Master thesis on Intelligent Interactive Systems

Universitat Pompeu Fabra

Combined Task and Motion Planning: Scaling up

Magí Dalmau Moreno

Supervisor: Hector Geffner, PhD

Co-Supervisor: Néstor García, PhD

July 2022

Contents

Dedication

I want to dedicate this work to my newborn nephew, Joan, to whom I hope that the indefatigable work that we all researchers carry out will bring him a better future to live in.

Acknowledgement

I would like to deeply thank my supervisors Hector Geffner and Néstor García for their work and contributions in this challenging project. Likewise, my gratitude to Eurecat Technological Center for supporting this work.

Finally, I would like to extend a heartfelt thank you to my family and friends for all their support during this journey.

Abstract

This master thesis contributes to provide planning capabilities to embodied agents so that they are able to decide what sequence of actions should be taken to solve a given complex problem, involving coupled reasoning both at symbolic and geometric level (i.e. combined task and motion planning). This work has investigated width-based search algorithms, and a new algorithm has been proposed which implements a novel *lazy-interleaved* approach to reduce the computational cost of the planning, without making any assumptions on the robot used or the environment objects. Note, then, that the proposed approach works completely online without needing any precomputation at all. This work effectively exploits domain knowledge provided via *sketches* to guide the search in problems with huge-combinatorial state-spaces. The proposed approach is able to work transparently with robotic continuous search state spaces with flexibility, using an adaptive sampling of the world to build the search space of the problem. Moreover, the problem dynamics are retrieved as a black box, so the developed planner is able to work directly with a simulator, and it does not need an explicit declaration of the action structure. The proposed approach has been validated in two problem families that illustrate the current challenges in combined task and motion planning (video demos showing plans calculated using the developed framework can be found in `https://drive.google.com/drive/folders/10goVJ8A86RGIGsbthdmak8L_mTdT6hby?usp=sharing`). Furthermore, the proposed approach has been compared with state-of-the-art approaches, obtaining significantly better results. Besides, the proposed approach has been combined and integrated within the ROS environment in order to provide the highest level of standardization and compatibility with the maximum number of robotic systems currently available.

Keywords: Combined Task and Motion Planning; Width-Based Search Algorithms; Sketches; Lazy Planning; Robotics

Chapter 1

Introduction

In this work, a contribution is made to provide reasoning capabilities to embodied agents so that they are able to solve complex tasks that involve physical interaction with their environment. These capabilities are developed in a planning approach, in which the agent(s) must take into account symbolic and geometric reasoning to decide what sequence of actions should be taken to solve a given task. This kind of problems are known as combined task and motion planning (CTMP) [1]. Thus, this research contributes to provide new tools to tackle this problem. Before describing in depth the methodologies, contributions and obtained results, the motivations for the work in the context of the CTMP problems are discussed. Also, a discussion of the state-of-the-Art in CTMP is included.

1.1 Motivation

The creation of intelligent agents that are able to operate autonomously is one of the challenges long pursued by humanity. The motivations for seeking this ancient objective are multiple, ranging from pragmatic approaches in which these agents will free mankind from its arduous tasks to visions that arise from philosophical questions, such as whether a synthetic conscious life form can be produced. In recent years, there have been great advances in multiple fields of artificial intelligence. The combination of these breakthroughs represent steps on the way to generating general

artificial intelligence. However, as is well known, this challenge has not yet been overcome and, in fact, there is still a long journey ahead.

With this ambitious goal in mind, one of the current challenges is to provide agents with the ability to solve novel complex tasks in an intelligent and reliable manner. With this, the agent is intended to be able to face, in a flexible way, new tasks, and solve them demonstrating an intelligent behavior (i.e. solving the tasks not randomly but attempting to improve on some metric such as, for instance, the duration of the plan to be carried out).

The task-reasoning capabilities of embodied agents must take into account the physical interactions with the environment. These interactions add complexity to the tasks in multiple dimensions, such as restrictions on the agent's actions, actions that require greater dexterity, or increasing the number of actions to be taken into account and performed to make a task physically possible.

Thus, it is natural to develop methodologies that take into account symbolic (i.e. on the semantic nature of the task) and physical (i.e. on the spatial, geometric and dynamic nature) reasoning to successfully address the challenge of providing embodied agents with the ability to autonomously perform complex tasks.

1.2 Context

In order to provide embodied agents with the ability to solve complex tasks autonomously, multiple approaches have been proposed. These approaches can be described as programming-based, model-based and learning-based [2].

Programming-Based approach

In this approach, the decisions' workflow is programmed by hand in advance by a human. Consequently, this approach lacks flexibility and fails facing unexpected situations. On the other hand, for routine situations, this approach can result into more reliable and predictable behaviors. One of the methodologies of this approach, which is gaining more popularity on the autonomous embodied agents field, is using

Behavior Trees [3]. In this methodology, the behavioral patterns that the agent will adopt, according to its objectives and perception of the environment, are entirely defined by hand. These behaviors are defined through hierarchical trees that control the flow of decisions and actions of the agent. This technique allows the definition of specific behaviors for different families of tasks, that can also be combined to generate more complex behaviors. Even if these patterns have to be defined by hand, the behavior trees maintain a certain level of flexibility and uncertainty handling. This is achieved with structures, such as fallbacks, that allow alternative actions to be given when the default action fails.

Model-Based approach

In the model-based approach for autonomous behavior, called planning [4], the agent is able to decide the sequence of actions to be performed to solve a specific task given a model. The model contains the state space in which the agent (or agents) can be found, which actions can be applied (depending on the state) and the results that applying an action entails (i.e. the transition from the previous state to the next one). From this model, by specifying a goal to the agent, the agent searches for a sequence of valid actions that allow the transition from the initial state to the goal state. This sequence of actions is known as a plan.

Learning-Based approach

In this approach, the autonomous agent learns the appropriate behavior (i.e. a policy) from experience. These experiences could come from an expert, as in Imitation Learning, or from its own interaction with the environment, as in Reinforcement Learning which is the most popular learning-based approach. On the Reinforcement Learning (RL) framework [5], the agent learns to take actions that maximize accumulated reward. Furthermore, the transitions between states (as a result of actions) are stochastic and, in general, unknown. That is, the agent cannot plan deterministically which actions to take, guaranteeing that it will reach the desired state (contrary to the classical AI-Planning approach). Thus, instead of calculating plans, the agent learns policies that select actions that maximize reward.

1.3 State of the Art

Embodied agents must reason about what plan to follow, expressed symbolically, and about the feasibility of the plan, given by geometrical constraints, in order to solve autonomously complex tasks that involve physical interaction with the environment. From an AI-Planning perspective, this problem is named Task and Motion Planning [6]. Thus, tackling this problem involves methodologies that emerge from both Motion-Planning and Task-Planning disciplines, and it cannot be solved completely by only one of them.

Motion planning

Motion planning is concerned with the problem of finding a continuous path in the agent's configuration-space joining an initial configuration with a target configuration without leaving the envelope of the agent's valid configurations [7]. Note that this agent, could be any robotic system like, for instance, a robotic arm, a mobile robot or even a diverse combination of agents. Thus, more precisely, the motion-planning problem can be specified in the following terms:

Given a robot with d degrees of freedom, its configuration space $\mathcal{Q} \in \mathbb{R}^d$ defines a kinematic state of the robot system. Thus, an initial configuration state $\boldsymbol{q}_s$ and a goal configuration subspace $\mathcal{Q}_g \subseteq \mathcal{Q}$ can be defined. Furthermore, constraints can be defined as $C : \mathcal{Q} \to \{0, 1\}$. These constraints, for example, denote that the robot must not collide with itself and that neither the robot nor the objects attached to it must collide with the surrounding bodies. Then, given the constraint set and the configuration space, the feasible configuration space is defined as that subspace of $\mathcal{Q}$ that satisfies the constraints, i.e. $\mathcal{Q}_f = \{\boldsymbol{q} \in \mathcal{Q} \mid C(\boldsymbol{q}) = 1\}$. Thus, the objective will be to find a continuous path $\mathcal{P}(s) : [0, L] \to \mathcal{Q}_f$ such that $\mathcal{P}(0) = \boldsymbol{q}_s$ and $\mathcal{P}(L) \in \mathcal{Q}_g$, where L is the arc-length of the path, i.e. finding a curve in the feasible space of configurations that leads from the initial configuration to a point belonging to the set of target configurations. Notice, however, that explicit representations of the free configuration space $\mathcal{Q}_f$ are generally infeasible to produce,

so instead black-box collision-checkers are used, to test the validity (freedom-from-collision) of configurations during planning [8].

Task planning

AI planners select sequences of symbolic actions to drive an initial state into a goal state. The fundamentals of AI Planning (also known as Automated Planning) are found in what is known as Classical Planning [9]. In this version of AI Planning, the following assumptions are taken. The state space is finite and discrete, the initial state is completely known, the effects of the actions are deterministic, and it is not necessary to sense the environment. Relaxing to some extent the different assumptions leads to more general approaches. However, for introducing the work related to the present thesis, it is sufficient to describe the classical approach. The classical planning model consists of:

- A discrete and finite state-space $\mathcal{S}$.
- A known initial state $\boldsymbol{s}_0 \in \mathcal{S}$.
- A set $\mathcal{S}_G \subseteq \mathcal{S}$ of goal states.
- Actions $\mathcal{A}(s) \subseteq \mathcal{A}$ applicable in each state $\boldsymbol{s} \in \mathcal{S}$.
- Deterministic state-transition functions, i.e. $\boldsymbol{s}' = f(a, \boldsymbol{s})$ for $a \in \mathcal{A}(s)$.
- Positive action costs $c(a, \boldsymbol{s}) > 0$.

The objective is to compute a plan which is a sequence of applicable actions that lead from $\boldsymbol{s}_0$ to $\mathcal{S}_G$ by means of the transition functions and the intermediate states of the sequence. Furthermore, a plan is optimal if it minimizes the total action costs $\sum_{i=0}^{n} c(a_i, \boldsymbol{s}_i)$.

In order to solve this problem, one common approach is to model it as a directed graph where states are the nodes (where the initial and goal states become the root and goal nodes, respectively), and the edges are the actions and the edge weights are the action costs. Thereby, finding a plan becomes a search-in-a-graph problem and

can be tackled with either blind or informed search strategies (i.e. using informative heuristics for guiding the search [10]). Furthermore, classical planning can be described with several formalisms that differ mainly in how the transitions are described. The main ones are STRIPS [11] and Simplified Action Structure (SAS) [12].

STRIPS is a language based on Boolean variables. These variables, or facts, describe the states of the system and, thus, a state $\boldsymbol{s}$ is defined by a set of facts that are satisfied for that state. Transitions between states are produced by actions. These actions consist of preconditions $prec(a)$ (i.e. propositions that must be satisfied for the action to be applicable) and addition $add(a)$ or deletion $del(a)$ effects (which cause propositions to be satisfied or not to be satisfied, respectively).

Alternatively, in SAS, the state is encoded by a set of variables $\mathcal{V}$ where each variable $v \in \mathcal{V}$ belongs in a finite set of possible values named the domain of the variable $dom(v)$. Similarly to STRIPS, actions are formed by preconditions and effects. Nevertheless, those structures have a certain different meaning. In SAS, actions are applicable for a certain state if the state is consistent with the preconditions. Furthermore, applying an action implies a transition in which the new state is equivalent to the previous state but with the variables values updated according to the action effects $eff(a)$.

However, both formalisms are grounded representations and can be lifted in order to gain expressiveness and be able to tackle similar problems (i.e. problems sharing a certain domain) in a more generic way and facilitate performance comparison between planners. For this reason, the Planning Domain Definition Language (PDDL) [13] was created in the context of International Planning Competition (ICAPS) and it is a *de facto* standard. Thus, PDDL is a first-order logic language where the state variables are Boolean. It splits the planning-problem definition in a domain file and a problem file. The domain file describes the object types, the possible predicates and the schematic operators (which encode the possible actions). On the other hand, the problem file is unique for a specific problem and describes the objects in the world, the initial state and the problem goal.

Nevertheless, PDDL and other purely declarative planning languages are not the only way to represent classical planning models [14]. In fact, the aforementioned model functions $A(s)$, $f(a, s)$, $c(a, s)$ can also be encoded in a general fashion as black boxes. This facilitates tackling problems that are not easy to model declaratively or whose dynamics are given via simulations. Thereby, simulators do not provide a declarative representation of actions, but simply return successor states. Nevertheless, note that planning with simulators does not imply that these functions are not representable in a declarative way, but rather that this kind of algorithms make no assumption about the form of such descriptions. However, this prevents using heuristics that rely on the action structures to guide the searches. Thus, other type of efficient algorithms that can work with the state and goal structures only, such as width-based search algorithms, should be used [15, 16, 17].

Width-based search algorithms exploit the fact that classical-planning problems can be characterized in terms of a *width* measure that is bounded and small for most planning domains when goals are restricted to single atoms. These algorithms search for solutions through a general definition of state *novelty*, achieving state-of-the-art performance in classical planning. Iterated-Width Search (IW) is the most basic width-based algorithm. It consists on a sequence of IW(k) calls, a plain Breadth-First search with one change: in each iteration k, newly generated states are pruned if they do not pass the novelty test (i.e. if its novelty is greater than k, and the state novelty being the size of the smallest tuple of atoms that is true in the state and false in the previously generated states).

In most standard benchmarking instances, IW(k) performs well solving the search in low-polynomial time when the goal is atomic. Nevertheless, IW(k) does not perform well with multi-atom goals. For tackling this problem another algorithm called Serialized Iterated-Width (SIW) could be used. SIW does multiple calls to IW(k) achieving one goal at a time. However, it is an incomplete algorithm that can get trapped into dead-ends when goals are not easily serializable. In order to overcome this limitation, a powerful language for expressing finer problem decomposition called policy sketches [18] can be used to boost SIW.

A policy sketch R is given by a collection of sketch rules that express how a set of Boolean and numerical features are supposed to change. These changes do not need to occur in one single step (unlike policy rules). Policy sketches can encode domain-specific knowledge facilitating the expression of general problem decomposition, which can be used to enhance SIW. SIW_R is a variant of the SIW algorithm that uses a given sketch R and performs IW searches to find the next state s' closest from the current state s, such that s' is a goal state of the problem or a sub-goal state for some given sketch rule $r \in R$.

Combined task and motion planning

In order to solve a CTMP problem, two challenges must be addressed [6]. From the task-planning part, solving which sequence of symbolic actions must be followed. On the other hand, for the motion-planning part, satisfying the geometric constrains that must be met for each action needs to be addressed [19]. However, these two challenges are not independent and must be addressed together. Thus, it is crucial to find a decision-making process that tackles the CTMP problem in an integrated fashion and minimizing the overall runtime of the algorithm.

The most consuming operation of the algorithm is checking the geometric feasibility of a plan. Therefore, CTMP approaches can be categorized by considering the different ways of combining the construction of plans with the validity checks that are expensive. Thus, there are three main strategies: sequence-before-satisfy, satisfy-before-sequence and interleaved.

In sequence-before-satisfy strategies [20, 21, 22], the symbolic action sequence is found first, and continuous parameter values are then searched in order to meet the geometric constrains. This approach tries to take advantage of the fact that, for some problems, the search of the actions sequence is relatively inexpensive. It focuses on finding a plausible action sequence and, then, solve the (more expensive) associated geometric planning for viable symbolic plans only.

On the other hand, satisfy-before-sequence approaches [23, 24, 25] solve first the geometric problem to find sets of satisfying assignments for individual constraints and,

afterwards, try to find action sequences that use those values. These approaches aim to get advantage of the fact that modern planners are capable of taking huge (but still finite) combinatorial state spaces, finding plans efficiently. Then, the CTMP problem is tackled by generating a representative-enough discretization of the continuous quantities, computing in advance which constraints they satisfy, and, then, finding (in the generated huge problem) an action sequence using those quantities.

Finally, in interleaved approaches [26, 27, 28], actions are added to the action sequence (i.e. the plan) and constraints are satisfied progressively. There are several ways to combine the two problem resolution steps. For example, it is possible to compute a discretization of the continuous state-space in such a way that a finite (but large) problem is obtained, but the computation of the satisfaction of the constraints is postponed. Conversely, it is also possible to work with the entire continuous space, but mechanisms must be defined to re-generate new states resulting from applying symbolic actions (note that they are infinite) if the search requires it. As for when to compute the constraint satisfaction, there are also a variety of options. These can range from doing this calculation for each action to be considered, to making partial plans and then verifying if there is a solution to the constraint satisfaction for the whole sub-plan. Furthermore, *lazy* approaches can be considered. On this kind of strategies, a relaxed version of the geometric problem is solved more frequently (e.g. for each action considered) and then a complete version is solved, as a final check, significantly less frequently (e.g. when a sub-plan has been found).

1.4 Scientific questions

This thesis is aimed at providing embodied agents with the ability to reason simultaneously symbolically and geometrically to solve complex tasks involving physical interaction with the environment. Taking into account the state of the art, the scientific questions to be solved in this work are the following:

- Q1: Are width-based search algorithms, in combination of the domain knowledge provided via sketches, a plausible way to tackle CTMP problems with huge combinatorial state-spaces?

- Q2: Is it feasible to solve complex CTMP tasks in an online fashion (i.e. without needing any pre-computing) taking advantage of a lazy-interleaved approach?
- Q3: Is it possible to work transparently with robotic continuous search state spaces with flexibility using a world sampling based approach?
- Q4: Can the approach developed on this work be combined and integrated with existing Robot Operating System (ROS) [29] ecosystem tools?

Chapter 2

Methods

This chapter explains the developments carried out to answer the posed scientific questions. First, a high-level exposition of the developed approach, together with its translation to a software architecture and the used resources, is given. Afterwards, an in-depth explanation of the core developments of this project is provided. These developments are the basis of the new proposed approach to Combined Task and Motion Planning problems (CTMP).

This chapter presents the new search algorithm, named Lazy Serialized Iterated-Width with Sketches (Lazy-SIW$_R$). This novel algorithm is based on the Sketched Serialized Iterated-Width (SIW$_R$) [18], which has been modified to be able to better fit CTMP problems, and exploits geometric reasoning to check solutions in a *lazy* fashion in order to speed up the search in problems with huge-combinatorial state-spaces.

This chapter also shows how the CTMP problems are tackled in the developed framework. This comprises, first, figuring out how continuous-state domains can be supported with adaptive sampling-based techniques, how the interaction with the world problem is considered then as a black box, and, finally, how the different functions and features that the developed search algorithm requires are implemented.

2.1 Overview of the approach

The developed framework is able to work directly on states belonging to continuous domains. In this way, it works transparently on real variables typical of robotic environments (e.g. the 3D position and orientation of the base of a mobile robot). In addition, it is capable to work with actions that correspond directly to valid robotic operations (e.g. movement plans associated with picking up an object). This implies that actions are not modeled declaratively (i.e. no symbolic pre- and post-conditions, neither costs, are specified). Instead, the world interaction when performing an action in a certain state is taken as a black box that provides the successor state (i.e. the state resulting from applying the given action). This enables working directly with simulators to capture the complex dynamics of the interactions between a robot and its environment. Indeed, some actions are unfeasible to model declaratively and only after applying the action in a digital twin an accurate-enough result estimation can be retrieved.

On the other hand, no metrics or information regarding the goals are taken for granted. Thus, again from a black-box perspective, it is only possible to query whether a state is a goal or not, nothing more. This precludes the use of guides or heuristics associated with this type of information.

To solve a CTMP problem in this context, the developed framework (outlined in Figure 1) requires certain specifications that could be classified according to the traditional task-planning categorization of domain and problem. Among these specifications, the most relevant are:

- The domain of available states and actions.
- The specific set-up of the problem, i.e. its initial state, the components of the environment (e.g. the object geometries or the robot kinematic model) and the goal of the task.
- A black-box module that allows to know, in a certain state, whether an action is valid, what cost it entails and which is the resulting state.

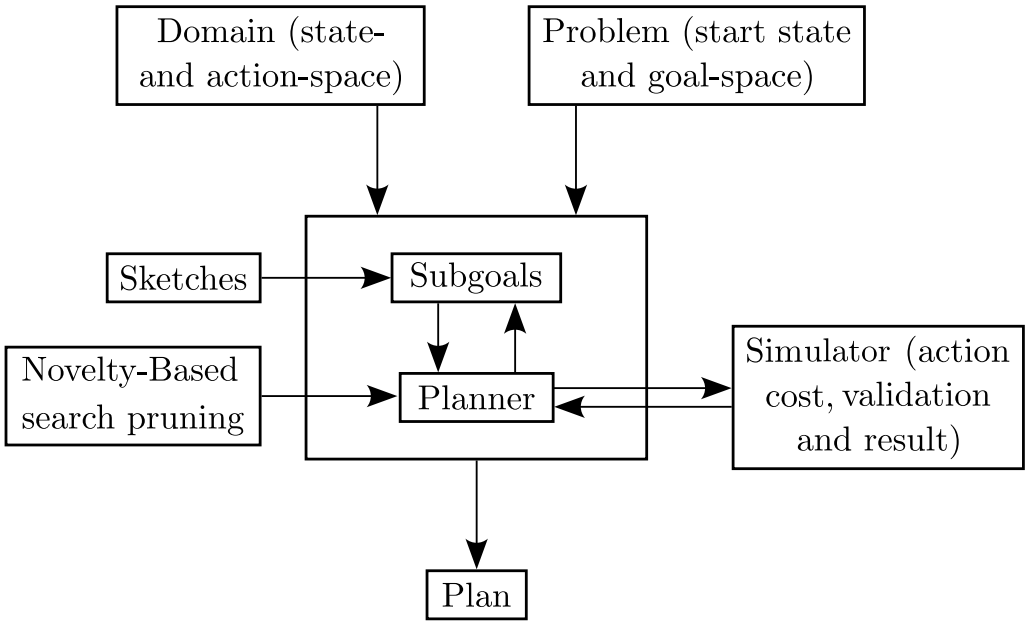


Figure 1: Block diagram of the proposed approach.

- Rules derived from domain knowledge that facilitate the generation of subgoals to serialize the problem.

- Features to factor the state and keep track of the novelty provided by each node discovered in the search.

It is noteworthy that, as the state and action domains are continuous and, thus, infinite, this framework performs an adaptive world- and action-parameter sampling. As an example, this sampling could include which are the locations in which the robot base can be or which are the gripper positions that allow to grasp a given object. The used sampling strategies are deeper explained at Section 2.3 below.

Once all the necessary inputs have been provided, the search for the solution is governed by a search algorithm that serializes the problem into subproblems using the rules or sketches provided. The series of subgoals obtained allow getting from the initial state to the goal states by solving subproblmes that are simpler (in certain cases of width 1 [16]). Concatenating sequentially each subproblem solution, the global solution is obtained.

In each subproblem, the search algorithm IW(k) builds a tree of nodes starting from

the root node (representing the start state) and expanding the successors in the corresponding order according to the algorithm used. However, any of the possible algorithms needs to have access to which actions are valid given a node (i.e. a state), its cost, and which successors it generates when applying each action.

In CTMP problems, while there may be purely symbolic actions, it is relevant to consider those actions that involve physical interaction with the robot's environment. In this sense, the developed framework integrates a communication between the search algorithm and a motion-planning module, which is the one that has contact with the physical world. This enables it to determine whether an action is valid, to map this action into a plan, executable by the robot, and to obtain the resulting state after applying the action. The complete explanation of the motion-planning pipeline can be found at Section 2.4 below. Notice that the following reasonable assumptions are made regarding the problem:

- **Geometric:** Motion planning only over positions is sufficient. Objects that are grasped or placed are kinematically coupled with their parent object and do not move or slide.

- **Quasi-static:** At the start and the end of each action, the velocity and acceleration of all the objects in the world is zero (i.e. objects move only when manipulated). Thus, the temporal variable plays no role in the state.

- **Deterministic:** The state is only changed by the planned actions. The robot motions and object grasp/release operations exactly follow the output of the path planner.

- **Fully-Observable:** The initial state is entirely known geometrically (e.g. positions, meshes and configurations) and semantically (e.g. mobility of objects, placement surfaces).

- **Fully-Controllable:** No external agents (i.e. non-controllable) intervene in the scene.

In order to perform the search in a reasonable time, it is unfeasible to validate all the actions by a complete motion planning during each expansion. Thus, this strategy, referred to in this thesis as the *lazy approach*, performs partial checks that are fast and informative enough during node expansions and only performs a complete check for those actions that are part of a potential plan (i.e. an action sequence that, if all are valid, lead from the initial to the final state). This implies modifications to the existing search algorithms that have been taken as the basis for the development of this framework. These modifications culminating in new algorithms are explained in Section 2.5 below.

Finally, the obtained (and fully validated) subplans are simply aggregated sequentially forming a resulting complete and valid robotic action plan which solves the complete problem.

2.2 Resources and architecture

The developed framework is implemented in the class-oriented programming language C++. It has been implemented seeking memory and computation efficiency, although it has not been implemented taking advantage at all of any GPU acceleration nor CPU multithreading parallelization as it is outside the scope of the project.

As it is represented in Figure 2, the proposed framework is developed under an inheritance pattern which provides modularity and flexibility. Thus, search algorithms are grounded on an abstract lazy-search class which provides the grounds of a search-schema that allows and optional lazy action-validation approach and node prunning (i.e. non-lazy non-pruning schemas can be also implemented using this same class). On top of this, the algorithms described in Section 2.5 have been implemented. The state, action and problem classes are the main ones that the user of the framework should specialize in order to personalize how the CTMP problems should be taken (e.g. the state variables, the available actions or the underlying motion-planner framework). A particularization of these clases is given in Section 3.2 below.

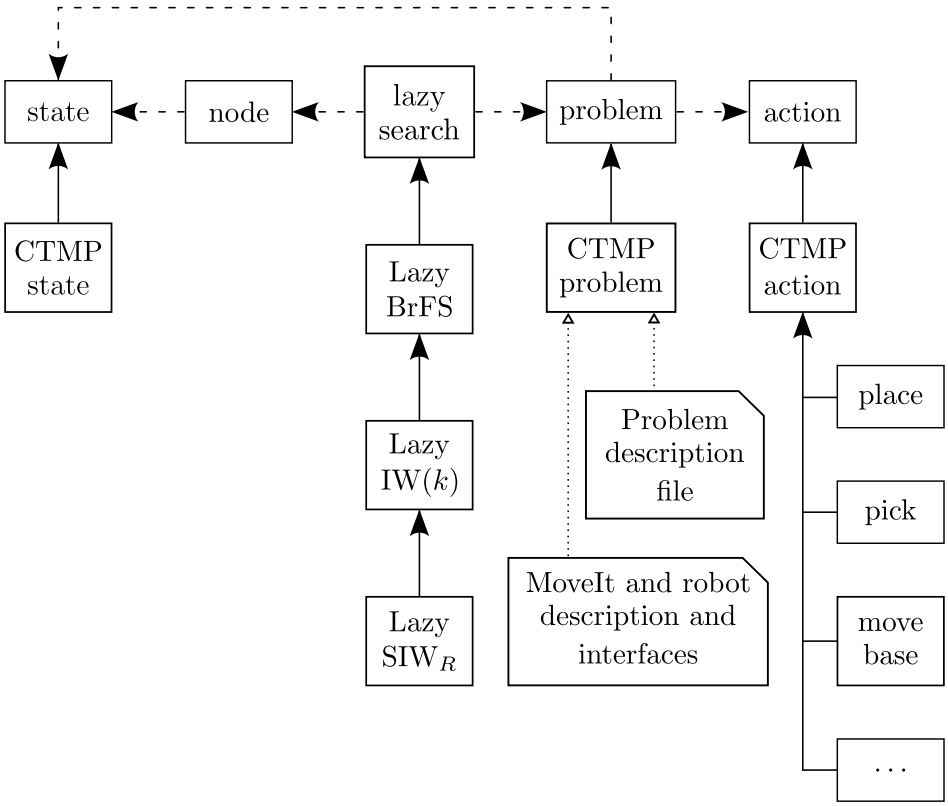


Figure 2: Architecture of the developed framework.

In order to solve CTMP problems, the search algorithm requires a module from which it can validate actions, get their cost and obtain the resulting state after the action application. For this reason, within this framework, an integration layer to interact with a motion-planner environment has been implemented in a specialization of the problem class. From this external module, the world planning-scene is maintained, the robot model is accessible, valid and collision-free robot configurations for a queried robot pose can be retrieved and, also motion plans to perform the action movements can be computed. The experiments presented in this thesis have been performed using the MoveIt [30] motion-planning framework. Note, nevertheless, that the developed framework is agnostic to the motion-planner environment used and, for example, it could work with the Kautham [31] framework.

The framework is completely integrated with the Robot Operating System (ROS) ecosystem [29]. ROS is an open-source flexible framework that aims to simplify the development of general-purpose collaborative software for robots. Although ROS is not an operating system, it provides services designed for a heterogeneous com-

Figure 3: TIAGo robot[1].

puter cluster such as hardware abstraction, implementation of commonly used functionality, message-passing between processes, low-level device control, and package management. In the ROS ecosystem, the processing takes place in nodes that are seamlessly distributed, in the same or in different cores and machines. These nodes, coordinated by a master node, send data streams to each other and can be dynamically configured. Although originally oriented to UNIX-like systems (e.g. Ubuntu), it is also adapted to work in other operating systems (e.g. macOS and Microsoft Windows). ROS has been the framework used to develop and to connect all the software modules designed for this work.

Although the developed framework is robot agnostic, it has been tested in a simulated TIAGo robot from the company PAL Robotics (see Figure 3). TIAGo is a service robot designed to work in indoor environments. TIAGo's features make it the ideal platform for research, especially on assisted living or light industry. It has mobility, perception, manipulation and also human-robot interaction capabilities. It is composed of a differential drive mobile platform that has a liftable torso with a 7-degrees-of-freedom arm mounted on top of it. In addition, the robot has a bi-articulated head with a camera.

[1]pal-robotics.com/robots/tiago/

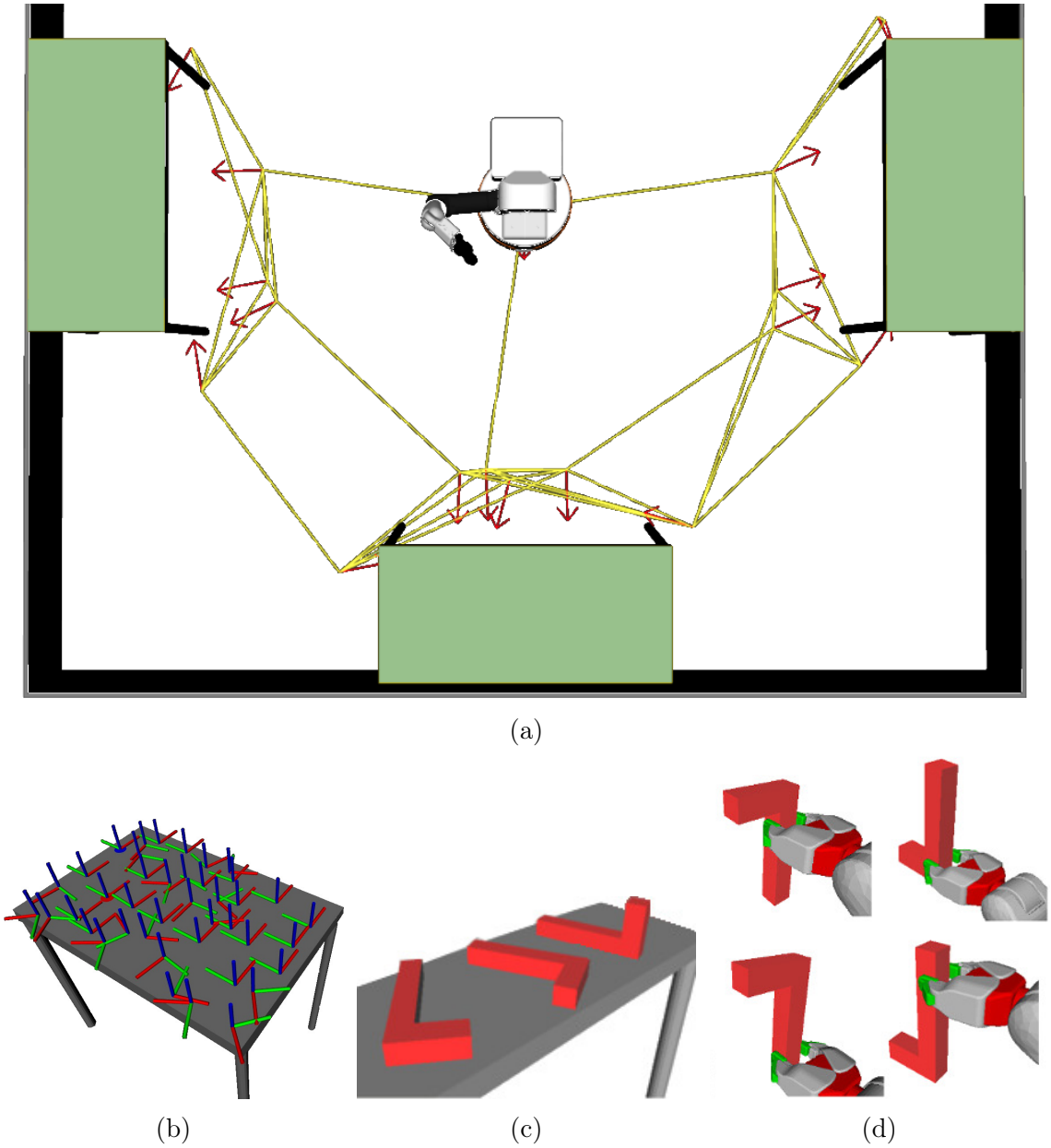
2.3 Search-Space sampling from physical world

One of the premises of the developed framework is working directly with state variables that capture aspects from the physical world. These variables can capture, for example, the position and orientation of the existing objects or the robot's joint configuration. However, this entails infinite-state domains that are beyond the scope of classical-planning search techniques. Therefore, in this framework, a sampling module has been developed in which a finite number of samples of continuous variables are drawn to build the search space directly from the physical world.

The variables to be sampled depend on the family of problem to be solved. For instance, the sampled variables may include:

- Robot-base locations: When working with mobile robots, different locations for the robot base are sampled close to the regions of interest of the problem (e.g. at a docking station so that the robot can be charged, near places where objects may be located so that the robot can manipulate them, and doors so that the robot can open them and change room location). The sampled base-locations are roadmapped to define the allowed navigation actions. However, not all the base locations are connected to each other. The maximum travel distance is constrained to a dynamically-computed value. Thus, limiting the connections reduces the number of actions (i.e. the size of the search space). Furthermore, this adds an implicit cost for traveling far which can help to search cheaper solutions without considering the real exact cost of the actions. Note that, during this process, the sampling density is checked to be sufficient to avoid having unconnected sampled bases (see Figure 4a).

- Robot configurations: Drawing samples from the robot configuration space may be needed in some problems. Specifying a robot configuration (i.e. setting a value for all robot joint) implies the characterization of the pose of all the robot bodies. This may be needed, for instance, to ensure that the robot is sufficiently folded to navigate through a narrow corridor. Another relevant



(a)

(b) (c) (d)

Figure 4: Examples of sampled variables: Roadmapped robot locations (a), object locations (b), stable object placements (c), and object grasps (d).

example could be that it may be necessary to select a collision-free target configuration after a pick action (considering the shape of the grasped object).

- Robot statuses: In some problems other variables than the strict positional or kinodynamic ones could be relevant and should be also sampled according to the problem needs (e.g. the robot battery level and temperature).

- Object locations: The location of mobile objects can change during the search. Therefore, it could be relevant sampling possible locations for these objects. In order to sample locations for articulated objects (e.g. drawers or doors), the corresponding kinematic constraints have to be respected. On the other hand, for non-articulated mobile objects (e.g. blocks), these can be in any non-collision position on a supporting surface (e.g. on the floor, a table, and even stacked on another moving object). In addition, when computing potential object locations, the sampling algorithm has to take into account not only that the object can be in any position (called *placement* here), but that the object can lie in various stable object poses (called *SOPs* here) depending on the geometry of the object (see Figures 4b and 4c, respectively). For example, a prism can lie on any of its faces and be rotated in the vertical direction, while a cylinder is only stable lying on its bases. Sampling object-locations is problem-dependent and should not be denser than necessary. For instance, it is pointless to sample infinitesimal degrees of openness of a drawer when for the problem it only matters whether the drawer is completely open or completely closed. In the same way, it is meaningless to have a density of placements within a supporting surface in which the distance between placements is smaller than the average size of the mobile objects. Otherwise, if a placement is occupied by an object, the robot will not be able to put any object in the neighboring placements because the objects will collide, but the search algorithm will not know this until it tries to perform the action (note that the actions structure is hidden as a black box).

- Ways of interacting with objects: In order to manipulate an object, physical contact is necessary and different ways can exist (and, then, be represented in the sampled variables). For example, to pull an object, depending on gripper position with respect to the object, it can be moved in one direction or another. In order to pick up an object, for instance, the robot can use different grasps (i.e. pose of the gripper with respect to the object, see Figure 4d).

Also, note that the initial state of the problem must be included in the set of

sampled states (i.e. robot location and state, object locations, etc.) as well as the goal placements of the target objects. Moreover, the sampling proposed in this environment is adaptive. Thus, during the global search, the sampling density of the variables can be increased to obtain new values. This is limited, however, to when a local search is initiated or restarted (e.g. between the achievement of subtargets or when a local search fails and is restarted). This adaptability in the sampling contributes to having a sufficiently dense sampling to ensure that the algorithm is complete (i.e. finding solution if it exists). If it were not, for example, by sampling insufficient robot locations, a certain object included in the goal would not be accessible without collisions among the available states.

2.4 Action validation

Actions involving interaction between the robot and its environment require a validation process to verify that the robot movement is feasible. For this, there must be a trajectory in the robot joint-space that respects its movement capabilities and does not involve self-collisions nor collisions with the environment. Such validations can be done using any appropriate motion-planning framework. Although the work done here is agnostic to the selected motion-planning framework and algorithm, MoveIt has been used here for the implementation and the corresponding experimentation, and, among the several planning algorithms offered by this framework, RRT-Connect has been chosen [32].

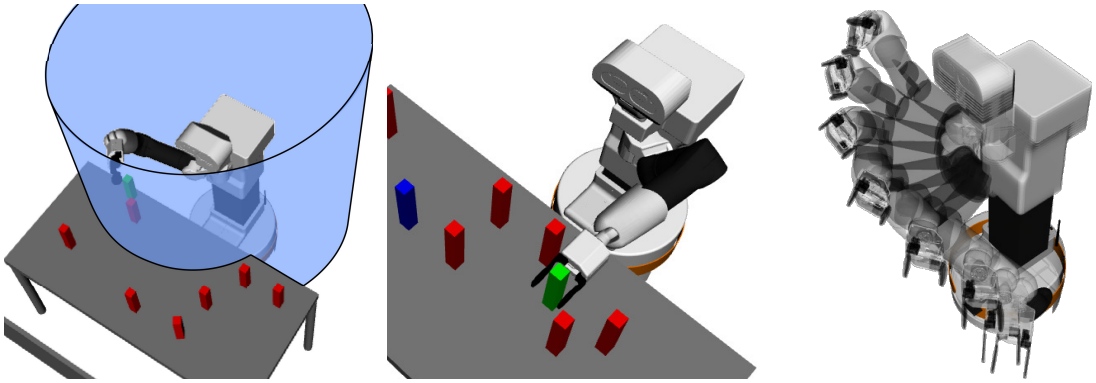
The selected sampling-based planner offers high versatility and a demonstrated good performance in complicated environments and highly-articulated robots, as it is the case in this work. The basic idea of the RRT-Connect algorithm is to build two sample trees, rooted at the initial and final joint configurations respectively. Each node of these trees is a valid robot joint configuration and each edge is a valid robot motion. The motion planner grows the trees towards each other by iteratively adding to the sample trees new collision-free joint configurations. Then, while exploring the configuration space, the sample trees finally connect and a collision-free path, through the sample trees, connecting the start and final configurations is obtained.

Using this motion-planning framework and an implemented integration layer, the action validation and its own mapping to a real and valid robot motion is computed simultaneously. The proposed validation pipeline checks sequentially the success of the following three functions:

1. *InArmWorkspace*: Returns true when, at a certain problem state, a target position is within the region of actuation of the robotic arm (keeping the robot base fixed), and false otherwise (see Figure 5a). For instance, the target position can represent an object to interact with. To speed up the calculation, the arm workspace is estimated as a cylinder of a given radius, and it is checked whether the target position is inside this cylinder.

2. *InverseKinematics*: Returns true when, at a certain problem state, a valid robot joint configuration is found to place the robot end-effector at a target pose without colliding with any object nor with the robot itself (see Figure 5b). For generalization purposes, a general iterative Inverse Kinematics (IK) solver algorithm is used [33], which makes no assumptions on the robot used or its kinematics (although robot-specific algorithms may be faster). If this algorithm does not find a solution in a given timeout, *InverseKinematics* returns false. Note that if the target pose is not in the arm workspace (i.e. *InArmWorkspace* fails), an IK solution does not exist (i.e. *InverseKinematics* will fail, after the timeout).

3. *MotionPlan*: Returns true when, there exists a valid motion plan to place the robot end effector in a target pose from the current arm configuration, perform the corresponding action and moving the arm back to the current state (see Figure 5c). Based on the checked action, the robot motions may imply, for instance, picking up, placing down, pushing or pulling an object, and, hence, different motion constraints should be considered. If the motion-planning algorithm does not find a solution in a given timeout, *MotionPlan* returns false. Note that if a valid robot joint configuration does not exist to grasp the object with the specified target pose (i.e. *InverseKinematics* fails), a motion-plan solution does not exist (i.e. *MotionPlan* will fail, after the timeout).



(a) (b) (c)

Figure 5: Examples of calls to the functions: *InArmWorkspace* (a), *InverseKinematics* (b), and *MotionPlan* (c).

In the traditional approach (i.e. non-lazy), all these three steps are all checked before considering that an action is valid. Nevertheless, in the proposed approach, only the first two steps are verified when checking the actions in a lazy fashion (as they are fast to compute), while the third step is only verified when checking the actions in a non-lazy fashion (to invest time only in fully-checking those actions that are part of a potential solution plan). Note that the introduced validation-pipeline refers only to manipulation actions. For other cases, other validation steps may be required. For instance, checking if moving the robot base is valid implies, in a lazy fashion, to check whether the initial and final locations are directly connected in the generated roadmap, while a valid motion plan will be investigated only when checking the action non-lazily.

2.5 Search algorithms

In order to select or design a search algorithm to tackle CTMP problems, some requirements must be considered. The first is due to the fact that the action structure of the problem is not known as the world interactions are given as a black box. This implies that the algorithm can not rely on heuristics or other techniques that require this information to solve the search problems like, for instance, the heuristics proposed by Bonet and Geffner [10] or the one proposed by Hoffman and Nebel [34]. Furthermore, the algorithm must be capable of tackling huge search spaces. This

requirement comes, on the one hand, since, due to the complexity of the real world, the search algorithm must work directly with continuous state-domains. And, on the other hand, to capture each robotic task accurately, it is needed not only an abstract conceptualization of the action but also a combinatorial action parametrization that implies different resulting states. Finally, in order to obtain a flexible approach the underlying search algorithm must be capable of working with multi-goal problems.

The SIW_R algorithm [18] is, then, a valid starting point since it meets these requirements. Nevertheless, action validation on CTMP problems requires a geometric reasoning that is computationally expensive. This issue, in combination with the huge search-space of the problems, provokes that performing complete motion-validations in each node expansion is infeasible. In the other hand, not checking at all the motion constraints would reduce the problem to an only symbolic task planning and produce potential infeasible robot plans. Thus, the proposed strategy is verifying only an informative but cheap part of the motion checks during the search and postpone the complete check until a potential plan is found. This strategy is referred in this thesis as a *lazy approach*.

Therefore, SIW_R must be adapted in order to fit to this lazy approach and to do so its building blocks, which are BrFS [35] and IW(k) [15] must be also adapted. In the following subsections, these new algorithm versions are explained in detail.

2.5.1 Lazy breadth-first search

Breadth-First Search, BrFS, is an algorithm for searching, within a tree data structure, for a node that satisfies a given property (i.e. to be a goal node). Note that the algorithm is blind in the sense that it does make any assumption on the solution neither the search is guided (i.e. it only knows what the goal looks like when it finds it). It starts at the tree root and sequentially explores all nodes at the present depth prior to moving on to the nodes at the next depth level. The search algorithm keeps track of the child nodes that were already encountered but not yet explored, and it checks whether a node has already been explored to not explore it twice, and, thus, not creating loops in the tree data (i.e. that each child node has only one parent

node). The algorithm uses a queue (First In First Out), called $OPEN$, to store the encountered but not explored yet nodes, and a list, called $CLOSED$, to store the already explored nodes.

The Breadth-First Search algorithm has been modified to fit the lazy approach. This new algorithm, called Lazy Breadth-First Search (Lazy-BrFS), is outlined in Algorithms 1 and 2. As it has been introduced, this lazy approach implies that the resulting graph search consists on nodes connected by provisional edges. These edges are called *provisional* since, until the complete action-validation is performed, an action may be not valid and, then, the corresponding edge must be dropped. In the original BrFS, each node has only one single parent, even when a node is reachable from other nodes (the parent node to be imposed is the one encountered first). Keeping only one parent node for each node, without checking if this connection is really valid, entails a number of problems. For example, if the connection between a parent and its successor is finally checked to be invalid, the child node and all its successors would be disconnected from the root node. Dropping all these nodes could make the problem solution to be completely inaccessible, leading to the failure of the algorithm. In order to mitigate this problem, two mechanisms has been designed:

On the one hand, in the proposed approach, nodes are allowed to have more than one parent. In this sense, when an already discovered node is found, this new parent-child relationship is established with the node that is already kept in memory (i.e. in $OPEN$ or $CLOSED$). This operation can be found at Lines 8-10 and Lines 13-15 of Algorithm 1. Nevertheless, the node parents are ordered in increasing order of cumulative cost (i.e. the addition of the edge cost to the cumulative cost of the parent) and only the better parent acts as such, to encourage cheaper (i.e. better) solutions. Thus, when a potential plan is under complete validation, if a certain edge is dropped (i.e. the edge was not really valid), a node gets disconnected from its main parent. Then, the following parent node takes the place of the main one and the plan validation follows (see Figure 6a). This process is repeated until a confirmed parent is found or there are no more parent candidates available at this moment. This mechanism can be found at Lines 12-22 from Algorithm 2.

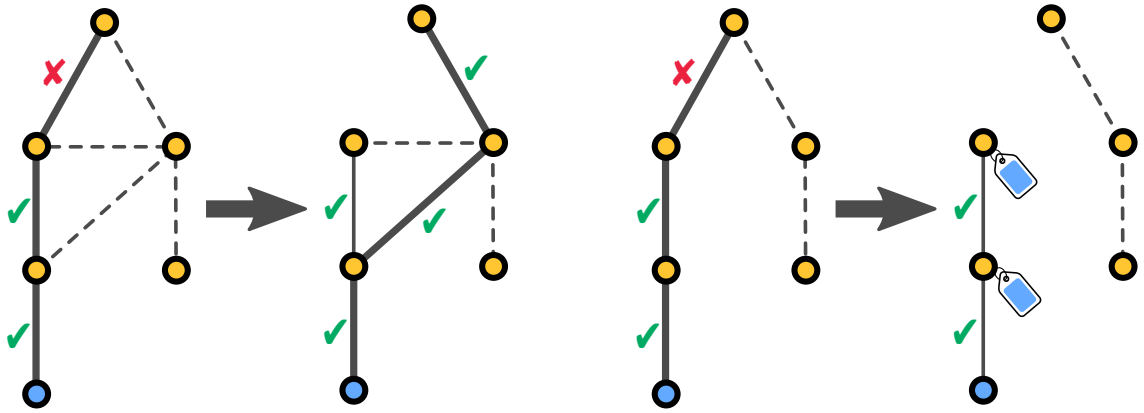
Algorithm 1: LazyBrFS(*start*, IsGoal)

```
1  OPEN ← Queue( );
2  CLOSED ← List( );
3  OPEN.Enqueue(start);
4  while OPEN ≠ ∅ do
5      parent ← OPEN.Dequeue( );
       // Actions are lazily checked in ValidActions
6      foreach action in ValidActions(parent, lazy) do
7          successor ← Successor(parent, action);
8          if ∃ duplicate ∈ OPEN : duplicate = successor then
9              parent.AddSuccessor(duplicate);
10             duplicate.AddParent(parent, action);
11             continue
12         end
13         if ∃ duplicate ∈ CLOSED : duplicate = successor then
14             parent.AddSuccessor(duplicate);
15             duplicate.AddParent(parent, action);
16             if duplicate.ChangedParent( ) and
                duplicate.IsConnectedToGoal( ) then
                   // Actions are non-lazily checked in GetPlan
17                 plan ← GetPlan(start, duplicate.ConnectedGoal( ));
18                 if plan ≠ ∅ then
19                     return plan;
20                 end
21             end
22             continue
23         end
24         parent.AddSuccessor(successor);
25         if IsGoal(successor) then
               // Actions are non-lazily checked in GetPlan
26             plan ← GetPlan(start, successor);
27             if plan ≠ ∅ then
28                 return plan;
29             end
30         else
31             OPEN.Enqueue(successor);
32         end
33     end
34     CLOSED.Append(parent);
35 end
36 return ∅;
```

Algorithm 2: GetPlan(*start*, *goal*)

```
1  current ← goal;
2  plan ← {states : List( ), actions : List( )};
3  plan.states.Append(current.GetState( ));
4  while current ≠ start do
5  |  action ← current.GetAction( );
6  |  parent ← current.GetParent( );
7  |  if IsActionValid(parent, action, non-lazy) then
8  |  |  plan.actions.Append(action);
9  |  |  current ← parent;
10 |  |  current.SetConnectedToGoal(goal);
11 |  |  plan.states.Append(current.GetState( ));
12 |  else
13 |  |  current.RemoveParent( );
14 |  |  if IsOrphan(current) then
15 |  |  |  ManageOrphan(current);
16 |  |  end
17 |  |  if DisconnectedFromRoot(goal) then
18 |  |  |  return ∅
19 |  |  else
20 |  |  |  return GetPlan(start, goal);
21 |  |  end
22 |  end
23 end
24 return Reverse(plan);
```

In both scenarios, the cost to reach the node from the root node could have changed, either because the new parent offers a higher cost or either because the node is now orphan (i.e. no more parents are available), setting its cost to infinite in this case. Also, notice that if the given node changes its cumulative cost, the cost of its successors (and the successors of the successors and so on) must be also updated according the new node cost. In the original Breadth-First Search algorithm, nodes have no information of their successors (only their parent), therefore, in order to be able to make the appropriate cost updates in this version of the algorithm, the nodes are doubly linked, meaning that the nodes keep pointers to its parents and also to its successors. This linking operation is performed at Lines 9, 14 and 24 of Algorithm 1.



(a) (b)

Figure 6: Tree examples: Recovered plan thanks to multi-parent strategy (a), and orphan chain with tags of *connected to goal* (b).

On the other hand, the second mechanism that has been developed to mitigate the side effects of the lazy approach is keeping in memory (i.e. in the $CLOSE$ list) the chains of nodes that are connected to a goal node, but its connection to the root node was dropped during a non-lazy validation process. These chains are formed during a plan validation when the goal ancestors are tried to be validated non-lazyly. Each time this validation is successful, the validated node is marked as *connected to goal* (as it can bee seen at Line 10 of Algorithm 2). Then, if any of the nodes of this chain is rediscovered during the search (i.e. a potential parent connected to the root node exists), the search is interrupted since a potential solution has been found. This can be found at Lines 16-21 of Algorithm 1. In this case, the plan between the rediscovered node and its connected goal was already validated and only is left the validation between the root and the rediscovered node (see Figure 6b). Thus, by means of this re-wiring mechanism, chains of nodes (and all the computations needed to form find it) can be reused.

2.5.2 Lazy iterated-width search

In the previous section, it has been seen that the blindness property of the Breadth-First Search algorithm allows it to make no assumptions about the actions of the problem and to be applied in a wide variety of domains. Nevertheless, the time complexity of BrFS is $\mathcal{O}(|V| + |E|)$, where $|V|$ is the number of graph vertices

(i.e. different problem states) and $|E|$ is the number of graph edges (i.e. problem actions), which can reach up to $\mathcal{O}(|V|^2)$. It is then clear that this same generalization capacity makes it explore too many nodes and lose efficiency, especially in very large search-spaces, as in the case of this project (the number of vertices can easily exceed 1×10^{60}). It is necessary to prune the search.

Without going into much detail, the *width* of a problem is a measure that provides a bound on the complexity of its solution. The fact is that most domain benchmarks appear to have a small width independent of the size of the problems, provided that the goal is restricted to a single atom (i.e. a single state property) [15]. This factor can be used as an opportunity to speed up the search for a solution. The Iterated-Width search algorithm, IW, manages to exploit the low width of these problems much better than other blind-search algorithms (being, at the same time, competitive with heuristic-based planners for single-atom goals) using a node pruning method based on the *novelty* of a state, described below.

If s is the first state generated in all the search that makes an atom true, its novelty is 1. If s does not generate a new atom (i.e. a 1-length atom tuple) but generates a new pair of atoms (i.e. a 2-length atom tuple), its novelty is 2, and so on. Likewise, if s does not generate a new tuple at all, then its novelty is set to $n + 1$, where n is the number of atoms in the problem (i.e. different properties characterizing the states). The iterations IW(k) are plain Breadth-First searches that treat newly generated states with novelty measure greater than k as if they were *duplicate* states. Notice that IW(n) just prunes truly duplicate states (i.e. is just a plain BrFS), and it is therefore complete. On the other hand, IW(k) for lower k values prunes many states and is not. Indeed, the number of states not pruned (i.e. explored) in IW(k) is $\mathcal{O}(n^k)$. Note that this results in a great improvement because, if we compare the number of states explored by IW(k) and BrFS, k with a value of 1 or 2 is enough to solve most of the problems and that n can be several tens of orders of magnitude lower than the number of vertices $|V|$. The resulting planning algorithm IW is just a series of k-width searches IW(k), for increasing values of k until the problem is solved or k exceeds the number of problem variables (i.e. a solution does not exist).

Algorithm 3: LazyIW(*start*, IsGoal, $k_{\max}$)

```
1  k ← 1;
2  while k ≤ k_max do
3  |   plan ← LazyPrunableBrFS(start, IsGoal, k);
4  |   if plan ≠ ∅ then
5  |   |   return plan;
6  |   else
7  |   |   k ← k + 1;
8  |   end
9  end
10 return ∅;
```

As stated before, the main building block of the IW(k) is the BrFS algorithm. Consequently, in order to fit the lazy approach in the developed Lazy IW(k) algorithm, the novelty-based node-pruning is implemented in the underlying Lazy Breadth-First Search algorithm (see Algorithms 3 and 4). Nevertheless, this change is not enough, and more adaptation must be done to implement the lazy approach in the IW(k) algorithm.

In order to implement the novelty-based pruning, a novelty table must be maintained, in which the atom tuples discovered during the search are stored. Nevertheless, in the lazy approach, the connections between the discovered nodes are provisional until its final validation during the plan checking. This implies, that a certain node (and, potentially, its successors) could lose its connection to the root node and thus not be an accessible state anymore. Then, if the node resulted to be inaccessible, the atom tuples introduced by the node should not have been taken into account in the search process. In addition to correcting the novelty table, some pruned nodes -stored without duplication in the $PRUNED$ list (Lines 3 and 23 of Algorithm 4)- will need to be recovered and explored, since these tuples could have mistakenly caused their pruning (as they were later discovered but providing the same tuples). Note that the stored pruned nodes, are still updated when rediscovered, updating their parent list (Lines 24-25 of Algorithm 4). This necessity implies changes in the novelty-table structure and in the algorithm itself, with respect to the original IW(k).

Algorithm 4: LazyPrunableBrFS(*start*, IsGoal, k)

```
1  OPEN ← Queue( );
2  CLOSED ← List( );
3  PRUNED ← List( );
4  OPEN.Enqueue(start);
5  while OPEN ≠ ∅ do
6      parent ← OPEN.Dequeue( );
       // Actions are lazily checked in ValidActions
7      foreach action in ValidActions(parent, lazy) do
8          successor ← Successor(parent, action);
9          if ∃ duplicate ∈ OPEN : duplicate = successor then
10             parent.AddSuccessor(duplicate);
11             duplicate.AddParent(parent, action);
12             continue
13         end
14         if ∃ duplicate ∈ CLOSED : duplicate = successor then
15             parent.AddSuccessor(duplicate);
16             duplicate.AddParent(parent, action);
17             if duplicate.ChangedParent( ) and
                duplicate.IsConnectedToGoal( ) then
                   // Actions are non-lazily checked in GetPlan
18                 plan ← GetPlan(start, duplicate.ConnectedGoal( ));
19                 if plan ≠ ∅ then return plan;
20             end
21             continue
22         end
23         if ∃ duplicate ∈ PRUNED : duplicate = successor then
24             parent.AddSuccessor(duplicate);
25             duplicate.AddParent(parent, action);
26             continue
27         end
28         parent.AddSuccessor(successor);
29         if IsGoal(successor) then
               // Actions are non-lazily checked in GetPlan
30             plan ← GetPlan(start, successor);
31             if plan ≠ ∅ then return plan;
32         else if Prune(k, successor) then
33             PRUNED.Append(successor);
34         else
35             OPEN.Enqueue(successor);
36         end
37     end
38     CLOSED.Append(parent);
39 end
40 return ∅;
```

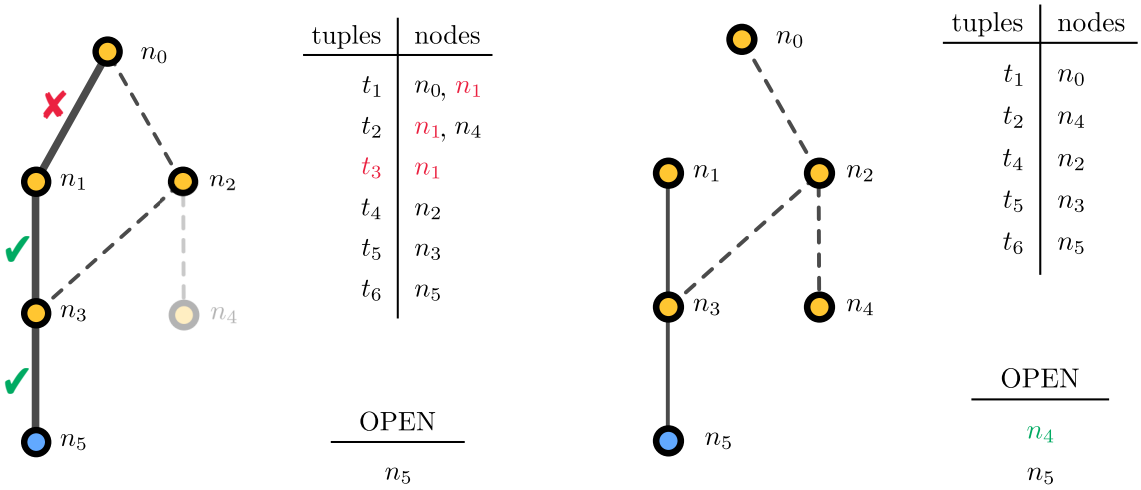


Figure 7: Novelty table and $OPEN$ queue before (left) and after (right) reparation.

In particular, in the developed Lazy-IW(k), the novelty table, in addition to containing the tuples seen so far within the search, contains information about the node that originated that tuple for the first time, as well as the nodes that would have also provided that tuple if the pioneer node would not have been discovered. Note that these non-pioneer nodes need not have been pruned since they could have provided any other novelty that would have ensured their survival. Therefore, in the case of non-lazily checking the actions of a potential plan and a node across the plan becoming orphan (i.e. it loses the connection to root because a movement planning has failed), the novelty table is repaired (see Figure 7). This repair implies taking into account the tuples that the orphan node originated but also those tuples originated by the successors that have also become orphans -not all the successors became orphans, since they can have another genealogical branch still connected to root. Those tuples that had been contributed only by the now-orphan nodes, must be deleted from the novelty table, leaving the novelty table as if those nodes had never been seen (because they are really not accessible from the root node). Furthermore, in the event that nodes had been discovered that also generated the affected tuples but had been pruned upon discovery for that very reason (Lines 32-33 at Algorithm 4), these nodes (and their successive descendant nodes) are inserted in front of the $OPEN$ queue (Lines 14-15 at Algorithm 2), to give the tree a chance to achieve the state it would have reached if the orphan nodes had never been seen (i.e. as if all the actions had been checked in a non-lazy way).

2.5.3 Lazy serialized iterated-width search with *sketches*

Iterated-Width search achieves good performance in problems with single-atom goals. Nevertheless, when the problem entails higher goal complexity, IW(k) is not powerful enough. However, the fact that most of these problems contain conjunctive goals can be exploited and, hence, they can be decomposed in single-goal subproblems. Thus, Serialized Iterated Width (SIW) tackles in a serialized way each of the conjunctive goals (i.e. one at a time) and applies IW(k) in each subproblem [15].

However, knowing how the goals should be serialized is not trivial. A novel approach, called SIW$_R$, solves this issue combining SIW with *policy sketches*, which allow expressing domain knowledge by turning it into rules used to decompose the problem. These rules operate with Boolean and numerical features and define a certain tendency that its values must follow. These rules do not need to be achieved in a single step, and it is not needed to express how the subgoals should be achieved. Rules contain a pre-condition part that must be fulfilled in order to be *active* and a target part that indicates how the values of certain features should evolve to consider that a subgoal has been found. Rules are mutually exclusive and, in each iteration of the SIW$_R$ algorithm, the active rule is selected by evaluating the rules pre-conditions. Then, the active rule generates a subgoal space and IW(k) is used to solve the associated subproblem.

A simple example is the trivial sketch rule $\{g > 0\} \mapsto \{g\downarrow, f_1?, f_2? \dots\}$, that operates on a single feature g, that here counts the number of pending subgoals (e.g. places still to be visited by the robot), while f_i are some other features. Without going into many details in the specification language of the sketch rules, this rule says that decreasing the goal counter (i.e. $g\downarrow$) is always *good* independently of the effects on other features (i.e. $f_i?$). Therefore, this sketch rule successively enforces the planner to decrease the number of pending subgoals, no matter in which order they are reached or how long it takes to achieve them. Note that this rule only applies (i.e. the rule is *active*) when the pre-condition is met (i.e. $g > 0$). If no rule is active (i.e. all the subgoals have been reached, in this example), the complete

Algorithm 5: LazySIW$_R$($start$, IsGoal, $Sketch$, $k_{\max}$)

```
1  plan ← {states : [start], actions : List( )};
2  current ← start;
3  while ¬IsGoal(current) do
4  |   IsSubGoal ← Sketch.GetActiveRule(current);
5  |   sub_plan ← LazyIW(start, IsSubGoal, k_max);
6  |   if sub_plan = ∅ then return ∅;
7  |   plan.Append(sub_plan);
8  |   current ← sub_plan.LastState( );
9  end
10 return plan;
```

problem has been solved. Some other more complex examples of sketch rules, applied to CTMP problems, are given in Section 3.2 below.

SIW$_R$ is, then, an algorithm that should be able to tackle CTMP problems from the perspective of its complexity. Nevertheless, as it has been explained, it is not feasible to make complete action-validations in each of the search steps in the proposed approach. For this reason, in the proposed framework, the lazy approach has been taken and SIW$_R$ has been adapted to use this approach. Therefore, SIW$_R$ is adapted to the lazy approach and named Lazy-SIW$_R$ (see Algorithm 5). Therefore, in order to solve a problem, Lazy-SIW$_R$ iteratively select the next subgoal based on provided sketches (Line 4), solves the subproblem (Line 5) and concatenates the subplans (Line 7), until the final goal is reached (Line 3). Note that, instead of calling IW(k) for solving each subproblem, Lazy-IW(k) is used and that all the subplans are completely valid (i.e. have been non-lazily checked).

Chapter 3

Experiments

After laying the theoretical foundations on which the proposed approach is based and explaining the developments made, this chapter shows how to particularize the framework on two different families of CTMP problems. These benchmarking problem families are also the validation scenarios in which the developed framework is tested and compared against different state-of-the-art approaches. Finally, the obtained results are thoroughly discussed.

3.1 Benchmarking problems

The proposed planning framework has been evaluated on two benchmarking problem families from the set proposed by Lagriffoul et al. [36], in an effort to ease comparison of the approach proposed in this work with future CTMP approaches (i.e. by using what is intended as a standard specification format and set of benchmarking problems for the field). This benchmarking problem set is independent of specific planners and specific robots to maximize portability to different platforms and support physical testing on available hardware. The selected problems imply pick-and-place actions in cluttered scenarios and are described as follows:

- **Sorting Objects:** A robot must arrange different objects standing on different tables, based on their color (see Figure 8). The goal constraints are that

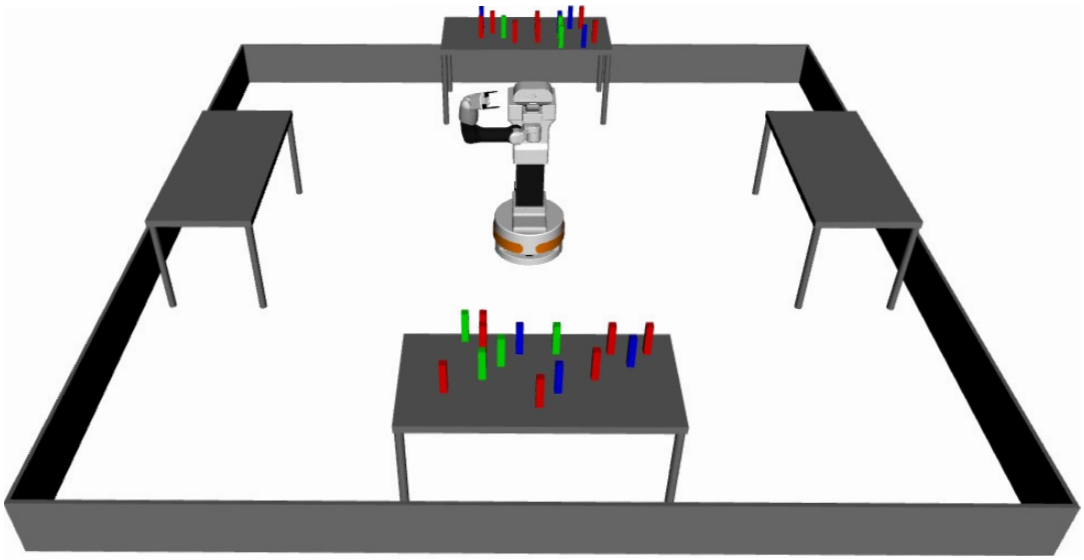


Figure 8: Scenario of the *Sorting Objects* problem.

all N blue blocks must be on the left table and all N green blocks must be on the right table. There are also $2N$ red blocks, acting as obstacles for reaching blue and green blocks, whose goal position is free. The robot is allowed to freely navigate around the tables, while keeping within the arena, picking and placing the blocks at the tables. The proximity between the blocks forces the planner to carefully order its operations, as well as to move red blocks out of the way without creating new obstructions (i.e. blocking objects). Therefore, solving the problem requires to move many objects, sometimes multiple times (i.e. large task-space).

- **Non-Monotonic:** A robot must move 3 green blocks standing on a table to their corresponding goal position on another table (see Figure 9). At the initial state, there are 4 red blocks obstructing the direct grasp of the green blocks and there are also 4 blue blocks obstructing the direct placement of the green blocks at their target position. Nevertheless, in this case, the red and blue blocks must end up being at the very same initial locations. The robot is allowed to freely navigate around the tables, while keeping within the arena, picking and placing the blocks at the tables. The goal condition of blue and red blocks requires to temporarily move them away and bring them back later on (non-monotonicity), in order to solve the problem.

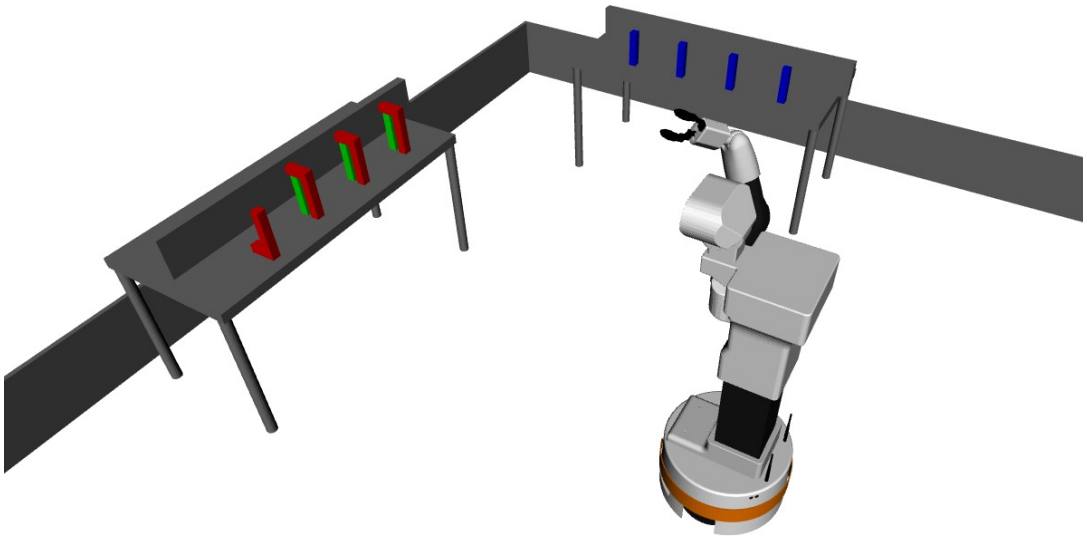


Figure 9: Scenario of the *Non-Monotonic* problem.

The selected problems are a good and broad representation of the challenges currently faced by the state of the art since they involve the following difficulties:

- **Infeasible task actions:** Some task actions are not possible, i.e. no corresponding motion-plan exists. Possible causes of unfeasibility include blocking objects and kinematic limits of the robot.
- **Large task-spaces:** The underlying task planning problem requires a substantial search effort.
- **Motion/Task trade-off:** The problem can be solved with fewer steps if grasps and placements are carefully chosen.
- **Non-monotonicity:** Some objects need to be moved more than once for achieving the goal.

From the current challenges addressed in the state of the art [36], the only not-tested difficulty here is needing non-geometric actions, i.e. actions which change discrete state but not robot or objects' configuration. Note that needing only non-geometric actions reduces the problem to stand-alone symbolic planning, which is not the focus of this master thesis.

3.2 Instantiation of the planning framework

The proposed framework makes no assumptions about which particular problem will be addressed. However, this general approach cannot be directly applied to solve a given specific problem without implementing the necessary planner components, introduced in Section 2.1. This section explains the specific implementation carried out for solving the selected problems with the developed framework. Note that this implementation is versatile enough to tackle both benchmarking problem families.

Firstly, the state and action spaces of the problems are specified. In addition, a definition is given about how the states are factored in atoms on which the planners evaluate the novelty-based pruning and about the space of atom tuples they spawn. In all cases, a study of their cardinality is reported as a function of the sampling parameters and the elements of the problem itself. Finally, the set of rules outlining the solution, as well as the variables upon which they operate, are given so that the planner makes use of the domain knowledge provided by the user to serialize and decompose the problem into subproblems.

State-Space

The state must completely specify the status of the environment. For the selected benchmarking problems, this implies that the problem state-space $\mathcal{S}$ contains the kinematic configuration of the robot and its location as well as the location of all the movable objects with respect a common coordinate system. Notice that no bits of symbolic state are needed here, as no non-geometric actions are involved in the considered problems. Formally,

$$\mathcal{S} = \mathbb{R}^d \times \underbrace{\mathrm{SE}(3) \times \cdots \times \mathrm{SE}(3)}_{O+1}, \tag{3.1}$$

where d is the number of degrees of freedom of the robot, O is the number of movable objects and SE(3) is the group of homogeneous transformations. This space may be high-dimensional for even moderate numbers of movable objects. Therefore, in order to obtain a sufficiently rich discretization that represents well the continuous

world of the problem but is still tractable, some assumptions have been made.

On the one hand, regarding the robot, a single *home* robot configuration is considered (see Figure 8.). Note that this does not imply that the robot cannot move its arm to pick up an object for example (which would make the problem unfeasible), but that the robot starts the *pick* action with its arm at *home* and that after picking up the object, the robot returns the arm to its original position (with the object grasped, though). In addition, a predefined number $B > 0$ of different robot locations are sampled around the tables (even if they are initially empty), ensuring that the robot may have access from different locations to any goal position and object standing in any table (providing maneuverability and facilitating the completeness of the approach). Note that the same base-location set is used for the entire search. An example of sampled robot locations can be found in Figure 4a.

On the other hand, regarding the movable objects (which do not have internal degrees of freedom in the considered problems), a set of $P \geq O$ different placements (i.e. locations) on the table surfaces are considered simultaneously for the object set, while $G_i > 0$ different grasps and $S_i > 0$ different SOPs (see Section 2.3) are allowed for each object i. The space of grasps and SOPs is defined in the benchmarking problem specification and the same predefined number of samples is drawn for each object and used for the entire search. However, to obtain the set of placements, an adaptive sampling has been implemented. At the start of each subsearch, a predefined number of placements are randomly sampled for each table in such a way that they are as distant as possible from the other placements, objects and goal positions on that surface (an example can be found in Figure 4b).

Therefore, formally,

$$|\mathcal{S}| = B \left(\prod_{i=1}^{O} S_i \right) \frac{P!}{(P-O)!} \left(1 + \frac{1}{P+1-O} \sum_{i=1}^{O} \frac{G_i}{S_i} \right), \tag{3.2}$$

which simplifies to, if $S_i = S$ and $G_i = G \; \forall i$,

$$\frac{B\, S^O P!}{(P-O)!} \left(1 + \frac{O\, G}{S\,(P+1-O)} \right) \approx \mathcal{O}\left(B\, G\, S^O P^O \right). \tag{3.3}$$

Table 1: Problem dimensions.

B	O	P	S_i	G_i	#States	#Actions	Avg. actions per state	#Atoms	#Atom Combinations
1	3	4	1	1	60	144	> 2	16	60
1	3	4	4	4	3840	36864	> 9	52	60
5	14	27	4	4	$> 4 \times 10^{27}$	$> 2 \times 10^{29}$	60	1587	$> 1 \times 10^{19}$
13	14	27	4	4	$> 1 \times 10^{28}$	$> 8 \times 10^{29}$	68	1707	$> 4 \times 10^{19}$
1	28	54	1	1	$> 1 \times 10^{45}$	$> 3 \times 10^{46}$	> 27	1541	$> 1 \times 10^{45}$
17	28	54	4	4	$> 1 \times 10^{63}$	$> 1 \times 10^{65}$	> 125	6541	$> 1 \times 10^{46}$

Notice, then, how the size of the state-space (and, therefore, also the problem complexity) increases exponentially with the number of movable objects (which are given by the problem definition), and polynomially with the sampling parameters (i.e. B, P, G_i and S_i). Examples of this growth can be observed in Table 1. Note that the start and the goal states are included in the defined state-space but, nevertheless, that not all states may be accessible from the initial state.

Action-Space

Actions connect problem states and are the only means by which the robot and objects can change internal state and location. In the selected problems, purely-symbolic actions are not necessary, so the action-space $\mathcal{A}$ is thus restricted to the following parametrized actions:

- *Pick* actions that allow the robot to pick up, from a given base location, a specific object, which is in a specific pose, using a specific grasp, as long as the robot is not already holding any object (in the considered problems, the robot has only one gripper). These actions end, when valid, with the given object in the robot gripper using the given grasp, while all other objects remain static.
- *Place* actions that allow the robot to place a particular object, which must be grasped with a given grasp, in a given placement using a given SOP. These actions finalize, if they are valid, with the specific object in the given target pose and the robot gripper being free, while all other objects remain static.

- *Move-Base* actions that, when valid, allow the robot to change its location from a given position to a different one, dragging with it the object it might be holding (while the rest of the objects remain static).

Notice, then, that combinations of these actions allow objects to change their position and orientation (using the robot), and that the effects of the described actions are always reversible applying some appropriate action sequence. Also, note that, for reasons of robot stability and robustness in the execution of the movements, the arm and the base are moved in an exclusive and sequential fashion. Thereby, the arm remains still during *move-base* actions and the robot base is immobile during *pick/place* actions.

Therefore, formally,

$$|\mathcal{A}| = B \left(\prod_{i=1}^{O} S_i \right) \frac{P!}{(P-O)!} \left(B - 1 + \frac{B-1}{P-O+1} \left(\sum_{i=1}^{O} \frac{G_i}{S_i} \right) + 2 \sum_{i=1}^{O} G_i \right), \quad (3.4)$$

which simplifies to, if $S_i = S$ and $G_i = G \; \forall i$,

$$\frac{B\, S^O P!}{(P-O)!} \left(B - 1 + \frac{(B-1)\, O\, G}{S\,(P+1-O)} + 2\, O\, G \right) \approx \mathcal{O}\left(B\, G\, S^O P^O \right). \quad (3.5)$$

Note how the size of the action-space also increases exponentially with the number of movable objects and polynomially with the sampling parameters (see some examples in Table 1). Notice, however, that not all actions may be valid due to geometric constraints (see Section 2.4), and that the feasibility of the actions is not known a priori (only when trying to plan them in the simulator), in the same way that the cost and effects of performing a given action are unknown.

Novelty atoms

The algorithm IW(k) prunes the discovered states that do not pass a novelty check. This filter operates on atoms which, in combination, factorize the states (see Subsection 2.5.2). For the considered benchmarking problems, the used atoms are of type `robot-at-`p and `object-`o`-at-`p, for some given poses p and objects o. This defini-

tion of atoms allows solving the benchmarking problems with a maximum novelty of 1 (as shown in the following Section). Note that, unlike the defined states, these atoms do not capture grasp information (reducing the atom-space size). Therefore, there exist $B\,(O+1) + P\sum_{i=1}^{O} S_i$ different atoms and, since each state is encoded as a combination of $O+1$ atoms, there exist

$$B\left(\prod_{i=1}^{O} S_i\right)\frac{P!}{(P-O)!}\left(1+\frac{1}{P+1-O}\sum_{i=1}^{O}\frac{1}{S_i}\right), \tag{3.6}$$

possible atom combinations. Then, the number of atoms grows linearly with the number of objects and the search parameters (see some examples in Table 1).

Sketch rules

The algorithm SIW_R is able to decompose the problem into sequential subproblem using a user-provided domain knowledge, encoded into a set of rules that outline a sketch of the solution (see Subsection 2.5.3). These rules operate on state features that capture relevant information about the problem status. For the benchmarking problems, the following features have been implemented:

- H: is a Boolean variable that equals true if the robot is currently holding an object, and false otherwise.
- m: is the number of *misplaced* objects, defined as the objects that are currently standing -and, hence, not held- outside their associated goal position/region (if the object has no associated goal position, it is never misplaced) or are held and their goal position/region is completely blocked. Furthermore, in the case of the *Non-Monotonic* problem, a hierarchy is established in which blue and red objects are not considered misplaced, even though they are not in their goal position, until the green objects are all in their desired position.
- n: is the number of blocking objects of the most accessible misplaced object (i.e. the one with fewer objects preventing it from being picked up from its current position and put down in its goal position/region), and equals 0 if there are no misplaced objects. To calculate the number of blocking objects

for a given object, the minimum number of objects that the robot could collide with when picking it up from its current position and putting it down in its goal zone is calculated. Note that approximations are used and no inverse kinematics is solved within this computation, to speed up the process. During the search for such a minimum value, it is considered the best robot location and grasp for the *pick* action and the best robot location, placement and SOP for the *place* action in the goal zone, while maintaining consistency (i.e. using the same grasp for the considered *pick* and *place* pair).

- s: is the sum of the number of objects currently obstructing all the misplaced objects and their associated goal position/region, and equals 0 if there are no misplaced objects.

Using these features, the goal in both family problems is repreresented by the expression $\neg H \wedge m = 0$ (i.e. the robot is not holding any object and all the objects are at their goal positions).

In adition, the following mutually-exclusive sketch rules have been established for both benchmarking problem families:

- $\{\neg H,\, m > 0,\, n = 0\} \mapsto \{H,\, m\!\downarrow,\, n?,\, s?\}$: It is active when there is at least one misplaced object ($m > 0$) that can be picked up directly ($\neg H$ and $n = 0$), enforcing to pick up any misplaced object (i.e. decrement m and make H true).

- $\{\neg H,\, m > 0,\, n > 0\} \mapsto \{H,\, m?,\, n?,\, s\!\downarrow\}$: It is active when there are still misplaced objects ($m > 0$) but they cannot be picked up directly -in theory- ($\neg H$ but $n > 0$). Here, the robot is asked to pick any obstructing object (i.e. decrement s). However, since the estimation of n may be not totally accurate, picking an unexpectedly-accessible misplaced object would also satisfy this rule (which is not an inconvenience at all).

- $\{H\} \mapsto \{\neg H\}$: It is active whenever the robot is holding an object. Then, the robot is asked to place the grasped object but not in a wrong position (i.e. do not modify the value of m) and without disturbing the access to the pending misplaced objects (do not change n or s).

3.3 Results and discussion

The proposed framework has been evaluated in different instances of the selected benchmarking families, to analyze its performance in various environments but also to allow direct comparison with other approaches in the same scenario. The framework has been run in an Intel® Core™ i7-10610U CPU at 1.80 GHz, with 8 processors and 16 Gb of RAM, on Ubuntu 20.04.4 and ROS Noetic. The results obtained by the proposed approach, together with the ones of some relevant algorithms, can be found on Table 2. Videos of the plans obtained with the proposed framework can be found in `https://drive.google.com/drive/folders/10goVJ8A86RGIGsbthdmak8L_mTdT6hby?usp=sharing`. Non-Lazy and/or non-goal-serialized versions have also been evaluated in these same scenarios to evaluate the performance improvement provided by each of the proposed strategies of this approach. However, the results obtained without using goal-serialization are not included since these planners cannot solve any problem in the allocated time. Note that these algorithms, i.e. lazy BrFS and IW(k), are still valuable as they are the building blocks of the proposed approach.

All the considered problems have been successfully solved within the maximum time budget of 1 h. Note that the budget time, which would not be admissible on most real applications, has set intentionally generous to show the soundness and completeness of the algorithm. During the experimentation, the effectiveness of the planner has been demonstrated by not requiring planning times longer than the execution time of the produced plans. Indeed, the proposed approach requires less time to obtain a solution plan compared with the results obtained by Ferrer-Mestres et al. [23], which also use a width-based algorithm, and Thomason and Knepper [37], which perform normal motion planning in a space including both geometric and symbolic states. Notice that, in order to make a fair comparison, any pre-computation or other planning effort conducted offline are also considered as planning time in Table 2. Note that the proposed approach specifically avoids this sort of offline work since, in a real application, there is no guarantee that the objects will be in the positions foreseen in the pre-processing, or that objects of different geometry will not be used.

Table 2: Average results in the benchmarking problems.

Problem family	#Tables	#Objects	#Goal objects	Clutter level	Planner	Success Ratio	Planning time	Execution time	Total time$^{\perp}$	#Expanded nodes	#Sub plans
Sorting Objects	1	20	2	High	**Lazy-SIW$_R$**	100%	3.16 min	3.67 min	6.83 min	60	14
					SIW$_R$	100%	13.05 min	3.73 min	16.78 min	54	14
					BFWS [23]	100%	5.25 min*	5.91 min	11.16 min	63.3k	1
	3	25	5	Medium	**Lazy-SIW$_R$**	100%	3.36 min	3.95 min	7.31 min	235	10
					SIW$_R$	100%	21.60 min	4.02 min	25.62 min	198	10
					BFWS [23]	100%	13.47 min*	7.61 min	21.08 min	3.5k	1
	4	28	14	Medium	**Lazy-SIW$_R$**	100%	6.52 min	9.24 min	15.76 min	630	34
					SIW$_R$	100%	47.38 min	9.17 min	56.55 min	587	34
					Planet [37]	N/A	>1 h	≈10 min	>1 h	N/A	1
Non-Monotonic	2	10	10	Low	**Lazy-SIW$_R$**	100%	3.53 min	8.10 min	11.63 min	565	30
					SIW$_R$	100%	17.69 min	7.96 min	25.65 min	534	30

$^{\perp}$ Considering that the path execution does not start until the path has been completely planned.

* Including also the pre-processing time, for a fair comparison.

Table 3: Profiling of the action-validation pipeline in the benchmarking problems.

Step	Function	Time on success	Timeout	Relative success ratio	Cumulative success ratio
1	*InArmWorkspace*	35 µs	N/A	5%-30%	5%-30%
2	*InverseKinematics*	9 ms	150 ms	20%-40%	1%-10%
3	*MotionPlan*	0.38 s	5 s	80%-100%	0.5%-10%

The planning time depends on both the number of expanded nodes and the required time for expanding each node. Regarding the first factor, even though the problem search-space is huge (see Table 1), the planner manages to find a solution expanding only a minimal fraction for all the problems. Thanks to the used atom features and sketch rules (introduced in Section 3.2), a novelty of 1 has been enough to solve all the problems. In this way, the underlying IW(k) algorithm has been able to apply the maximum level of node-pruning (thus, minimizing the number of expanded nodes). In addition, the generated decomposition in subproblems has been very useful to guide and focus the search by obtaining a sequence of subgoals very close to each other (i.e. they require limited exploration to reach them consecutively).

Furthermore, the followed strategy of adaptive sampling has allowed to use a reduced but useful selection of placements in each subproblem (see Section 3.2). With fewer placements, the search-space is reduced but, in addition, as the sampled placements have been automatically adapted to the specific situation of the subproblem, the subgoals are reached quickly.

The node-expansion time in the proposed approach mostly depends on the geometric validation of each of the actions, which includes three steps: *InArmWorkspace*, *InverseKinematics* and *MotionPlan* (see Section 2.4). These steps act as sequential filters (i.e. if a step fails, the next ones are not computed and the action is set as unfeasible). Note that they act in increasing order of computational load and decreasing order of relative discriminative power (i.e. rejection ratio over the non-filtered actions in the previous step). This permits discarding quickly the majority of unfeasible actions with a very small computational cost (see some illustrative values in Table 3). In addition, decomposing the action-validation has enabled delaying

the computation of the most expensive step (i.e. *MotionPlan*). Thus, the complete validation is only computed on those actions that are part of a potential plan (i.e. *lazy approach*). Note that satisfying the *InverseKinematics* step almost ensures that the action is valid. This is essential to make using the lazy approach advantageous. Indeed, as it can be seen on Table 2, following the lazy approach (i.e. Lazy-SIW$_R$) reduces drastically the planning time when compared to plain SIW$_R$.

Implementing the lazy-approach, however, has implied some challenges. Since the actions are all provisional (until a plan is validated), the planner needs to keep all the found parents in each node to prevent discarding connections that are part of the solution. Furthermore, the planner keeps record on node chains that are connected to a goal but got sometime disconnected from the root node within a plan validation. These two mechanisms become relevant when many nodes need to be expanded in a subsearch. Another important challenge of the lazy approach is the adaptation of the IW(k) to keep track of the novelty seen by the algorithm. Thus, when a provisional action turns to be unfeasible, if the node remains orphan (i.e. disconnected from the root node) the novelty introduced by this node has to be revised (see Subsection 2.5.2). Running the algorithm over the different problems has implied the activation of this mechanism in multiple occasions. This has prevented to wrongly prune graph branches that are part of the solution.

As well as the planning time, another important metric is the quality of the produced plans. After experimenting with the proposed algorithm in several problems, it is observed that the produced solutions are near-optimal. Although optimality can not be formally ensured, the proposed algorithm performs notably on avoiding unuseful actions. This is facilitated by the proposed sketch since its rules discourages moving non-misplaced objects that do not block any goal object or goal placement but promotes moving the easier goal-objects first. Hence, after moving these objects, often other goal objects became accessible without needing to move non-goal objects. Indeed, when comparing the results with the work of Ferrer-Mestres et al. [23] and the work of Thomason and Knepper [37], plans containing fewer actions are obtained for the same problems with the proposed approach.

Moreover, from the obtained results, it can be concluded that the proposed algorithm performs well even with a high number of objects and goals. However, high-cluttered problems are challenging. On these problems, the goal objects can be obstructed by many objects, which must be moved away. Nevertheless, there is a limited space where the obstructing objects could be set aside (without blocking other necessary areas). In this scenario, most of the inverse-kinematic computations fail, reaching the function timeout. This implies a slower node expansion and a higher difficulty to find valid actions. In addition, because there are so few gaps, in these problems the placements must be sampled more densely to ensure that there exists an accessible placement. On the contrary, the approach of Thomason and Knepper [37] does not scale well with the number of objects. The work of Ferrer-Mestres et al. [23] scales well, though, but at the cost of rigidity as it pre-compiles and forces the objects to be in a fixed grid distribution during the search process. With this inflexible approach, the planner manages to reuse many inverse-kinematic and motion-planning computation because the grid is always in the same coordinates relative to the robot even if the robot is in a different location.

Finally, it is important to remark that fairly benchmarking CTMP planners is still an unsolved challenge. Here, the work from Lagriffoul et al. [36], which aims to become a CTMP benchmarking standard, has been intentionally used to facilitate present and future comparisons. Currently, this standard (nor any other) has not reached enough popularity and only few works have reported results in some of the proposed problems.

After analyzing the results, it can be concluded that the proposed approach outperforms the available approaches which implement the selected benchmarking problems. In addition, this work has proven that its planning time is reasonable with respect to the plan execution time and that the produced plans are near-optimal.

Chapter 4

Conclusions

This chapter summarizes the main contributions of this work and discusses the possible future work.

4.1 Contributions

This master thesis has addressed the problem of planning simultaneously the tasks (i.e. symbolic level) and movements (i.e. geometric level) of robots to solve a given problem where these two issues cannot be tackled in a decoupled manner (i.e. combined task and motion planning, CTMP, is needed). Thus, a new framework has been developed, capable of tackling CTMP problems, obtaining robot plans that seem apparently reasonable (i.e. near-optimal), quickly and efficiently, even for complex problems. The developed framework includes a new algorithm based on width-based search algorithms and implements a novel *lazy-interleaved* strategy to reduce the computational cost of the planning, without making any assumptions on the robot used or the number, types, locations and geometries of the objects. Therefore, the efficiency of the approach is increased by focusing motion-planning efforts only in potential solutions, thus, obtaining a fully-online and flexible approach that does not require any pre-computing at all. Besides, domain knowledge provided via sketch language has been integrated in the proposed approach, proving to be very effective to tackle CTMP problems with huge-combinatorial state-spaces. The pro-

posed approach is able to work transparently with robotic continuous state-spaces with flexibility using an adaptive sampling of the world to build the search space of the problem. Moreover, the problem dynamics are retrieved as a black box, so the developed planner is able to work directly with a simulator, and it does not need an explicit declaration of the action structure. The proposed approach has been satisfyingly validated in two problems families, which imply several difficulties characterizing the current challenges addressed in the state of the Art. Furthermore, the proposed approach has been compared with state-of-the-art approaches, obtaining significantly better results. Finally, the proposed approach has been combined and integrated within the ROS environment in order to provide the highest level of standardization and compatibility with the maximum number of robotic systems currently available. In conclusion, it can be confirmed that all the scientific questions posed in this master thesis have been positively answered, thus achieving the pursued objectives.

4.2 Future work

The combined task and motion planning in mobile manipulators (i.e. mobile robotic platform equipped with one or more robotic arm and gripper) involves several levels of complexity and new features of practical relevance can be addressed. In this work, some novel solutions to the presented problem have been proposed, nevertheless, there are still several interesting topics that could be treated in future work:

- The validation of the proposed approach in more problems, considering non-geometric actions (e.g. activating the robot camera), that may involve dual-arm manipulation (e.g. considering hand-over) or more than one robot simultaneously.
- The definition of sketches to tackle new problem families and the inclusion of learning AI-based techniques into the proposed approach to obtain self-learned rules, features and sampling strategies.
- The extension of the proposed approach to consider optimality and being

able to tackle problems with state uncertainty and partial observability, where dynamics- or physics-based planning is needed.

- The validation of the proposed approach with a real robot, performing simultaneous planning and execution (i.e. the robot starts the execution as soon as a subplan is available while the planner searches for the next subplan in parallel).

- Post-processing and smoothing of the obtained plans by using optimal motion-planning algorithms (e.g. Informed RRT* [38]) and merging and shortening the trajectories of consecutive actions.

- The automatic generation of grasps [39] and stable object placements [40], based on the object and gripper meshes, instead of sampling them from a given set.

- Improving the performance of the proposed approach with code optimization, CPU parallelization and GPU acceleration.

- Obtaining a better probabilistic completeness of the proposed approach than just re-running the planner (note that the sampling implicit in the proposed approach is random, offering at each iteration a different set of placements, inverse-kinematic solutions, motion plans ...). Some improvements could include, for instance, dynamic-density sampling and online resampling within a planner run if the planner is struggling to find a solution.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] Mansouri, M., Pecora, F. & Schüller, P. Combining task and motion planning: Challenges and guidelines. *Frontiers in Robotics and AI* **8** (2021).

[2] Geffner, H. The model-based approach to autonomous behavior: A personal view. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, 1709–1712 (2010).

[3] Colledanchise, M. & Ögren, P. How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees. *IEEE Transactions on robotics* **33**, 372–389 (2017).

[4] Ghallab, M., Nau, D. & Traverso, P. *Automated Planning and Acting* (Cambridge University Press, 2016).

[5] Kaelbling, L. P., Littman, M. L. & Moore, A. W. Reinforcement learning: A survey. *J. Artif. Int. Res.* **4**, 237–285 (1996).

[6] Garrett, C. R. *et al.* Integrated task and motion planning. *Annual Review of Control, Robotics, and Autonomous Systems* **4**, 265–293 (2021).

[7] Latombe, J.-C. *Robot Motion Planning* (Kluwer Academic Publishers, USA, 1991).

[8] Pan, J., Chitta, S. & Manocha, D. FCL: A general purpose library for collision and proximity queries. In *2012 IEEE International Conference on Robotics and Automation*, 3859–3866 (2012).

[9] Geffner, H. & Bonet, B. *A Concise Introduction to Models and Methods for Automated Planning* (Springer Cham, 2013).

[10] Bonet, B. & Geffner, H. Planning as heuristic search. *Artificial Intelligence* **129**, 5–33 (2001).

[11] Fikes, R. E. & Nilsson, N. J. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* **2**, 189–208 (1971).

[12] Klein, I. & Bäckström, C. Planning in polynomial time : The SAS-PUBS class. Tech. Rep. 1372, Linköping University, Automatic Control (1992).

[13] McDermott, D. *et al.* PDDL: The planning domain definition language. *Technical Report* (1998).

[14] Francès, G., Ramírez, M., Lipovetzky, N. & Geffner, H. Purely declarative action representations are overrated: Classical planning with simulators. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 4294–4301 (2017).

[15] Lipovetzky, N. & Geffner, H. Width and serialization of classical planning problems. In *Frontiers in Artificial Intelligence and Applications*, vol. 242, 540–545 (2012).

[16] Lipovetzky, N. & Geffner, H. Width-Based algorithms for classical planning: New results. In *ECAI* (2014).

[17] Lipovetzky, N. Width-Based algorithms for common problems in control, planning and reinforcement learning. In *IJCAI* (2021).

[18] Drexler, D., Seipp, J. & Geffner, H. Expressing and exploiting the common subgoal structure of classical planning domains using sketches: Extended version. In *KR* (2021).

[19] Dechter, R. *Constraint processing* (Morgan Kaufmann Publishers, San Francisco (Calif.)[etc, 2003).

[20] Toussaint, M. Logic-Geometric programming: An optimization-based approach to combined task and motion planning. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, 1930–1936 (AAAI Press, 2015).

[21] Srivastava, S., Riano, L., Russell, S. & Abbeel, P. Using classical planners for tasks with continuous operators in robotics. *AAAI Workshop - Technical Report* 85–91 (2013).

[22] Lagriffoul, F., Dimitrov, D., Bidot, J., Saffiotti, A. & Karlsson, L. Efficiently combining task and motion planning using geometric constraints. *The International Journal of Robotics Research* **33**, 1726–1747 (2014).

[23] Ferrer-Mestres, J., Francès, G. & Geffner, H. Combined task and motion planning as classical AI planning. *ArXiv* **abs/1706.06927** (2017).

[24] Akbari, A., Muhayyuddin & Rosell, J. Task planning using physics-based heuristics on manipulation actions. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, 1–8 (2016).

[25] Garrett, C. R., Lozano-Perez, T. & Kaelbling, L. P. Ffrob: Leveraging symbolic planning for efficient task and motion planning. *The International Journal of Robotics Research* **37**, 104 – 136 (2018).

[26] Lozano-Pérez, T. & Kaelbling, L. P. A constraint-based method for solving sequential manipulation planning problems. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 3684–3691 (2014).

[27] Dornhege, C. *et al.* Semantic attachments for domain-independent planning systems. In *ICAPS* (2009).

[28] Braun, C. V., Ortiz-Haro, J., Toussaint, M. & Oguz, O. S. RHH-LGP: Receding horizon and heuristics-based logic-geometric programming for task and motion planning. *ArXiv* **abs/2110.03420** (2021).

[29] Quigley, M. *et al.* ROS: An open-source robot operating system. In *ICRA Workshop on Open Source Software*, vol. 3 (IEEE, 2009).

[30] Coleman, D., Sucan, I., Chitta, S. & Correll, N. Reducing the barrier to entry of complex robotic software: a moveit! case study (2014).

[31] Rosell, J. *et al.* The Kautham project: A teaching and research tool for robot motion planning. In *19th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2014* (2014).

[32] Kuffner, J. & LaValle, S. 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, 995–1001 vol.2 (2000).

[33] Chiaverini, S., Siciliano, B. & Egeland, O. Review of the damped least-squares inverse kinematics with experiments on an industrial robot manipulator. *Control Systems Technology, IEEE Transactions on* **2**, 123 – 134 (1994).

[34] Hoffmann, J. & Nebel, B. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research - JAIR* **14** (2011).

[35] Moore, E. F. The shortest path through a maze. In *Proceedings of the International Symposium on the Theory of Switching*, 285–292 (Harvard University Press, 1959).

[36] Lagriffoul, F. *et al.* Platform-Independent benchmarks for task and motion planning. *IEEE Robotics and Automation Letters* **3**, 3765–3772 (2018).

[37] Thomason, W. & Knepper, R. A unified sampling-based approach to integrated task and motion planning. In *Proceedings of the International Symposium on Robotics Research (ISRR)* (2019).

[38] Gammell, J. D., Srinivasa, S. S. & Barfoot, T. D. Informed RRT: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2997–3004 (2014).

[39] Pas, A. t., Gualtieri, M., Saenko, K. & Platt, R. Grasp pose detection in point clouds (2017).

[40] Jiang, Y., Lim, M., Zheng, C. & Saxena, A. Learning to place new objects in a scene. *The International Journal of Robotics Research* **31**, 1021–1043 (2012).