001)**	**0.028 (± 0.003)**

Table 5.4: Evaluation of the generalizability to 3D environments with a higher number of objects compared to SOTA methods, when trained on the **Panda-3D-4** dataset.

Test set	Panda-3D-10		Panda-3D-15		Panda-3D-20	
Task	**Action (F1)**	**Grasp (F1)**	**Action (F1)**	**Grasp (F1)**	**Action (F1)**	**Grasp (F1)**
MLP	0.773	0.624 (± 0.028)	0.766	0.616 (± 0.046)	0.768	0.609 (± 0.047)
DVH	0.820	0.697 (± 0.115)	0.819	0.686 (± 0.127)	0.825	0.676 (± 0.140)
AGFPNet	0.858	0.770 (± 0.079)	0.862	0.768 (± 0.078)	0.864	0.755 (± 0.087)
F-GCN	0.794	0.633 (± 0.036)	0.771	0.595 (± 0.044)	0.764	0.565 (± 0.055)
F-GAT	0.829	0.738 (± 0.026)	0.826	0.725 (± 0.032)	0.825	0.715 (± 0.038)
GRN (Ours)	**0.894**	**0.909 (± 0.012)**	**0.891**	**0.906 (± 0.013)**	**0.890**	**0.903 (± 0.012)**

process helps action and grasp types feasibility prediction. Moreover, we conduct two ablations to demonstrate the effectiveness of our training strategy. Training the model without the proposed data augmentation method, yields a lower performance on all tasks, particularly on grasp types feasibility prediction and GO estimation with a 3.6% difference in F1 score, and 1.6 % in MAE. Finally, training the full model from scratch, rather than pre-training each module before fine-tuning the complete network, yields a slightly lower performance across all tasks. Without pre-training, jointly training all modules leads to convergence to spurious local optima due to the greedy nature of weight optimization. Pre-training helps mitigate this issue by providing a strong initialization, which ensures more effective fine-tuning during joint training.

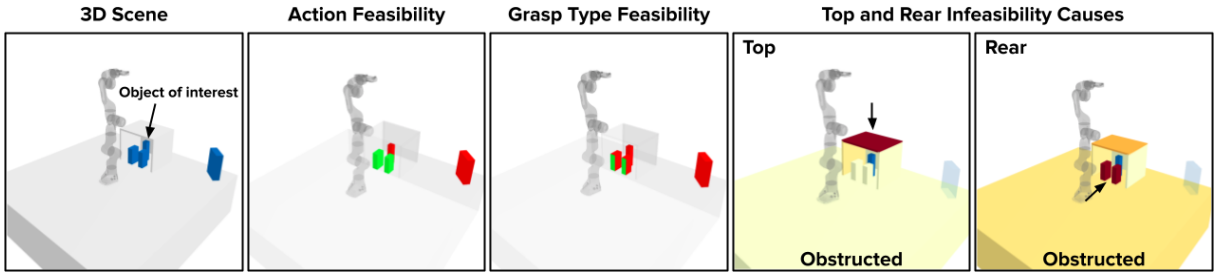
5.5.3 Generalizability Evaluation

Applicability to other robots. Table 5.1 shows our model's performance on the **PR2-3D-4** compared to prior methods. The results show that **GRN** achieves a better performance than the state-of-the-art on robots with various kinematics. A consistent decrease in performance can be noticed across all methods (ours included), compared to when trained on the **Panda-3D-4** dataset. This is due to the smaller number of training data of the PR2 dataset and the harder kinematics of the PR2 robot. However, our method still outperforms other methods, with an F1 score of 0.908 for action feasibility prediction, and 0.903 for grasp types feasibility prediction. This shows that our method is applicable to different robotic manipulators. Moreover, additional data generation effort can be made to improve the performance of our method on the PR2 robot, since the number of training scenes in the **PR2-3D-4** dataset is lower than in **Panda-3D-4**.

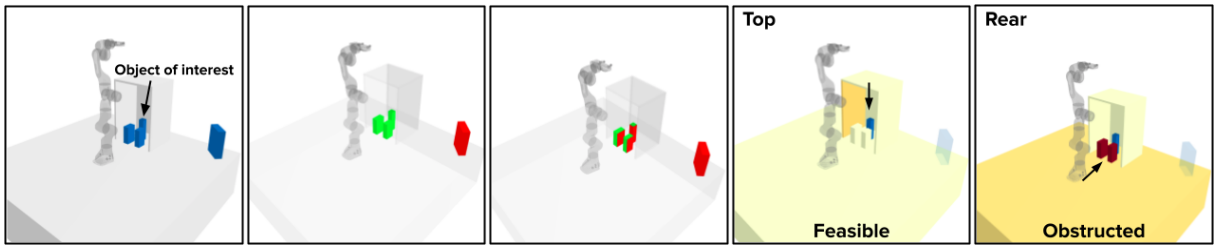
Generalizability to more complex environments. Table 5.4 reports the performance of our method and the baselines on the **Panda-3D-10**, **Panda-3D-15** and **Panda-3D-20** test sets, when trained on the **Panda-3D-4** dataset. Results show that, although there is a decrease in F1-scores compared to the one obtained on **Panda-3D-4**, **GRN** maintains a good performance on 3D environments with a higher number of fixed and movable objects, with an F1-score on **Panda-3D-20** of 0.89 for action feasibility prediction, and 0.903 for grasp types feasibility prediction. Particularly, our model shows a better generalization capability on the latter than previous CNN-based or GNN-based methods. Note that these results also show a better generalization of our method to out-of-distribution object dimensions, since the dimensions' range used during data generation for these test sets is larger than the one used for the training set, resulting in larger objects than the ones seen during training.

5.5.4 Interpretability Analysis

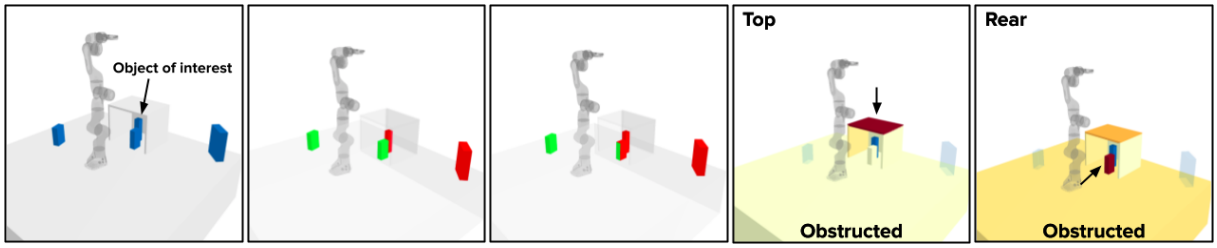
To assess the interpretability of our model, we conduct an evaluation using a 3D environment containing two objects of interest for which the action and grasps are initially infeasible, as illustrated in Figure 5.8. Based on **GRN**'s predicted infeasibility causes, we systematically



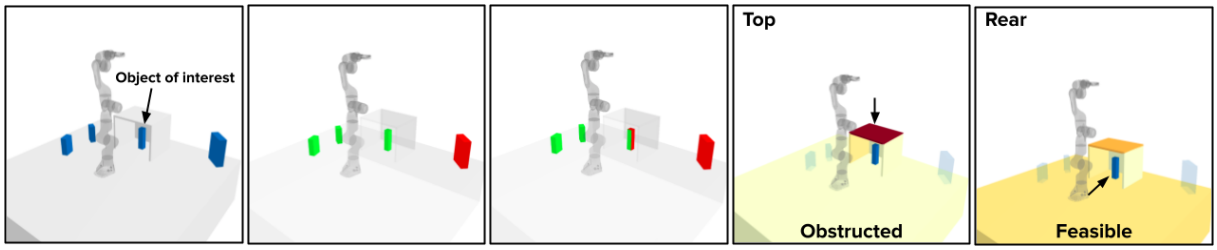
(a) Top grasps are obstructed by the shelf, Rear grasps are obstructed by two movable objects.



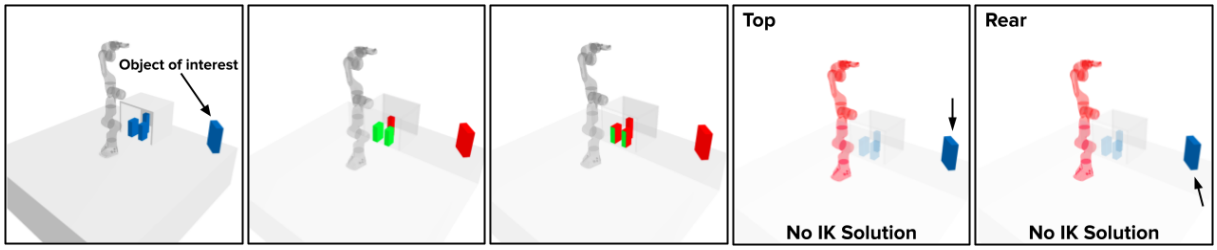
(b) With a higher shelf height, Top grasps become feasible, which makes the action feasible



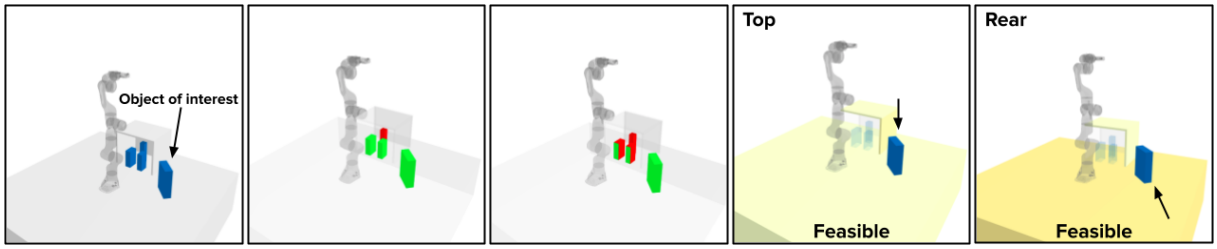
(c) Removing only one of the two obstructing objects does not free access to the object via Rear grasps



(d) Removing both obstructing objects makes Rear grasps feasible, hence the action becomes feasible



(e) The object of interest is unreachable, the action and grasps are infeasible due to the absence of IK solutions



(f) Making the object closer to the robot allows reachability, thus the action and grasps become feasible.

Figure 5.8: Interpretability evaluation of **GRN**'s predictions on an example 3D environments.

modify the environment to observe how our model's predictions change in response. This allows us to evaluate the interpretability of our method by showcasing how the predicted reasons of infeasibility explain action and grasp feasibility predictions, and how changing the input scene based on these infeasibility causes affects the outputs of the model.

Figure 5.8a shows **GRN**'s predictions on the initial 3D environment, and the predicted reasons of infeasibility for the first object of interest. For simplicity, we show infeasibility causes for the *Top* and *Rear* grasp types only. The obtained predictions show that the action involving the object of interest is infeasible, with *Top* grasps being obstructed by the shelf, while *Rear* grasps are obstructed by two movable objects. Figure 5.8b shows that increasing the height of the shelf removes the obstruction of the *Top* grasp type, with **GRN** predictions indicating that *Top* grasps and the action are now feasible. For the *Rear* grasp type, Figure 5.8c shows that removing only one of the two obstructing objects does not make the grasp type feasible. Removing the second object as well is necessary to allow access to the object of interest, as shown in Figure 5.8d.

Regarding the second object of interest, the corresponding reasons of infeasibility are shown in Figure 5.8e. Due to the object's unreachability by the robot, all grasp types are infeasible due to the absence of IK solutions. Figure 5.8f shows that decreasing the distance between the object and robot allows reachability, with **GRN** predictions showing that the action as well as the *Top* and *Rear* grasp types become feasible.

This evaluation demonstrates **GRN**'s strong interpretability capabilities. By predicting specific infeasibility causes, namely grasp obstructions and IK infeasibility, our model's feasibility predictions are not only informed, but also interpretable. Moreover, these explanations provide actionable insights into how modifying the environment affects feasibility. Its outputs consistently align with expected changes, sustaining **GRN**'s reliability and coherence. These interpretability features are critical for real-world task and motion planning, where understanding the reasons behind infeasibility is as important as the feasibility predictions themselves.

5.6 Visualizations

In this section, we present a visual comparison between the predictions of our model, GRN, and the ground truth. Figure 5.9 illustrates GRN's predictions in a test environment from **Panda-3D-4**, while Figure 5.10 showcases predictions from **Panda-3D-10**. In these figures, action feasibility predictions and ground truth are represented through object colors in the top images: green indicates feasibility for pick or place actions, while red denotes infeasibility. The feasibility of different grasp types is visualized through the coloration of the corresponding object's faces. Finally, the reasons of infeasibility are shown for one of the objects in the environment (in blue), for each grasp type. There are two types of infeasibility causes, IK infeasibility (with the robot depicted in red) and grasp obstructions (where obstructing objects are displayed in full opacity, with their colors reflecting obstruction ratios). These comparisons highlight that GRN accurately predicts action and grasp feasibility for most objects. It also effectively identifies the reasons for infeasibility and estimates grasp obstruction ratios with minimal error. A notable exception occurs in the 10-object environment, where the *Rear* grasp type of one object is incorrectly classified as infeasible. Figure 5.10 details the predicted infeasibility reasons for this misclassification. A closer examination of the corresponding *Rear* grasp type plot reveals a slight overestimation of the obstruction ratio for a large block, which likely contributes to the error. However, GRN correctly predicts the feasibility of other grasp types for the same object, including the feasible *Top* grasp type, ensuring accurate action feasibility predictions overall.

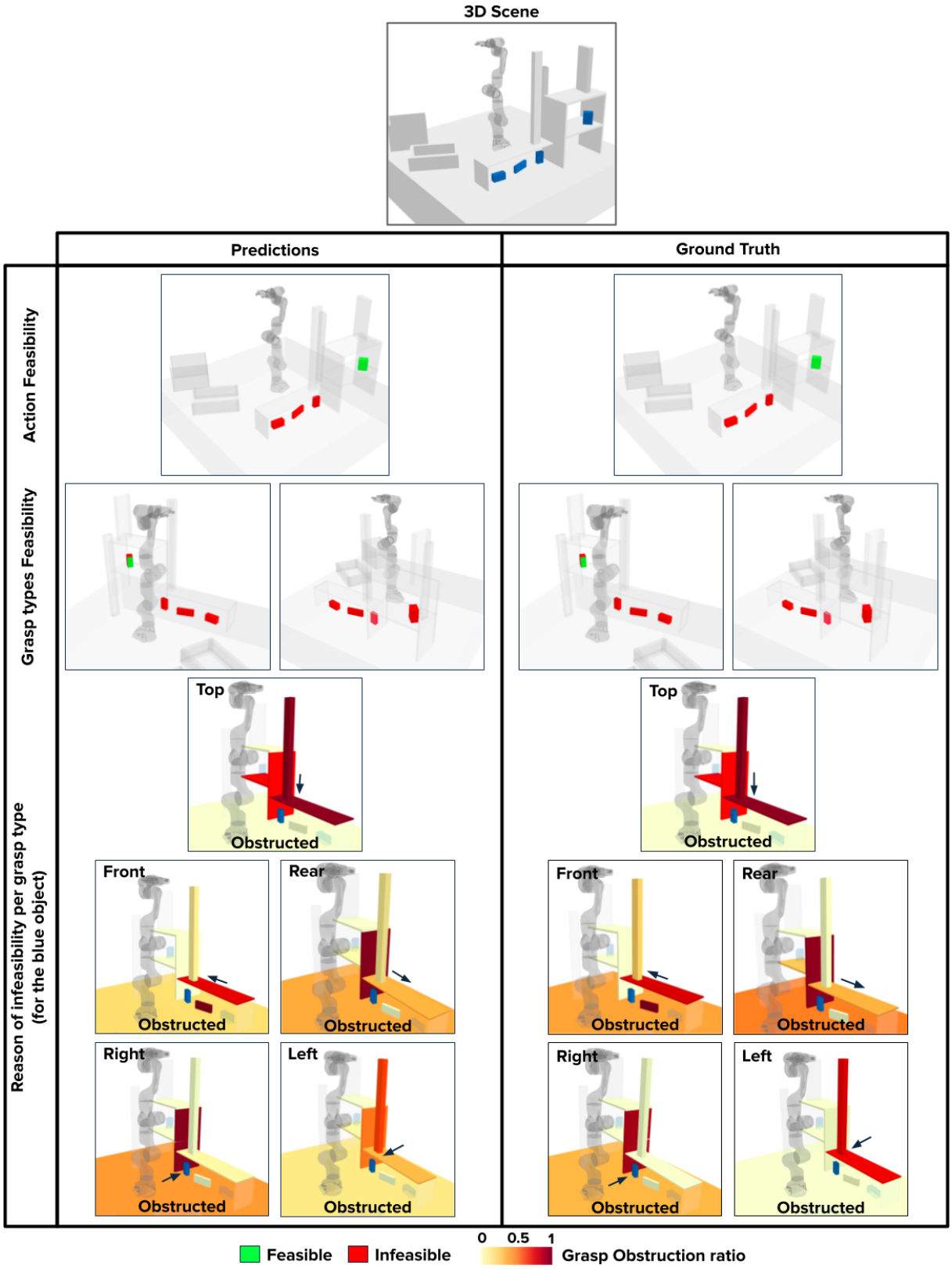


Figure 5.9: Visualization of **GRN** prediction compared to the ground truth on a test environment from **Panda-3D-4**.

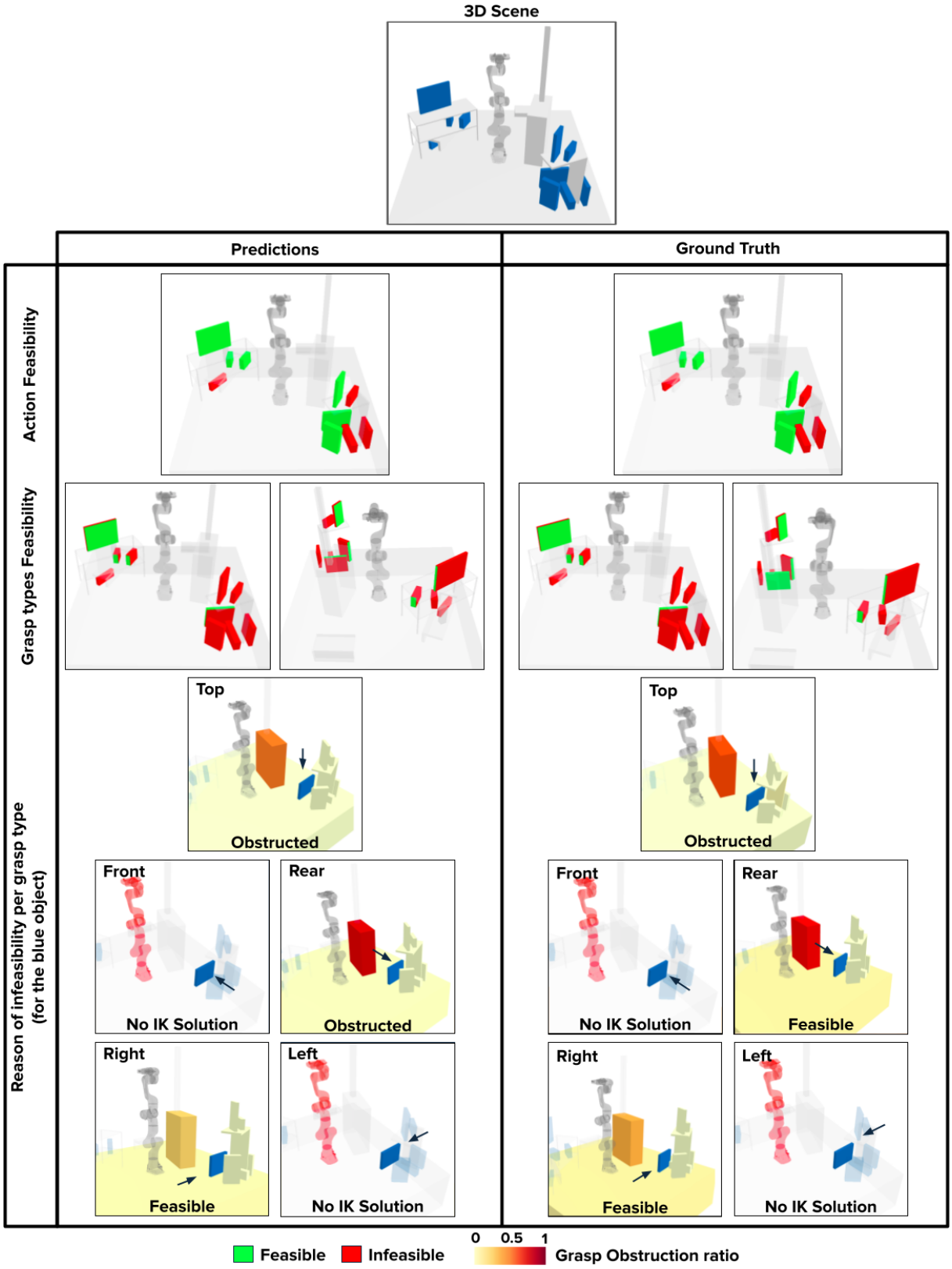


Figure 5.10: Visualization of **GRN** prediction compared to the ground truth on a test environment from **Panda-3D-10**.

5.7 Robustness to Noise

We evaluate the robustness of **GRN** by introducing varying levels of noise to the input data. Specifically, noise is added to the node features, i.e. the dimensions and poses of all objects in the environment. The noise is sampled from a normal distribution with the following parameters: (i) **1cm, 1° Noise Level:** Noise with a mean of 0 and a standard deviation of 1 cm for bounding box dimensions and translation coordinates, and 1° for object orientation, (ii) **2cm, 2° Noise Level:** Noise with a mean of 0 and a standard deviation of 2 cm for bounding box dimensions and translation coordinates, and 2° for object orientation. Table 5.5 presents the performance of GRN on the Panda-3D-4 test set under these noise conditions.

Table 5.5: Performance of **GRN** on the **Panda-3D-4** test set with different levels of noise.

Noise Level	Action (F1)	Grasp (F1)	IK (F1)	GO (MAE)
No Noise	0.939	0.940 (± 0.009)	0.995 (± 0.001)	0.028 (± 0.003)
1cm, 1°	0.912	0.891 (± 0.020)	0.985 (± 0.001)	0.039 (± 0.003)
2cm, 2°	0.864	0.820 (± 0.029)	0.971 (± 0.002)	0.057 (± 0.003)

While there is a slight decrease in performance as noise levels increase, **GRN** maintains a good accuracy in both action and grasp feasibility predictions. Importantly, even under the highest noise condition (**2 cm, 2°**), our method still achieves a better performance than the one yielded by most baselines on noise-free inputs, shown in Table 5.1. More precisely, under both noise levels, **GRN** outperforms **MLP**, **DVH** and **F-GCN** on action and grasp feasibility prediction, even when these baselines are given noise-free inputs. Under the **(1cm, 1°)** noise level, our model also provides more accurate predictions than the ones obtained by **AGFPNet** and **F-GAT** under zero noise. Under the highest noise level, **GRN** achieves similar performance to noise-free **F-GAT** and a slightly worse performance than zero-noise **AGFPNet** on action feasibility prediction, but still yields a better performance than both on grasp type feasibility prediction. Also, comparing standard deviations shows that even when noise is applied, **GRN** results in a more consistent prediction quality across grasp types.

Furthermore, we highlight that in addition to a better accuracy under different noise level, our model also provides richer information thanks to our two interpretation modules. Table 5.5 shows that **GRN** maintains a good performance across noise levels on IK feasibility and Grasp Obstruction (GO) predictions as well. These results emphasize the robustness of GRN to real-world imperfections, while still delivering state-of-the-art performance compared to other methods.

5.8 Application to Task and Motion Planning

5.8.1 Single-shot Manipulation Planning

The proposed model allows through its two interpretability mechanisms to solve in a single shot a class of manipulation planning problems where the goal is to move a single object initially placed in a complex setting. To demonstrate this capability, we develop a simple planner that uses **GRN**'s predictions to propose geometrically feasible plans. From a single query to **GRN**, it leverages the predicted reasons of infeasibility to find a sequence of actions that allows the robot to reach the object of interest. Algorithm 16 shows the pseudo-code for the GRN-based planner. Given the **GRN** predictions on the initial state of the environment, a feasibility threshold and a grasp obstructions threshold, it recursively builds a task plan by moving objects when it is feasible, or trying to free access to grasp types based on grasp obstruction and IK feasibility

Algorithm 16 GRNPlanner

Input: $F_a, \boldsymbol{F}_\mathcal{G}, \boldsymbol{\kappa}_\mathcal{G}, \boldsymbol{\rho}_\mathcal{G}, \mathcal{T}_F, \mathcal{T}_{GO}, \mathcal{O}$ ▷ **GRN** predictions, feasibility threshold, grasp obstruction threshold, desired object

$plan \leftarrow []$
if $F_a(\mathcal{O}) > \mathcal{T}_F$ **then**
  $feasibleGrasps = \emptyset$
  for each $g \in \mathcal{G}$ such that $F_g(\mathcal{O}) > \mathcal{T}_F$ **do**
    $feasibleGrasps \leftarrow feasibleGrasps \cup g$
  end for
  $plan.append(Move(\mathcal{O}, feasibleGrasps))$
  return $plan$
else
  $BlockersPerGraspType \leftarrow [\emptyset \quad \forall g \in \mathcal{G}]$
  $freeableGraspTypes \leftarrow \mathcal{G}$
  for each $g \in \mathcal{G}$ **do**
    if $\kappa_g(\mathcal{O}) < \mathcal{T}_F$ **then**
      $freeableGraspTypes \leftarrow freeableGraspTypes \setminus g$
    else
      for $\mathcal{O}' \in \mathcal{N}(\mathcal{O})$ such that $\rho_g(\mathcal{O}, \mathcal{O}') > \mathcal{T}_{GO}$ **do**
        if $\mathcal{O}'$ is fixed **then**
          $freeableGraspTypes \leftarrow freeableGraspTypes \setminus g$
          break
        else
          $BlockersPerGraspType[g] \leftarrow BlockersPerGraspType[g] \cup \mathcal{O}'$
        end if
      end for
    end if
  end for
  if $freeableGraspTypes$ is empty **then**
    return []
  else
    for each $g \in freeableGraspTypes$ **do**
      $subplan \leftarrow []$
      $graspTypeFreed \leftarrow True$
      for each $\mathcal{O}' \in BlockersPerGraspType[g]$ **do**
        $result \leftarrow GRNPlanner(F_a, \boldsymbol{F}_\mathcal{G}, \boldsymbol{\kappa}_\mathcal{G}, \boldsymbol{\rho}_\mathcal{G}, \mathcal{T}_F, \mathcal{T}_{GO}, \mathcal{O}')$
        if $result$ is empty **then**
          $graspTypeFreed \leftarrow False$
          break
        else
          $subplan.append(result)$
        end if
      end for
      if $graspTypeFreed$ is True **then**
        $plan.append(subplan)$
        return $plan$
      end if
    end for
    if $graspTypeFreed$ is False **then**
      return []
    end if
  end if
end if

information otherwise. Once a task plan is found, we query the geometric planner used during data annotation to verify its geometric feasibility, and plan the corresponding robot motions.

We compare this algorithm to a the baseline TAMP planner, described in Chapter 3, on two different problems shown in Figure 5.1. The first is the **Access-monotonic** problem, where a single Panda robot has to move a small bottle, to which a number of fixed and movable objects block access. This is a monotonic version of the **Access** problem described in Section 3.4, in which the robot does not have to move blockers back to their initial placement. The second is a multi-robot **Clutter** problem, in which both arms of the PR2 robot can collaborate to pick an object surrounded by grasp-obstructing objects. For simplicity, Algorithm 16 shows the planner used for the single robot problem only. For the multi-robot **Clutter** problem using both arms of the PR2 robot, we choose the robot with the higher feasibility prediction to execute

feasible actions. If an action is infeasible for both arms, we consider both of them when trying to free grasp types. Objects used in these problems are a mix of box-shaped and mesh objects. Although these tasks might seem simple, they represent challenging TAMP problem with high combinatorial complexity. Indeed, in both problems, multiple objects not only block access to the goal object, they also block access to each other. The number of geometrically feasible solutions is very limited, requiring the planner to find the specific order in which to move the objects and which grasp to choose at each step.

Table 5.6: Performance of **GRN** planner compared to a non-informed planner on the **Access-monotonic** and **Clutter problems**. Results are average over 10 runs on 10 different instances of each problem.

Problem	Method	Success Rate	Planning time (s)	Nb Geometric Planner Calls
Access-monotonic	Baseline	100%	26.5	41.1
	GRN planner	90%	3.17	6
Clutter	Baseline	100%	558.9	89.5
	GRN planner	100%	14.8	7.2

Table 5.6 shows that despite its simplicity, the **GRN**-based planner not only achieves a 90% and 100% success rate but also reduces planning time by 88% on the **Access-monotonic** problem, and 97% on the **Clutter** problem. This is possible thanks to the two interpretation mechanisms, which allow the planner to reason over why an action or a grasp type is infeasible, and recursively decide which objects should be moved to rectify that. We also test **GRN** planner in real-world setups, on both a Panda and a PR2 robots. Objects' poses are estimated using a separate perception module. The proposed model is able to accurately predict action and grasp feasibility, as well as reasons of infeasibility, from estimated objects' poses in both setups, allowing the planner to compute feasible solutions that are successfully executed by the robots.

5.8.2 Additional Simulation Experiments

Access problem with a large number of objects. We conduct an additional experiment in simulation to test the scalability of our method to problems requiring long-horizon planning. As in the previous problems, the goal is to move a single object (shown in pink in Figure 5.11), to which a total of 27 objects block access. This task requires removing at least 18 objects to access the desired one, resulting in a total of 19 necessary Pick-Place actions. Figure 5.11 shows **GRN**'s predictions on the initial state of this problem. The model accurately predicts that, initially

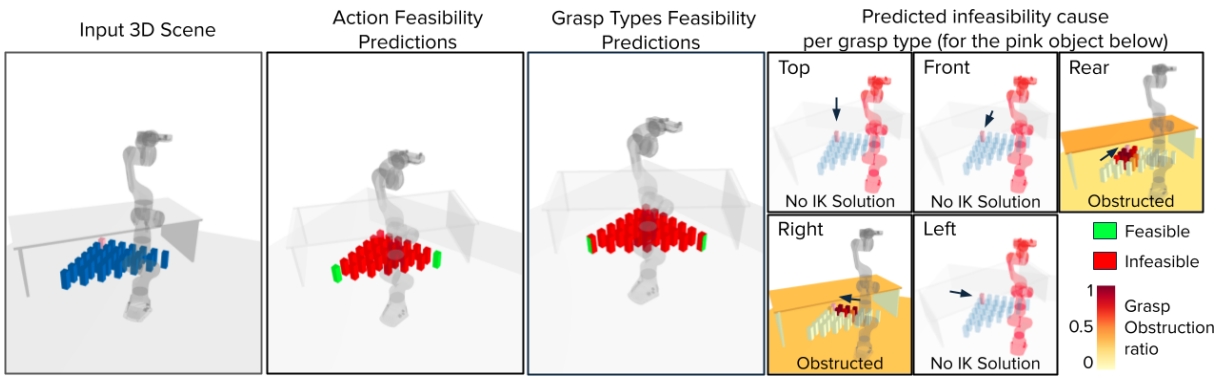


Figure 5.11: Visualization of **GRN** predictions on the initial state of a larger version of the **Access-monotonic** problem.

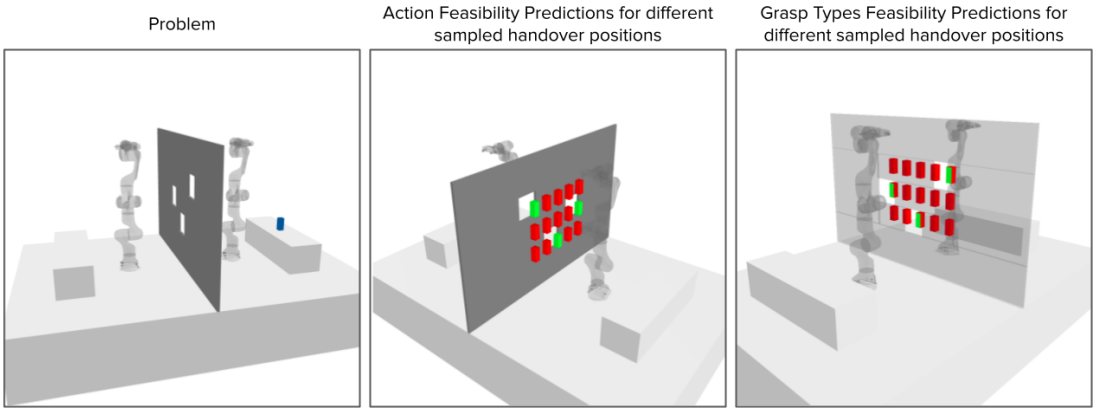


Figure 5.12: Visualization of **GRN** predictions on different sampled handover positions in the **Handover** problem.

accessing the desired object is infeasible, and that only two blockers can be moved, using a single grasp type each. For simplicity, we show the obtained reasons of infeasibility of the pink object only. However, after the same initial query, our model also predicts the reasons of infeasibility for all the blockers as well. The baseline planner completely fails to solve this problem, even after 2 hours of planning, while the **GRN**-based planner find a feasible solution in under 15s.

Inter-robot Handover. Our proposed approach can also be used to predict the feasibility of handover actions between two robots. Indeed, given the robot-centric nature of our model, we can divide a handover action into two separate Pick-Place actions, and query **GRN** for each robot individually to predict the feasibility of each sub-action. One of the main challenges of handover tasks is to determine where to perform the object exchange. Our model can be used to efficiently find feasible handover positions by evaluating different sampled positions. In order to showcase this ability, we define the **Handover** problem, in which two Panda robots have to perform a handover task for a single object. A wall separates the two robots leaving only 3 "windows" through which the object can be exchanged. We sample a number of potential handover positions that lie in the intersection between the two robots workspaces, and query **GRN** to evaluate their feasibility. As shown in Figure 5.12, our model is able to accurately predict that the wall blocks most of the sampled positions, and that only those in front of the windows are feasible. One key advantage of our method is that all the sampled positions can be evaluated simultaneously by adding a node to the input scene graph for each one, while making sure that proximity edges are not created between these nodes. This enables an efficient search for feasible handover positions, requiring a single query to **GRN**. In contrast, traditional methods need to query the geometric planner for each sampled position independently, resulting in long planning times.

5.8.3 Real-world Experiments

We test the GRN-based planner in two real-world setups, using a Panda arm and a PR2 robot. We use a perception module based on Overworld (Sarthou 2023) to identify objects and estimate their poses. The shapes of the object are known and are not subject to detection. The GRN-based planner was able to generate geometrically feasible plans on both the **Access-monotonic** and **Clutter** problems, which were successfully executed by the robots. Figures 5.13 and 5.14 show the rollouts of both executions.

Figure 5.13: Rollout of the solution found by the **GRN**-based planner on the **Access-monotonic** problem.

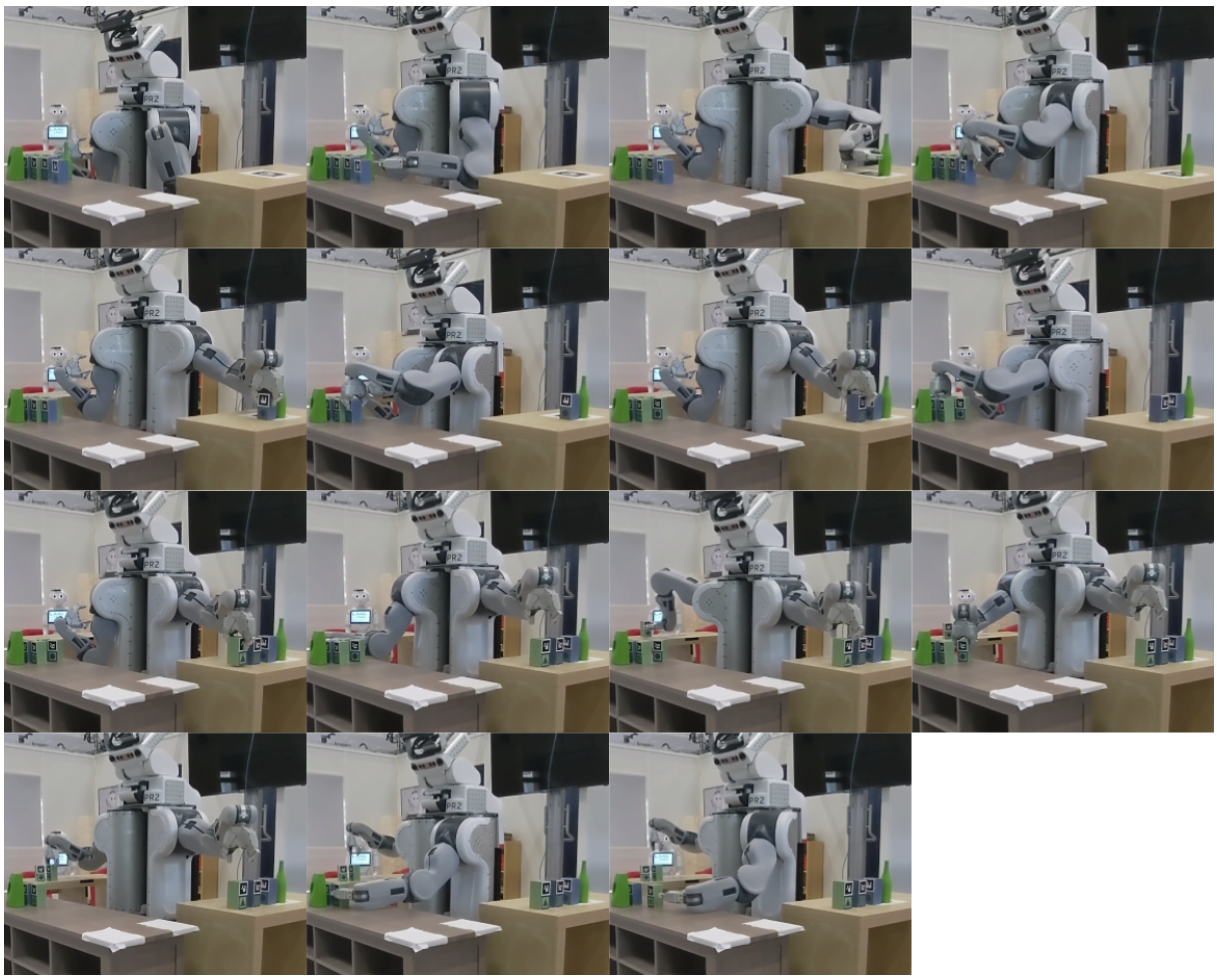


Figure 5.14: Rollout of the solution found by the **GRN**-based planner on the **Clutter** problem.

5.9 Discussion and Limitations

In this chapter, we propose a framework for action and grasp feasibility prediction in 3D environments. Leveraging a GNN-based neural network and two interpretation mechanisms, our model predicts the feasibility of pick or place actions, different grasp types, and the infeasibility cause for each grasp type from a scene graph representation. Results demonstrate that our approach outperforms state-of-the-art methods, generalizing better to complex environments and robots. Additionally, through IK feasibility and grasp obstruction predictions, a class of manipulation planning problems can be solved with a single query to our model, significantly reducing planning time compared to traditional TAMP planners. Our design also enables straightforward extensions to diverse object shapes via bounding boxes and multi-robot feasibility predictions due to the robot-centric nature of **GRN**.

Thanks to the two interpretation mechanisms predicting reasons of infeasibility, our model can efficiently find feasible solutions to a class of complex manipulation planning problems in a single shot. However, the GRN-based planner is not complete, and can fail to find a solution in case of misclassifications. Moreover, this planner is limited to problems where the one object only has a specified goal. An integration into a more robust TAMP planner with better error management is necessary to tackle a wider range of TAMP problems while ensuring probabilistic completeness. Specifically, the GRN-based planner provides an efficient method for better estimating the cost-to-goal in the baseline TAMP planner presented in Section 3.3.1. In the next chapter, we present an improved feasibility-informed TAMP planner that integrates learned geometric reasoning to prioritize feasible actions, better estimate the cost-to-goal, and reduce planning time.

Chapter 6

Task and Motion Planning with Learned Geometric Reasoning

Contents

6.1 Introduction

Geometric reasoning is a crucial component of task and motion planning (TAMP). It allows the robot to reason about the feasibility of actions, and more importantly the reason for their infeasibility. By understanding the infeasibility causes, the TAMP planner can avoid exploring infeasible plans, thus repeated geometric planning failures. As a result, the planner can focus on action sequences that are feasible, or allow the robot to rectify an infeasible action. Traditional geometric reasoning, however, still requires computationally expensive calls to the geometric planner to determine the reasons of infeasibility. In the previous chapter, we introduced a method for simultaneously predicting the feasibility of actions and grasps, as well as the causes of infeasibility. **GRN** offers an efficient geometric reasoning method that can be used to improve the efficiency of TAMP planners, without the need for repeated calls to the geometric planner. In this chapter, we present an integration of **GRN** with the previously introduced TAMP pipeline. We first propose an informed backtracking method that leverages infeasibility causes as planning

constraints to improve search tree pruning and cost-to-goal computation. We then present an improved TAMP algorithm that uses predicted infeasibility causes as relaxed planning constraints in combination with feasibility predictions to accelerate the planning process.

6.2 Infeasibility Causes as Planning Constraints

The reference task and motion planning pipeline presented in Chapter 3 Section 3.3.1 relies on geometric planning to determine the feasibility of actions and plan their motions. The only feedback provided to the task planner is the success or failure of the geometric planner, without any information on why a specific action failed. This lack of interpretability makes it challenging to improve planning efficiency, especially in complex environments where multiple objects interact. As a result, during backtracking, the task planner simply moves to the next best state, without considering the infeasibility causes encountered during geometric planning. This can lead to extensive effort spent on exploring infeasible task plans and repeating the same infeasible actions. In this section, we propose an informed backtracking method that records infeasibility causes during geometric planning as planning constraints, and leverages them during task planning to better guide the search toward a geometrically feasible solution.

6.2.1 Planning Constraints Definition

During geometric planning, the geometric planner may encounter multiple infeasibility causes that prevent the robot from successfully executing an action. These causes can be due to the absence of a valid inverse kinematics solution, the obstruction of grasps by neighboring objects, or the absence of a collision-free path. By recording these infeasibility causes, the task planner can avoid exploring infeasible task plans that involve the same infeasibility causes. This can significantly reduce the number of calls to the geometric planner and help reduce the combinatorial complexity of TAMP problems. For instance, if an object $\mathcal{O}'$ is identified as the one blocking access to the wanted object $\mathcal{O}$, hence the reason of infeasibility of an action involving $\mathcal{O}$, this information can be valuable to the task planner as it can avoid exploring task plans that attempt to move $\mathcal{O}$ while $\mathcal{O}'$ is still at the same configuration. Moreover, the task planner can use this information to guide the search process towards prioritizing actions that move $\mathcal{O}'$ out of the way first. In order to accomplish this, we propose to represent the reasons of infeasibility as planning constraints that can be used to prune actions where the same infeasibility causes occur without the need for repeated calls to the geometric planner. Furthermore, they can be used during the computation of the cost-to-goal to obtain a more accurate estimation that takes into account the necessary actions needed to rectify the infeasibility causes.

Given a state s, let $a_{infeasible}$ be an action that is infeasible at state s such that:

$$a_{infeasible} = Move(\mathcal{R}, \mathcal{O}_{blocked}, s(\mathcal{O}_{blocked}) \rightarrow \mathcal{P}) \tag{6.1}$$

where $\mathcal{R}$ is the robot performing the action, $\mathcal{O}_{blocked}$ is the object manipulated by the infeasible action, $s(\mathcal{O}_{blocked})$ is its configuration at state s, and $\mathcal{P}$ is the placement action $a_{infeasible}$ aims to place $\mathcal{O}_{blocked}$ at. Let $\mathcal{G}$ be the set of grasp types as defined in Section 4.4.3, where each grasp type $\gamma \in \mathcal{G}$ is associated with a side of the object $\mathcal{O}_{blocked}$. The goal is to record the infeasibility causes for each grasp type $\gamma \in \mathcal{G}$. Specifically, we are only interested in the infeasibility causes that can be rectified, meaning that the infeasible action can be made feasible by performing other actions. For example, if a grasp type is infeasible due to the absence of an inverse kinematics solution or due to a fixed object fully obstructing it, then the infeasibility cause can not be rectified. In contrast, if a grasp type is infeasible due to an obstructing movable object, then the

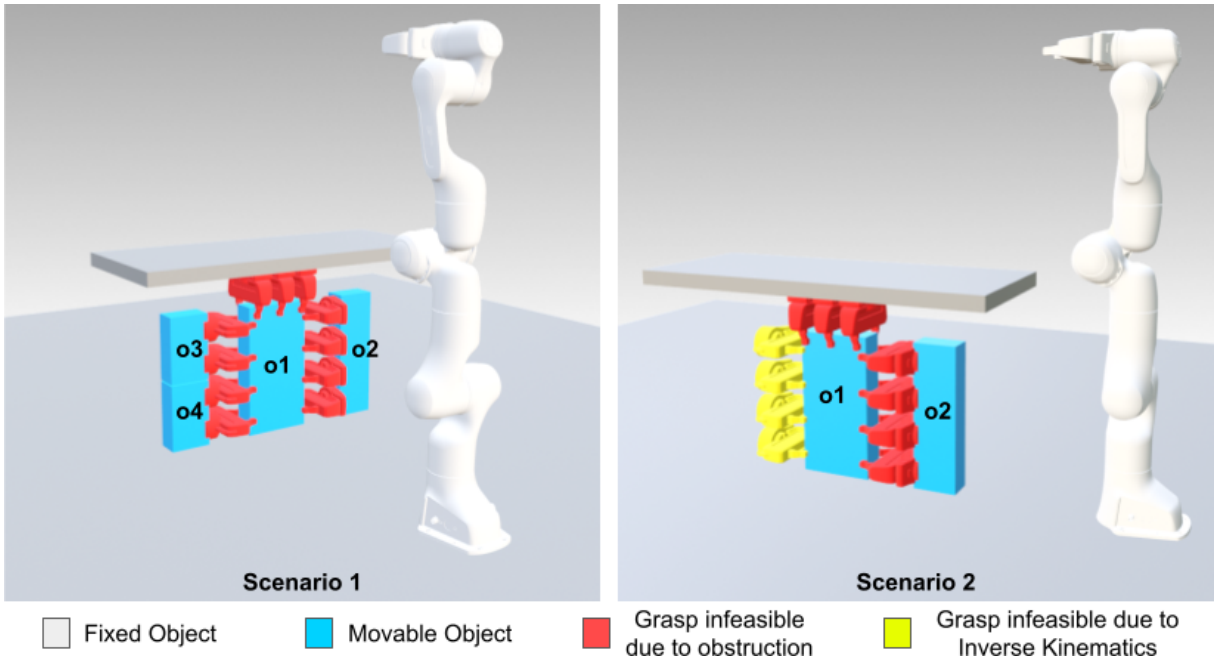


Figure 6.1: Illustration of how infeasibility causes are differentiated depending on rectifiability and obstruction ratio. **Left:** The object is obstructed from all sides. All top grasps are obstructed by a fixed object, hence $\mathcal{I}_{top} = True$. Right grasps are fully obstructed by the movable object $o2$, meaning that $\mathcal{I}_{right} = False$ and $\mathcal{B}^{full}_{right} = \{o2\}$. Left grasps are partially obstructed by two movable objects $o3$ and $o4$, then $\mathcal{I}_{left} = False$ and $\mathcal{B}^{partial}_{left} = \{o3, o4\}$. **Right:** Grasps from the front of the object are infeasible due to the absence of inverse kinematics solutions, thus $\mathcal{I}_{front} = True$. Top and Rear grasps are similar to the first scenario.

infeasibility cause can be rectified by moving the obstructing object out of the way. Following this reasoning, we define $\mathcal{I}_\gamma$ as a predicate that indicates whether a grasp type γ is non-rectifiable:

$$\mathcal{I}_\gamma = \begin{cases} True & \text{if } \gamma \text{ is non-rectifiable} \\ False & \text{otherwise} \end{cases} \tag{6.2}$$

We hence obtain a set $\mathcal{I}_\mathcal{G} = \{\mathcal{I}_\gamma | \gamma \in \mathcal{G}\}$ that contains information on the rectifiability of infeasibility causes for each grasp type. In case the infeasibility is rectifiable, let $\mathcal{B}^{full}_\gamma$ and $\mathcal{B}^{partial}_\gamma$ be the sets of movable objects fully and partially obstructing the grasp type γ, respectively. In other words, $\mathcal{B}^{full}_\gamma$ contains the objects that obstruct all of the grasps of type γ, while $B^{partial}_\gamma$ contains the objects that obstruct only a subset of grasps of type γ. Similarly to $\mathcal{I}_\mathcal{G}$, we define $\mathcal{B}^{full}_\mathcal{G} = \{\mathcal{B}^{full}_\gamma | \gamma \in \mathcal{G}\}$ and $\mathcal{B}^{partial}_\mathcal{G} = \{\mathcal{B}^{partial}_\gamma | \gamma \in \mathcal{G}\}$. Figure 6.1 provides an example of how $\mathcal{I}_\mathcal{G}$, $\mathcal{B}^{full}_\mathcal{G}$ and $\mathcal{B}^{partial}_\mathcal{G}$ are populated.

As a reminder, the action $a_{infeasible}$ can be decomposed into a *Pick* component and a *Place* component as follows:

$$a_{infeasible} = Pick(\mathcal{R}, \mathcal{O}_{blocked}, s(\mathcal{O}_{blocked})) \rightarrow Place(\mathcal{R}, \mathcal{O}_{blocked}, \mathcal{P}) \tag{6.3}$$

We distinguish two infeasibility cases for $a_{infeasible}$, the first being the infeasibility of the *Pick* component of $a_{infeasible}$, and the second being the infeasibility of the *Place* component. When an action is infeasible due to the *Pick* component, the infeasibility depends on the configuration of the blocked object $s(\mathcal{O}_{blocked})$ and the current configuration of the objects in $\mathcal{B}^{full}_\mathcal{G}$ and $\mathcal{B}^{partial}_\mathcal{G}$

Algorithm 17 buildConstraint

Input: $t, \mathcal{R}, \mathcal{O}, s, \mathcal{P}, \boldsymbol{\kappa_G}, \boldsymbol{\rho_G}$
1: $\mathcal{C} \leftarrow \emptyset$
2: $\mathcal{I}_\mathcal{G} \leftarrow \{False|\gamma \in \mathcal{G}\}, \quad \mathcal{B}_\mathcal{G}^{full} \leftarrow \{\emptyset|\gamma \in \mathcal{G}\}, \quad \mathcal{B}_\mathcal{G}^{partial} \leftarrow \{\emptyset|\gamma \in \mathcal{G}\}$
3: **for** each $\gamma \in \mathcal{G}$ **do**
4: **if** $\boldsymbol{\kappa_G}(\gamma) = 0$ or ($\exists \mathcal{O}'$ s.t. $\mathcal{O}'$ is fixed and $\boldsymbol{\rho_G}(\gamma, \mathcal{O}') = 1$) **then**
5: $\mathcal{I}_\mathcal{G}(\gamma) \leftarrow True$
6: **continue**
7: **else**
8: $\mathcal{I}_\mathcal{G}(\gamma) \leftarrow False$
9: **end if**
10: **for** each obstructing object $\mathcal{O}'$ such that $\mathcal{O}'$ is movable **do**
11: **if** $\boldsymbol{\rho_G}(\gamma, \mathcal{O}') = 1$ **then**
12: $\mathcal{B}_\mathcal{G}^{full}(\gamma) \leftarrow \mathcal{B}_\mathcal{G}^{full}(\gamma) \cup \{\mathcal{O}'\}$
13: **else**
14: $\mathcal{B}_\mathcal{G}^{partial}(\gamma) \leftarrow \mathcal{B}_\mathcal{G}^{partial}(\gamma) \cup \{\mathcal{O}'\}$
15: **end if**
16: **end for**
17: **end for**
18: $\mathcal{C} \leftarrow \{t, \mathcal{R}, \mathcal{O}, s, \mathcal{P}, \mathcal{I}_\mathcal{G}, \mathcal{B}_\mathcal{G}^{full}, \mathcal{B}_\mathcal{G}^{partial}\}$
19: **return** $\mathcal{C}$

only. It does not depend on the target placement $\mathcal{P}$, since no matter the target placement, if the robot is not able to pick the object, then the action will still be infeasible. On the other hand, when the infeasibility is due to the *Place* component, the infeasibility depends on both the current configuration of the blocked object $s(\mathcal{O}_{blocked})$ and the target placement $\mathcal{P}$, as well as the current configuration of the objects in $\mathcal{B}_\mathcal{G}^{full}$ and $\mathcal{B}_\mathcal{G}^{partial}$. Indeed, considering the target placement only is not sufficient, since the only grasps (and hence inverse kinematics solutions and motions) tested at the *Place* stage are those that are feasible at the *Pick* component, given the current configuration of the manipulated object and the potentially obstructing objects. This means that if new grasps are made feasible at the *Pick* stage, then the *Place* stage can become feasible as well, even if the infeasibility causes of the *Place* component are still present. In order to differentiate between these two cases, we define $t \in \{Pick, Place\}$ as the type of infeasiblility constraint. Given the above definitions, we can now define an infeasibility constraint as follows:

$$\mathcal{C} = \{t, \mathcal{R}, \mathcal{O}_{blocked}, s, \mathcal{P}, \mathcal{I}_\mathcal{G}, \mathcal{B}_\mathcal{G}^{full}, \mathcal{B}_\mathcal{G}^{partial}\} \tag{6.4}$$

The constraint construction process is detailed in the function *buildConstraint* (cf. Algorithm 17). For each grasp type $\gamma \in \mathcal{G}$, if no IK solutions were found or one of the objects **fully** obstructing γ is fixed, we set $\mathcal{I}_\mathcal{G}(\gamma)$ to True, otherwise, we set $\mathcal{I}_\mathcal{G}(\gamma)$ to False (cf. lines 4-9). We then populate $\mathcal{B}_\mathcal{G}^{full}(\gamma)$ and $\mathcal{B}_\mathcal{G}^{partial}(\gamma)$ with the movable objects obstructing γ depending on whether they fully or partially obstruct it (cf. lines 10-16). If the infeasibility is due to the *Pick* component, the constraint is recorded as a *Pick* constraint. Otherwise, it is recorded as a *Place* constraint. The constraint is then returned to be used as a planning constraint during the search.

6.2.2 Hard Constraints Vs. Soft Constraints

In this work, we have two different sources of infeasibility causes: the ones provided by the geometric planner and those predicted by the **GRN** model. The infeasibility causes provided by the geometric planner are considered as hard constraints, meaning that they are certain to be true. On the other hand, the infeasibility causes predicted by **GRN** are considered as soft constraints, since they are approximations obtained by the neural network and may not always be accurate. Since hard constraints are certain to be true, they can be used to prune infeasible actions from the search tree. However, this is not the case for soft constraints, as it may lead discard feasible actions and hence prevent the planner from finding a solution. Be that as it may,

both hard and soft constraints can still be used to guide the search process by providing a better estimation of the cost-to-goal. During planning, we maintain two sets of constraints: ζ_{hard} and ζ_{soft}, which contain the hard and soft constraints, respectively.

Hard Constraints. Hard constraints can be constructed at the geometric planning stage, as

Algorithm 18 GeometricPlanner (Updated Alg. 14)

Input: $E, s, a, n_{IK}, \tau_{local}, \tau_{global}, \boldsymbol{p}^a_{\mathcal{G}}, \zeta_{hard}$ ▷ Environment, state, action, max. IK solutions, local and global timeout, grasp types feasibility, hard constraints set

$\mathcal{R}, \mathcal{O}, s(\mathcal{O}), \mathcal{P} \leftarrow extractActionParams(a)$
$\boldsymbol{\kappa_{\mathcal{G}}} \leftarrow \mathbf{0}$, $\boldsymbol{\rho_{\mathcal{G}}} \leftarrow \mathbf{0}$, $\boldsymbol{n_{\mathcal{G}}} \leftarrow 0$
while τ_{global} is not reached **do**
  $g \leftarrow sampleGrasp(\mathcal{O}, \boldsymbol{p}^a_{\mathcal{G}})$, $\gamma \leftarrow getGraspType(g)$, $n_{\mathcal{G}}(\gamma) \leftarrow n_{\mathcal{G}}(\gamma) + 1$
  $\mathcal{Q}^{Pick}_g \leftarrow solveIK(\mathcal{R}, g, s(\mathcal{O}), n_{IK})$
  if $\mathcal{Q}^{Pick}_g$ is empty **then**
    continue
  end if
  $\boldsymbol{\kappa_{\mathcal{G}}}(\gamma) \leftarrow \boldsymbol{\kappa_{\mathcal{G}}}(\gamma) + 1$
  for each $\boldsymbol{q}_{Pick}$ in $\mathcal{Q}^{Pick}_g$ **do**
    $collision, obstructors \leftarrow checkCollision(\mathcal{R}, \boldsymbol{q}_{Pick}, E)$
    if *collision* is True **then**
      for $\mathcal{O}' \in obstructors$ **do**
        $\boldsymbol{\rho_{\mathcal{G}}}(\gamma, \mathcal{O}') \leftarrow \boldsymbol{\rho_{\mathcal{G}}}(\gamma, \mathcal{O}') + 1$
      end for
      continue
    end if
    $m_{Pick} \leftarrow planMotion(\mathcal{R}, s(\mathcal{R}), \boldsymbol{q}_{Pick}, E, \tau_{local})$
    if m_{Pick} is infeasible **then**
      continue
    end if
    $\mathcal{Q}^{Place}_g \leftarrow solveIK(\mathcal{R}, g, \mathcal{P}, n_{IK})$
    for each $\boldsymbol{q}_{Place}$ in $\mathcal{Q}^{Place}_g$ **do**
      $collision, obstructors \leftarrow checkCollision(\mathcal{R}, \boldsymbol{q}_{Place}, E)$
      if *collision* is True **then**
        for $\mathcal{O}' \in obstructors$ **do**
          $\boldsymbol{\rho_{\mathcal{G}}}(\gamma, \mathcal{O}') \leftarrow \boldsymbol{\rho_{\mathcal{G}}}(\gamma, \mathcal{O}') + 1$
        end for
        continue
      end if
      $m_{Place} \leftarrow planMotion(\mathcal{R}, \boldsymbol{q}_{Pick}, \boldsymbol{q}_{Place}, E, \tau_{local})$
      if m_{Place} is infeasible **then**
        continue
      end if
      $m \leftarrow \{m_{Pick}, m_{Place}\}$
      return m
    end for
  end for
end while
for each $\gamma \in \mathcal{G}$ **do**
  if $\boldsymbol{\kappa_{\mathcal{G}}}(\gamma) > 0$ **then**
    $\boldsymbol{\kappa_{\mathcal{G}}}(\gamma) \leftarrow 1$
  end if
  for each $\mathcal{O}' \in obstructors$ **do**
    $\boldsymbol{\rho_{\mathcal{G}}}(\gamma, \mathcal{O}') \leftarrow \boldsymbol{\rho_{\mathcal{G}}}(\gamma, \mathcal{O}') / \boldsymbol{n_{\mathcal{G}}}(\gamma)$
  end for
end for
if Infeasibility is due to *Pick* **then**
  $t \leftarrow Pick$
else
  $t \leftarrow Place$
end if
$\mathcal{C} \leftarrow buildConstraint(t, \mathcal{R}, \mathcal{O}, s, \mathcal{P}, \boldsymbol{\kappa_{\mathcal{G}}}, \boldsymbol{\rho_{\mathcal{G}}}, obstructors)$
if $\mathcal{C} \notin \zeta_{hard}$ **then**
  $\zeta_{hard} \leftarrow \zeta_{hard} \cup \mathcal{C}$
end if
return Action Infeasible

shown in Algorithms 18. Similarly to the data annotation procedure detailed in Chapter 5 (Algorithm 15), during geometric planning, we record the presence of inverse kinematic solutions for each grasp type $\gamma \in \mathcal{G}$ as a binary value $\boldsymbol{\kappa}_{\mathcal{G}}(\gamma)$ (cf. lines 9, 41-43). We also record the objects obstructing each grasp type γ as well as the grasp obstruction ratio of each obstructing object $\mathcal{O}'$ as the value $\boldsymbol{\rho}_{\mathcal{G}}(\gamma, \mathcal{O}')$ (cf. lines 11-17, 24-30). For a grasp type $\gamma \in \mathcal{G}$, the grasp obstruction ratio is defined as the number of grasps obstructed by $\mathcal{O}'$ divided by the total number of grasps (cf. lines 44-46). If geometric planning fails, these values are given to the function *buildConstraint* that constructs the infeasibility constraint as detailed in Algorithm 17. Once the constraint constructed, we add it to the set of hard constraints ζ_{hard} if it is not already present (Algorithm 18, lines 53-56).

Soft Constraints. Soft constraints, on the other hand, are constructed at the feasibility prediction stage. As explained in Chapter 5, **GRN** predicts the feasibility of actions and grasps, as well as the infeasibility causes. These reasons of infeasibility are provided as a binary value $\boldsymbol{\kappa}_{\mathcal{G}}(\gamma)$ for each grasp type $\gamma \in \mathcal{G}$ and a grasp obstruction ratio $\boldsymbol{\rho}_{\mathcal{G}}(\gamma, \mathcal{O}')$ for each obstructing object $\mathcal{O}'$. During feasibility computation (cf. Algorithm 19), we query the **GRN** model to obtain these predictions for the *Pick* and *Place* components of the action, and potentially for the future *Pick* if it is a *Pass* action (cf. lines 2, 9, 21). If the predicted feasibility of each subaction

Algorithm 19 computeFeasibility (Updated Alg. 12)

Input: $E, s, a, \mathcal{T}_F, \zeta_{hard}, \zeta_{soft}$

1: $\mathcal{R}, \mathcal{O}, s(\mathcal{O}), \mathcal{P} \leftarrow extractActionParams(a)$
2: $[p_F^{Pick}, \boldsymbol{p}_{\mathcal{G}}^{Pick}, \boldsymbol{\kappa}_{\mathcal{G}}^{Pick}, \boldsymbol{\rho}_{\mathcal{G}}^{Pick}] \leftarrow$ **GRN**$(\mathcal{R}, \mathcal{O}, s(\mathcal{O}), E)$
3: **if** $p_F^{Pick} < \mathcal{T}_F$ **then**
4: $\quad \mathcal{C} \leftarrow buildConstraint(Pick, \mathcal{R}, \mathcal{O}, s, \mathcal{P}, \boldsymbol{\kappa}_{\mathcal{G}}^{Pick}, \boldsymbol{\rho}_{\mathcal{G}}^{Pick}, \mathcal{N}(\mathcal{O}))$
5: $\quad$ **if** $\mathcal{C} \notin \zeta_{hard}$ and $\mathcal{C} \notin \zeta_{soft}$ **then**
6: $\quad\quad \zeta_{soft} \leftarrow \zeta_{soft} \cup \mathcal{C}$
7: $\quad$ **end if**
8: **end if**
9: $[p_F^{Place}, \boldsymbol{p}_{\mathcal{G}}^{Place}, \boldsymbol{\kappa}_{\mathcal{G}}^{Place}, \boldsymbol{\rho}_{\mathcal{G}}^{Place}] \leftarrow$ **GRN**$(\mathcal{R}, \mathcal{O}, \mathcal{P}, E)$
10: **if** $p_F^{Place} < \mathcal{T}_F$ **then**
11: $\quad s' \leftarrow applyAction(s, a)$
12: $\quad \mathcal{C} \leftarrow buildConstraint(Pick, \mathcal{R}, \mathcal{O}, s', \mathcal{P}, \boldsymbol{\kappa}_{\mathcal{G}}^{Place}, \boldsymbol{\rho}_{\mathcal{G}}^{Place}, \mathcal{N}(\mathcal{O}))$
13: $\quad$ **if** $\mathcal{C} \notin \zeta_{hard}$ and $\mathcal{C} \notin \zeta_{soft}$ **then**
14: $\quad\quad \zeta_{soft} \leftarrow \zeta_{soft} \cup \mathcal{C}$
15: $\quad$ **end if**
16: **end if**
17: $\boldsymbol{p}_{\mathcal{G}}^{a} \leftarrow \boldsymbol{p}_{\mathcal{G}}^{Pick} \otimes \boldsymbol{p}_{\mathcal{G}}^{Place}$
18: $p_F^a \leftarrow p_F^{Pick} \times p_F^{Place} \times max(\boldsymbol{p}_{\mathcal{G}}^{a})$
19: **if** a is *Pass* **then**
20: $\quad \mathcal{R}' \leftarrow getReceivingRobot(a)$
21: $\quad [p_F^{Pick'}, \boldsymbol{p}_{\mathcal{G}}^{Pick'}, \boldsymbol{\kappa}_{\mathcal{G}}^{Pick'}, \boldsymbol{\rho}_{\mathcal{G}}^{Pick'}] \leftarrow$ **GRN**$(\mathcal{R}', \mathcal{O}, s(\mathcal{O}), E)$
22: $\quad$ **if** $p_F^{Pick'} < \mathcal{T}_F$ **then**
23: $\quad\quad s' \leftarrow applyAction(s, a)$
24: $\quad\quad \mathcal{C} \leftarrow buildConstraint(Pick, \mathcal{R}', \mathcal{O}, s', \mathcal{P}, \boldsymbol{\kappa}_{\mathcal{G}}^{Pick'}, \boldsymbol{\rho}_{\mathcal{G}}^{Pick'}, \mathcal{N}(\mathcal{O}))$
25: $\quad\quad$ **if** $\mathcal{C} \notin \zeta_{hard}$ and $\mathcal{C} \notin \zeta_{soft}$ **then**
26: $\quad\quad\quad \zeta_{soft} \leftarrow \zeta_{soft} \cup \mathcal{C}$
27: $\quad\quad$ **end if**
28: $\quad$ **end if**
29: $\quad p_{CF}^a \leftarrow p_F^a \times p_F^{Pick'}$
30: $\quad$ **return** $p_{CF}^a, \boldsymbol{p}_{\mathcal{G}}^a$
31: **else**
32: $\quad$ **return** $p_F^a, \boldsymbol{p}_{\mathcal{G}}^a$
33: **end if**
34: **return** *children*

is below a user-defined threshold $\mathcal{T}_F$, we call the function *buildConstraint* (Algorithm 17) to construct the infeasibility constraint (cf. lines 3-4, 10-12, 22-24). Note that all soft constraints are considered as *Pick* constraints. This is because, contrary to the geometric planner, **GRN** predicts the feasibility of the *Pick* and *Place* components independently, meaning that the reasons of infeasibility of the *Place* component do not depend on which grasps are feasible at the *Pick* component. For that reason, for the *Place* component, we first compute the next state s' after applying the action a at state s, and construct the constraint as a *Pick* constraint using s' instead of s (cf. lines 11-12). For *Pass* actions between robots, we use the receiving robot $\mathcal{R}'$ and the next state s' to construct the constraint (cf. lines 23-24). If the constraints obtained are not already present in the set of soft constraints ζ_{soft}, we add them to the set (cf. lines 5-7, 13-15, 25-27). The threshold $\mathcal{T}_F$ determines the feasibility score for an action to be considered infeasible. This threshold controls the number of soft constraints that are added to the set ζ_{soft}. If the threshold is too high, we may end up with too many constraints, which can slow down planning. On the other hand, if the threshold is too low, we may overlook important constraints that could help guide the search process.

Given the combinatorial complexity of TAMP problems, the number of hard and soft constraints can grow exponentially with the number of geometric planning calls and feasibility checks. This can lead to a large number of constraints that can slow down the planning process. To address this issue, we do not record constraints for *Temporary* actions[1]. Since the same *Temporary* placement is never sampled twice, recording constraints for these actions would not provide any useful information for the task planner.

Note that hard constraints are collected incrementally every time the task planner generates an infeasible task plan and calls the geometric planner. This means that early exploration of the search space can not leverage these constraints to guide planning towards a feasible solution. Moreover, collecting hard constraints comes at the cost of computationally expensive geometric planning calls. Soft constraints, on the other hand, are collected during the feasibility prediction stage thanks to **GRN**'s ability to predict infeasibility causes. These efficient feasibility checks are performed for each expanded node in the search tree, meaning that soft constraints can be constructed at the earliest stages of the search process. This allows the task planner to leverage these constraints from the beginning of the search to guide the exploration towards a feasible solution and accelerate the planning process.

6.3 Informed Backtracking

Now that we have defined how infeasibility causes are recoded as planning constraints, we can use them to improve the backtracking process and better guide the search towards a geometrically feasible solution. We propose to leverage these constraints in two ways: by using hard constraints to prune the search tree more effectively, and by using both hard and soft constraints to compute a more accurate cost-to-goal.

6.3.1 Informed Search Tree Pruning

The first way we leverage these constraints is by using them to prune the search tree more effectively. In the reference TAMP algorithm, search tree pruning is performed after an unsuccessful geometric planning attempt, by removing all children nodes of the infeasible action. However, other actions that involve the same infeasibility causes can still be present in the search tree, leading to repeated geometric planning failures. By using the infeasibility constraints, we can

[1] Only in cases where infeasibility is due to the sampled *Temporary* Placement.

Algorithm 20 verifiesConstraint

Input: s, a, $\mathcal{C}$ ▷ state, action, constraint

1: **if** constraintApplies(s, a, $\mathcal{C}$) is False **then**
2: **return** True
3: **end if**
4: $s_{\mathcal{C}}, \mathcal{I}_{\mathcal{G}}, \mathcal{B}_{\mathcal{G}}^{full}, \mathcal{B}_{\mathcal{G}}^{partial} \leftarrow$ extractInfeasibilityInfo($\mathcal{C}$)
5: **for** each $\gamma \in \mathcal{G}$ **do**
6: **if** $\mathcal{I}_{\mathcal{G}}(\gamma) = True$ **then** ▷ all grasps of type γ are infeasible and unrectifiable
7: **continue**
8: **end if**
9: **if** $\mathcal{B}_{\mathcal{G}}^{full}(\gamma) \neq \emptyset$ **then**
10: $moved \leftarrow 0$
11: **for** each $\mathcal{O} \in \mathcal{B}_{\mathcal{G}}^{full}(\gamma)$ **do**
12: **if** $s(\mathcal{O}) \neq s_{\mathcal{C}}(\mathcal{O})$ **then**
13: $moved \leftarrow moved + 1$
14: **end if**
15: **end for**
16: **if** $moved = |\mathcal{B}_{\mathcal{G}}^{full}(\gamma)|$ **then** ▷ all objects in $\mathcal{B}_{\mathcal{G}}^{full}(\gamma)$ have been moved
17: **return** True
18: **end if**
19: **else if** $\mathcal{B}_{\mathcal{G}}^{partial}(\gamma) \neq \emptyset$ **then**
20: **for** each $\mathcal{O} \in \mathcal{B}_{\mathcal{G}}^{partial}(\gamma)$ **do**
21: **if** $s(\mathcal{O}) \neq s_{\mathcal{C}}(\mathcal{O})$ **then** ▷ At least one object in $\mathcal{B}_{\mathcal{G}}^{partial}(\gamma)$ has been moved
22: **return** True
23: **end if**
24: **end for**
25: **end if**
26: **end for**
27: **return** False

Algorithm 21 verifiesAllConstraints

Input: s, ζ_{hard} ▷ state, constraints

1: **if** s is initial state **then**
2: **return** True
3: **end if**
4: $a \leftarrow getLastAction(s)$
5: **for** each $\mathcal{C} \in \zeta_{hard}$ **do**
6: **if** verifiesConstraint(s, a, $\mathcal{C}$) is False **then**
7: **return** False
8: **end if**
9: **end for**
10: $s' \leftarrow getParent(s)$
11: **return** verifiesAllConstraints(s', ζ_{hard})

prune all actions that involve the same infeasibility causes. This can be done by checking all the edges in the search tree against the infeasibility constraints, and removing the children of actions that do not verify them.

In order to do this, we need a method to check if an action verifies a given constraint. A constraint is verified by an action if the constraint applies to the action, and the infeasibility causes of at least one grasp type have been rectified. This is done by the function $verifiesConstraint$ detailed in Algorithm 20. Given a state s, an action a and a constraint $\mathcal{C}$, we first check if the constraint applies to the action . The constraint C applies to the action a if the robot $\mathcal{R}$ performing the action, the object $\mathcal{O}$ manipulated by the action, and the configuration of that object at the state s denoted $s(\mathcal{O})$ are the same as the ones recorded in the constraint denoted $\mathcal{R}_{\mathcal{C}}$, $\mathcal{O}_{\mathcal{C}}$ and $s_{\mathcal{C}}(\mathcal{O})$, respectively. If the constraint is a *Place* constraint, then the target placement $\mathcal{P}$ of the action must also be the same as the placement $\mathcal{P}_{\mathcal{C}}$ recorded in the constraint. If the constraint does not apply to the action, we consider that it is verified automatically (cf. line 1-3).

Otherwise, if the constraint $\mathcal{C}$ applies to the action a, we then check if the infeasibility causes of at least one grasp type have been rectified. For each grasp type $\gamma \in \mathcal{G}$, we first check if the

grasp type is rectifiable (cf. line 6). If it is not, we continue to the next grasp type. If γ is rectifiable, we then check if the objects obstructing the grasp type γ have been moved. If γ has fully obstructing objects, we check if all of them have been moved (cf. lines 9-18). If they have, then the constraint is verified. If γ has partially obstructing objects, we simply check if at least one of them has been moved (cf. lines 20-24). If that is the case, then the constraint is verified. If none of the grasp types have been rectified, it means that the constraint is not verified, and the action should be pruned from the search tree. A given state should be kept in the search tree only if all the actions leading to it verify all the constraints in the set of hard constraints ζ_{hard}. The function $verifiesAllConstraints$ defined in Algorithm 21 performs this check recursively, starting from a given state s and going up the tree until the initial state is reached. If at any point a constraint is not verified, the function returns False, meaning that the state should be pruned from the search tree. If all actions in the branch verify all constraints, the function returns True, meaning that the state should be kept in the search tree.

This verification and pruning process can be computationally expensive, especially as the size of the search tree and the number of constraints increase. To alleviate this issue, we propose to perform the constraint-guided search tree pruning in a continuous manner. As shown in Algorithm 22, rather than traversing the search tree completely and pruning all nodes emanating from infeasible branches, we perform the pruning at each step of the search process. When selecting the next node to expand, we first check if it verifies all constraints and simply skip it if it does not, moving to the next node (cf. lines 6-7). This method allows us to save computation time by avoiding the verification of all nodes in the search tree, and focus only on those that are considered for expansion.

Algorithm 22 Task and motion planner (Updated Alg. 2)

Input: $E, s_0, s_{goal}, n_{IK}, n_{goal}, n_{temp}, n_{pass}, \tau_{local}, \tau_{global}$
1: $\mathbf{\Gamma} \leftarrow buildRobotGraph(E)$
2: $\zeta_{hard} \leftarrow \emptyset$, $\zeta_{soft} \leftarrow \emptyset$ ▷ Set of hard constraints
3: $Q \leftarrow \{s_0\}$ ▷ Set of nodes to expand
4: **while** Solution not found **do**
5: $s \leftarrow argmin_{cost} Q$
6: **if** verifiesAllConstraints(s, ζ_{hard}) is False **then**
7: **continue**
8: **end if**
9: **if** $s \in s_{goal}$ **then**
10: $\mathcal{A} \leftarrow retrieveSolution(s)$
11: $\mathcal{M} \leftarrow \emptyset$
12: **for** each action a in $\mathcal{A}$ **do**
13: $m \leftarrow GeometricPlanner(E, s, a, n_{IK}, \tau_{local}, \tau_{global}, \zeta_{hard})$
14: **if** m is feasible **then**
15: $\mathcal{M} \leftarrow \mathcal{M} \cup m$
16: **else**
17: $Q \leftarrow updateSearchTree(Q, a, s_{goal}, E, \mathbf{\Gamma}, \zeta_{hard}, \zeta_{soft})$
18: **break**
19: **end if**
20: **end for**
21: **if** $\mathcal{M}$ is feasible **then**
22: **return** $\mathcal{A}$, $\mathcal{M}$
23: **end if**
24: **else**
25: $Q \leftarrow Q \cup findChildren(s, E, s_{goal}, \mathbf{\Gamma}, n_{goal}, n_{temp}, n_{pass}, \zeta_{hard}, \zeta_{soft})$
26: **end if**
27: **end while**

6.3.2 Informed Cost-to-Goal Computation

In addition to improving search tree pruning, we propose to use infeasibility constraints to compute a more accurate cost-to-goal. Up until now, the cost-to-goal has been computed as the number of objects which are not in their goal configuration, augmented by the number of potential *Pass* actions necessary. However, this measure tends to largely underestimate the true cost-to-goal. For instance, consider a TAMP problem which involves moving an object to a goal placement, with a second object preventing the robot from reaching the desired object. In this case, the reference planner (cf. Chapter 3) would consider the cost-to-goal as 1, since only one object is not in its goal configuration. However, the true cost-to-goal is 2, since the robot must first move the obstructing object before moving the desired object. By using the infeasibility constraints, we can compute a more accurate cost-to-goal by taking into account the infeasibility causes of each action. This increased accuracy in cost-to-goal estimation can greatly reduce the number of nodes expanded during the search. As a result, the number of neural feasibility checks and geometric planning calls is reduced, thus minimizing the overhead introduced by feasibility prediction and the overall planning time.

The proposed informed cost-to-goal computation method is detailed in Algorithm 23. The function *computeCostToGoal* remains similar to the one explained in Section 3.3.1. Given a state s, for each object $\mathcal{O}$ that is not in its goal configuration, we first compute the number of actions required to move the object to its goal configuration without considering infeasibility constraints (cf. lines 4-8). The novelty is the addition of a constraint-added cost, which considers the infeasibility causes of each object that has not reached its goal. This cost represents the minimum number of actions necessary to allow object $\mathcal{O}$ to be picked from its current configuration $s(\mathcal{O})$, and placed at its goal configuration $s_{goal}(\mathcal{O})$. Therefore, the constraint-added cost is computed in two steps. First, we find the number of actions necessary to allow the *Pick* subaction $\mathcal{C}_{Pick}$ (cf. line 10), and then we do the same for the *Place* subaction to obtain $\mathcal{C}_{place}$ (cf. lines 12-16). The latter is computed only if the goal configuration of the object is fully specified, since, in this context, planning constraints can only be leveraged on fully instantiated goal placements. For both cases, the constraint-added cost is computed in the same manner, by calling the function *computeConstraintsAddedCost* (Algorithm 25). The only difference resides in the relevant constraints to consider.

Algorithm 23 computeCostToGoal (Updated Alg. 7)

Input: $s, s_{goal}, E, \mathbf{\Gamma}, \zeta_{hard}, \zeta_{soft}$
1: $C_{ToGoal} \leftarrow 0$
2: **for** each movable object $\mathcal{O}$ in E **do**
3: **if** $s(\mathcal{O}) \notin s_{goal}(\mathcal{O})$ **then**
4: **if** $s(\mathcal{O})$ and $s_{goal}(\mathcal{O})$ are in the same robot's workspace **then**
5: $C_{ToGoal} \leftarrow C_{ToGoal} + 1$
6: **else**
7: $C_{ToGoal} \leftarrow C_{ToGoal} + BFS(\mathbf{\Gamma}, s(\mathcal{O}), s_{goal}(\mathcal{O}))$
8: **end if**
9: *// Number of actions necessary to allow the object to be picked*
10: $C_{Pick} \leftarrow computeConstraintsAddedCost(s, s_{goal}, E, \mathcal{O}, \zeta_{hard}, \zeta_{soft}, Pick)$
11: *// Number of actions necessary to allow the object to be placed at its goal*
12: **if** $s_{goal}(\mathcal{O})$ is fully specified **then**
13: $C_{Place} \leftarrow computeConstraintsAddedCost(s, s_{goal}, E, \mathcal{O}, \zeta_{hard}, \zeta_{soft}, Place)$
14: **else**
15: $C_{Place} \leftarrow 0$
16: **end if**
17: $C_{ToGoal} \leftarrow C_{ToGoal} + C_{Pick} + C_{Place}$
18: **end if**
19: **end for**
20: **return** C_{ToGoal}

Indeed, given the fact that there are two types of constraints, the set of relevant constraints to consider depends on the considered subaction. If the subaction considered is the *Pick* component, a relevant constraint is one that is of type *Pick* (i.e. $t_{\mathcal{C}} = Pick$) and applies to the object $\mathcal{O}$ at state s, meaning that the object $\mathcal{O}$ and its configuration at state s are the same as the ones recorded in the constraint. More formally, the relevant constraints for the *Pick* subaction $\zeta_{relevant}|_{Pick}$ are defined as:

$$\zeta_{relevant}|_{Pick} = \{\mathcal{C} \in \zeta_{hard} \cup \zeta_{soft} \mid t_{\mathcal{C}} = Pick \wedge \mathcal{O}_{\mathcal{C}} = \mathcal{O} \wedge s_{\mathcal{C}}(\mathcal{O}) = s(\mathcal{O})\} \tag{6.5}$$

where $\wedge$ denotes the logical AND operator. On the other hand, when the subaction considered is the *Place* component, both *Pick* and *Place* constraints that apply to the object $\mathcal{O}$ at its goal configuration $s_{goal}(\mathcal{O})$ should be considered. Hence, in this case, the relevant constraints are two-fold:

- Constraints of type *Pick* that apply to the object $\mathcal{O}$ at its goal placement $s_{goal}(\mathcal{O})$, meaning that the object $\mathcal{O}$ and its goal configuration are the same as the ones recorded in the constraint.
- Constraints of type *Place* that apply to the object $\mathcal{O}$ at state s and for which the recorded target placement $\mathcal{P}_{\mathcal{C}}$ is the same as the object's goal configuration $s_{goal}(\mathcal{O})$.

Formally, the relevant constraints for the *Place* subaction $\zeta_{relevant}|_{Place}$ are defined as:

$$\zeta_{relevant}|_{Place} = \left\{ \mathcal{C} \in \zeta_{hard} \cup \zeta_{soft} \;\middle|\; \begin{array}{l} [t_{\mathcal{C}} = Pick \wedge \mathcal{O}_{\mathcal{C}} = \mathcal{O} \wedge s_{\mathcal{C}}(\mathcal{O}) = s_{goal}(\mathcal{O})] \\ \vee\, [t_{\mathcal{C}} = Place \wedge \mathcal{O}_{\mathcal{C}} = \mathcal{O} \wedge s_{\mathcal{C}}(\mathcal{O}) = s(\mathcal{O}) \wedge \mathcal{P}_{\mathcal{C}} = s_{goal}(\mathcal{O})] \end{array} \right\} \tag{6.6}$$

where $\vee$ denotes the logical OR operator.

Once the relevant constraints are found, the function *computeConstraintsAddedCost* (cf. Algorithm 25) proceeds to find the minimum number of actions required to rectify the infeasibility causes of the considered subaction. We first check if an unconstrained robot can perform the tested subaction. If at least one robot that is not constrained by the relevant constraints found can achieve the considered subaction, then the minimum number of actions to allow that subaction is 0 (cf. lines 2-6). Otherwise, computing the constraints-added cost operates as a recursive branch-and-bound search, where the minimum number of actions is found by considering all the possible ways to rectify the infeasibility causes. This amounts to looping over the relevant constraints, grasp types, and obstructing objects (branching), and recursively calling the function *computeConstraintsAddedCost* to find the minimum number of actions required to rectify the infeasibility causes of the obstructing objects (bounding). This recursion allows our method to handle deeply non-monotonic scenarios (e.g, object A obstructs object B, which obstructs object C, etc).

Let c_{min} be the constraint-added cost to find, which we initially set to ∞ (cf. line 7). For each relevant constraint $\mathcal{C}$, we iterate over each grasp type γ and check if the grasp type is rectifiable (cf. lines 11-13). If it is not, we continue to the next grasp type. If the grasp type is rectifiable, we check if the grasp type has fully obstructing objects. Otherwise, we explore the reasons of infeasibility of γ, in order to find the minimum number of actions required to rectify it denoted c_γ. Here, two cases can be distinguished:

- **Grasp type γ has fully obstructing objects:** In this case, all objects in $\mathcal{B}_{\mathcal{G}}^{full}(\gamma)$ must be moved before the grasp type's infeasibility can be considered rectified. Hence, c_γ is

used as a counter to keep track of the number of actions required to rectify the grasp type. Thus, in this case, c_γ can be formulated as:

$$c_\gamma = \sum_{\mathcal{O}' \in \mathcal{B}_\mathcal{G}^{full}(\gamma)} c_{\mathcal{O}'} \tag{6.7}$$

where $c_{\mathcal{O}'}$ is the minimum number of actions required to rectify picking a fully obstructing object $\mathcal{O}'$. For each obstructor $\mathcal{O}'$ for which $s(\mathcal{O}') = s_\mathcal{C}(\mathcal{O}')$ (i.e $\mathcal{O}'$ is still obstructing $\mathcal{O}$ at state s), we perform a recursive call to the function *computeConstraintsAddedCost* to find $c_{\mathcal{O}'}$ (cf. line 21). If $\mathcal{O}'$ has a specified goal and is at its goal at state s, then $\mathcal{O}'$ must be moved then put back to its goal configuration. In this case, $c_{\mathcal{O}'}$ is incremented by 2 actions (cf. lines 22-23). If $\mathcal{O}'$ has a specified goal but is not at its goal at state s, then moving $\mathcal{O}'$ simply means moving it to its goal configuration, which is already accounted for in the cost-to-goal (cf. Alg. 23, lines 4-8). Hence, there is no need to increment $c_{\mathcal{O}'}$. On the other hand, if $\mathcal{O}'$ has no specified goal, then moving $\mathcal{O}'$ simply means moving it to a configuration that allows moving $\mathcal{O}$. In this case, $c_{\mathcal{O}'}$ is incremented by 1 action (cf. lines 24-25). Finally, $c_{\mathcal{O}'}$ is added to c_γ (cf. line 28), and we proceed to the next fully obstructing object.

- **Grasp type γ has partially obstructing objects:** In this case, contrary to fully obstructing objects, we only need to move one of the partially obstructing objects to rectify the grasp type's infeasibility. However, each partially obstructing object requires a different number of actions to be moved. Hence, we need to evaluate each one of these object and choose the one that requires the least number of actions to be moved. Therefore, in this case, c_γ can be expressed as:

$$c_\gamma = \min_{\mathcal{O}' \in \mathcal{B}_\mathcal{G}^{partial}(\gamma)} c_{\mathcal{O}'} \tag{6.8}$$

  For each partially obstructing object $\mathcal{O}'$, we check if the object has already been moved at state s. If it has, then no action is required to rectify the grasp type's infeasibility, and we can move on to next grasp type (cf. lines 33-36). If it has not, we recursively compute the number of actions $c_{\mathcal{O}'}$ required to allow $\mathcal{O}'$ to be picked (cf. line 37). Similarly to fully obstructing objects, we increment $c_{\mathcal{O}'}$ by 2 actions if $\mathcal{O}'$ has a specified goal and is at its goal at state s (cf. lines 38-39), or by 1 action if $\mathcal{O}'$ has no specified goal (cf. lines 40-41). Finally, we compare $c_{\mathcal{O}'}$ to the current minimum c_γ and update the latter if $c_{\mathcal{O}'}$ is smaller (cf. line 43).

Once the number of actions required to rectify the grasp type's infeasibility c_γ is found, we compare it to the current global minimum c_{min} and update the latter if c_γ is smaller (cf. line 46). At the end of the process, c_{min} represents the minimum number of actions required to rectify the infeasibility causes of picking the object $\mathcal{O}$ at state s, or placing it at its goal configuration s_{goal}, depending on the subaction considered. Theses constraint-added costs are then added to the cost-to-goal (Alg. 23, line 17), and the process is repeated for the next object that is not in its goal configuration. Note that during the computation of the constraint-added costs, it is possible to encounter a cycle in the constraints (e.g. object A obstructs object B, which obstructs object A). To avoid infinite recursion, we keep track of the pending objects in the recursion stack, and consider the constraint-added cost of an object as infinite if it is already in the stack. Furthermore, to avoid counting the same actions multiple times, we keep track of the objects that have already been moved in the recursion stack, and do not count them more than once. These two mechanisms ensure that the constraint-added costs are computed correctly and efficiently, making sure that the number of actions required to rectify the infeasibility causes is not overestimated. Since the informed cost-to-goal computation depends on the infeasibility

constraints, in addition to being computed at each node expansion during the search (Alg. 22, line 25), it is important to update the cost of each node in the search tree after each geometric planning failure. This is done during backtracking (cf. Alg. 22, line 17), following Algorithm 24.

Algorithm 24 updateSearchTree (Updated Alg. 3)

Input: $Q, a_{infeasible}, s_{goal}, E, \mathbf{\Gamma}, \zeta_{hard}, \zeta_{soft}$ ▷ List of open nodes, Infeasible action
1: **for** each s in Q **do**
2: **if** s is a descendant of $a_{infeasible}$ **then**
3: $Q \leftarrow Q \setminus s$
4: **continue**
5: **end if**
6: $C_{ToGoal}(s) \leftarrow computeCost(s, s_{goal}, E, \mathbf{\Gamma}, \zeta_{hard}, \zeta_{soft})$
7: **end for**
8: **return** Q

Algorithm 25 computeConstraintsAddedCost

Input: $s, s_{goal}, \mathcal{O}, E, \zeta_{hard}, \zeta_{soft}, subaction$ ▷ state, goal, object, env, constraints, subaction $\in \{Pick, Place\}$
1: $\zeta_{relevant} \leftarrow getRelevantConstraints(\mathcal{O}, \zeta_{hard}, \zeta_{soft}, subaction)$ ▷ If no relevant constraints are found, we return 0
2: **if** $subaction = Pick$ and $\exists \mathcal{R} \in findReachingRobots(s(\mathcal{O}), E)$ such that $\mathcal{R}$ is not constrained **then**
3: **return** 0 ▷ Object can be picked by non-constrained robot
4: **else if** $subaction = Place$ and $\exists \mathcal{R} \in findReachingRobots(s_{goal}(\mathcal{O}), E)$ such that $\mathcal{R}$ is not constrained **then**
5: **return** 0 ▷ Object can be placed by non-constrained robot
6: **end if**
7: $c_{min} \leftarrow \infty$
8: **for** each $\mathcal{C} \in \zeta_{relevant}$ **do**
9: $s_{\mathcal{C}}, \mathcal{I}_{\mathcal{G}}, \mathcal{B}_{\mathcal{G}}^{full}, \mathcal{B}_{\mathcal{G}}^{partial} \leftarrow$ extractInfeasibilityInfo($\mathcal{C}$)
10: **for** each $\gamma \in \mathcal{G}$ **do**
11: **if** $\mathcal{I}_{\mathcal{G}}(\gamma) = True$ **then** ▷ Grasp type γ is unrectifiable
12: **continue**
13: **end if**
14: **if** $\mathcal{B}_{\mathcal{G}}^{full}(\gamma) \neq \emptyset$ **then**
15: $c_\gamma \leftarrow 0$ ▷ Number of actions to rectify grasp type γ
16: **for** each $\mathcal{O}' \in \mathcal{B}_{\mathcal{G}}^{full}(\gamma)$ **do** ▷ All objects in $\mathcal{B}_{\mathcal{G}}^{full}(\gamma)$ must be moved
17: **if** $s(\mathcal{O}') \neq s_{\mathcal{C}}(\mathcal{O}')$ **then** ▷ Obstructing object has been moved
18: **continue**
19: **end if**
20: *// Recursively find the number of actions to rectify picking $\mathcal{O}'$*
21: $c_{\mathcal{O}'} \leftarrow computeConstraintsAddedCost(s, s_{goal}, E, \mathcal{O}', \zeta_{hard}, \zeta_{soft}, Pick)$
22: **if** $\mathcal{O}'$ has a specified goal and $s(\mathcal{O}') \in s_{goal}(\mathcal{O}')$ **then**
23: $c_{\mathcal{O}'} \leftarrow c_{\mathcal{O}'} + 2$ ▷ Object is at its goal, must be removed then put back
24: **else if** $\mathcal{O}'$ has no specified goal **then**
25: $c_{\mathcal{O}'} \leftarrow c_{\mathcal{O}'} + 1$ ▷ Object must be removed without put back
26: **end if**
27: *//if $\mathcal{O}'$ has a goal but has not reached it, its removal is already counted by Alg 23 (cf. lines 4-8)*
28: $c_\gamma \leftarrow c_\gamma + c_{\mathcal{O}'}$
29: **end for**
30: **else if** $\mathcal{B}_{\mathcal{G}}^{partial}(\gamma) \neq \emptyset$ **then**
31: $c_\gamma \leftarrow \infty$
32: **for** each $\mathcal{O}' \in \mathcal{B}_{\mathcal{G}}^{partial}(\gamma)$ **do** ▷ Only one object in $\mathcal{B}_{\mathcal{G}}^{partial}(\gamma)$ must be moved
33: **if** $s(\mathcal{O}') \neq s_{\mathcal{C}}(\mathcal{O}')$ **then** ▷ One of the obstructing objects was already moved
34: $c_\gamma \leftarrow 0$
35: **break**
36: **end if**
37: $c_{\mathcal{O}'} \leftarrow computeConstraintsAddedCost(s, s_{goal}, E, \mathcal{O}', \zeta_{hard}, \zeta_{soft}, Pick)$
38: **if** $\mathcal{O}'$ has a specified goal and $s(\mathcal{O}') \in s_{goal}(\mathcal{O}')$ **then**
39: $c_{\mathcal{O}'} \leftarrow c_{\mathcal{O}'} + 2$
40: **else if** $\mathcal{O}'$ has no specified goal **then**
41: $c_{\mathcal{O}'} \leftarrow c_{\mathcal{O}'} + 1$
42: **end if**
43: $c_\gamma \leftarrow min(c_\gamma, c_{\mathcal{O}'})$
44: **end for**
45: **end if**
46: $c_{min} \leftarrow min(c_{min}, c_\gamma)$
47: **end for**
48: **end for**
49: **return** c_{min}

Due to the recursive nature of Algorithm 25, the complexity of computing the constraint-added cost is exponential in the worst-case scenario. In fact, the function loops over three components:

- The relevant constraints $\zeta_{relevant}$, which can be equal to the total number of constraints in $\zeta_{hard} \cup \zeta_{soft}$ in the worst-case scenario.
- The rectifiable grasp types which have a maximum size of 6 in our case.
- The obstructing objects $\mathcal{B}_{\mathcal{G}}^{full}(\gamma)$ and $\mathcal{B}_{\mathcal{G}}^{partial}(\gamma)$ which can be equal to the total number of objects (except the object considered) in the worst-case scenario.

In addition, the function is called recursively for each obstructing object. This recursion can itself have a maximum depth equal to the total number of objects in the environment. Therefore, a *very* pessimistic upper bound to the complexity of the function is:

$$O((|\zeta_{hard} \cup \zeta_{soft}| \cdot 6 \cdot (n_{movable} - 1))^{n_{movable}})$$

where $|\zeta_{hard} \cup \zeta_{soft}|$ is the total number of constraints and $n_{movable}$ is the number of movable objects.

However, in practice, the number of constraints and obstructing objects is usually much smaller than the total number of objects in the environment. Moreover, the recursive calls are usually limited to a few objects, since the recursion stack keeps track of the objects that have already been moved. As a result, the actual complexity of the function is much lower than the upper bound, and the computation of the constraint-added cost is done in a reasonable amount of time. Moreover, since the feasibility cost $C_{Feasibility}$ defined in equation 4.17 (cf. Chapter 4) tends to be very large for actions predicted as infeasible, there is no need to compute the constraint-added cost for these actions, as it would be negligible compared to the feasibility cost. The constraint-added cost is thus only computed for actions that are predicted as feasible by the neural network, which further reduces the computation time. Note that compared to the complexity of the task and motion planner (cf. Section 3.3.2), which is exponential in the number of objects and the number of robots, the complexity of the informed cost-to-goal computation is exponential in the number of movable objects only. The number of robots affects the complexity of the constraints-added cost computation only through the number of relevant constraints.

6.4 Results

6.4.1 Experimental Setup

In order to evaluate the performance of the proposed task and motion planner with informed backtracking compared to the baseline planner and the feasibility-informed planner using **AGFPNet**. We reuse the same experimental setup described in Sections 3.5.1 and 4.5.1. In order to measure the impact of the introduced informed backtracking, we present two versions of each planner: the first one does not use infeasibility constraints, while the second one leverages infeasibility constraints (IC) for informed backtracking. As a result, we obtain six variants of the planner:

- **Baseline**: The original baseline planner described in Section 3.3, which does not feasibility prediction nor informed backtracking.
- **Baseline+IC**: The baseline planner with informed backtracking using hard infeasibility constraints (IC).
- **AGFPNet**: The feasibility-informed planner presented in Chapter 4, which uses **AGFPNet** for feasibility prediction, but does not use informed backtracking.

- **AGFPNet+IC**: The **AGFPNet** informed planner leveraging both feasibility prediction and informed backtracking using hard infeasibility constraints (IC).
- **GRN**: A version of the feasibility-informed planner replacing **AGFPNet** with **GRN** for feasibility prediction, but without leveraging infeasibility constraints.
- **GRN+IC**: The TAMP planner proposed in this chapter, which uses **GRN** for predicting action and grasp feasibility, as well as reasons of infeasibility, and leverages informed backtracking using both hard and soft infeasibility constraints.

Note that in the case of the **Baseline+IC** and **AGFPNet+IC** planners, the only constraints used are hard constraints, since the only source of infeasibility causes is the geometric planner. In contrast, the **GRN+IC** planner uses both hard and soft constraints, since the reasons of infeasibility are both predicted by the neural network and obtained from the geometric planner in case of an infeasible geometric planing call. The experiments are conducted on the same benchmark TAMP problems described in Section 3.4.

6.4.2 Planning Performance on the Benchmark TAMP Problems

Tables 6.1 and 6.2 compare the planning performances of the different planners on each one of the benchmark TAMP problems, while Figure 6.2 shows the decomposition of the total planning time. The planning performances are averaged over 10 trials. In order to review the obtained results in detail, we focus on different aspects of the planning performances.

GRN-based Feasibility-Informed Planner. We first focus on the performance gain yielded by replacing **AGFPNet** with **GRN** for feasibility prediction, without considering informed backtracking. Table 6.1 shows that the **GRN** planner outperforms both the baseline planner and the **AGFPNet** planner on all problems, and achieve 100% success rate on 5 of the 6 benchmarks. Thanks to the more accurate feasibility prediction of the **GRN** model, as well as it much faster inference time, the **GRN**-based planner is able to reduce planning time by at least 60% on the **Access** and **Clear** problems compared to the **AGFPNet** planner. On the **Sort** problem, in addition to a 10% improvement in success rate, the **GRN** planner is able to find a geometrically feasible solution 7 times faster than the **AGFPNet** planner. **GRN**'s higher accuracy in feasibility prediction translates into less infeasible task plans generated, which in turn reduces the number of geometric planning calls as shown in Table 6.2. Moreover, the **GRN** planner expands less nodes than the **AGFPNet** planner, hence requiring much less collision and feasibility checks. Therefore, the **GRN** planner is not only able to perform faster feasibility checks, it also requires less of them, which results in a significant reduction in the feasibility prediction time. This is illustrated in Figure 6.2, where the feasibility prediction time of the **GRN** planner is negligible compared to the one yielded by the **AGFPNet** planner.

Impact of Informed Backtracking. The second aspect we consider is the impact of informed backtracking on the planning performances. As shown in Table 6.1, leveraging infeasibility constraints improves planning performance for all planners. On the **Access** problem, the use of informed backtracking increases the success rate of the baseline planner from 0% to 80%, and from 90% to 100% for the **AGFPNet** planner. Also, thanks to informed backtracking, the success rate of the baseline planner improves by 50% on the **Clear** problem, 70% on the **Sort** problem, and 100% on the **Longswap** and **Middleman-3** problems. Using infeasibility constraints also increases the success rate of the **AGFPNet** planner on the **Longswap** problem from 90% to 100%. This improvement in success rate comes at a relatively low overhead in planning time, sometimes even decreasing the total planning time. For instance, the **AGFPNet+IC** planner is able to reduce the planning time by 12% on the **Clear** domain and by 7% on the

Sort problem. Figure 6.2 shows that the informed backtracking time is significant only for the **Baseline+IC** planner, while being negligible for the **AGFPNet+IC** planner. This is expected since feasibility prediction time already filters out most of the infeasible actions, leading to a low number of infeasible geometric planning calls, thus reducing the number of hard constraints constructed. In fact, in the case of the **AGFPNet+IC** planner, feasibility prediction and informed backtracking are not complementary, but rather concurrent. Feasibility prediction aims to filter out all infeasible actions in order to find a geometrically feasible task plan, eliminating infeasible geometric planning calls. Informed backtracking using hard constraints, on the other hand, needs these infeasible geometric planning calls to construct the infeasibility constraints and thus better guide the search. This is the main motivation behind infeasibility cause prediction introduced by **GRN** (cf. Chapter 5), which allows the planner to construct soft infeasibility constraints without the need for infeasible geometric planning calls.

Table 6.1: Planning performances of the different planners on the benchmark TAMP problems averaged over 10 trials, using a timeout of 300s.

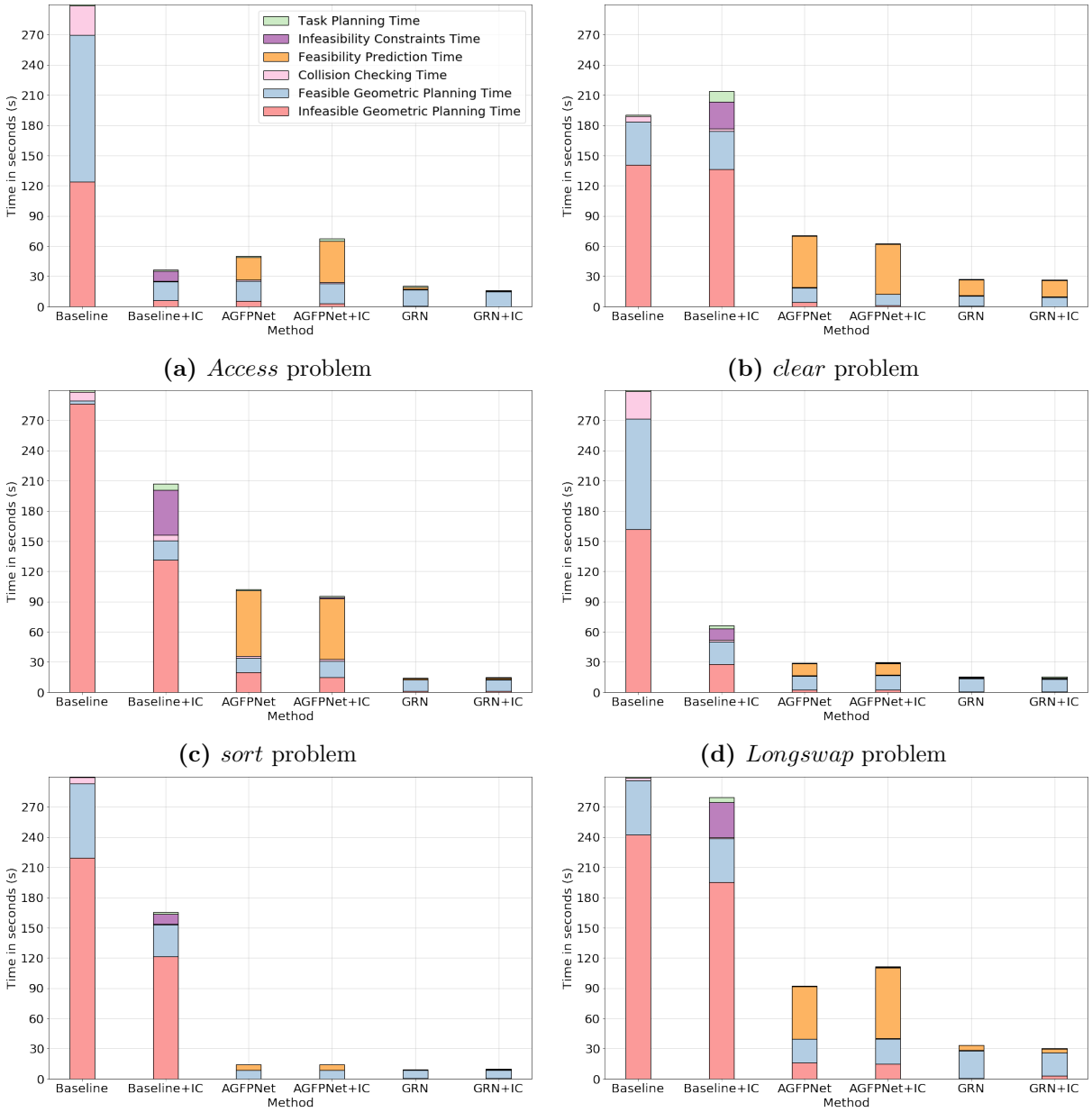
Problem	Method	Success Rate	Total Planning Time (s)	Infeasible Task Plans	Expanded Nodes
Access	Baseline	0%	> 300	> 204.4 (+/-18.4)	> 6848.0 (+/-832.3)
	Baseline+IC	80%	36.7 (+/-16.1)	6.4 (+/-1.0)	632.4 (+/-505.7)
	AGFPNet	90%	50.1 (+/-49.3)	2.6 (+/-5.6)	2067.2 (+/-2978.9)
	AGFPNet+IC	100%	67.3 (+/-75.0)	2.0 (+/-2.9)	3926.0 (+/-7747.6)
	GRN	100%	20.2 (+/-3.9)	0.2 (+/-0.4)	732.7 (+/-204.7)
	GRN+IC	100%	16.2 (+/-2.3)	0.1 (+/-0.3)	108.4 (+/-14.8)
Clear	Baseline	30%	190.0 (+/-51.9)	124.7 (+/-20.2)	5873.3 (+/-869.2)
	Baseline+IC	80%	213.7 (+/-41.8)	40.1 (+/-6.5)	4158.1 (+/-715.7)
	AGFPNet	100%	70.5 (+/-27.9)	3.9 (+/-6.6)	1278.4 (+/-604.7)
	AGFPNet+IC	100%	62.5 (+/-17.1)	0.9 (+/-0.7)	1267.1 (+/-460.2)
	GRN	100%	26.9 (+/-8.1)	0.6 (+/-1.8)	3453.7 (+/-1217.2)
	GRN+IC	100%	26.4 (+/-6.4)	0.1 (+/-0.3)	3404.5 (+/-1177.3)
Sort	Baseline	0%	> 300	> 49.0 (+/-5.8)	> 7452.6 (+/-704.9)
	Baseline+IC	70%	206.8 (+/-57.2)	28.3 (+/-6.2)	3849.0 (+/-514.4)
	AGFPNet	90%	102.1 (+/-47.3)	2.6 (+/-1.5)	2443.0 (+/-2396.6)
	AGFPNet+IC	90%	95.2 (+/-43.9)	2.1 (+/-0.9)	2185.3 (+/-2063.9)
	GRN	100%	14.4 (+/-3.7)	0.3 (+/-0.5)	322.8 (+/-117.1)
	GRN+IC	100%	14.8 (+/-3.2)	0.6 (+/-0.8)	320.5 (+/-106.6)
Longswap	Baseline	0%	> 300	> 200.7 (+/-34.2)	> 4674.4 (+/-2590.8)
	Baseline+IC	100%	66.4 (+/-23.0)	9.9 (+/-3.7)	874.9 (+/-295.2)
	AGFPNet	90%	29.3 (+/-8.1)	0.7 (+/-0.7)	321.2 (+/-103.0)
	AGFPNet+IC	100%	29.7 (+/-6.1)	0.8 (+/-0.8)	332.5 (+/-104.1)
	GRN	100%	15.6 (+/-2.7)	0.5 (+/-1.0)	221.1 (+/-48.7)
	GRN+IC	100%	15.2 (+/-2.6)	0.3 (+/-0.6)	214.4 (+/-39.1)
Middleman-3	Baseline	0%	> 300	> 155.8 (+/-18.4)	> 2394.2 (+/-372.0)
	Baseline+IC	100%	165.7 (+/-28.4)	20.8 (+/-3.6)	1029.1 (+/-152.7)
	AGFPNet	100%	14.3 (+/-1.4)	0.0 (+/-0.0)	166.2 (+/-31.1)
	AGFPNet+IC	100%	14.4 (+/-1.5)	0.0 (+/-0.0)	166.2 (+/-31.1)
	GRN	100%	9.4 (+/-0.9)	0.1 (+/-0.3)	158.7 (+/-45.1)
	GRN+IC	100%	9.6 (+/-1.2)	0.1 (+/-0.3)	158.7 (+/-45.1)
Middleman-5	Baseline	0%	> 300	> 116.1 (+/-40.1)	> 8649.3 (+/-2339.3)
	Baseline+IC	20%	279.8 (+/-8.4)	31.0 (+/-1.0)	4778.0 (+/-171.0)
	AGFPNet	80%	92.2 (+/-29.8)	1.5 (+/-0.9)	1162.5 (+/-577.5)
	AGFPNet+IC	80%	111.3 (+/-54.0)	1.5 (+/-0.9)	1064.0 (+/-370.3)
	GRN	80%	33.4 (+/-6.6)	0.3 (+/-0.4)	715.9 (+/-219.1)
	GRN+IC	100%	30.1 (+/-8.3)	0.6 (+/-0.8)	789.9 (+/-226.2)

GRN-based Planner with Informed Backtracking. Finally, we focus on the performance of the **GRN+IC** planner, which combines the **GRN** model for predicting action/grasp feasibility and infeasibility causes, with informed backtracking using both hard and soft infeasibility constraints. The **GRN+IC** planner achieves 100% success rate on all problems, outperforming all other planners. On the **Access** problem, the **GRN+IC** planner is able to find a geometrically feasible solution in 16s, which is at least 19 times lower than the baseline planner, 4 times lower than **AGFPNet+IC**, and 4s faster than the **GRN** planner. In most trials, the first task plan generated is a geometrically feasible one. Moreover, the improved cost-to-goal estimation provided by predicted soft constraints reduces the number of expanded nodes by 85% compared to the **GRN** planner which does not use informed backtracking. This directly translates into a significant reduction in the number of feasibility checks and collision checks. On the **Middleman-5** problem, the **GRN+IC** planner improves the planning time by 20% compared to the **AGFPNet**, **AGFPNet+IC** and **GRN** planners. Furthermore, the total planning time is reduced by 89% compared to **Baseline+IC**, 72% compared to **AGFPNet+IC**, and by 10% compared to the **GRN** planner. On this problem, however, the number of expanded nodes is not reduced, which means that the gain in performance is not mainly due to the improved cost-to-goal estimation, but rather to the informed search tree pruning. The lack in

Table 6.2: Planning statistics of the different planners on the benchmark TAMP problems, averaged over 10 trials.

Problem	Method	Geometric Planner Calls	Feasibility Checks	Collision Checks	Pruned Branches
Access	Baseline	> 315.1 (+/-28.3)	-	> 12355.7 (+/-1718.9)	> 6068.4 (+/-556.0)
	Baseline+IC	21.125 (+/-4.106)	-	812.1 (+/-526.5)	197.3 (+/-31.5)
	AGFPNet	16.7 (+/-12.0)	4134.4 (+/-5957.9)	2541.7 (+/-3404.3)	1229.0 (+/-3115.7)
	AGFPNet+IC	15.8 (+/-7.4)	7852.0 (+/-15495.1)	4443.2 (+/-8204.9)	3104.9 (+/-7849.7)
	GRN	11.7 (+/-1.4)	1465.4 (+/-409.4)	1274.4 (+/-367.5)	29.1 (+/-58.4)
	GRN+IC	11.1 (+/-0.3)	216.8 (+/-29.7)	178.4 (+/-20.4)	4.0 (+/-12.0)
Clear	Baseline	172.3 (+/-27.4)	-	13297.7 (+/-1926.6)	5279.7 (+/-782.7)
	Baseline+IC	79.8 (+/-9.6)	-	10043.3 (+/-1730.4)	3374.3 (+/-631.0)
	AGFPNet	19.4 (+/-12.2)	2996.0 (+/-1333.2)	3131.1 (+/-1373.0)	420.3 (+/-707.9)
	AGFPNet+IC	13.7 (+/-2.2)	2997.4 (+/-1076.2)	3148.6 (+/-1124.2)	232.1 (+/-306.6)
	GRN	11.8 (+/-4.5)	8292.8 (+/-2925.4)	8843.0 (+/-3110.9)	51.1 (+/-153.3)
	GRN+IC	10.3 (+/-0.9)	8177.9 (+/-2834.4)	8719.8 (+/-3015.3)	18.3 (+/-54.9)
Sort	Baseline	> 52.1 (+/-6.1)	-	> 25393.1 (+/-2909.4)	> 7439.7 (+/-712.4)
	Baseline+IC	49.0 (+/-10.5)	-	16091.1 (+/-2195.2)	3476.1 (+/-432.3)
	AGFPNet	15.9 (+/-2.3)	4951.0 (+/-4780.7)	6930.4 (+/-6184.6)	756.0 (+/-649.9)
	AGFPNet+IC	17.3 (+/-3.3)	4430.3 (+/-4111.3)	6200.7 (+/-5428.6)	493.4 (+/-365.9)
	GRN	11.9 (+/-1.4)	697.8 (+/-257.1)	852.5 (+/-330.9)	9.4 (+/-25.0)
	GRN+IC	12.7 (+/-2.2)	694.5 (+/-235.7)	841.8 (+/-297.9)	11.4 (+/-26.6)
Longswap	Baseline	> 319.3 (+/-39.0)	-	> 9775.0 (+/-5704.2)	> 4402.4 (+/-2622.2)
	Baseline+IC	30.2 (+/-9.2)	-	2037.3 (+/-709.7)	591.6 (+/-289.1)
	AGFPNet	13.9 (+/-1.1)	805.1 (+/-260.4)	760.3 (+/-253.9)	65.9 (+/-85.0)
	AGFPNet+IC	14.9 (+/-2.9)	835.0 (+/-264.8)	788.9 (+/-256.8)	68.6 (+/-81.0)
	GRN	13.5 (+/-2.4)	532.8 (+/-112.3)	508.6 (+/-105.8)	13.5 (+/-39.2)
	GRN+IC	13.3 (+/-2.3)	518.8 (+/-94.1)	494.5 (+/-81.9)	7.6 (+/-21.5)
Middleman-3	Baseline	> 225.2 (+/-26.1)	-	> 4984.3 (+/-814.2)	> 2259.7 (+/-362.7)
	Baseline+IC	41.7 (+/-8.6)	-	2597.3 (+/-397.5)	755.2 (+/-122.1)
	AGFPNet	9.7 (+/-0.8)	437.1 (+/-88.9)	429.2 (+/-78.4)	0.0 (+/-0.0)
	AGFPNet+IC	9.7 (+/-0.8)	437.1 (+/-88.9)	429.2 (+/-78.4)	0.0 (+/-0.0)
	GRN	9.1 (+/-0.3)	408.3 (+/-108.7)	389.3 (+/-88.7)	6.0 (+/-18.0)
	GRN+IC	9.1 (+/-0.3)	408.3 (+/-108.7)	389.3 (+/-88.7)	6.0 (+/-18.0)
Middleman-5	Baseline	> 154.9 (+/-54.6)	-	> 18456.2 (+/-4591.4)	> 8362.9 (+/-2326.1)
	Baseline+IC	62.5 (+/-0.5)	-	11254.5 (+/-626.5)	4190.0 (+/-102.0)
	AGFPNet	22.6 (+/-1.7)	3107.6 (+/-1572.1)	2907.4 (+/-1438.8)	402.6 (+/-230.1)
	AGFPNet+IC	22.6 (+/-1.3)	2855.6 (+/-1031.4)	2669.8 (+/-943.9)	472.6 (+/-328.6)
	GRN	21.1 (+/-1.4)	1937.3 (+/-661.2)	1749.9 (+/-526.2)	40.8 (+/-75.9)
	GRN+IC	22.4 (+/-2.7)	2141.8 (+/-666.3)	1937.9 (+/-558.9)	102.0 (+/-135.6)

significant performance gain compared to the **GRN** planner on the **Clear**, **Sort**, **Longswap** and **Middleman-3** problems is due to the fact that the **GRN** planner on its own is already able to achieve the best performance possible on these problems. Indeed, as shown in Figure 6.2, nearly all of the planning time is spent in planning the solution's motion plan, which is the minimum planning time achievable, meaning that no further significant improvement can be made through informed backtracking. Actually, although these problems present a high combinatorial complexity, accurate feasibility prediction is sufficient to find a geometrically feasible solution in a near-optimal amount of time, without the need to leverage infeasibility constraints. In the next section, we introduce new, much more complex TAMP problems to further highlight the benefits of the proposed planner.



(a) *Access* problem

(b) *clear* problem

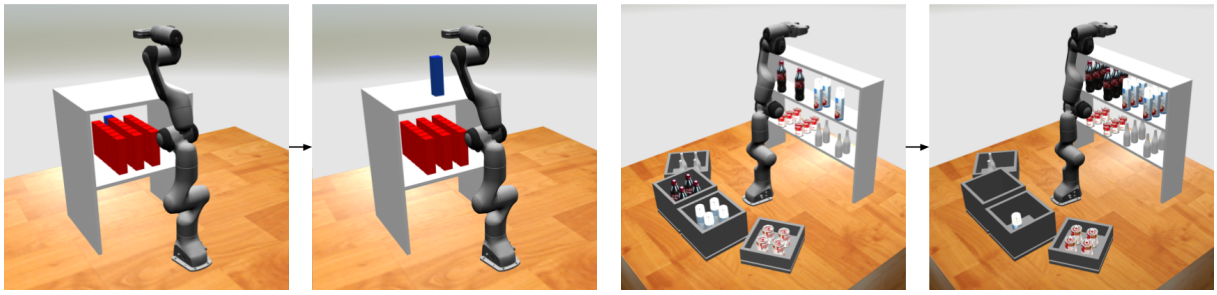
(c) *sort* problem

(d) *Longswap* problem

(e) *middleman*3 problem

(f) *middleman*5 problem

Figure 6.2: Comparison of the planning time decomposition of the different planners on each benchmark TAMP problem.



(a) *Access* – 19 problem **(b)** *Reshelf* problem

Figure 6.3: Visualization of the initial and goal states of the harder TAMP problems **Access-19** and **Reshelf**.

6.4.3 Performance on Harder TAMP Problems

In order to further demonstrate the capabilities of the proposed **GRN+IC** planner, we introduce two new, more complex TAMP problems. These problems are designed to be more challenging than the previous ones, by increasing the number of objects, the number of tasks to be performed, and the complexity of objects interactions. The first problem is a 19-object version of the **Access** problem, denoted as **Access-19** (cf. Figure 6.3a), where a single robot has to move a target object to which a total of 18 objects are blocking access. Furthermore, blocking objects also obstruct access to each other, requiring the robot to remove these objects and put them back in a specific order in order to ensure geometric feasibility. The planner has to find at geometrically feasible sequence of at least 37 actions to solve this problem. Also, following equation 3.11, the number of the potential task plans for this problem is in the order of 10^{34}, which highlights the very high combinatorial complexity of this problem. The second problem is the **Reshelf** problem (cf. Figure 6.3b), which simulates the illustrative scenario described in Section 1.1. In this 31-object problem, a robot has to resupply grocery store shelves by picking up objects from cartons and placing them on the shelves. In order to minimize waste, these objects must be placed behind the objects already on the shelves. In order to accomplish this task, the robot has to move old objects from the shelves to temporary placements, place new objects on the shelves, and then move the old objects back to their initial positions. Moreover, new objects are initially placed in cartons, meaning that only top grasps are feasible. However, goal placements on the shelves are obstructed from the top, meaning that objects must be placed from the side. This

Table 6.3: Planning performances of the different planners on the harder TAMP problems, averaged over 10 trials, using a timeout of 300s.

Problem	Method	Success Rate	Total Planning Time (s)	Infeasible Task Plans	Expanded Nodes
Access-19	Baseline	0%	> 300	> 318.7 (+/-34.2)	> 51641.0 (+/-6613.0)
	Baseline+IC	0%	> 300	> 27.900 (+/-1.640)	> 3856.300 (+/-492.408)
	AGFPNet	0%	> 300	> 11.4 (+/-2.7)	> 1257.0 (+/-292.1)
	AGFPNet+IC	0%	> 300	> 21.8 (+/-2.0)	> 3617.7 (+/-347.0)
	GRN	0%	> 300	> 1.0 (+/-0.0)	> 58986.1 (+/-640.2)
	GRN+IC	90%	47.9 (+/-13.6)	0.3 (+/-0.5)	1814.2 (+/-1625.3)
Reshelf	Baseline	0%	> 300	> 43.6 (+/-1.3)	> 19131.6 (+/-663.5)
	Baseline+IC	0%	> 300	> 31.9 (+/-2.3)	> 14353.8 (+/-1199.5)
	AGFPNet	0%	> 300	> 0.0 (+/-0.0)	> 335.9 (+/-20.4)
	AGFPNet+IC	0%	> 300	> 2.0 (+/-1.8)	> 4923.6 (+/-490.4)
	GRN	0%	> 300	> 0.0 (+/-0.0)	> 37734.0 (+/-472.7)
	GRN+IC	80%	92.0 (+/-13.2)	0.4 (+/-0.5)	1871.6 (+/-427.2)

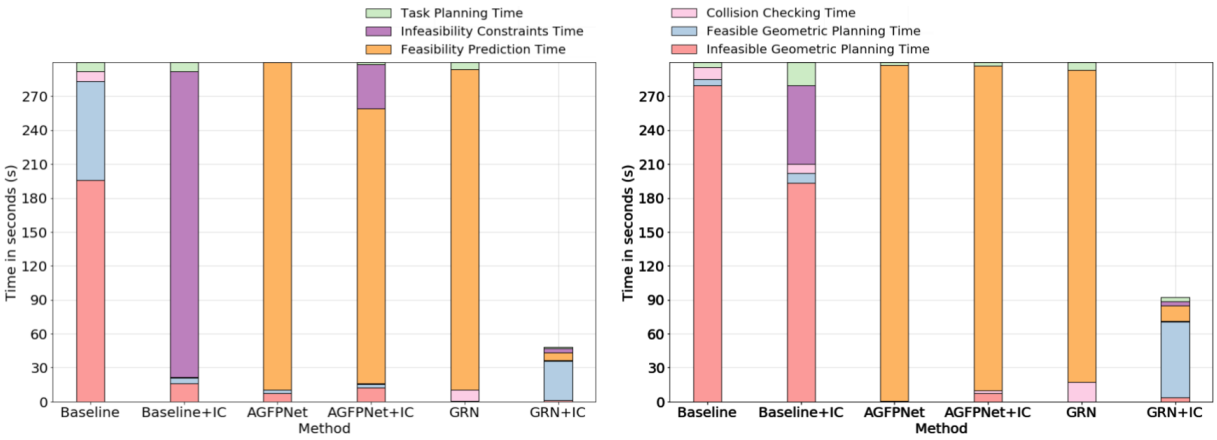
Table 6.4: Planning statistics of the different planners on the harder TAMP problems, averaged over 10 trials.

Problem	Method	Geometric Planner Calls	Feasibility Checks	Collision Checks	Pruned Branches
Access-19	Baseline	> 402.0 (+/-43.4)	-	> 103513.3 (+/-13232.1)	> 48328.4 (+/-6290.8)
	Baseline+IC	> 34.6 (+/-2.2)	-	> 7761.4 (+/-1002.1)	> 3128.5 (+/-231.7)
	AGFPNet	> 16.0 (+/-4.1)	> 2514.0 (+/-584.2)	> 2529.2 (+/-584.7)	> 899.0 (+/-227.6)
	AGFPNet+IC	> 27.0 (+/-2.9)	> 7235.4 (+/-694.0)	> 7299.0 (+/-706.1)	> 2854.2 (+/-479.9)
	GRN	> 1.0 (+/-0.0)	> 117972.2 (+/-1280.5)	> 119557.4 (+/-1256.8)	> 4.0 (+/-0.0)
	GRN+IC	37.4 (+/-0.7)	3628.4 (+/-3250.6)	2954.1 (+/-2488.0)	129.0 (+/-205.1)
Reshelf	Baseline	> 47.1 (+/-1.7)	-	> 42099.1 (+/-1437.6)	> 18861.7 (+/-661.6)
	Baseline+IC	> 36.1 (+/-2.2)	-	> 31737.9 (+/-2605.8)	> 13973.6 (+/-1183.3)
	AGFPNet	> 0.0 (+/-0.0)	> 671.8 (+/-40.8)	> 715.4 (+/-43.0)	> 0.0 (+/-0.0)
	AGFPNet+IC	> 2.0 (+/-1.8)	> 9847.2 (+/-980.8)	> 10358.9 (+/-889.9)	> 1782.9 (+/-1657.7)
	GRN	> 0.0 (+/-0.0)	> 75468.0 (+/-945.4)	> 80654.9 (+/-985.0)	> 0.0 (+/-0.0)
	GRN+IC	36.6 (+/-0.9)	3743.3 (+/-854.5)	3651.6 (+/-804.4)	312.1 (+/-446.7)

introduces an additional level of difficulty, as the robot has to perform a regrasp of each object before placing it at its goal position, by first moving it to a temporary placement, regrasping it, and then placing it at its goal placement. Consequently, the **Reshelf** problem requires each object to be moved at least twice, leading to a minimum solution length of 36 actions.

Tables 6.3 and 6.4 compare the planning performances and statistics of the different planners on the **Access-19** and **Reshelf** problems. As shown in Table 6.3, the **GRN+IC** planner is the only planner capable of solving these problems within the timeout, achieving a 90% success rate on the **Access-19** problem and an 80% success rate on the **Reshelf** problem. Despite the significant combinatorial complexity of these problems, the **GRN+IC** planner is able to solve the **Access-19** problem in under $50s$, while the **Reshelf** problem is solved in $92s$. All other variants of the planner fail to find a geometrically feasible solution to these problems. The reasons behind this failure vary depending on the planner as shown in Figure 6.4. The **Baseline** planner fails due to a high number of computationally expensive geometric planning calls, reaching 402 calls on the **Access-19** problem and 47 calls on the **Reshelf** problem. The high computational cost of the geometric planner is particularly visible on the latter with an average of $6s$ per geometric planning call.[2] The **Baseline+IC**, on the other hand, spends too much time on informed backtracking. For instance, on the **Access-19** domain, most of the total planning time is spent on hard constraints construction, search tree pruning and informed cost-to-goal computation. As a reminder, the latter has an exponential complexity in the number of objects. Conversely, the feasibility-informed planners (**AGFPNet**, **AGFPNet+IC** and **GRN**) spend almost the entire planning time on feasibility prediction. Regarding the models using **AGFPNet**, feasibility predictions are not reliable as both problems present extreme out-of-distribution cases for the neural network. Indeed, **AGFPNet** is trained on environments containing 2 movable objects and up to 4 support surfaces, while the **Access-19** and **Reshelf** problems contain 19 and 31 movable objects. Moreover, the increased number of objects causes a high number of occlusions, with many objects completely invisible from all the input views of the neural network. This leads to a high number of misclassifications, mainly false negatives, which prevent the planner from finding a geometrically feasible solution. In the case of the **GRN** planner, even though the model shows a better generalization capability than **AGFPNet** (cf. Section 5.5.3), the planner still fails to solve the **Access-19** and **Reshelf** problems, with most of planning time spent on feasibility prediction. Even though the neural network is able to filter out many infeasible actions, a large number of feasible actions remain to be explored. The naive cost-to-goal estimation described in Chapter 3 causes the planner to get lost exploring a very large number of potential actions while

[2]Mesh objects in the **Reshelf** problem have a high number of grasps with up to 200 grasps considered.



(a) *Access* – 19 **(b)** *Reshelf*

Figure 6.4: Comparison of the planning time decomposition of the different planners on the harder TAMP problems.

rarely reaching a potential solution that could be checked by the geometric planner. Table 6.3 shows that the **GRN** planner, without informed backtracking, finds only one infeasible task plan on the **Access-19** problem, and never finds any potential task plan on the **Reshelf** problem. Instead, the planner gets stuck expanding a large number of nodes (58′986 and 37′734 expanded nodes, respectively), which translates into a high number of feasibility checks exceeding 110′000 on the **Access-19** problem and 75′000 on the **Reshelf** problem.

These results further motivate the need for combining accurate feasibility prediction with informed backtracking, particularly informed cost-to-goal estimation, to solve complex TAMP problems. The **GRN+IC** planner strikes a good balance between these two heuristics, leveraging feasibility prediction to discard infeasible actions and informed backtracking to select the most promising feasible actions to explore. This allows the planner to find geometrically feasible solutions to the **Access-19** and **Reshelf** problems in a relatively short amount of time, without getting lost in the combinatorial complexity of these problems, and without any infeasible task plans generated in most trials as shown in Table 6.3. Despite the high combinatorial complexity of these problems, the **GRN+IC** planner is able to keep the size of the search tree tractable, with an average of 1814 and 1871 expanded nodes on the **Access-19** and **Reshelf** problems, respectively. This leads to a significant reduction in the number of feasibility checks and collision checks compared to the **GRN** planner. Furthermore, Table 6.4 shows that in most trials, the only geometric planner calls necessary are the ones used to plan motion for the final solution of 37 actions for the **Access-19** problem and 36 actions for the **Reshelf** problem. These results demonstrate the effectiveness and reliability of the proposed planner in solving complex TAMP problems, and highlight the benefits of combining accurate feasibility prediction with informed backtracking to achieve high planning performance.

6.4.4 Comparison with Previous Works

In order to further validate the performance of the proposed planner, we compare it with the state-of-the-art TAMP planners described in Section 4.5.5 on the benchmark problems introduced in Khodeir et al. 2023a. Following the same evaluation protocol, we finetune the **GRN** model on the combined training data from Khodeir et al. 2023a for 100 epochs using a learning rate of $1e-4$. We then evaluate the **GRN+IC** planner on the **Non-Monotonic**, **Sorting**, **Clutter** and **Stacking** problems. Table 6.5 shows the planning performances of the **GRN+IC** planner

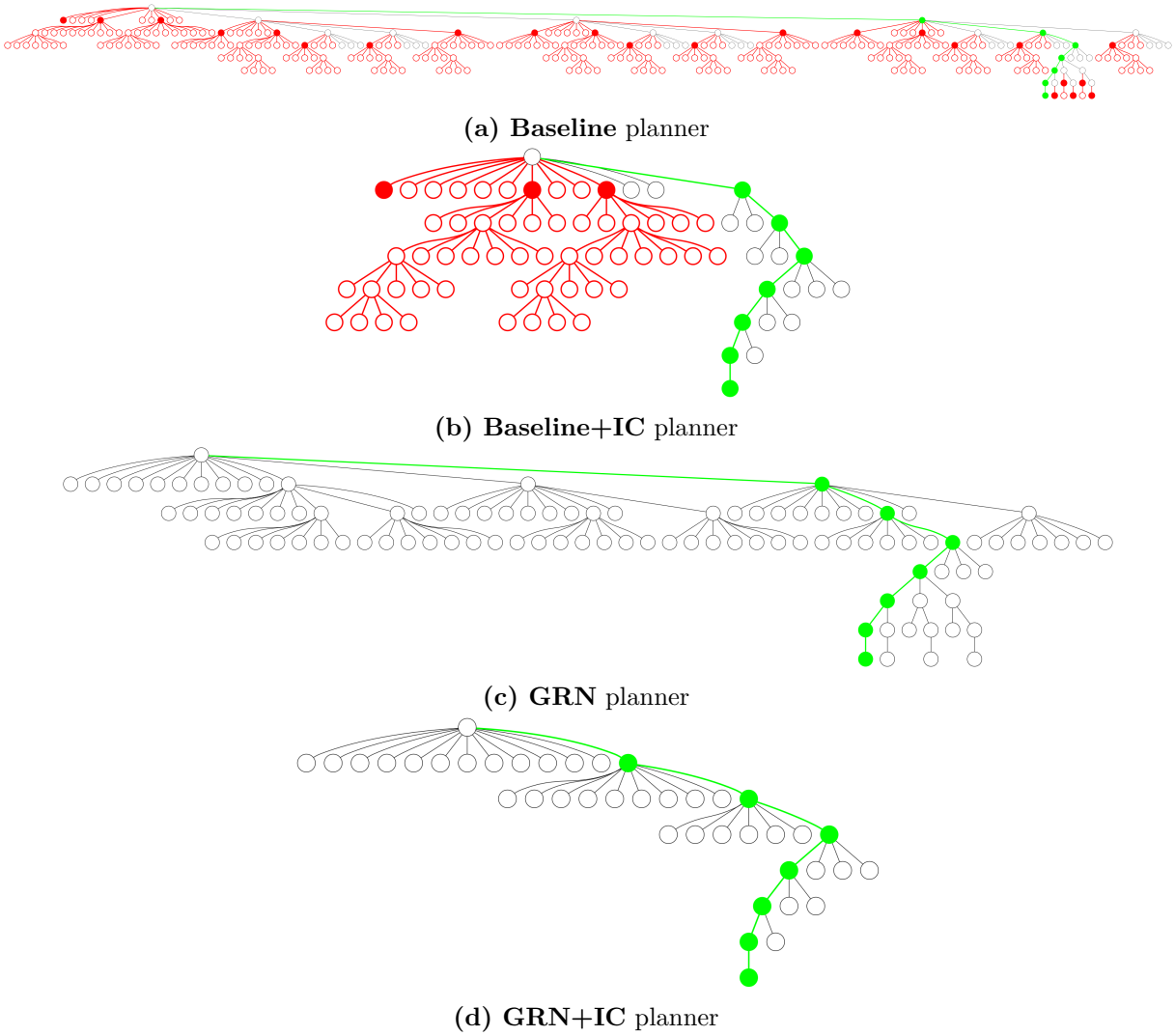
Table 6.5: Comparison of the planning performance with previous works. For the **Adaptive**, **Informed** and **PLOI** algorithms, results are taken from Khodeir et al. 2023a.

Problem	Adaptive		Informed		PLOI		AGFPNet+IC		GRN+IC	
	Solved	Time	Solved	Time	Solved	Time	Solved	Time	Solved	Time
Non-Monotonic	27	31.45	58	33.88	25	29.71	89	23,88	**98**	7,27
Sorting	66	16.95	77	21.07	68	20.47	69	20,35	**95**	7,04
Clutter	54	30.99	54	12.15	49	16.11	62	23,72	**91**	6,63
Stacking	47	16.95	54	7.94	47	9.20	87	5,38	**89**	6,15

compared to the **Adaptive** (Garrett et al. 2020), the **Informed** (Khodeir et al. 2023a), and the **PLOI** (Silver et al. 2021a) algorithms. We also include the results of the **AGFPNet+IC** planner for comparison. The **GRN+IC** planner outperforms all previous works on all problems. Compared to the best performing baseline, our proposed approach achieves a 40% improvement in success rate on the **Non-Monotonic** problem, an 18% improvement on the **Sorting** problem, a 37% improvement on the **Clutter** problem, and a 35% improvement on the **Stacking** problem. The **GRN+IC** planner is also significantly faster than all previous methods, maintaining an average planning time of 6.77s on all problems combined. Our proposed planner is 5 times faster than the **Informed** planner proposed by Khodeir et al. 2023a on the **Non-Monotonic** problem, 3 times faster on the **Sorting** problem, 4 times faster on the **Clutter** problem, and 2 times faster on the **Clutter** problem. In accordance with previous results, the **GRN+IC** planner outperforms the **AGFPNet+IC** planner on all problems, with an average improvement of 16% in success rate and 63% in planning time. These results show that our proposed planner is able to achieve state-of-the-art performance on a wide range of TAMP problems, outperforming previous works in terms of both planning performance and efficiency.

6.4.5 Qualitative Results

Finally, we conduct a qualitative analysis of the planning process of the **GRN+IC** planner compared to baseline planners on a 4-object version of the **Access** problem. Figure 6.5 shows the search trees constructed by the **Baseline**, **Baseline+IC**, **GRN** and **GRN+IC** planners. As previously observed, the **Baseline** planner expands a large number of nodes and generates a large number of infeasible task plans, leading to a high number of calls to the geometric planner. Leveraging hard infeasibility constraints allows the **Baseline+IC** planner to reduce the number of infeasible task plans generated, and consequently the number of geometric planner calls. However, the planner still requires computationally expensive geometric planner calls to verify the feasibility of the generated task plans and construct hard constraints in case of infeasibility. Thanks to accurate feasibility prediction, the **GRN** planner is able to eliminate infeasible calls to the geometric planner, though it tends to explore a high number nodes and expands more nodes than necessary. Through the combination of feasibility prediction with informed cost-to-goal estimation using soft constraints, the **GRN+IC** planner is able to significantly reduce the size of the search tree, focusing the search on the most promising feasible actions. On the **Access** problem, the only actions that are explored are the ones that are part of the final solution, without any unnecessary exploration of other actions. This illustrates how the **GRN+IC** planner is able to achieve state-of-the-art performance by leveraging the benefits of both fast and accurate feasibility prediction and informed backtracking.



(a) Baseline planner

(b) Baseline+IC planner

(c) GRN planner

(d) GRN+IC planner

Figure 6.5: Comparison of the search trees constructed by the different variants of the planner on a 4-object version of the **Access** problem. Black nodes represent nodes that have been expanded, filled red nodes/edges represent the actions for which geometric planning failed, while non-filled ones represent node/edges that have been pruned, and green nodes/edges represent nodes/edges that are part of the solution.

6.5 Discussion and Limitations

In this chapter, we have introduced a novel approach for task and motion planning with learned geometric reasoning. We proposed a method for recording infeasibility causes, obtained either from a geometric planner or from our geometric reasoning neural network (**GRN**), as planning constraints that can be used to guide the search process. Thanks to these infeasibility constraints, we were able to develop a novel informed backtracking approach that operates on two levels: (i) informed search tree pruning which serves as a filter to discard infeasible actions using hard infeasibility constraints, and (ii) informed cost-to-goal estimation which guides the search process towards the most promising feasible actions using both hard and soft constraints, by improving the estimation of the true number of actions remaining to reach the goal. We have shown that our proposed method is able to achieve state-of-the-art performance on a wide range of TAMP problems, outperforming the previous contributions of this thesis as well as previous methods

from the literature in terms of both planning performance and efficiency. By leveraging fast and accurate feasibility prediction with informed backtracking, our proposed planner is able to solve complex TAMP problems with high combinatorial complexity and a large number of objects, while maintaining a low planning time and a high success rate. Our planner is able to find geometrically feasible solutions in a near-optimal amount of time, without getting lost in the combinatorial complexity of the problem, and without infeasible calls to the geometric planner.

For now, the proposed informed backtracking method is limited to fully grounded actions. To maintain probabilistic completeness and avoid overestimating the cost-to-goal, infeasibility constraints can not be recorded for cases where multiple placement options are available for an object, such semi-specified Goal placements or *Pass* placements. In such cases, the robot is not always required to rectify infeasibility, as other placements may be feasible. In future work, we plan to extend the informed backtracking method to handle such cases by reasoning about infeasibility in a statistical manner. By keeping track of the number of sampled placements that are infeasible due an obstruction by a particular object or due to kinematic constraints, we can decide to record an infeasibility constraint if this number exceeds a certain threshold, switching planning efforts towards rectifying infeasibility rather than exploring new placements. This will allow the planner to handle more complex TAMP problems with semi-specified goals and multiple placement options, while maintaining a low planning time and a high success rate. Another potential improvement of the proposed planner is to leverage infeasibility constraints to determine when intermediate actions affect the feasibility of downstream actions. This would allow the planner to avoid placing objects in configurations that would make it impossible to perform subsequent actions, and to focus the search on actions that are compatible with the remaining tasks to perform.

Chapter 7

Conclusion

7.1 Summary

Throughout this thesis, we explored how learned geometric reasoning can accelerate and enhance task and motion planning (TAMP) for manipulation tasks in cluttered 3D environments. We first motivated the need for an efficient geometric reasoning module in manipulation planning, given the combinatorial complexity of TAMP problems, and the high computational cost of geometric planning. We then established that feasibility prediction is a crucial component of TAMP, as it provides fast geometric feedback to the symbolic planner, thus reducing the dependency on computationally expensive collision checkers. Moreover, we have shown that reasoning about geometric infeasibility causes can further improve the performance of task and motion planning through informed backtracking. The main contributions of this thesis are as follows:

- **Action and Grasp Feasibility Prediction (AGFPNet):** We propose a robot-centric convolutional network architecture, **AGFPNet**, capable of predicting the feasibility of *Pick* and Place actions and grasps in cluttered 3D environments. By representing the scene using multi-view depth images, **AGFPNet** can encode enough information to predict feasibility with high accuracy. We have presented a framework for computing the feasibility of *Move* action by combining the feasibility predictions of *Pick* and *Place* actions, as well as extensions to mesh-shaped objects and multi-robot scenarios, with the introduction of collaborative feasibility estimation for *Pass* actions. We have shown that **AGFPNet** can achieve high accuracy in predicting the feasibility of actions and grasps, and notable generalizability across different object shapes and scenes with varying clutter levels, although being trained on much simpler environments. We have proposed an integration of **AGFPNet** in a TAMP pipeline, by using the feasibility predictions as a cost component during task planning in order to prioritize feasible actions. We have shown that the proposed feasibility-informed planner is capable of significantly improving planning performance on complex single and multi-robot TAMP problems, by reducing the number of calls to the geometric planner, hence achieving high success rates, and considerably reducing planning time. We have also shown that our approach outperforms prior task and motion planning methods across different TAMP problems.
- **Geometric Reasoning Networks (GRN):** Building on the feasibility prediction capabilities of **AGFPNet**, we proposed a novel architecture, **GRN**, which leverages a graph-based representation of the scene to tackle the occlusion issues image-based representations face as the number of objects in the environment increases. **GRN** uses an edge-enhanced graph attention network to reason about the geometric relationships between objects in the scene

for action and grasp feasibility prediction. Furthermore, it introduces two interpretability mechanisms, Inverse Kinematics (IK) feasibility prediction and grasp obstruction (GO) estimation, which provide insights into the reasons behind action infeasibility. We have shown that **GRN** outperforms state-of-the-art methods action and grasp feasibility predictions, while operating at a fraction of the computational cost of geometric planners. The proposed model also demonstrates better generalizability across environments with an increasing number of object, as well as a better robustness to input noise. Thanks to the interpretability mechanisms, we introduced a single-shot task and motion planner that leverages the predicted reasons of infeasibility to efficiently solve a class of manipulation planning problems, from a single query to the neural network. The **GRN**-based planner has shown to be capable of solving complex manipulation tasks in cluttered environments with up to 28 objects, as well as non-trivial *Handover* tasks. Moreover, we have demonstrated the models seamless transfer to a real-world setting on two different robotic platforms, the single-arm Franka Emika Panda and the dual-arm PR2 robot, showcasing **GRN**'s ability to generalize to real-world scenarios.

- **Task and Motion Planning with Learned Geometric Reasoning:** We have proposed a task and motion planning framework which leverages the learned geometric reasoning capabilities of **GRN** to not only accelerate the planning process but also tackle much more complex TAMP problems. By recording reason of infeasibility obtained from both the geometric planner and the neural network, we introduce a novel informed backtracking method that better guides the search towards geometrically feasible solution in two ways. The first is by using constraints to prune the search tree more effectively and avoiding repeated geometric planning failures. The second is by leveraging the recorded infeasibility constraints to compute an informed cost-to-goal heuristic, which better approximates the true number of actions required to reach the goal. We proposed a recursive branch-and-bound algorithm that reasons about how different grasp types can be rectified, in order to compute the informed cost-to-goal heuristic, taking into account deeply non-monotonic scenarios. Through extensive experiments, we have shown that the proposed informed backtracking method can significantly improve the performance of task and motion planning, even without learned geometric reasoning, by keeping the size of the search tree tractable thanks to informed search tree pruning and informed cost-to-goal computation. The **GRN**-informed planner with informed backtracking **GRN+IC** demonstrates a significant improvement in planning performance, achieving 100% success rate on all benchmark TAMP problems, while considerably reducing planning time. Moreover, by leveraging fast and accurate feasibility prediction with informed backtracking, **GRN+IC** is the only planner capable of solving harder long-horizon TAMP problems, achieving near-optimal performance. Moreover, our approach significantly outperforms prior task and motion planning methods across different TAMP problems.

7.2 Future Work

The work presented in this thesis opens up several avenues for future research. Here, we outline some of the most promising directions:

Placement Sampling Guidance: In this thesis, we have shown that accurate action and grasp feasibility prediction and informed backtracking can significantly improve the performance of task and motion planning. However, the current approach is dependent on the quality of sampled placements during the search, which can be a bottleneck for problems where feasible placements are scarce (e.g. placing an object in a cluttered table where all objects are fixed).

Future work could focus on mapping the space of feasible object placements to guide sampling towards more promising regions. This could be achieved either by using distance-based methods such as Voronoi maps, or by adapting the feasibility prediction models to output the distribution of feasible placements, rather than the feasibility of individual placements. This would allow the planner to sample placements from the predicted distribution, thus increasing the likelihood of finding a feasible object placement.

More Detailed Grasp Representation: Throughout this thesis, we have represented grasps using a set of grasp types, each one grouping all grasps from a specific side of the object. This representation simplifies the grasp feasibility prediction problem as a classification of the feasibility of each grasp type. However, this representation is limited in the sense that it does not capture the full richness of the grasp space, especially for non-box shaped objects. Future work could focus on defining a more detailed grasp representation that allows the model to predict the feasibility of individual grasps, further improving the accuracy grasp feasibility prediction and thus accelerating geometric planning.

Extension to Mobile Manipulators: In this thesis, we have focused on manipulation planning for fixed-base manipulators. However, the robot-centric nature of the proposed neural networks allows a seamless extension to mobile manipulators. By adding the robot's base location as a sampled geometric parameter during the search, **AGFPNet** and **GRN** can be used to predict the feasibility of actions and grasps by simply expressing object poses in the robot's reference frame when the latter is at the sampled location. Future work could also extend learned geometric reasoning to navigation tasks, allowing the planner's applicability to Navigation Among Movable Obstacles (NAMO) problems. The graph-based representation of the scene proposed by **GRN** could be leveraged to reason about the feasibility of robot 2D motions in cluttered environments, thus providing a more comprehensive solution to mobile manipulation planning. In this thesis, we focused on infeasibility due grasp obstructions and kinematic constraints. An important future research direction involves reasoning about motion planning infeasibility, which is more challenging to predict due to the high dimensionality of the robot's configuration space.

Extension to Diverse Actions: Another promising direction for future work is to extend the proposed learned geometric reasoning to predict the feasibility of more diverse actions, such as pushing, pulling, or sliding objects. In this thesis, we have focused action and grasp feasibility prediction for *Pick* and *Place* actions. However, the proposed neural networks can be extended to predict the feasibility of other actions by adapting the input representation to include the necessary information for the new action. For instance, pushing actions could be represented by using two nodes in **GRN**'s input graph, one for the initial object pose and one for the final object pose after the push. The edge between these two nodes would represent the pushing action, thus allowing feasibility prediction as a link prediction problem.

TAMP Problem Decomposition: Finally, an interesting direction for future work is to investigate how complex TAMP problems can be decomposed into simpler subproblems that can be solved independently. This could further speed up the planning process by greatly reducing the search space, focusing only on objects and actions that are relevant to the subproblem under consideration. The learning-based geometric reasoning approach proposed in this thesis could be used for this purpose, particularly the predicted reasons of infeasibility as they can help identify the most critical objects and actions necessary to solve a specific subproblem.

Appendix A

Résumé en Français

Ces dernières années, le domaine de la robotique a connu une croissance remarquable, avec des robots jouant un rôle essentiel dans des applications variées telles que l'industrie, la santé, la logistique et l'assistance domestique. Cette évolution est rendue possible grâce à des avancées en hardware, en intelligence artificielle, en algorithmes de planification et de raisonnement, ainsi qu'en systèmes de contrôle. Cependant, à mesure que les applications robotiques deviennent plus complexes, les robots doivent développer des capacités avancées en planification, raisonnement et prise de décision, notamment pour exécuter des tâches de manipulation complexes sur de longs horizons. La planification de tâches et de mouvements (Task and Motion Planning, TAMP) est au cœur de ces capacités, permettant aux robots de planifier et d'exécuter des séquences d'actions tout en gérant les interactions physiques de manière sûre, efficace et adaptable. Ce problème est particulièrement difficile en raison de la haute dimensionnalité des espaces d'états et d'actions, de la nécessité de raisonner à la fois sur les aspects symboliques (discrets) et géométriques (continus) de l'environnement, et de produire des plans complexes en temps opportun. Cette thèse se concentre sur le développement d'algorithmes et de méthodes pour la planification de tâches et de mouvements, capables de relever ces défis.

Le chapitre 2 présente un état de l'art des méthodes TAMP, en mettant en lumière les approches classiques et récentes. Les méthodes basées sur les graphes de manipulation, et plus généralement la planification de mouvements multimodale modélisent les interactions entre le robot et les objets comme des transitions entre différents modes cinématiques, permettant une plus grande adaptabilité à différents types d'actions. Mais elles souffrent de problèmes de complexité combinatoire lorsque le nombre d'objets augmente. Les méthodes basées sur l'optimisation formulent la planification comme un problème d'optimisation non linéaire. Bien qu'elles offrent des solutions optimales, elles sont limitées par leur complexité computationnelle. Les approches combinées, qui intègrent un planificateur symbolique discret et un planificateur géométrique continu, permettent de vérifier la faisabilité des actions tout en explorant des séquences symboliques. Ces méthodes incluent des stratégies de rétroaction et des heuristiques pour améliorer l'efficacité. Des travaux récents se concentrent également sur la planification sous incertitude, en tenant compte des erreurs de perception ou d'exécution, ainsi que sur la planification multi-robots et la collaboration homme-robot.

Les approches basées sur l'apprentissage ont récemment émergé comme une solution prometteuse pour surmonter les limitations des méthodes classiques de planification de tâches et de mouvements. Ces méthodes exploitent des modèles d'apprentissage automatique pour accélérer la planification que ce soit en apprenant des politiques de planification bout en bout, ou en prédisant des heuristiques pour guider la recherche. Une classe de méthodes qui se distingue est la prédiction

de faisabilité des actions et des prises. Ces approches permettent de réduire la complexité combinatoire en filtrant les actions infaisables dès les premières étapes de la planification. Malgré les avancées significatives, les méthodes TAMP font face à des défis liés à leur évolutivité, leur adaptabilité et leur efficacité, en particulier dans des environnements encombrés ou à haute dimensionnalité. Ces limitations soulignent la nécessité de développer des méthodes plus performantes pour résoudre des problèmes complexes dans des scénarios réels.

Le chapitre 3 définit la planification de manipulation hors ligne comme un problème de planification de tâches et de mouvements, où deux composantes de planification sont combinées : un niveau de planification symbolique qui raisonne sur des états et actions symboliques pour générer un plan de tâches, et un niveau de planification géométrique qui valide la faisabilité géométrique de chaque action dans le plan de tâches généré et calcule le plan de mouvement correspondant. Une interface symbolique-géométrique comble l'écart entre les deux niveaux de planification, en ancrant les actions symboliques et en revenant en arrière dans l'arbre de recherche pour élaguer les branches non faisables après un échec d'appel à la planification géométrique. Nous avons présenté l'algorithme de référence de planification de tâches et de mouvements utilisé tout au long de cette thèse, et introduit un ensemble de problèmes de référence TAMP avec divers défis utilisés pour évaluer les performances du planificateur et l'impact de nos contributions. Les résultats montrent que le planificateur de référence a des difficultés à trouver des solutions géométriquement faisables sur les problèmes de référence, en raison de leur grande complexité combinatoire. Le planificateur génère un grand nombre de plans de tâches non faisables et nécessite un grand nombre d'appels au planificateur géométrique pour trouver une solution faisable. La composante de planification géométrique constitue le principal goulot d'étranglement dans le processus de planification, et la taille de l'arbre de recherche croît de manière exponentielle avec le nombre d'objets et de robots dans l'environnement. Compte tenu de ces résultats, il est nécessaire de disposer de stratégies de recherche et d'heuristiques plus efficaces pour guider l'exploration vers des solutions faisables de manière plus efficiente.

Le chapitre 4 présente **AGFPNet**, une méthode pour prédire la faisabilité des actions et des types de prises dans les problèmes de TAMP, sans recourir à une planification géométrique coûteuse en calcul. En représentant l'environnement 3D à l'aide d'images de profondeur multi-vues, nous sommes en mesure d'encoder les informations géométriques pour une prédiction précise de la faisabilité des actions et des prises. Nous avons démontré que le modèle proposé atteint une grande précision et est capable de se généraliser à des environnements inconnus, à des formes d'objets variées ainsi qu'à des configurations multi-robots, tout en étant entraîné sur un ensemble de données entièrement synthétique. De plus, nous avons proposé un planificateur de tâches et de mouvements informé par la faisabilité, qui exploite les prédictions de faisabilité pour accélérer la recherche d'un plan de tâches géométriquement faisable. En intégrant la faisabilité prédite dans le coût des nœuds, le planificateur peut éviter d'explorer des branches infaisables et prioriser les actions prometteuses. Nous avons également introduit un mécanisme de faisabilité collaborative permettant au planificateur d'estimer la faisabilité des actions impliquant deux robots. Les résultats montrent que ce planificateur informé par la faisabilité améliore significativement le taux de réussite et réduit le temps de planification sur les problèmes de TAMP de référence. Grâce à l'utilisation de **AGFPNet**, le planificateur est capable de réduire le nombre de plans de tâches infaisables générés, diminuant ainsi le temps consacré à la planification géométrique tout en ajoutant un faible surcoût en termes de vérifications de faisabilité. L'approche proposée surpasse également les méthodes précédentes sur plusieurs problèmes de TAMP, démontrant son efficacité dans la réduction de l'espace de recherche et du temps de planification Cependant, la méthode décrite dans ce chapitre présente encore certaines limitations. Tout d'abord, les images d'entrée de **AGFPNet** peuvent contenir des occultations à mesure que le nombre d'objets augmente.

Cela peut entraîner une mauvaise classification des actions et des types de prises, et aboutir à des plans infaisables. En outre, le modèle proposé ne fournit pas d'informations sur les raisons de l'infaisabilité d'une action, ce qui limite la capacité du planificateur à prioriser les actions qui influencent la faisabilité des autres.

Le chapitre 5 présente les **Geometric Reasoning Networks (GRN)**, un nouveau modèle de prédiction de faisabilité des actions et des prises dans des environnements 3D. En exploitant un réseau neuronal basé sur les graphs (GNN) et deux mécanismes d'interprétation, notre modèle prédit la faisabilité des actions de prise ou de placement, des différents types de prises, ainsi que la cause d'infaisabilité pour chaque type de prise à partir d'une représentation sous forme de graphe de scène. Les résultats démontrent que notre approche surpasse les méthodes de l'état de l'art, en se généralisant mieux aux environnements complexes et à différents robots. De plus, grâce aux prédictions de faisabilité IK et d'obstruction des prises, une classe de problèmes de planification de manipulation peut être résolue avec une seule requête à notre modèle, réduisant ainsi significativement le temps de planification par rapport aux planificateurs TAMP traditionnels. Notre approche permet également des extensions simples à des formes d'objets variées via des boîtes englobantes et des prédictions de faisabilité multi-robots grâce à la nature centrée sur le robot de **GRN**. Grâce aux deux mécanismes d'interprétation prédisant les raisons d'infaisabilité, notre modèle peut trouver efficacement des solutions faisables à une classe de problèmes complexes de planification de manipulation en une seule étape. Cependant, une planification uniquement basée sur GRN peut échouer à trouver une solution en cas de mauvaises classifications. De plus, ce planificateur est limité aux problèmes où un seul objet a un objectif spécifié. Une intégration dans un planificateur TAMP plus robuste avec une meilleure gestion des erreurs est nécessaire pour aborder une gamme plus large de problèmes TAMP tout en garantissant la complétude probabilistique.

Le chapitre 6 introduit une nouvelle approche pour la planification des tâches et des mouvements avec un raisonnement géométrique appris. Nous avons proposé une méthode pour enregistrer les causes d'infaisabilité, obtenues soit à partir d'un planificateur géométrique, ou grâce au réseau neuronal de raisonnement géométrique (**GRN**), sous forme de contraintes de planification pouvant être utilisées pour guider le processus de recherche. Grâce à ces contraintes d'infaisabilité, nous avons pu développer une nouvelle approche de backtracking informé qui opère à deux niveaux : (i) l'élagage informé de l'arbre de recherche, qui sert de filtre pour écarter les actions infaisables en utilisant des contraintes d'infaisabilité strictes, et (ii) l'estimation informée du cost-to-goal, qui guide le processus de recherche vers les actions faisables les plus prometteuses en utilisant à la fois des contraintes strictes et souples, en améliorant l'estimation du nombre réel d'actions restantes pour atteindre l'objectif. Nous avons montré que la méthode proposée s'avère très performante sur un large éventail de problèmes TAMP, surpassant les contributions précédentes de cette thèse ainsi que les méthodes antérieures de la littérature en termes de performance et d'efficacité de planification. En exploitant une prédiction rapide et précise de la faisabilité avec un backtracking informé, notre planificateur proposé est capable de résoudre des problèmes TAMP complexes avec une grande complexité combinatoire et un grand nombre d'objets, tout en maintenant un temps de planification réduit et un taux de réussite élevé. Notre planificateur est capable de trouver des solutions géométriquement faisables en un temps quasi-optimal, sans se perdre dans la complexité combinatoire du problème, et quasiment sans appels infaisables au planificateur géométrique.

Cette thèse a exploré des approches novatrices pour résoudre les problèmes de planification de tâches et des mouvements (TAMP) dans des environnements robotiques complexes. Les contributions principales incluent le développement de méthodes d'apprentissage permettant de prédire la faisabilité des actions et des prises, tout en identifiant les causes d'infaisabilité, ainsi que leur inté-

gration dans un algorithme de planification TAMP multi-robot. En combinant des techniques de raisonnement géométrique basées sur l'apprentissage avec des stratégies de planification avancées, nous avons réussi à améliorer la performance des planificateurs TAMP en réduisant fortement, grâce aux prédictions du **GRN**, les appels couteux au planificateur géométrique, en particulier pour tester la validité d'actions infaisables. Ces avancées ont permis d'améliorer l'efficacité, la robustesse et l'interprétabilité des processus de planification. Parmi les perspectives futures, une direction prometteuse consiste à décomposer les problèmes TAMP complexes en sous-problèmes plus simples, pouvant être résolus indépendamment. Cette décomposition pourrait accélérer le processus de planification en réduisant l'espace de recherche et en se concentrant sur les objets et actions pertinents. Les méthodes de raisonnement géométrique basées sur l'apprentissage, développées dans cette thèse, pourraient jouer un rôle clé dans cette approche, en exploitant les prédictions sur les causes d'infaisabilité pour identifier les éléments critiques. Ces travaux ouvrent ainsi la voie à des systèmes de planification encore plus performants et adaptatifs pour des applications robotiques réelles.

Bibliography

Shannon, Claude Elwood (1948). 'A mathematical theory of communication'. In: *The Bell system technical journal* 27.3, pp. 379–423 (cit. on p. 11).

Bellman, Richard (1957). 'A Markovian decision process'. In: *Journal of mathematics and mechanics*, pp. 679–684 (cit. on p. 12).

Dijkstra, Edsger Wybe (1959). 'A Note on Two Problems in Connexion with Graphs'. In: *Numerische Mathematik* 1, pp. 269–271 (cit. on p. 9).

Moore, Edward F (1959). 'The shortest path through a maze'. In: *Proc. of the International Symposium on the Theory of Switching*. Harvard University Press, pp. 285–292 (cit. on p. 11).

Hart, Peter E, Nils J Nilsson and Bertram Raphael (1968). 'A formal basis for the heuristic determination of minimum cost paths'. In: *IEEE transactions on Systems Science and Cybernetics* 4.2, pp. 100–107 (cit. on pp. 9, 11).

Fikes, Richard E and Nils J Nilsson (1971). 'STRIPS: A new approach to the application of theorem proving to problem solving'. In: *Artificial intelligence* 2.3-4, pp. 189–208 (cit. on p. 11).

Lozano-Pérez, Tomás and Michael A Wesley (1979). 'An algorithm for planning collision-free paths among polyhedral obstacles'. In: *Communications of the ACM* 22.10, pp. 560–570 (cit. on p. 10).

Khatib, Oussama (1986). 'Real-time obstacle avoidance for manipulators and mobile robots'. In: *The international journal of robotics research* 5.1, pp. 90–98 (cit. on p. 10).

Lozano-Pérez, Tomás, Joseph Jones, Emmanuel Mazer, Patrick O'Donnell, W Grimson, Pierre Tournassoud and Alain Lanusse (1987). 'Handey: A robot system that recognizes, plans, and manipulates'. In: *Proceedings. 1987 IEEE international conference on robotics and automation*. Vol. 4. IEEE, pp. 843–849 (cit. on p. 13).

Alami, Rachid, Thierry Simeon and Jean-Paul Laumond (1990). 'A geometrical approach to planning manipulation tasks. the case of discrete placements and grasps'. In: *The fifth international symposium on Robotics research*. MIT Press, pp. 453–463 (cit. on p. 13).

Erol, Kutluhan, James Hendler and Dana S Nau (1994). 'HTN planning: Complexity and expressivity'. In: *AAAI*. Vol. 94, pp. 1123–1128 (cit. on p. 12).

Koga, Yoshihito and J-C Latombe (1994). 'On multi-arm manipulation planning'. In: *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. IEEE, pp. 945–952 (cit. on p. 13).

Kavraki, Lydia E, Petr Svestka, J-C Latombe and Mark H Overmars (1996). 'Probabilistic roadmaps for path planning in high-dimensional configuration spaces'. In: *IEEE transactions on Robotics and Automation* 12.4, pp. 566–580 (cit. on p. 10).

Lynch, Kevin M and Matthew T Mason (1996). 'Stable pushing: Mechanics, controllability, and planning'. In: *The international journal of robotics research* 15.6, pp. 533–556 (cit. on p. 13).

Shimoga, Karun B (1996). 'Robot grasp synthesis algorithms: A survey'. In: *The International Journal of Robotics Research* 15.3, pp. 230–266 (cit. on pp. 13, 24).

Ahuactzin, Juan Manuel, Kamal Gupta and Emmanuel Mazer (1998). 'Manipulation planning for redundant robots: a practical approach'. In: *The International Journal of Robotics Research* 17.7, pp. 731–747 (cit. on p. 13).

Kaelbling, Leslie Pack, Michael L Littman and Anthony R Cassandra (1998). 'Planning and acting in partially observable stochastic domains'. In: *Artificial intelligence* 101.1-2, pp. 99–134 (cit. on p. 12).

LaValle, Steven (1998). 'Rapidly-exploring random trees: A new tool for path planning'. In: *Research Report 9811* (cit. on p. 10).

McDermott, Drew, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld and David Wilkins (1998). 'Pddl: the planning domain definition language'. In: *Technical report, Yale Center for Computational Vision and Control* (cit. on p. 11).

Nau, Dana, Yue Cao, Amnon Lotem and Hector Munoz-Avila (1999). 'SHOP: Simple hierarchical ordered planner'. In: *Proceedings of the 16th international joint conference on Artificial intelligence-Volume 2*, pp. 968–973 (cit. on p. 12).

Bicchi, Antonio and Vijay Kumar (2000). 'Robotic grasping and contact: A review'. In: *Proceedings 2000 ICRA. Millennium conference. IEEE international conference on robotics and automation. Symposia proceedings (Cat. No. 00CH37065)*. Vol. 1. IEEE, pp. 348–353 (cit. on p. 13).

Bohlin, Robert and Lydia E Kavraki (2000). 'Path planning using lazy PRM'. In: *Proceedings 2000 ICRA. Millennium conference. IEEE international conference on robotics and automation. Symposia proceedings (Cat. No. 00CH37065)*. Vol. 1. IEEE, pp. 521–528 (cit. on p. 11).

Kuffner, James J and Steven M LaValle (2000). 'RRT-connect: An efficient approach to single-query path planning'. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. Vol. 2. IEEE, pp. 995–1001 (cit. on p. 11).

Nau, Dana S, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J William Murdock, Dan Wu and Fusun Yaman (2003). 'SHOP2: An HTN planning system'. In: *Journal of artificial intelligence research* 20, pp. 379–404 (cit. on p. 12).

Ghallab, Malik, Dana Nau and Paolo Traverso (2004). *Automated Planning: theory and practice*. Elsevier (cit. on p. 12).

Siméon, Thierry, Jean-Paul Laumond, Juan Cortés and Anis Sahbani (2004). 'Manipulation planning with probabilistic roadmaps'. In: *The International Journal of Robotics Research* 23.7-8, pp. 729–746 (cit. on pp. 13, 15).

Choset, Howie, Kevin M Lynch, Seth Hutchinson, George A Kantor and Wolfram Burgard (2005). *Principles of robot motion: theory, algorithms, and implementations*. MIT press (cit. on p. 11).

Stilman, Mike and James J Kuffner (2005). 'Navigation among movable obstacles: Real-time reasoning in complex environments'. In: *International Journal of Humanoid Robotics* 2.04, pp. 479–503 (cit. on p. 15).

LaValle, Steven M (2006). *Planning algorithms*. Cambridge university press (cit. on p. 11).

Amir, Eyal and Allen Chang (2008). 'Learning partially observable deterministic action models'. In: *Journal of Artificial Intelligence Research* 33, pp. 349–402 (cit. on p. 12).

Stilman, Mike and James Kuffner (2008). 'Planning among movable obstacles with artificial constraints'. In: *The International Journal of Robotics Research* 27.11-12, pp. 1295–1307 (cit. on p. 15).

Argall, Brenna D, Sonia Chernova, Manuela Veloso and Brett Browning (2009). 'A survey of robot learning from demonstration'. In: *Robotics and autonomous systems* 57.5, pp. 469–483 (cit. on p. 12).

Cambon, Stephane, Rachid Alami and Fabien Gravot (2009). 'A hybrid approach to intricate motion, manipulation and task planning'. In: *The International Journal of Robotics Research* 28.1, pp. 104–126 (cit. on p. 15).

Ratliff, Nathan, Matt Zucker, J Andrew Bagnell and Siddhartha Srinivasa (2009). 'CHOMP: Gradient optimization techniques for efficient motion planning'. In: *2009 IEEE international conference on robotics and automation*. IEEE, pp. 489–494 (cit. on p. 10).

Hauser, Kris (2010). 'Randomized belief-space replanning in partially-observable continuous spaces'. In: *Algorithmic Foundations of Robotics IX*. Springer, pp. 193–209 (cit. on p. 14).

Hauser, Kris and Jean-Claude Latombe (2010). 'Multi-modal motion planning in non-expansive spaces'. In: *The International Journal of Robotics Research* 29.7, pp. 897–915 (cit. on pp. 13, 14).

Jaillet, Léonard, Juan Cortés and Thierry Siméon (2010). 'Sampling-based path planning on configuration-space costmaps'. In: *IEEE Transactions on Robotics* 26.4, pp. 635–646 (cit. on p. 11).

Yoshida, Eiichi, Mathieu Poirier, Jean-Paul Laumond, Oussama Kanoun, Florent Lamiraux, Rachid Alami and Kazuhito Yokoi (2010). 'Pivoting based manipulation by a humanoid robot'. In: *Autonomous Robots* 28, pp. 77–88 (cit. on p. 14).

Arfaee, Shahab Jabbari, Sandra Zilles and Robert C Holte (2011). 'Learning heuristic functions for large state spaces'. In: *Artificial Intelligence* 175.16-17, pp. 2075–2098 (cit. on p. 12).

Cosgun, Akansel, Tucker Hermans, Victor Emeli and Mike Stilman (2011). 'Push planning for object placement on cluttered table surfaces'. In: *2011 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, pp. 4627–4632 (cit. on p. 13).

Hauser, Kris and Victor Ng-Thow-Hing (2011). 'Randomized multi-modal motion planning for a humanoid robot manipulation task'. In: *The International Journal of Robotics Research* 30.6, pp. 678–698 (cit. on p. 14).

Kalakrishnan, Mrinal, Sachin Chitta, Evangelos Theodorou, Peter Pastor and Stefan Schaal (2011). 'STOMP: Stochastic trajectory optimization for motion planning'. In: *2011 IEEE international conference on robotics and automation*. IEEE, pp. 4569–4574 (cit. on p. 10).

Karaman, Sertac and Emilio Frazzoli (2011). 'Sampling-based algorithms for optimal motion planning'. In: *The international journal of robotics research* 30.7, pp. 846–894 (cit. on p. 10).

Burbridge, Chris, Zeyn Saigol, Florian Schmidt, Christoph Borst and Richard Dearden (2012). 'Learning operators for manipulation planning'. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 686–693 (cit. on p. 17).

Harada, Kensuke, Tokuo Tsuji, Kazuyuki Nagata, Natsuki Yamanobe, Hiromu Onda, Takashi Yoshimi and Yoshihiro Kawai (2012). 'Object placement planner for robotic pick and place tasks'. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 980–985 (cit. on p. 13).

Jäkel, Rainer, Sven R Schmidt-Rohr, Steffen W Rühl, Alexander Kasper, Zhixing Xue and Rüdiger Dillmann (2012). 'Learning of planning models for dexterous manipulation based on human demonstrations'. In: *International Journal of Social Robotics* 4, pp. 437–448 (cit. on p. 19).

Kasper, Alexander, Zhixing Xue and Rüdiger Dillmann (2012). 'The kit object models database: An object model database for object recognition, localization and manipulation in service robotics'. In: *The International Journal of Robotics Research* 31.8, pp. 927–934 (cit. on p. 32).

Latombe, Jean-Claude (2012). *Robot motion planning*. Vol. 124. Springer Science & Business Media (cit. on p. 11).

Lipovetzky, Nir and Hector Geffner (2012). 'Width and serialization of classical planning problems'. In: *ECAI 2012*. IOS Press, pp. 540–545 (cit. on p. 15).

Pan, Jia, Sachin Chitta and Dinesh Manocha (2012). 'FCL: A general purpose library for collision and proximity queries'. In: *2012 IEEE International Conference on Robotics and Automation*. IEEE, pp. 3859–3866 (cit. on pp. 35, 71, 72).

Sucan, Ioan A, Mark Moll and Lydia E Kavraki (2012). 'The open motion planning library'. In: *IEEE Robotics & Automation Magazine* 19.4, pp. 72–82 (cit. on p. 25).

Şucan, Ioan A and Lydia E Kavraki (2012). 'Accounting for uncertainty in simultaneous task and motion planning using task motion multigraphs'. In: *2012 IEEE International Conference on Robotics and Automation*. IEEE, pp. 4822–4828 (cit. on p. 16).

Abdo, Nichola, Henrik Kretzschmar, Luciano Spinello and Cyrill Stachniss (2013). 'Learning manipulation actions from a few demonstrations'. In: *2013 IEEE International Conference on Robotics and Automation*. IEEE, pp. 1268–1275 (cit. on p. 17).

Barry, Jennifer, Kaijen Hsiao, Leslie Pack Kaelbling and Tomás Lozano-Pérez (2013a). 'Manipulation with multiple action types'. In: *Experimental Robotics*. Springer, pp. 531–545 (cit. on p. 13).

Barry, Jennifer, Leslie Pack Kaelbling and Tomás Lozano-Pérez (2013b). 'A hierarchical approach to manipulation with diverse actions'. In: *2013 IEEE International Conference on Robotics and Automation*. IEEE, pp. 1799–1806 (cit. on p. 14).

Geffner, Hector and Blai Bonet (2013). *A concise introduction to models and methods for automated planning*. Morgan & Claypool Publishers (cit. on p. 12).

Holladay, Anne, Jennifer Barry, Leslie Pack Kaelbling and Tomás Lozano-Pérez (2013). 'Object placement as inverse motion planning'. In: *2013 IEEE International Conference on Robotics and Automation*. IEEE, pp. 3715–3721 (cit. on p. 13).

Kaelbling, Leslie Pack and Tomás Lozano-Pérez (2013). 'Integrated task and motion planning in belief space'. In: *The International Journal of Robotics Research* 32.9-10, pp. 1194–1227 (cit. on p. 16).

Coleman, David, Ioan Sucan, Sachin Chitta and Nikolaus Correll (2014). 'Reducing the barrier to entry of complex robotic software: a moveit! case study'. In: *arXiv preprint arXiv:1404.3785* (cit. on pp. 24, 35).

De Silva, Lavindra, Mamoun Gharbi, Amit Kumar Pandey and Rachid Alami (2014). 'A new approach to combined symbolic-geometric backtracking in the context of human-robot interaction'. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3757–3763 (cit. on p. 16).

Dearden, Richard and Chris Burbridge (2014). 'Manipulation planning using learned symbolic state abstractions'. In: *Robotics and Autonomous Systems* 62.3, pp. 355–365 (cit. on p. 17).

Kingma, Diederik P (2014). 'Adam: A method for stochastic optimization'. In: *arXiv preprint arXiv:1412.6980* (cit. on pp. 55, 76).

Lagriffoul, Fabien, Dimitar Dimitrov, Julien Bidot, Alessandro Saffiotti and Lars Karlsson (2014). 'Efficiently combining task and motion planning using geometric constraints'. In: *The International Journal of Robotics Research* 33.14, pp. 1726–1747 (cit. on p. 15).

Schulman, John et al. (2014). 'Motion planning with sequential convex optimization and convex collision checking'. In: *The International Journal of Robotics Research* 33.9, pp. 1251–1270 (cit. on p. 10).

Srivastava, Siddharth, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell and Pieter Abbeel (2014). 'Combined task and motion planning through an extensible planner-independent interface layer'. In: *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 639–646 (cit. on p. 15).

Calli, Berk, Arjun Singh, Aaron Walsman, Siddhartha Srinivasa, Pieter Abbeel and Aaron M Dollar (2015). 'The ycb object and model set: Towards common benchmarks for manipulation research'. In: *2015 international conference on advanced robotics (ICAR)*. IEEE, pp. 510–517 (cit. on p. 32).

Devaurs, Didier, Thierry Siméon and Juan Cortés (2015). 'Optimal path planning in complex cost spaces with sampling-based algorithms'. In: *IEEE Transactions on Automation Science and Engineering* 13.2, pp. 415–424 (cit. on pp. 25, 35, 72).

Gammell, Jonathan D, Siddhartha S Srinivasa and Timothy D Barfoot (2015). '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)*. IEEE, pp. 3067–3074 (cit. on p. 11).

Gharbi, Mamoun, Raphaël Lallement and Rachid Alami (2015). 'Combining symbolic and geometric planning to synthesize human-aware plans: toward more efficient combined search'. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 6360–6365 (cit. on p. 16).

Toussaint, Marc (2015). 'Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning.' In: *IJCAI*, pp. 1930–1936 (cit. on pp. 14, 16).

Chitnis, Rohan, Dylan Hadfield-Menell, Abhishek Gupta, Siddharth Srivastava, Edward Groshev, Christopher Lin and Pieter Abbeel (2016). 'Guided search for task and motion plans using

learned heuristics'. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 447–454 (cit. on p. 19).

Dantam, Neil T, Zachary K Kingston, Swarat Chaudhuri and Lydia E Kavraki (2016). 'Incremental task and motion planning: A constraint-based approach.' In: *Robotics: Science and systems*. Vol. 12. Ann Arbor, MI, USA, p. 00052 (cit. on p. 15).

Ghallab, Malik, Dana Nau and Paolo Traverso (2016). *Automated planning and acting*. Cambridge University Press (cit. on p. 12).

He, Kaiming, Xiangyu Zhang, Shaoqing Ren and Jian Sun (2016). 'Deep residual learning for image recognition'. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778 (cit. on p. 45).

Mirabel, Joseph, Steve Tonneau, Pierre Fernbach, Anna-Kaarina Seppälä, Mylene Campana, Nicolas Mansard and Florent Lamiraux (2016). 'HPP: A new software for constrained motion planning'. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 383–389 (cit. on p. 14).

Ferrer-Mestres, Jonathan, Guillem Frances and Hector Geffner (2017). 'Combined task and motion planning as classical AI planning'. In: *arXiv preprint arXiv:1706.06927* (cit. on p. 15).

Mirabel, Joseph and Florent Lamiraux (2017). 'Manipulation planning: addressing the crossed foliation issue'. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 4032–4037 (cit. on p. 14).

Aineto, Diego, Sergio Jiménez and Eva Onaindia (2018). 'Learning STRIPS action models with classical planning'. In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 28, pp. 399–407 (cit. on p. 12).

Caldera, Shehan, Alexander Rassau and Douglas Chai (2018). 'Review of deep learning methods in robotic grasp detection'. In: *Multimodal Technologies and Interaction* 2.3, p. 57 (cit. on p. 24).

Everett, Michael, Yu Fan Chen and Jonathan P How (2018). 'Motion planning among dynamic, decision-making agents with deep reinforcement learning'. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 3052–3059 (cit. on p. 11).

Garrett, Caelan Reed, Tomas Lozano-Perez and Leslie Pack Kaelbling (2018a). 'Ffrob: Leveraging symbolic planning for efficient task and motion planning'. In: *The International Journal of Robotics Research* 37.1, pp. 104–136 (cit. on pp. 15, 32, 33).

Garrett, Caelan Reed, Tomás Lozano-Pérez and Leslie Pack Kaelbling (2018b). 'Sampling-based methods for factored task and motion planning'. In: *The International Journal of Robotics Research* 37.13-14, pp. 1796–1825 (cit. on p. 15).

Lagriffoul, Fabien, Neil T Dantam, Caelan Garrett, Aliakbar Akbari, Siddharth Srivastava and Lydia E Kavraki (2018). 'Platform-independent benchmarks for task and motion planning'. In: *IEEE Robotics and Automation Letters* 3.4, pp. 3765–3772 (cit. on p. 33).

Sutton, Richard S (2018). 'Reinforcement learning: An introduction'. In: *A Bradford Book* (cit. on p. 12).

Veličković, Petar, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò and Yoshua Bengio (2018). 'Graph Attention Networks'. In: *International Conference on Learning Representations* (cit. on pp. 70, 78).

Wang, Zi, Caelan Reed Garrett, Leslie Pack Kaelbling and Tomás Lozano-Pérez (2018). 'Active model learning and diverse action sampling for task and motion planning'. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 4107–4114 (cit. on p. 17).

Fey, Matthias and Jan Eric Lenssen (2019). 'Fast graph representation learning with PyTorch Geometric'. In: *arXiv preprint arXiv:1903.02428* (cit. on pp. 75, 76).

Görner, Michael, Robert Haschke, Helge Ritter and Jianwei Zhang (2019). 'Moveit! task constructor for task-level motion planning'. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 190–196 (cit. on pp. 35, 72).

Haustein, Joshua A, Isac Arnekvist, Johannes Stork, Kaiyu Hang and Danica Kragic (2019a). 'Learning manipulation states and actions for efficient non-prehensile rearrangement planning'. In: *arXiv preprint arXiv:1901.03557* (cit. on p. 17).

Haustein, Joshua A, Kaiyu Hang, Johannes Stork and Danica Kragic (2019b). 'Object placement planning and optimization for robot manipulators'. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 7417–7424 (cit. on p. 13).

Kim, Beomjoon, Leslie Pack Kaelbling and Tomás Lozano-Pérez (2019). 'Adversarial actor-critic method for task and motion planning problems using planning experience'. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01, pp. 8017–8024 (cit. on p. 19).

Paszke, Adam et al. (2019). 'Pytorch: An imperative style, high-performance deep learning library'. In: *Advances in neural information processing systems* 32 (cit. on p. 52).

Phiquepal, Camille and Marc Toussaint (2019). 'Combined task and motion planning under partial observability: An optimization-based approach'. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 9000–9006 (cit. on p. 14).

Umay, Ilknur, Baris Fidan and William Melek (2019). 'An integrated task and motion planning technique for multi-robot-systems'. In: *2019 IEEE International Symposium on Robotic and Sensors Environments (ROSE)*. IEEE, pp. 1–7 (cit. on p. 16).

Wang, Angelina, Thanard Kurutach, Kara Liu, Pieter Abbeel and Aviv Tamar (2019). 'Learning robotic manipulation through visual planning and acting'. In: *arXiv preprint arXiv:1905.04411* (cit. on p. 19).

Wells, Andrew M, Neil T Dantam, Anshumali Shrivastava and Lydia E Kavraki (2019). 'Learning feasibility for task and motion planning in tabletop environments'. In: *IEEE robotics and automation letters* 4.2, pp. 1255–1262 (cit. on pp. 17, 42, 66, 73, 76).

Yuan, Weihao, Kaiyu Hang, Danica Kragic, Michael Y Wang and Johannes A Stork (2019). 'End-to-end nonprehensile rearrangement with deep reinforcement learning and simulation-to-reality transfer'. In: *Robotics and Autonomous Systems* 119, pp. 119–134 (cit. on p. 19).

Akbari, Aliakbar, Mohammed Diab and Jan Rosell (2020). 'Contingent task and motion planning under uncertainty for human–robot interactions'. In: *Applied Sciences* 10.5, p. 1665 (cit. on p. 16).

Anderson, Greg, Abhinav Verma, Isil Dillig and Swarat Chaudhuri (2020). 'Neurosymbolic reinforcement learning with formally verified exploration'. In: *Advances in neural information processing systems* 33, pp. 6172–6183 (cit. on p. 12).

Chen, Dian, Brady Zhou, Vladlen Koltun and Philipp Krähenbühl (2020). 'Learning by cheating'. In: *Conference on Robot Learning*. PMLR, pp. 66–75 (cit. on p. 71).

Driess, Danny, Jung-Su Ha and Marc Toussaint (2020a). 'Deep visual reasoning: Learning to predict action sequences for task and motion planning from an initial scene image'. In: *arXiv preprint arXiv:2006.05398* (cit. on pp. 18, 42, 43, 63).

Driess, Danny, Ozgur Oguz, Jung-Su Ha and Marc Toussaint (2020b). 'Deep visual heuristics: Learning feasibility of mixed-integer programs for manipulation planning'. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 9563–9569 (cit. on pp. 18, 42, 43, 63, 66, 73, 76).

Englert, Peter, Isabel M Rayas Fernández, Ragesh K Ramachandran and Gaurav S Sukhatme (2020). 'Sampling-based motion planning on sequenced manifolds'. In: *arXiv preprint arXiv:2006.02027* (cit. on p. 14).

Garrett, Caelan Reed, Tomás Lozano-Pérez and Leslie Pack Kaelbling (2020). 'Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning'. In: *Proceedings of the international conference on automated planning and scheduling*. Vol. 30, pp. 440–448 (cit. on pp. 16, 60, 112).

Ha, Jung-Su, Danny Driess and Marc Toussaint (2020). 'A probabilistic framework for constrained manipulations and task and motion planning under uncertainty'. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 6745–6751 (cit. on p. 16).

Kim, Beomjoon and Luke Shimanuki (2020). 'Learning value functions with relational state representations for guiding task-and-motion planning'. In: *Conference on robot learning*. PMLR, pp. 955–968 (cit. on p. 17).

Strub, Marlin P and Jonathan D Gammell (2020). 'Adaptively Informed Trees (AIT*): Fast asymptotically optimal path planning through adaptive heuristics'. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3191–3198 (cit. on p. 11).

Bejjani, Wissam, Matteo Leonetti and Mehmet R Dogar (2021). 'Learning image-based receding horizon planning for manipulation in clutter'. In: *Robotics and Autonomous Systems* 138, p. 103730 (cit. on p. 19).

Garrett, Caelan Reed, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling and Tomás Lozano-Pérez (2021). 'Integrated task and motion planning'. In: *Annual review of control, robotics, and autonomous systems* 4, pp. 265–293 (cit. on pp. 14, 16).

Lamiraux, Florent and Joseph Mirabel (2021). 'Prehensile Manipulation Planning: Modeling, Algorithms and Implementation'. In: *IEEE Transactions on Robotics* (cit. on p. 14).

Li, Sihui and Neil T Dantam (2021). 'Learning proofs of motion planning infeasibility'. In: *Robotics: Science and Systems* (cit. on p. 18).

Pan, Tianyang, Andrew M Wells, Rahul Shome and Lydia E Kavraki (2021). 'A general task and motion planning framework for multiple manipulators'. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 3168–3174 (cit. on p. 16).

Qureshi, Ahmed H, Arsalan Mousavian, Chris Paxton, Michael C Yip and Dieter Fox (2021). 'Nerp: Neural rearrangement planning for unknown objects'. In: *arXiv preprint arXiv:2106.01352* (cit. on p. 18).

Silver, Tom, Rohan Chitnis, Aidan Curtis, Joshua B Tenenbaum, Tomás Lozano-Pérez and Leslie Pack Kaelbling (2021a). 'Planning with learned object importance in large problem instances using graph neural networks'. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 35. 13, pp. 11962–11971 (cit. on pp. 17, 60, 76, 77, 112).

Silver, Tom, Rohan Chitnis, Joshua Tenenbaum, Leslie Pack Kaelbling and Tomás Lozano-Pérez (2021b). 'Learning symbolic operators for task and motion planning'. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 3182–3189 (cit. on p. 17).

Wang, Zi, Caelan Reed Garrett, Leslie Pack Kaelbling and Tomás Lozano-Pérez (2021). 'Learning compositional models of robot skills for task and motion planning'. In: *The International Journal of Robotics Research* 40.6-7, pp. 866–894 (cit. on p. 17).

Wang, Ziming, Jun Chen and Haopeng Chen (2021). 'EGAT: Edge-featured graph attention network'. In: *Artificial Neural Networks and Machine Learning–ICANN 2021: 30th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 14–17, 2021, Proceedings, Part I 30*. Springer, pp. 253–264 (cit. on p. 70).

Xu, Danfei, Ajay Mandlekar, Roberto Martín-Martín, Yuke Zhu, Silvio Savarese and Li Fei-Fei (2021). 'Deep affordance foresight: Planning through what can be done in the future'. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 6206–6213 (cit. on p. 17).

Zhang, Tianyi, Jiankun Wang and Max Q-H Meng (2021). 'Generative adversarial network based heuristics for sampling-based path planning'. In: *IEEE/CAA Journal of Automatica Sinica* 9.1, pp. 64–74 (cit. on p. 11).

Brody, Shaked, Uri Alon and Eran Yahav (2022). 'How Attentive are Graph Attention Networks?' In: *International Conference on Learning Representations* (cit. on p. 70).

Curtis, Aidan, Xiaolin Fang, Leslie Pack Kaelbling, Tomás Lozano-Pérez and Caelan Reed Garrett (2022a). 'Long-horizon manipulation of unknown objects via task and motion planning with estimated affordances'. In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 1940–1946 (cit. on p. 17).

Curtis, Aidan, Tom Silver, Joshua B Tenenbaum, Tomás Lozano-Pérez and Leslie Kaelbling (2022b). 'Discovering state and action abstractions for generalized task and motion planning'. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 36. 5, pp. 5377–5384 (cit. on p. 17).

Ding, Yan, Xiaohan Zhang, Xingyue Zhan and Shiqi Zhang (2022). 'Learning to ground objects for robot task and motion planning'. In: *IEEE Robotics and Automation Letters* 7.2, pp. 5536–5543 (cit. on p. 17).

Kim, Beomjoon, Luke Shimanuki, Leslie Pack Kaelbling and Tomás Lozano-Pérez (2022). 'Representation, learning, and planning algorithms for geometric task and motion planning'. In: *The International Journal of Robotics Research* 41.2, pp. 210–231 (cit. on p. 17).

Li, Xing and Oliver Brock (2022). 'Learning from demonstration based on environmental constraints'. In: *IEEE Robotics and Automation Letters* 7.4, pp. 10938–10945 (cit. on p. 17).

McDonald, Michael James and Dylan Hadfield-Menell (2022). 'Guided imitation of task and motion planning'. In: *Conference on Robot Learning*. PMLR, pp. 630–640 (cit. on p. 19).

Park, Suhan, Hyoung Cheol Kim, Jiyeong Baek and Jaeheung Park (2022). 'Scalable learned geometric feasibility for cooperative grasp and motion planning'. In: *IEEE Robotics and Automation Letters* 7.4, pp. 11545–11552 (cit. on p. 18).

Ren, Tianyu, Alexander Imani Cowen-Rivers, Haitham Bou Ammar and Jan Peters (2022). 'Learning geometric constraints in task and motion planning'. In: *arXiv preprint arXiv:2201.09612* (cit. on p. 17).

Russell, Stuart (2022). 'Artificial Intelligence and the Problem of Control.' In: *Perspectives on Digital Humanism* 19, pp. 1–322 (cit. on p. 12).

Xu, Lei, Tianyu Ren, Georgia Chalvatzaki and Jan Peters (2022). 'Accelerating Integrated Task and Motion Planning with Neural Feasibility Checking'. In: *arXiv preprint arXiv:2203.10568* (cit. on pp. 18, 42).

Yang, Zhutian, Caelan Reed Garrett, Tomás Lozano-Pérez, Leslie Kaelbling and Dieter Fox (2022). 'Sequence-based plan feasibility prediction for efficient task and motion planning'. In: *arXiv preprint arXiv:2211.01576* (cit. on p. 18).

Antonyshyn, Luke, Jefferson Silveira, Sidney Givigi and Joshua Marshall (2023). 'Multiple mobile robot task and motion planning: A survey'. In: *ACM Computing Surveys* 55.10, pp. 1–35 (cit. on p. 16).

Bouhsain, Smail Ait, Rachid Alami and Thierry Simeon (2023). 'Simultaneous action and grasp feasibility prediction for task and motion planning through multi-task learning'. In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 2042–2048.

Dalal, Murtaza, Ajay Mandlekar, Caelan Garrett, Ankur Handa, Ruslan Salakhutdinov and Dieter Fox (2023). 'Imitating task and motion planning with visuomotor transformers'. In: *arXiv preprint arXiv:2305.16309* (cit. on p. 19).

Ding, Yan, Xiaohan Zhang, Chris Paxton and Shiqi Zhang (2023). 'Task and motion planning with large language models for object rearrangement'. In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 2086–2092 (cit. on p. 19).

Dong, Minglun and Jian Zhang (2023). 'A review of robotic grasp detection technology'. In: *Robotica*, pp. 1–40 (cit. on p. 24).

Khodeir, Mohamed, Ben Agro and Florian Shkurti (2023a). 'Learning to search in task and motion planning with streams'. In: *IEEE Robotics and Automation Letters* 8.4, pp. 1983–1990 (cit. on pp. 18, 33, 60, 61, 66, 76, 77, 111, 112).

Khodeir, Mohamed, Atharv Sonwane, Ruthrash Hari and Florian Shkurti (2023b). 'Policy-guided lazy search with feedback for task and motion planning'. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3743–3749 (cit. on pp. 76, 77).

Lin, Kevin, Christopher Agia, Toki Migimatsu, Marco Pavone and Jeannette Bohg (2023). 'Text2motion: From natural language instructions to feasible plans'. In: *Autonomous Robots* 47.8, pp. 1345–1365 (cit. on p. 19).

Mendez-Mendez, Jorge, Leslie Pack Kaelbling and Tomás Lozano-Pérez (2023). 'Embodied lifelong learning for task and motion planning'. In: *Conference on Robot Learning*. PMLR, pp. 2134–2150 (cit. on p. 19).

Motes, James, Tan Chen, Timothy Bretl, Marco Morales Aguirre and Nancy M Amato (2023). 'Hypergraph-based multi-robot task and motion planning'. In: *IEEE Transactions on Robotics* (cit. on p. 16).

Orthey, Andreas, Constantinos Chamzas and Lydia E Kavraki (2023). 'Sampling-based motion planning: A comparative review'. In: *Annual Review of Control, Robotics, and Autonomous Systems* 7 (cit. on p. 11).

Sarthou, Guillaume (2023). 'Overworld: Assessing the geometry of the world for Human-Robot Interaction'. In: *IEEE Robotics and Automation Letters* 8.3, pp. 1874–1880 (cit. on p. 87).

Chen, Yongchao, Jacob Arkin, Charles Dawson, Yang Zhang, Nicholas Roy and Chuchu Fan (2024). 'Autotamp: Autoregressive task and motion planning with llms as translators and checkers'. In: *2024 IEEE International conference on robotics and automation (ICRA)*. IEEE, pp. 6695–6702 (cit. on p. 19).

Huang, Haojie, Owen Howell, Dian Wang, Xupeng Zhu, Robin Walters and Robert Platt (2024). 'Fourier transporter: Bi-equivariant robotic manipulation in 3d'. In: *arXiv preprint arXiv:2401.12046* (cit. on p. 18).

Huang, Yixuan, Christopher Agia, Jimmy Wu, Tucker Hermans and Jeannette Bohg (2024). 'Points2Plans: From Point Clouds to Long-Horizon Plans with Composable Relational Dynamics'. In: *arXiv preprint arXiv:2408.14769* (cit. on p. 19).

Ren, Tianyu, Georgia Chalvatzaki and Jan Peters (2024). 'Extended tree search for robot task and motion planning'. In: *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 12048–12055 (cit. on p. 16).

Renault, Benoit, Jacques Saraydaryan, David Brown and Olivier Simonin (2024). 'Multi-Robot Navigation Among Movable Obstacles: Implicit Coordination to Deal with Conflicts and Deadlocks'. In: *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3505–3511 (cit. on p. 15).

Wang, Shu, Muzhi Han, Ziyuan Jiao, Zeyu Zhang, Ying Nian Wu, Song-Chun Zhu and Hangxin Liu (2024). 'LLMˆ 3: Large Language Model-based Task and Motion Planning with Motion Failure Reasoning'. In: *Multi-modal Foundation Model meets Embodied AI Workshop in the 41 st International Conference on Machine Learning* (cit. on p. 19).

Zhang, Yan, Amirreza Razmjoo and Sylvain Calinon (2024). 'Learn2Decompose: Learning Problem Decomposition for Efficient Task and Motion Planning'. In: *arXiv preprint arXiv:2408.06843* (cit. on p. 19).

Zhao, Zhigen, Shuo Cheng, Yan Ding, Ziyi Zhou, Shiqi Zhang, Danfei Xu and Ye Zhao (2024). 'A survey of optimization-based task and motion planning: From classical to learning approaches'. In: *IEEE/ASME Transactions on Mechatronics* (cit. on p. 14).

Ghallab, Malik, Dana Nau and Paolo Traverso (2025). *Acting, Planning, and Learning*. Cambridge University Press (cit. on p. 12).

Hu, Yihan, Siqi Chai, Zhening Yang, Jingyu Qian, Kun Li, Wenxin Shao, Haichao Zhang, Wei Xu and Qiang Liu (2025). 'Solving motion planning tasks with a scalable generative model'. In: *European Conference on Computer Vision*. Springer, pp. 386–404 (cit. on p. 11).