UNIVERSITY OF GENOVA

PHD PROGRAM IN BIOENGINEERING AND ROBOTICS

Keep the planner in the loop: parallel planning and execution using Large Language Models

by

Alessio Capitanelli

Thesis submitted for the degree of *Doctor of Philosophy* (36° cycle)

May 2024

Fulvio Mastrogiovanni — Supervisor
Paolo Massobrio — Head of the PhD program

Thesis Jury:
Antonio Chella, *Università degli Studi di Palermo* — External examiner
Dimitri Ognibene, *Università degli Studi Milano-Bicocca* — External examiner
Antonio Sgorbissa, *Università degli Studi di Genova* — Internal examiner

Dibris

Department of Informatics, Bioengineering, Robotics and Systems Engineering

Dedicated to Enzo Capitanelli

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Alessio Capitanelli
May 2024

Acknowledgements

First and foremost, I would like to say thank you to my parents, Massimo and Ermenia. It goes without saying that without them I would not be here to write this thesis. Less obviously, they have been by far the people that have encouraged me the most to undertake a PhD, and, literally, any other study before. I know for sure that in their mind a good education has always been the most valuable asset that they could provide me. I agree with them.

Close seconds, I am truly grateful to all the amazing people that share a home with me. Giorgia, Isadora and Federico, all had to stand my frustration during the last three years, as well as my enthusiasm in those rare days when something was working. Giorgia in particular had to bear me more than anyone else. Her loving and unwavering support gave me all the strength I needed to succeed. Federico, being an amazing researcher in Neuroscience, was forced to listen to my ideas way too many times. Several of the insights in this thesis would not be there without us having those discussions.

From now on, in no particular order, there are many other people to whom I would like to extend my gratitude.

To my fellow researchers in the Engine Room lab, thank you. Mohamad Shaaban collaborated with me in order to obtain the results presented in Chapter 3 and has shown great patience when, while I was working on the results presented in Chapter 7 and I occupied most of the lab's computational power. Simone Macciò, having started our PhD projects together, has accompanied me through every step of this journey, and it was great to work with when we had the chance to collaborate on other research projects. Alessandro Carfì, as the most senior postdoc in the lab, provided much needed feedback at all stages of this project.

To Fulvio Mastrogiovanni, Mauro Vallati e Marco Maratea, thank you. Fulvio being my supervisor does not require much of an explanation. He encouraged me to undertake this PhD project and has left me great freedom to pivot it in the direction I wanted. I hope this bet paid out well. Much of the work in Chapter 2 and in Chapter 3, are the results of the joint effort of the four of us. Both Mauro and Marco provided invaluable expertise and

guidance on all issues regarding symbolic task planning, and Mauro in particular provided some insightful feedback after the results in Chapter 6 were first made public.

To Teseo and its team, thank you. Teseo is the company that I have been working for the last six years. Despite me moving to a less technical position over time, the company has not only funded my PhD studies, but also left me great freedom on the direction that my research should take and provided me much needed time to work on it.

To Antonio Chella of the University of Palermo and Lucia Passaro of the University of Pisa, thank you for the time spent reviewing the present manuscript. Your feedback has been highly appreciated.

For no particular reason, except their friendship during the last few years, thank you Alessandro and Fosca.

Abstract

Task planning is a popular approach for autonomous agents due to its understandability, predictability, and ease of deployment. However, it is difficult to scale in real-world, human-robot cooperation scenarios due to the poor performance scaling in complex planning domains and when frequent re-planning is needed. Longer planning times can hinder the robot's efficiency and adversely affect the interaction's fluency. Our objective in this PhD project is to develop novel methods to address this issue and favor keeping task planning in the execution loop as much as possible. First, we explore the use of traditional planning techniques, and in particular the use of macros, to optimize total planning and execution time. Macros are known to reduce planning time, but at the cost of plan optimality and thus execution time. We provide evidence that by selecting an appropriate level of macro abstraction and by implementing ad-hoc grounding for said macros, it is possible to reduce average planning time by 85% with little impact on execution time. Then, we proceed to explore more innovative approaches based on the latest advancement in generative AI. In particular, we propose a method, Teriyaki, to bridge the gap between symbolic task planning and machine learning methods, by training Large Language Models (LLMs), namely GPT-3, into a neurosymbolic planner compatible with the Planning Domain Definition Language (PDDL). Potential benefits include better scalability, as LLMs' response time scales with the combined length of the input and the output regardless of the symbols involved; and the ability to generate a plan action-by-action, which in turn enables simultaneous planning and execution, reducing wait times. In the past year, significant effort has been devoted by the AI community to evaluate the overall cognitive abilities of LLMs, but success rate has been limited. Instead, we focus on providing a success rate comparable to traditional planners in specific planning domains, while improving other real-world metrics. Preliminary results in two domains selected from those developed in the first part of this project, show that our method can: (i) solve 95.5% of problems in a test data set of 1000 samples, a result comparable with that of the baseline heuristic-search planner; (ii) produce plans up to 13.5% shorter than a traditional planner; (iii) reduce average waiting times for a plan by 61,4% and its standard deviation by 96.6% through parallel planning and execution.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Before moving to the industry and later returning to academia as an industrial PhD candidate, I was a research fellow at the University of Genoa for two years. It was a good time. A Ph.D. project bounds your body and soul to a single research project, but at the time I had the privilege and freedom to work on many, and often drastically different, aspects of Robotics and AI: from machine learning techniques for pallet detection and tracking in warehouse scenarios, to simultaneous coverage and relay positioning algorithms for swarms of UAVs. Nevertheless, one project still occupied most of my time as a research fellow: automated Planning and Reasoning for articulated object manipulation in Human-Robot Cooperation (HRC) scenarios. That sounds oddly specific (it is), but there was a good reason for it.

During my master's thesis, I developed an ontology-based hybrid architecture for planning and robust execution in tabletop scenarios and became passionate about classical AI methods such as ontologies and high-level symbolic task planning. One of the features of that first work was the capability to receive desired states, understand when they were unsatisfied, and plan and act accordingly. This fact made my architecture extremely resilient to unexpected failures or interventions from humans around the robot: if in the belief system of the robot, there was the axiom "all blue spheres must be on the left", but signals from the perception contradicted such prescription, the robot would wake up and hurry to correct the mistake. The same would happen if the ball slipped from the robot's hands, or if a malevolent researcher nearby decided to move the ball while the robot was busy with another task. The robot was constantly re-planning and adjusting its course of action to satisfy the prescriptions in its belief system, so that it could later come to rest. It was 2015 and while it was a nice concept at the time, the tabletop setting was very limited and I could not help to joke that I had built a generalized version of a Minsky's useless box.

My work on articulated objects was a direct continuation of that first project. The idea was to polish and scale the tools and methods that I had developed and tested in the simple tabletop scenario and bring them closer to the real world. Manipulating an articulated object was still a toy problem in many ways, but it was a fair approximation of a complex shop-floor operation, namely cable harnessing. It offered many more interesting Human-Robot Cooperation opportunities and also presented computational complexities meant to stress the architecture, especially the underlying heuristic-search planner. This work eventually resulted in a publication on Robotics and Autonomous Systems Capitanelli et al. (2018) and now constitutes a large part of the groundwork detailed in Chapter 2, but despite its merits in terms of ease of understanding and deployment, as well as HRC capabilities, it also shown that we were reaching the limits of what was achievable with symbolic planning in the loop when frequent re-planning was necessary.

My passion for symbolic planning was turning into frustration as it became clear that clever modeling and architectural design can only help us so much in the face of a brutal combinatorial explosion as planning domains become more complex in the real world. In the articulated object manipulation scenario, this meant that increasing the number of links of the object and their allowed angle values, rapidly deteriorated performance. When planning with Probe, a state-of-the-art PDDL planner, our most complete model would lead to a median planning time of around 10s to reconfigure a 10-link object while considering only 6 possible angle states. The variance was even worse.

A few seconds might not sound that awful at first sight, but they are an issue when frequent re-planning is required, especially when humans are involved in the loop. Hoffman (2019) defined the concept of fluency in HRC and demonstrated that the delay experienced by a human immediately after completing an action, as incurred by their teammate, has a strong correlation with subjective fluency perception. I can confirm that when I was working with my Baxter robot in 2017 and every interaction I had with the robot caused a 10s pause, I was not delighted by the experience.

My frustration grew further as I witnessed first-hand the spring of Machine Learning methods and how the other half of the AI community was tackling problems in computer vision that, until a couple of years before, were widely considered computationally prohibitive. At the time, the logical reasoning capabilities of machine learning methods were still limited and, while I never believed that they would eventually replace all other AI techniques, I started to mature the conviction that a hybrid approach, merging neural and symbolic AI, could be highly effective for the kind of problems I was interested in. An AI that can sequence high-level actions, easy to deploy, explainable, and understandable by humans in

HRC scenarios; but also an AI that is resistant to combinatorial explosion and that could leverage the power of highly parallel computing. At the time, this was little more than a vague idea, but as the state of the art progressed, it became increasingly concrete and reasonably achievable in the span of a Ph.D. project.

Fast forward to 2020, after a couple of years break from research, I decided to start my Ph.D. As you might have guessed from this lengthy introduction, the main objective of this project is to keep the high-level planning in the execution loop of a robotic architecture by building on the groundwork laid down in my previous projects, and in particular in the domain of articulated object manipulation. This translates into four main scientific questions, which I will try to answer in this pages: (i) is it possible to still squeeze some performance in terms of total planning and execution time through clever modeling?; (ii) Can the latest neurosymbolic AI methods such as Large Language Models be used to emulate heuristic-search planners while maintaining comparable accuracy levels?; (iii) What performance can we expect from such a hypothetical neurosymbolic planner and how does it scale?; (iv) Can we use it to improve the fluency of the interaction in HRC scenarios?

Fortunately, the answers to these questions are mostly positive and can be found on these pages. We will start in Chapter 2 by better defining the articulated object scenario, introducing the base planning models that we will use throughout the rest of the thesis, and highlighting the performance issues that motivated this work. As anticipated, a large part of this chapter derives from the article *"On the manipulation of articulated objects in human-robot cooperation scenarios"* which I published on Robotics and Autonomous Systems in 2018. In Chapter 3 we propose an extension of the base planning models which use macros, test it in simulation, discuss the trade-off existing between planning and execution times, and show that a good choice of macro can be highly beneficial to total planning and execution time. This work is still unpublished as successive works were prioritized, but it is important to present to the reader as one of the models developed in it is the basis for successive advancements. In Chapter 4 we go beyond traditional heuristic-search planners and introduce the state of the art of Large Language Models and how their logical capabilities have been assessed so far. In Chapter 5 we introduce Teriyaki, a framework method to generate PDDL-compliant neurosymbolic planners using LLM, and specifically GPT-3. In Chapter 6 we analyze the performance of Teriyaki and we highlight how, unlike traditional heuristic-search planners, it allows to generate a plan action-by-action, enabling simultaneous planning and execution and improving HRC interaction fluency. Chapters 4, 5, and 6 are largely based on the article *"A Framework for Neurosymbolic Robot Action Planning using Large Language Models"* submitted to Frontiers in Neurobotics in 2023. In Chapter 7, we

explore the possibility of replicating Teriyaki using smaller, local LLM instead of GPT-3. While in my experience so far local models have proved unable to solve planning problems consistently, the technology is maturing fast and the interested reader might find these preliminary results useful to select the right model, size and training hyperparameters before venturing in this field. Conclusion follows in Chapter 8.

Chapter 2

The scalability issue: the articulated object example

2.1 What is an architecture for Human-Robot Cooperation?

The Industry 4.0 paradigm is expected to redefine the nature of shop-floor environments, including the role played by robots in the manufacturing process Heyer (2010a); Krüger et al. (2009). One of its main tenets is the increased customer satisfaction via product personalization and just-in-time delivery. A higher level of flexibility in manufacturing processes is needed to cope with diversified demands, especially in low-automation tasks. Collaborative robots are regarded as a valuable aid to shop-floor operators, who can supervise robots' work and intervene when needed Tsarouchi et al. (2016), whereas robots can be tasked with difficult or stressful operations.

Human-robot cooperation (HRC) in the shop-floor is a specific form of human-robot interaction (HRI) with two important specificities. The first is the fact that cooperation is targeted to a well-defined objective (e.g., an assemblage, a unit test, a cable harnessing operation), which must be typically achieved in a short amount of time. The second has to do with the fact that humans need to feel in control Baraglia et al. (2016); Munzer et al. (2017): human behaviour could be unpredictable in specific cases, with obvious concerns about safety Denei et al. (2015); Haddadin and Croft (2016), humans may not fully understand robot goals Chakraborti et al. (2017), and robot actions may not be considered appropriate for the peculiar cooperation objectives Goodrich and Schultz (2007); Munzer et al. (2017).

Figure 2.1 Two examples of a cable harnessing operation.

As far as the cooperation process is concerned, two high-level directives must be considered:

D_1 it is necessary to adopt human-robot *cooperation models* and the associated robot action planning techniques to meet cooperation objectives Darvish et al. (2017); Johannsmeier and Haddadin (2017);

D_2 robots must be flexible enough to adapt to human actions while (i) fulfilling the overall cooperation objectives Dautenhahn (2007); Prewett et al. (2010), and (ii) making their intentions clear to human operators Clair and Matarić (2015); Roncone et al. (2017).

These directives lead to three functional requirements for a HRC architecture. Collaborative robots must be able to:

R_1 recognize the effects of human operator actions Liu and Wang (2017);

R_2 adapt their behaviour considering human actions and the whole cooperation objectives;

R_3 employ planning techniques allowing for an appropriate action re-planning when needed, e.g., when planned actions cannot be executed for sudden changes in the environment or inaccurate modeling assumptions Srivastava et al. (2014).

2.2 Manipulating articulated objects in a HRC scenario

Among typical shop-floor tasks, the manipulation of articulated or flexible objects, e.g., cable harnessing operations, is particularly challenging Henrich and Worn (2000); Jimenez (2012);

Saadat and Nan (2002); Smith et al. (2012), as can be seen in Figure 2.1. In this example, it is required to plan the expected cable configurations on the harnessing table *in advance*, thus confirming requirement R_3. Furthermore, it is necessary to keep a cable firm using more than two grasping points and to re-route the wiring pattern, which – when done collaboratively with a robot, for instance to place bundle retainers or junction fixtures – leads to requirements R_1 and R_2 above.

In the literature, the problem of determining the 2D or 3D configuration of articulated or flexible objects has received much attention in the past few years Nair et al. (2017); Wakamatsu et al. (2006), whereas the problem of obtaining a target configuration via manipulation has been explored in motion planning Bodenhagen et al. (2014); Schulman et al. (2016); Yamakawa et al. (2013). In the context of HRC, perception and manipulation are only part of the challenges to address. Conceptually, the outcome of such approaches is a *continuous mapping* from an initial to a target object's configuration Ahmadzadeh et al. (2015); Berenson (2013); Bodenhagen et al. (2014); Miller et al. (2011), subject to simplifying hypotheses related to object models Elbrechter et al. (2011, 2012); Frank et al. (2010); Howard and Bekey (1997); Mastrogiovanni et al. (2004). This remark leads to two further requirements. Collaborative robots must be able to:

R_4 adopt a representation to be used by action planners, and segment the whole manipulation problem in simpler actions, each action leading to a new intermediate configuration;

R_5 represent actions using a formalism allowing for plan executions that are robust with respect to unexpected events (e.g., the human operator suddenly intervenes), and modeling errors (e.g., not modelled objects to be removed from the workspace).

In this Chapter, we provide to the reader some background on the human-robot cooperative manipulation of articulated objects Yamakawa et al. (2013), which is necessary to fully grasp the importance of the performance of the integrated planning module and will be taken as an example in following chapters. In particular, we will discuss:

- The design and development of two representation and planning models for the specification of articulated object configurations and the sequencing of manipulation actions;
- The use of two state-of-the-art PDDL planners, namely Probe Lipovetzky and Geffner (2011a) and Madagascar Rintanen (2014) as well as the VAL plan validator Fox et al. (2005), to generate manipulation plans using such models;

- The design and development of a novel reactive/deliberative architecture for HRC, which we call PLANHRC, allowing human operators to intervene as they wish during the cooperation process, and implemented on top of the ROSPlan Cashmore et al. (2015a) and MoveIt! Coleman et al. (2014) frameworks;
- A discussion about how robot perception and object representation impact on action planning and execution in HRC scenarios, which is peculiar for the use case we consider.

2.3 Background

A number of studies have been conducted to investigate the role and the acceptability of automated planning techniques in HRC scenarios. Gombolay and colleagues highlight two factors as important to maximise human satisfaction in HRC Gombolay et al. (2013): (i) humans should be allowed to choose their own tasks freely, i.e., not assigned by an algorithm, subject to the fact that the cooperation is successful; (ii) the overall human-robot team's performance must be at high standards. These two factors may conflict in case of a *lazy* or *not focused* human operator's attitude. However, when required to trade-off between them, humans show a strong preference for team's performance over their own freedom. This study well fits with the requirements R_1, R_2 and R_3 outlined above, and opens up to an idea of collaborative robots as devices not only able to *aid* human workers, but also capable of *keeping them in focus* and steering the cooperation towards its objectives if deviations occur.

As a follow-up of the work discussed in Gombolay et al. (2013), a study about the actual amount of control a human operator would like to have when collaborating with a robot has been reported in Gombolay et al. (2014): human workers tend not to prefer a total control of the cooperation process, rather they opt for partial control. This is confirmed by the fact that the overall team's performance seems higher when the robot determines what actions must be carried out by the human operator. A key factor for the acceptance of collaborative robots is finding a sensible – yet efficient – trade-off between performance and human control.

In order to determine such trade-off, one possibility is to encode human operator preferences in the planning process Gombolay et al. (2015). In a first series of experiments, the use of human preferences in the planning algorithm led to an overall *decrease* in performance, correlated with human subjective perception of robots not in line with the main cooperation objectives. This suggests that a subjective assessment of the HRC process tends to attribute major inefficiencies to robots, and confirms that this is a crucial aspect for the applicability

of collaborative robots in industrial scenarios. Many techniques for HRC available in the literature target these issues only to a partial extent, and in limited contexts. In particular, it is possible to identify two relevant activity trends that the approach presented here is related to.

Approaches in the first class aim at defining cooperation models, i.e., data structures modeling the task to be jointly carried out, and algorithms operating on such data structures for the cooperation process to unfold, while keeping flexibility and human preferences into account Cirillo et al. (2010); Darvish et al. (2017); Johannsmeier and Haddadin (2017); Roncone et al. (2017); Sebastiani et al. (2017); Tsarouchi et al. (2016); Wilcox et al. (2012).

A probabilistic planner is used in Cirillo et al. (2010) to sequence available partial plans, which include indications about human preferred actions. Once determined, the sequence of partial plans cannot be changed, therefore no flexibility for the human is allowed. Such a limitation is partially overcome by the approach described in Wilcox et al. (2012), where an algorithm to adapt on-line both the action sequence and the number of action parameters is described. This is achieved using a temporal formulation making use of preferences among actions, and using optimization techniques to identify the sequence best coping with preferences and constraints. The algorithm weighs more plan optimality (in terms of a reduced number of actions, or the time to complete the plan), and uses human preferences as soft constraints. The approach by Tsarouchi and colleagues Tsarouchi et al. (2016) assumes that a human and a robot co-worker operate in different workspaces. The focus is on allocating tasks to the human or to the robot depending on their preferences, suitability and availability, and the cooperation model is represented using an AND/OR graph Sanderson et al. (1988). Although human preferences are taken into account, task allocation is *a priori* fixed and cannot be changed at run-time. A similar approach is considered in Johannsmeier and Haddadin (2017), where the assumption about the separate workspaces is relaxed. Hierarchical Task Models (HTMs) are used in Roncone et al. (2017), where the robot is given control on task allocation and execution is modelled using Partially Observable Markov Decision Processes (POMDPs). However, the focus of this approach is on robot communication actions to enhance *trust* in the human counterpart and to share a mutual understanding about the cooperation objectives. A similar approach is adopted in Sebastiani et al. (2017), where HTMs are substituted by Hierarchical Agent-based Task Planners and POMDPs are replaced by Petri Network Plans. However, differently from the approach in Roncone et al. (2017), the work by Sebastiani and colleagues support on-line changes during plan execution. Finally, the work by Darvish and colleagues represents cooperation models using AND/OR graphs, and allows for a switch among different cooperation sequences at runtime Darvish et al. (2017), therefore allowing humans to redefine the sequence of tasks

among a predefined set of choices. The human operator does not have to explicitly signal the switch to the robot, whereas the robot adapts to the new cooperation context reactively.

The second class includes techniques focused on understanding, anticipating or learning human behaviors on-line Agostini et al. (2011); Caccavale and Finzi (2017); Karpas et al. (2015); Koppula et al. (2013); Kwon and Suh (2014); Liu and Fisac (2015).

The work by Agostini and colleagues adopts classical planning approaches to determine an appropriate sequence of actions, given a model of the cooperation defined as a domain and a specific problem to solve Agostini et al. (2011). At runtime, the system ranks a predefined series of cause-effect events, e.g., observing their frequency as outcomes of human activities, and updates the cooperation model accordingly. Markov Decision Processes (MDPs) are used in Koppula et al. (2013) to model the cooperation. In particular, the human and the robot are part of a Markov decision game, and must cooperatively conclude the game, i.e., carrying out the cooperation process. Human actions are detected on-line, which influences robot's behaviour at run-time. A similar approach, which takes into account temporal constraints among tasks, is discussed in Karpas et al. (2015). Statistical techniques to recognise human actions and to adapt an already available plan accordingly are presented in Liu and Fisac (2015). Human deviations from the plan are detected. When this happens, re-planning (including task allocation) occurs to achieve the cooperation objectives. While the approaches discussed so far are quite conservative as far as robot's autonomy in the cooperation process is concerned, the work discussed in Kwon and Suh (2014) exploits Bayesian networks to predict the occurrence and the timing of human actions. Such a prediction is used to perform preparatory actions before an event even occurs. While the overall system's performance is greatly improved, humans tend to be confused by the seemingly anticipative robot's behaviour. Hierarchical Task Networks are used in Caccavale and Finzi (2017) to embed communication actions in the cooperation process. When specific deviations from the plan are detected, such communication actions enforce the adherence to the plan.

From this analysis of the literature, the PLANHRC architecture presented here differs in the following aspects:

- While the majority of cooperation models described in the literature do not allow human operators to decide what actions to carry out, or they do so only to a very limited extent, PLANHRC foresees a cooperation process informed by optimality in the planning process (therefore adhering to the first findings of Gombolay and colleagues), but allows humans to intervene freely, up to the limit situation where all the plan is executed by the human operator as he or she wishes (i.e., humans are given partial or total control);

- In PLANHRC the robot does not have to explicitly recognise human operator actions, as it is prescribed by approaches in the literature, but it focuses on their effects in the planning model, and treats any perturbation as violations with respect to the normal plan unfolding.In particular, PLANHRC takes inspiration from the findings in Gombolay et al. (2014, 2015, 2013) to devise a cooperation model and an interaction model with the human operator with the following characteristics:
  - similarly to the work in Agostini et al. (2011), the robot plans an appropriate, *optimal*, sequence of actions to determine relevant intermediate configurations for an articulated object (considered as a simplified model for a flexible object like a cable), in order to determine a final target configuration, therefore coping with requirement R_4;
  - during plan execution, the robot always monitors the outcome of each action, and compares it with the target configuration to achieve, therefore limiting the burden on the human side Gombolay et al. (2014);
  - normally, humans can supervise robot actions: when a robot action is not successful, or a plan cannot be found, humans can intervene on the robot's behalf performing their preferred action sequence Gombolay et al. (2015), therefore meeting R_1 and R_2;
  - at any time, a human operator can intervene (e.g., performing an action the robot was tasked with, or changing the articulated object's configuration), and the robot adapts to the new situation, in accordance with Gombolay et al. (2015) and requirements R_3 and R_5.

2.4 Problem Statement and Reference Scenario

The problem that PLANHRC solves is three-fold:

- Given a target articulated object's configuration, determining a plan to attain such configuration as an ordered set of actions:

$$a = \{a_1, \ldots, a_i, \ldots, a_N; \prec\}, \tag{2.1}$$

  where each action a_i involves one or more manipulation operations to be executed by a dual-arm robot;

- Designing a planning and execution architecture for the manipulation of articulated objects, which is *efficient* and *flexible* in terms of perceptual features, their representation and action planning;
- Seamlessly integrating human actions *in the loop*, allowing the robot to adapt to novel, not planned beforehand, object's configurations on-line.

In order to provide PLANHRC with such a features, we pose a number of assumptions:

A_1 articulated objects (Figure 2.2) are characterised by an inertial behaviour, i.e., rotating one link causes the movement of all upstream or downstream links, depending on the rotation joint;

A_2 the effects of gravity on the articulated object's configurations are not considered, and the object is located on a table during all operations;

A_3 we do not assume any specific grasping or manipulation strategies to obtain a target object's configuration starting from another configuration; however, we do consider when an action a_i cannot be completed because of unexpected events or modeling omissions;

A_4 perception of articulated objects is affected by noise, but the *symbol grounding problem*, i.e., the association between perceptual features and the corresponding symbols in the robot's knowledge representation system Harnad (1990), is assumed to be given.

As anticipated above, we need to represent articulated object's configurations. We define an articulated object as a 2-ple

$$\alpha = (\mathscr{L}, \mathscr{J}) \tag{2.2}$$

where $\mathscr{L}$ is the ordered set of its $|L|$ links, i.e.,

$$\mathscr{L} = \{l_1, \ldots, l_j, \ldots, l_{|L|}; \prec\} \tag{2.3}$$

and $\mathscr{J}$ is the ordered set of its $|J|$ joints, i.e.,

$$\mathscr{J} = \{j_1, \ldots, j_j, \ldots, j_{|J|}; \prec\} \tag{2.4}$$

Each link $l_j \in \mathscr{L}$ is characterized by two parameters, namely a length λ_l and an orientation θ_l. We allow only for a limited number of possible orientations. This induces an ordered set O of $|O|$ allowed orientation values, i.e.,

$$O = \{o_1, \ldots, o_{|O|}; \prec\} \tag{2.5}$$

such that an orientation θ_l can assume values in O. Given a link l_j, we define two sets, namely $up(l_j)$ and $down(l_j)$, such that the former is made of upstream links, i.e., from l_1 to l_{j-1}, whereas the latter includes downstream links from l_{j+1} to $l_{|J|}$.

Orientations can be expressed with respect to an *absolute*, possibly robot-centred reference frame, or – less intuitively – *relative* to each other, for instance θ_{l_i} can represent the rotation with respect to $\theta_{l_{i-1}}$. At a first glance, the absolute representation seems preferable because it leads to the direct perception of links and their orientations with respect to a robot-centred reference frame, whereas the set of absolute orientations constitute the overall object's configuration. When a sequence of manipulation actions are planned, changing one absolute orientation requires – in principle – the *propagation* of such change upstream or downstream the object via joint connections, which (hypothesis H_1) is expected to increase the computational burden on the reasoner and (H_2) may lead to suboptimal or redundant action sequences, because the propagation may jeopardise the effects of previous actions in the plan, or to sequences which cannot be fully understood by the human operator. On the contrary, the less intuitive *relative* approach assumes the direct perception of the relative orientations between pairwise links, and thus the overall object's configuration is made up of *incremental* rotations. In this case, (H_3) computation is expected to be less demanding, since there is no need to propagate one change in orientation to upstream or downstream links, and therefore (H_4) actions on different links *tend* to be planned sequentially. This has obvious advantages since it leads to shorter plans (on average), which could be further shortened by combining together action sub-sequences (e.g., two subsequent reorientations of 45 *deg* consolidated as one 90 *deg* single action), and to easy-to-understand plans.

If an articulated object is represented using absolute orientations (Figure 2.2 on the top), then its configuration is a $|L|$-ple:

$$\mathscr{C}_{\alpha,absolute} = \left(\theta_1^a, \ldots, \theta_l^a, \ldots, \theta_{|L|}^a\right), \tag{2.6}$$

where it is intended that the generic element θ_l^a is expressed with respect to an absolute reference frame. Otherwise, if relative angles are used (Figure 2.2 on the bottom), then the configuration must be *augmented* with an initial *virtual* link l_0 in order to define a reference frame, and therefore:

$$\mathscr{C}_{\alpha,relative} = \left(\theta_{0,virtual}^r, \theta_1^r, \ldots, \theta_l^r, \ldots, \theta_{|L|}^r\right). \tag{2.7}$$

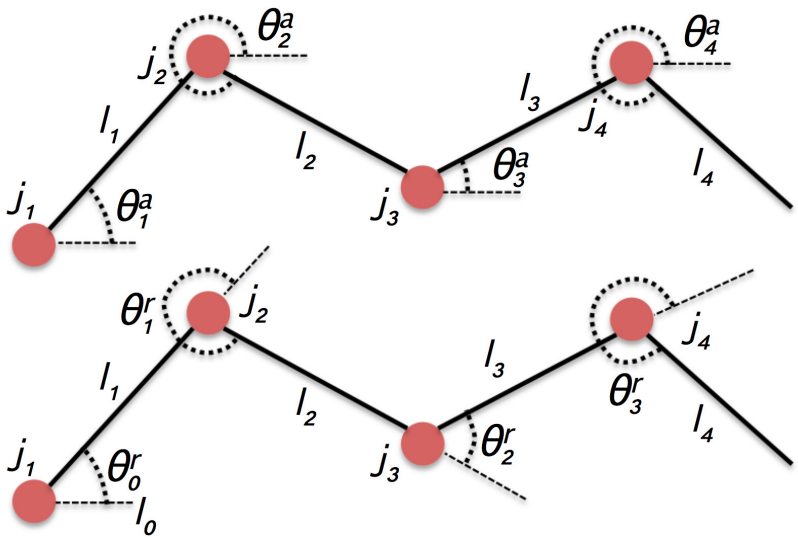

Figure 2.2 Two possible representations: absolute (top) and relative (bottom).

In principle, while the relative representation could model an object's configuration with one joint less compared to the absolute representation, the resulting configuration would not be unique (indeed there were infinitely many), since the object would maintain pairwise relative orientations between its links even when rotated *as a whole*. Therefore, an initial virtual reference link is added to the chain.

In order to comply with assumption A_2, we set up an experimental scenario where a Baxter dual-arm manipulator operates on articulated objects located on a table in front of it (Figure 2.3). Rotation operations occur only around axes centred on the object's joints and perpendicular to the table where the object is located. We have crafted a wooden articulated object made up of $|L| = 5$ 15.5 *cm* long links, connected by $|J| = 4$ joints. Links are 3 *cm* thick. The object can be easily manipulated by the Baxter's standard grippers, which complies with assumption A_3. To this aim, we adopt the MoveIt! framework. The robot is equipped with an RGB-D device located on top of its *head* pointing downward to the table. Only RGB information is used. QR tags are fixed to each object's link, which is aimed at meeting assumption A_4. Each QR code provides a 6D link pose, which directly maps to an absolute link orientation θ_l^a. Finally, if relative orientations are employed, we compute them by performing an algebraic sum between the two absolute poses of two consequent links:

$$\theta_1^r = |\theta_2^a - \theta_1^a| \tag{2.8}$$

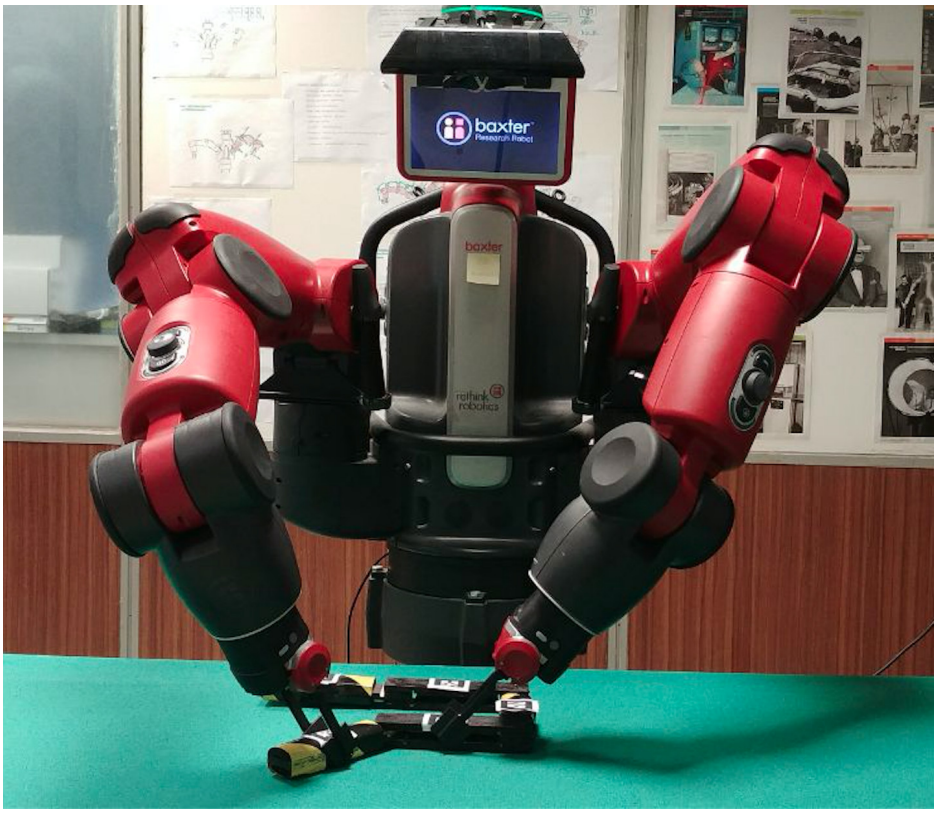

Figure 2.3 The experimental scenario: a Baxter dual-arm manipulator operating on an articulated object.

A human can supervise robot operations and intervene when necessary from the other side of the table[1].

2.5 PLANHRC's Architecture

2.5.1 Information Flow

PLANHRC is organised as a number of parallel loops orchestrating the behaviour of different modules (Figure 2.4). Assuming that an articulated object α is located on the table in front of the robot, we want to modify its *current* configuration c^c_α to obtain a goal configuration c^g_α, which can be expressed using (2.6) or (2.7).

The goal configuration c^g_α is encoded as assertional knowledge in an OWL-based *Ontology* module Krotzsch et al. (2013). When this happens, the *Perception* module is activated,

[1] A video is available at `https://www.youtube.com/watch?v=dMdzCB5FBMI`.

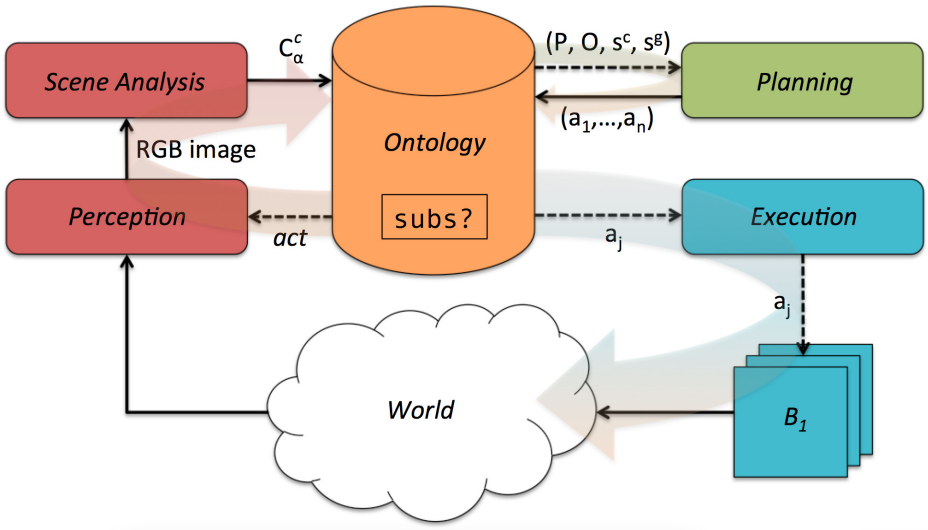

Figure 2.4 The information flow in PLANHRC.

and the Baxter's camera acquires an image of the workspace[2], which is fed to the *Scene Analysis* module. A *perceived* configuration c^p_α (i.e., the current configuration c^c_α) is extracted from the image, and a representation of it stored in the *Ontology* module. Both c^c_α and c^g_α are represented using conjunctions of class instances, which model such predicates as Connected, to indicate whether two links are connected by a Joint, or HasOrientation, to define angle orientations. If c^c_α and c^g_α *are different* then a planning process occurs. In order to determine such a difference, we assume the availability of a logic operator $\mathscr{D}$ that, given an element in the ontology, returns its description in OWL formalism. If the description of c^c_α is not *subsumed* by the description of c^g_α, i.e., it does not hold that $\mathscr{D}(c^c_\alpha) \sqsubseteq \mathscr{D}(c^g_\alpha)$, the *Planner* module is activated, which requires the definition of relevant predicates $\mathscr{P}_1,\ldots,\mathscr{P}_{|P|}$, and possible action types $\mathscr{A}_1,\ldots,\mathscr{A}_j,\ldots,\mathscr{A}_{|A|}$ in the form:

$$\mathscr{A}_j = \left(pre(\mathscr{A}_j), eff^-(\mathscr{A}_j), eff^+(\mathscr{A}_j)\right), \tag{2.9}$$

where $pre(\mathscr{A}_j)$ is the set of preconditions (in the form of predicates) for the action to be executable, $eff^-(\mathscr{A}_j)$ is the set of negative effects, i.e., predicates becoming false after action execution and $eff^+(\mathscr{A}_j)$ is the set of positive effects, i.e., predicates becoming true after execution. For certain domains, it is useful to extend (2.9) to allow for additional positive or negative effects, i.e., predicates becoming true or false in case certain additional

[2]The *Perception* module acquires images continuously, but for the sake of simplicity we treat each acquisition as if it were synchronous with action execution.

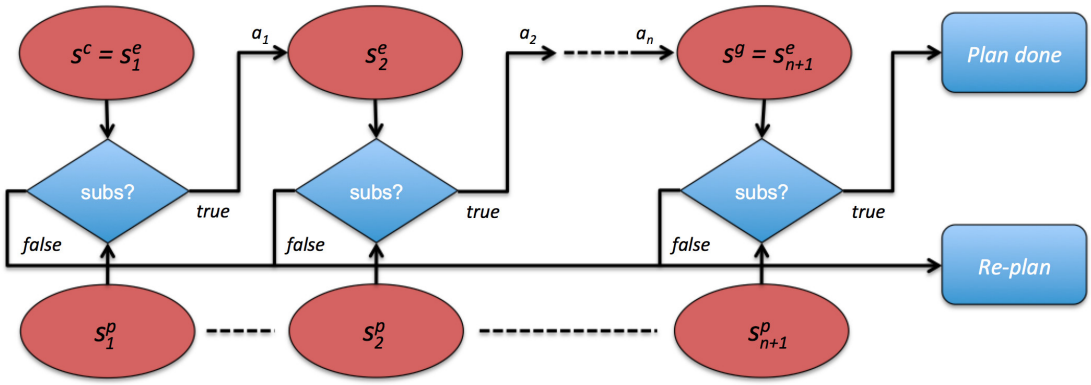

Figure 2.5 The planning and execution pipeline.

conditions hold. A conditional action can be modelled as:

$$\mathscr{A}_j = \left(pre(\mathscr{A}_j), eff^-(\mathscr{A}_j), eff^+(\mathscr{A}_j), pre_a(\mathscr{A}_j), eff_a^-(\mathscr{A}_j), eff_a^+(\mathscr{A}_j)\right), \tag{2.10}$$

where $pre(\mathscr{A}_j)$, $eff^-(\mathscr{A}_j)$ and $eff^+(\mathscr{A}_j)$ are defined as before, $pre_a(\mathscr{A}_j)$ is the set of additional preconditions, whereas $eff_a^-(\mathscr{A}_j)$ and $eff_a^+(\mathscr{A}_j)$ are the sets of additional effects subject to the validity of predicates in $pre_a(\mathscr{A}_j)$. Furthermore, the *Planner* requires a suitable description of the current state s^c (including a description of c^c_α) and the goal state s^g (including c^g_α), described using an appropriate set of ground predicates $p_1, \ldots, p_{|p|}$. This information, encoded partly in the terminological section and partly in the assertional section of the *Ontology* module, is translated in a format the *Planner* module can use, namely the Planning Domain Definition Language (PDDL) McDermott (1998).

A plan, as formally described in (2.1), is an ordered sequence of N actions whose execution changes the current state from s^c to s^g through a set of intermediate states. In a plan, each action corresponds to one or more scripted robot behaviors. For example, rotating a link l_{j+1} requires the robot to (i) keep the upstream link l_j steady with its left gripper, and (ii) rotate l_{j+1} of a certain amount with the right gripper. Such sequence shall not be encoded in the planning process, thereby reducing planning cost, but demanded to an action execution module. If a plan is found, each action is encoded in the ontology, along with all the *expected* intermediate states $s^c = s^e_1, s^e_2, \ldots, s^g = s^e_{N+1}$, which result from actions. The *Execution* module executes action by action activating the proper modules in the architecture, e.g., such *behaviors* as motion planning, motion execution, obstacle avoidance or grasping.

Each action a_j in a plan is assumed to transform a state s^e_j into a state s^e_{j+1}, such that:

$$s^e_{j+1} = \left(s^e_j \setminus eff^-(a_j)\right) \cup eff^+(a_j). \tag{2.11}$$

If a_j has additional conditions, then (2.11) is modified as:

$$s^e_{j+1} = \left(s^e_j \setminus \left(eff^-(a_j) \cup C^-(pre_a(a_j))\right)\right) \cup \left(eff^+(a_j) \cup C^+(pre_a(a_j))\right), \tag{2.12}$$

where conditions C^- and C^+ return the sets $eff^-_a(a_j)$ and $eff^+_a(a_j)$, respectively, if the conditions in $pre_a(a_j)$ hold, and $\emptyset$ otherwise. Before the action is executed, the *Ontology* module activates *Perception* to acquire a new image. Again, this induces a new perceived, current configuration c^c_α. Every time this happens, two situations can happen: if c^c_α corresponds to a current perceived state s^c whose description is subsumed by the description of a state s^e_{j-1} possibly generated applying an action a_{j-1} or as a consequence of human intervention, i.e., $\mathscr{D}(s^c) \sqsubseteq \mathscr{D}(s^e_{j-1})$, then the execution continues with action a_j until a state is reached which is subsumed by $\mathscr{D}(s^g)$; otherwise, a new planning process occurs, considering the current state s^c as a new initial state and keeping the previous goal state s^g.

A few remarks can be made. When an action a_j is executed, the expected intermediate state s^e_j is treated as a set of *normative* ground predicates, i.e., it defines the normal, expected state for a_j to be feasible. Whether s^e_j is obtained as a result of a previous action, or with the help of the human operator is not relevant for a_j. On the contrary, deviations from it are treated as violations and therefore the system tries to re-plan in order to reach a state compatible with s^g starting from the current state. As discussed above, violations can be of two kinds:

- Human interventions (i.e., object manipulations on robot's behalf) may lead to a current state s^c not compatible with the expected intermediate state s^e_j, and therefore the robot should adapt by re-planning;
- A robot may not be able to complete action a_j, e.g., due to a cluttered workspace Srivastava et al. (2014) or the obstructing presence of the human operator Haddadin and Croft (2016).

In the second case, if such an event were detected, the robot would re-plan starting from the current state, and possibly ask for the human operator's help to achieve a workable object's configuration. As a consequence, PLANHRC implements a policy according to which the overall system's performance is ensured by the use of state-of-the-art planning techniques, but it allows at any time the human operator to intervene and forces the robot to adapt its plan accordingly.

Figure 2.5 shows a graphical model of the information flow from the perspective of the planning process.

2.5.2 Reasoning in the Ontology and the Cooperation Model

In PLANHRC, the *Ontology* module is used both *off-line* and *on-line* for different purposes[3]. The off-line use is related to modeling the domain of articulated objects manipulation, in terms of *types*, *predicates*, *operators*, *states*, *problems* and *plans*. The on-line use serves two purposes: on the one hand, to represent relevant object's configurations, such as the current c^c_α and the goal c^g_α configurations, as well as specific actions to perform using classes and relationships defined in the ontology; on the other hand, to apply such reasoning techniques as *instance checking* to the representation, e.g., to determine whether an action a_j assumes an expected planning state s^e_j which is compatible with the perceived current state s^c, as described in Figure 2.5.

As anticipated in Section 2.4, and in accordance with the findings in Gombolay et al. (2014, 2015, 2013), the human-robot cooperation model implemented in PLANHRC foresees that:

1. The robot determines a plan maximizing some performance indicator in terms of number of actions and/or time-to-completion;
2. The robot executes and monitors each action in the plan;
3. During normal work flow, the human operator supervises robot actions;
4. The human operator can intervene to cope with robot's failures in action planning or execution, or to perform tasks asynchronously and in parallel to robot activities.

The model unfolding is based on monitoring the state transitions in (2.11) and (2.12) and their failures. Independently of the presence of conditional effects in an action a_j, two cases are possible after the action is submitted to the *Execution* module: it cannot be executed (or it is executed only in part) or it is carried out successfully.

The first case originates from motion planning or execution issues, e.g., because of a cluttered workspace Srivastava et al. (2014) or to prevent any harm to the human operator Darvish et al. (2017); Denei et al. (2015); Haddadin and Croft (2016). If motion issues occur, PLANHRC does not generate a state compatible with s^e_{j+1}. However, this does not necessarily mean that the current state s^c is still compatible with the previous assessed state s^e_j, i.e., $\mathscr{D}(s^c) \sqsubseteq \mathscr{D}(s^e_j)$ may not hold, because the robot may have completed only part

[3]A more detailed description of the ontology is present in Appendix 1, whereas the full OWL ontology is available at `https://github.com/EMAROLab/OWL-ROSPlan/tree/master/rosplan_knowledge_base/`.

of the action. In this case, a new current state s^c is acquired. If there is an intermediate expected state s^e_i comparable with s^c, then execution resumes from action a_{i+1}; otherwise, it is necessary to invoke again the *Planner* module using s^c and s^g, and obtain a new plan.

In the second case, action a_j is considered to be successful from the point of view of motion execution. Still, the outcome may or may not be compatible with the expected state s^e_{j+1}, e.g., due to not modelled effects. This state is observable as the current state s^c. However, although $\mathscr{D}(s^c) \sqsubseteq \mathscr{D}(s^e_{j+1})$ does not hold, it may happen that s^c could be appropriate for the next action a_{j+1} to occur. In particular, for a_{j+1} to be executable, it must hold that $\mathscr{D}(s^c) \sqsubseteq \mathscr{D}(pre(a_{j+1}))$. We treat the set of predicates in $pre(a_{j+1})$ as normative conditions for a_{j+1}, regardless whether the expected state s^e_{j+1} is generated as the outcome of the previous action a_j. If $\mathscr{D}(s^c) \sqsubseteq \mathscr{D}(pre(a_{j+1}))$ does not hold, we must check whether there is any intermediate expected state s^e_i comparable with s^c: if it is the case, execution resumes from action a_{i+1}; otherwise, re-planning is necessary.

In summary, human intervention is strictly necessary when a plan cannot be found. However, any human action is implicitly considered every time the current state does not comply with normative predicates. In other words, human intervention is always expected, and in real-world conditions, it might be extremely frequent, so much so to trigger a re-planning every single iteration of the architecture loop. In such conditions, even small delays introduced by the planning module can deteriorate performance and compound to significant wait times for the human operator.

2.5.3 Planning Models

As anticipated in Section 2.4, orientations can be expressed using an absolute or relative reference frame. These two possibilities lead to two planning models, which are characterized by different properties as far as (i) obtained plan, (ii) computational load of the planning process, and (iii) ease of execution for the robot, are concerned.

For the sake of description, we present the relative formulation first, and then the absolute one. The relative formulation employs the `:STRIPS` subset of PDDL, extended with `:equalities` and `:negative-preconditions`, whereas the absolute version requires also the use of `:conditional-effects`. Notably, the problem we are interested in induces a sort of granularity discretization of angular orientations, hence there is no practical necessity for continuous or hybrid planning models Fox and Long (2006). Therefore, PDDL constitutes an appropriate level of abstraction[4].

[4]Examples of planning domains and problems can be found at **?**.

```
(:action RotateClockwise
  :parameters (?l1 ?l2 - Link
    ?j1 - Joint ?o1 ?o2 - Orientation)
  :precondition (and
    (Connected ?j1 ?l1)
    (Connected ?j1 ?l2)
    (not (= ?l1 ?l2))
    (HasOrientation ?o1 ?j1)
    (OrientationOrd ?o1 ?o2))
  :effect (and
    (not (HasOrientation ?o1 ?j1))
    (HasOrientation ?o2 ?j1))
)
```

Figure 2.6 The *relative* version of RotateClockwise in PDDL.

As discussed when introducing assumption A_1, our model assumes inertial behaviour, i.e., rotating one link affects the orientation of upstream or downstream links as well. Given a link l_j to rotate (clockwise or anticlockwise), two rotation actions are possible:

1. If link l_{j-1} is kept still and l_j is rotated (clockwise or anticlockwise), then all links in $down(l_j)$ rotate (clockwise or anticlockwise) and are displaced as well;

2. If link l_{j+1} is kept still, all links in $up(l_j)$ are rotated (clockwise or anticlockwise) and displaced.

Each rotation action (either clockwise or anticlockwise) changing an angle θ_j^r referring to a relative orientation does not affect any other orientations of links in $up(l_j)$ or $down(l_j)$, since all of them are relative to each other, and therefore the planning process is computationally less demanding. However, since actions are expected to be based on link orientations grounded with respect to a robot-centred reference frame, i.e., absolute in terms of pairwise link orientations, a conversion must be performed, which may be greatly affected by perceptual noise, therefore leading to inaccurate or even inconsistent representations. In the absolute formulation, θ_j^a is considered absolute, and therefore it can be associated directly with robot actions. Unfortunately, this means that each action changing θ_j^a does affect numerically all other orientations of links in $up(l_j)$ or $down(l_j)$ in the representation, which must be kept track of using conditional effects in the planning domain.

2.5.4 Relative formulation

As described in Section 2.4, an articulated object α is represented using two ordered sets of links and joints. We use a `Connected` predicate modeled as described in

$$\begin{aligned} \mathsf{Connected} \sqsubseteq\ & \mathsf{Predicate} \sqcap \\ & \exists \mathsf{arg1}.\mathsf{Joint} \sqcap =_1 \mathsf{arg1} \sqcap \\ & \exists \mathsf{arg2}.\mathsf{Link} \sqcap =_1 \mathsf{arg2}. \end{aligned} \tag{2.13}$$

to describe the sequence of links in terms of binary relationships each one involving a link l_j and a joint j_{j+1}, which induces a pairwise connection between two links, namely l_j and l_{j+1}, since they share the same joint j_{j+1}. The orientation of a link l_j is associated with the corresponding joint j_j and corresponds to an angle θ_j^r, which ranges between 0 and 359 deg, using the predicate `HasOrientation` as specified in

$$\begin{aligned} \mathsf{HasOrientation} \sqsubseteq\ & \mathsf{Predicate} \sqcap \\ & \exists \mathsf{arg1}.\mathsf{Joint} \sqcap =_1 \mathsf{arg1} \sqcap \\ & \exists \mathsf{arg2}.\mathsf{Orientation} \sqcap =_1 \mathsf{arg2}, \end{aligned} \tag{2.14}$$

This formulation assumes that link orientations are expressed incrementally relative to each other, and it implies that the robot's perception system is expected to provide the *Ontology* module with the set of relative link orientations as primitive information. If absolute link orientations are not available, the object's configuration $\mathscr{C}_{\alpha,absolute}$ can be computed applying forward kinematics formulas using relative orientations and link lengths. If noise affects the perception of link orientations, as it typically does, the reconstruction of the object's configuration may differ from the real one, and this worsens with link lengths. However, this model significantly simplifies the planning model's complexity: from a planner's perspective, the modification of any link orientations does not impact on other relative joint angles, and therefore rotation actions can be sequenced *in any order* the planner deems fit.

Angles are specified using *constants*, and are ordered using the predicate `OrientationOrd` as described by

$$\begin{aligned}\mathsf{OrientationOrd} \sqsubseteq\ & \mathsf{Predicate} \sqcap \\ & \exists \mathsf{arg1}.\mathsf{Orientation} \sqcap =_1 \mathsf{arg1} \sqcap \\ & \exists \mathsf{arg2}.\mathsf{Orientation} \sqcap =_1 \mathsf{arg2}.\end{aligned} \tag{2.15}$$

The difference between constant values is the *granularity* of the resolution associated with modelled orientations. For example, if `30` and `45` are used as constants representing, respectively, a 30 and a 45 *deg* angle, then a predicate `(OrientationOrd 30 45)` is used to encode the fact that `30` precedes `45` in the orientation granularity, and corresponds to the description in

$$\begin{aligned}&\mathsf{arg1}(\mathsf{ord\_30\_45}, 30), \\ &\mathsf{arg2}(\mathsf{ord\_30\_45}, 45),\end{aligned} \tag{2.16}$$

Independently of what part of the articulated object is rotated, the domain model includes two actions, namely `RotateClockwise` (Figure 2.6) and `RotateAntiClockwise`. In the definition of `RotateClockwise`, `?l1` and `?l2` represent any two links l_j and l_{j+1}, `?j1` is the joint j_{j+1} connecting them, whereas `?o1` and `?o2` are the current and the obtained link orientations, respectively. If `?j1` connected two *different* links `?l1` and `?l2`, the angle `?o1` of `?l1` associated with `?j1` would be increased of a certain step (depending on the next orientation value) therefore leading to `?o2`. A similar description can be provided for `RotateAntiClockwise`.

A problem is defined by specifying the initial and final states. The former includes the *topology* of the articulated object in terms of `Connected` predicates, and its initial configuration using `HasOrientation` predicates; the latter describes its goal configuration using relevant `HasOrientation` predicates.

2.5.5 Absolute formulation

The absolute formulation differs from the relative one in that link orientations are expressed with respect to a unique, typically robot-centred, reference frame. If a rotation action modifies a given link orientation θ_j^a, all orientations of links in $up(l_j)$ or $down(l_j)$ must be consistently updated as well, i.e., it is necessary to propagate such change upstream or downstream. Such a representation increases the complexity of the planning task but it is more robust to errors: perceiving independent link orientations induces an *upper bound* on the error associated with

```
(:action RotateClockwise
  :parameters (?l1 ?l2 - Link
    ?j1 - Joint ?o1 ?o2 - Orientation)
  :precondition (and
    (Connected ?j1 ?l1)
    (Connected ?j1 ?l2)
    (not (= ?l1 ?l2))
    (HasOrientation ?o1 ?j1)
    (OrientationOrd ?o1 ?o2))
  :effect
    (and
      (not (HasOrientation ?o1 ?j1))
      (OrientationOrd ?o2 ?j1)
      (forall (?j2 - Joint ?o3 ?o4 - Orientation)
        (when (and
           (Affected ?j2 ?l1 ?j1)
           (not (= ?j2 ?j1))
           (HasOrientation ?o3 ?j2)
           (OrientationOrd ?o3 ?o4))
        (and
           (not (HasOrientation ?o3 ?j2))
           (HasOrientation ?o4 ?j2)))
    )
)
```

Figure 2.7 The *conditional* version of `RotateClockwise` in PDDL.

their inner angle. The `Connected`, `HasOrientation` and `OrientationOrd` predicates are the same as in the relative formulation, subject to the different semantics associated with link orientations. However, with respect to the relative formulation, the effects of the actions differ. In particular, the model assumes that we can represent which joints are affected when a link is rotated around one of the corresponding joints. This is done using the `Affected` predicate, i.e., a ternary predicate `(Affected ?j2 ?l1 ?j1)`, where `?l1` is the rotated link, `?j1` is the joint around which `?l1` rotates, and `?j2` is a joint affected by this rotation. Therefore, if `?j2` were affected, the angle of the corresponding link would be modified as well in the conditional statement and, as such, it would affect other joints via the corresponding links. For each couple `?l1`, `?j1`, the list of joints affected by the corresponding movement should be provided under the form of multiple `Affected` predicates. With reference to the action

described in Figure 2.7, as in the previous case, the joint `?j1`, located between `?l1` and `?l2`, is increased by a quantity defined by a specific granularity, according to the `OrientationOrd` predicate. If rotating `?l2` around `?j1` affects `?j2`, the latter is updated, as well as all other joints upstream or downstream. This is encoded by the `forall` part of the PDDL encoding. Following the semantics of the language, the `forall` statement requires the planner to update the state of all joints `?j2` that are affected by the performed action – checked conditions are specified via the `when` statement. The `HasOrientation` predicate of identified affected joints is then updated accordingly. A similar definition for `RotateAntiClockwise` can be easily given.

In terms of problem definition, beside `Connected` and `HasOrientation` predicates, it is necessary to include the list of appropriately defined `Affected` predicates.

It is noteworthy that the two action definitions, namely `RotateClockwise` and `RotateAntiClockwise`, are functionally equivalent. Furthermore, any problem we target here could be solved – in principle – with just one action, as long as discretized angles were ring-connected. We decided to introduce two different actions for two reasons: on the one hand, it is rare that joints can rotate freely for 360 *deg* or more; on the other hand, this model leads to shorter plans (on average) in terms of number of actions and cleaner, more natural executions, at the expense of a slightly longer planning time.

2.6 Performance of the PLANHRC architecture

2.6.1 System Design

PLANHRC has been implemented integrating existing modules and novel *ad hoc* solutions. The experiments reported here have been carried out using a dual-arm Baxter manipulator. The *Perception* and *Scene Analysis* modules are custom nodes developed using the Robot Operating System (ROS) framework. They integrate the Alvar tracker library to read QR codes [5]. Different solutions are equally legitimate, and the use of QR codes is not a fundamental feature of the proposed framework. Images are collected using the standard RGB camera of a Kinect device, which is mounted on the Baxter's *head* and points downward to capture what happens on a table in front of the robot. The *Ontology* and *Planning* modules have been implemented on top of ROSPlan Cashmore et al. (2015a). A custom ontology describing the domain of articulated object manipulation has been developed and validated.

[5] `http://wiki.ros.org/ar_track_alvar`

Ontology management is done using the ARMOR framework [6], which has been integrated with ROSPlan. Two existing planners have been interfaced with the system and evaluated, namely Probe Lipovetzky and Geffner (2011a) and Madagascar Rintanen (2014). In principle, any existing PDDL-based planner with the features discussed above could be used. The two planners have been selected on the basis of their performance in the agile track of the 2014 International Planning Competition, as well as following a computational assessment of their performance Capitanelli et al. (2017). The *Execution* module and the various activated behaviors have been implemented using the well-known MoveIt! framework.

On-line, the architecture runs on a 8× Intel Core i7-4790 CPU 3.60 GHz processors workstation, with 8 GB of RAM, running a Linux Ubuntu 14.04.5 LTS operating system. Off-line performance tests about the planning process have been carried out on a workstation equipped with 2.5 GHz Intel Core 2 Quad processor, 4 GB of RAM, running a Linux 2.6.32 kernel operating system.

Problem formulations, as well as all generated instances, including domain, problems and plans, are freely available [7].

2.6.2 Planning Performance

Tests with synthetic problem instances have been performed to stress the two planning formulations. For the tests, we varied the number of links $|L|$ from 4 to 20 and the number of allowed orientations $|O|$ a link can take from 4 (i.e., with a resolution of 90 *deg*) to 12 (i.e., with a resolution of 30 *deg*). As outlined above, such a resolution has a different meaning depending on whether we employ the absolute or relative formulations.

Figures 2.8 to 2.11 represent means and variances, in *seconds*, for different problem instances, for all the combinations of formulation and planner. Problem instances are labelled as $x-y$, where $x \leq |L|$ defines the number of links and $y \leq |O|$ specifies the orientation resolution. For each instance, planners have been executed 10 times to take into account the randomness associated with the employed heuristics. A 300 *sec* upper bound to the solution time has been set. If a planner is unable to find a solution before such time limit is reached, it is stopped. Figures only contain data related to problems solved within the time limit.

As it can be seen in Figure 2.8, when we use the absolute formulation and Probe, 73.5% of the instances are solved, i.e., 125 out of 170. It is possible to observe that problem instances with up to $x \leq 10$ and $y \leq 4$ are solved in roughly less than 1 *sec*, with a relatively small variance. When the number of links increase, planning time significantly increases

[6] `https://github.com/EMAROLab/ARMOR`
[7] `https://github.com/EMAROLab/paco_actions`

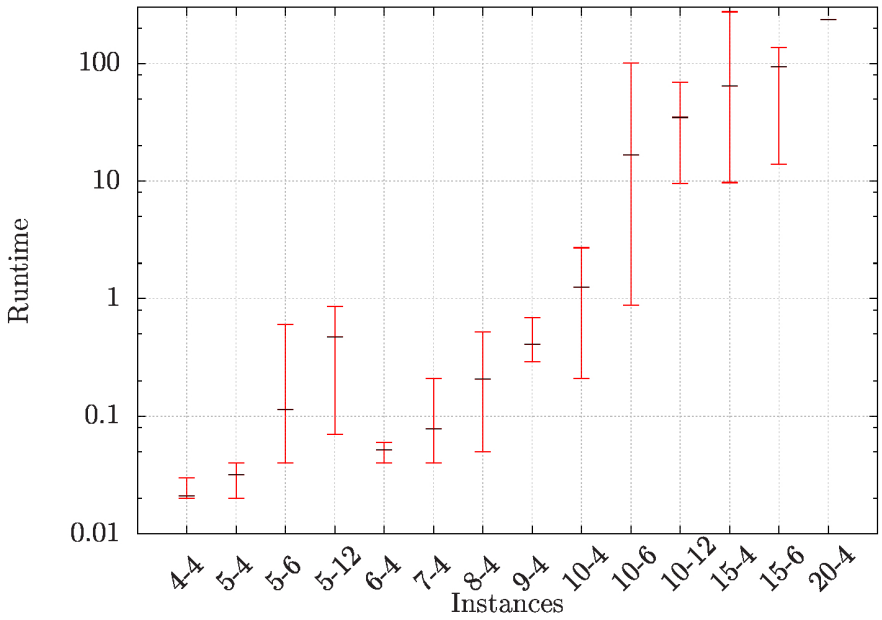

Figure 2.8 Means and variances of solution times for different problem instances using the absolute formulation and Probe: on the x-axis, the first value indicates the number of links, the second the number of allowed orientations. Runtime is reported in *seconds*.

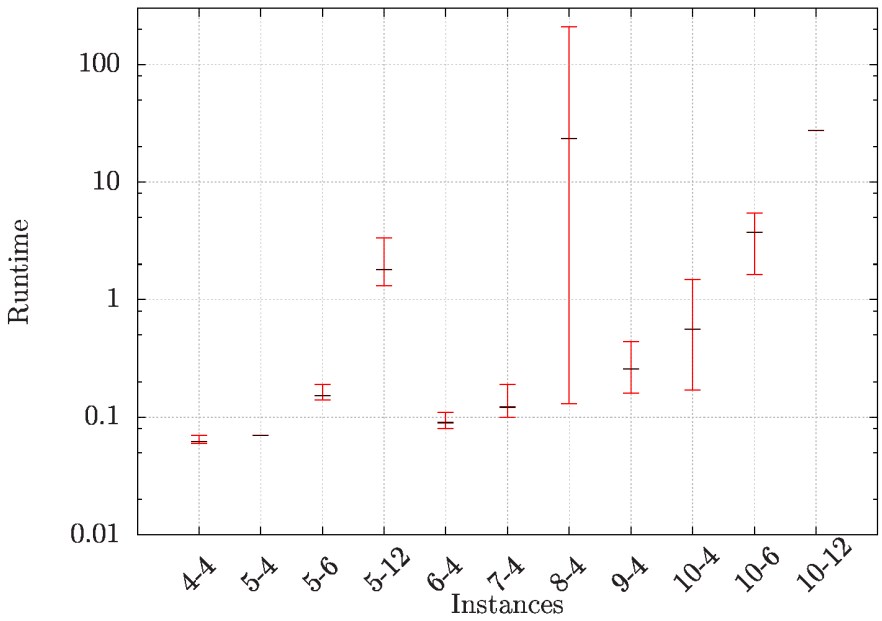

Figure 2.9 Means and variances of solution times for different problem instances using the absolute formulation and Madagascar: on the x-axis, the first value indicates the number of links, the second the number of allowed orientations. Runtime is reported in *seconds*.

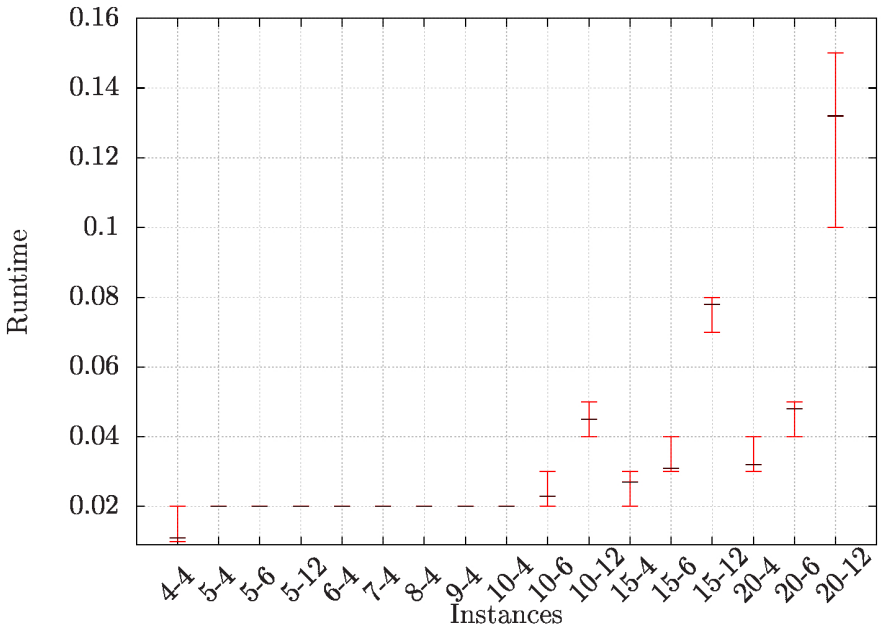

Figure 2.10 Means and variances of solution times for different problem instances using the relative formulation and Probe: on the x-axis, the first value indicates the number of links, the second the number of allowed orientations. Runtime is reported in *seconds*.

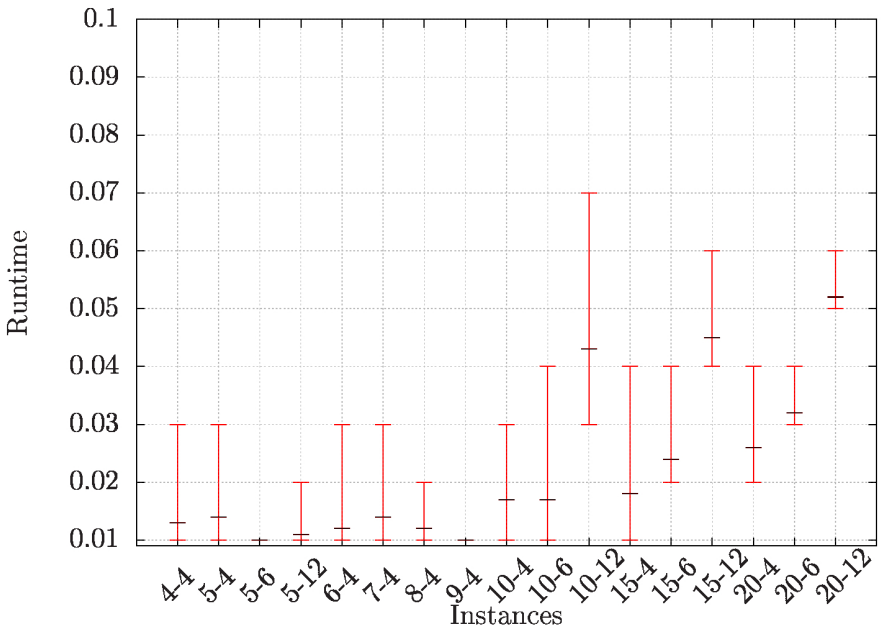

Figure 2.11 Means and variances of solution times for different problem instances using the relative formulation and Madagascar: on the x-axis, the first value indicates the number of links, the second the number of allowed orientations. Runtime is reported in *seconds*.

as well, and thus the variance. In the same situation, as depicted in Figure 2.9, Madagascar shows a more unpredictable behaviour: for small problem instances, it can quickly find a solution, and with a small temporal variance; however, the employed heuristics may cause large variances in specific cases, e.g., the instance labelled $8-4$. It is worthy to note that larger instances are rarely solved and, in general, the number of solved instances is lower when compared to Probe, i.e., only 53.5% (91 out of 170). As it will be also showed in the next Section, these results seem to confirm hypothesis H_1, i.e., the more intuitive absolute formulation leads to more complex reasoning processes. This is due to the fact that planners need to propagate the effects of each action to upstream or downstream links, which can be done only by employing a complex formulation involving conditional effects.

If we consider the relative formulation, then both Probe (Figure 2.10) and Madagascar (Figure 2.11) are very efficient, with Madagascar outperforming Probe to a small extent. Both planners are capable of solving all the instances (170 out of 170) in less that 0.2 *sec*, and exhibit a very good scalability, as well as a very limited variance. These results support hypothesis H_3, i.e., the reduced planning effort is reflected by the simpler formulation.

2.6.3 Examples

In this Section, we provide examples of plans generated by Probe and Madagascar using the two formulations introduced above. Furthermore, we show and discuss what happens in a number of human-robot cooperation use cases.

In order to discuss how the different planners deal with the absolute and the relative formulation, we focus the discussion on a specific instance with 3 links and 3 joints. Figure 2.12 shows two possible solutions, obtained respectively using Probe (first two rows) and Madagascar (last two rows), when the absolute formulation is adopted. In each solution, the top-leftmost configuration is the initial one, whereas the bottom-rightmost configuration is final one. It can be observed that both plans are characterized by a number of seemingly unnecessary actions, since the planners must continuously maintain the representation consistency. The plan obtained using Madagascar (on the bottom) also loops over two configurations, which is probably due to the employed heuristics. This example seems to confirm H_2, i.e., the absolute approach leads to suboptimal plans, or plans which may not easily understood by human co-workers.

Figure 2.13 shows how Probe (top) and Madagascar (bottom) solve the same problem when a relative formulation is adopted. Both planners generate solutions that are shorter than those obtained using the absolute formulation, and no seemingly unnecessary actions are

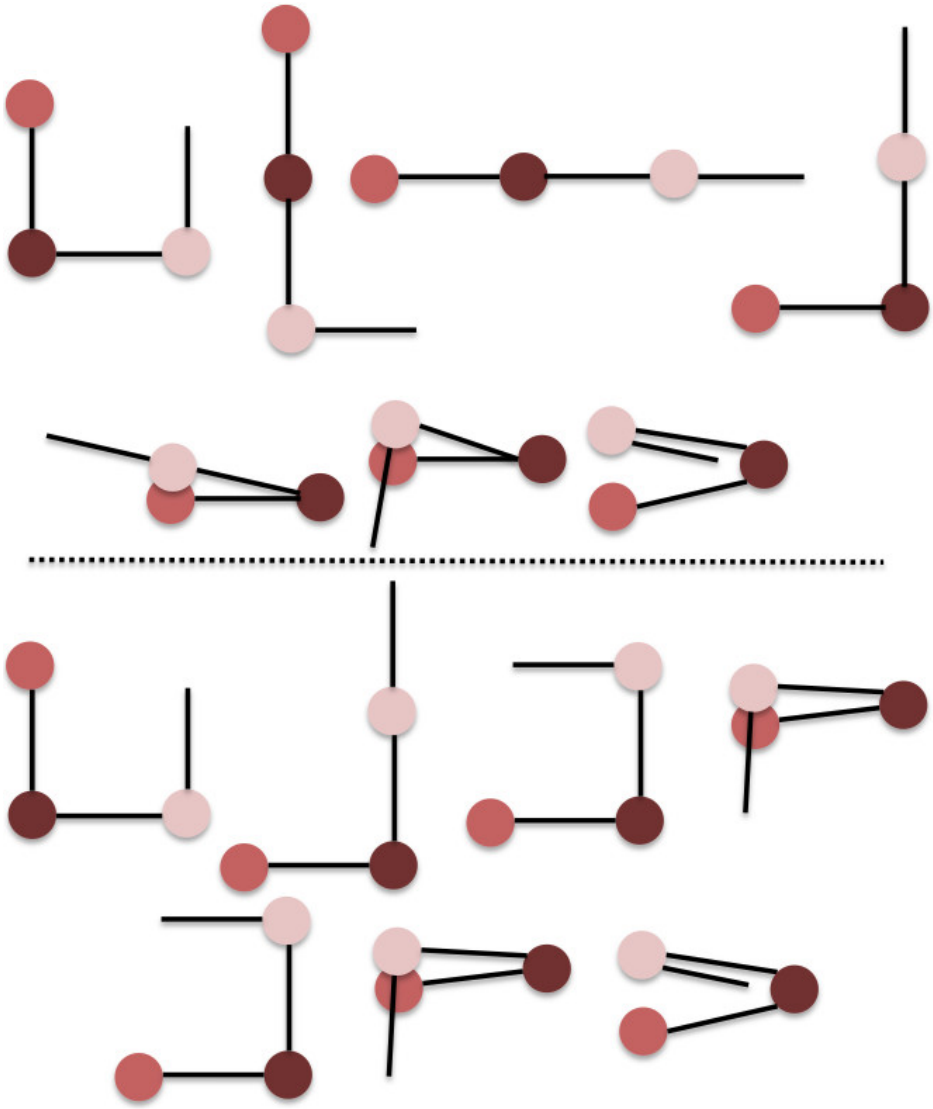

Figure 2.12 A sequence of configurations for a $3-3$ problem, using the absolute formulation with Probe (first two rows, from left to right and top to bottom) and Madagascar (second two rows, from left to right and top to bottom).

planned. In the plan generated by Madagascar, it is possible to observe that actions involving the same link tend to be performed sequentially, i.e., H_4 seems to be verified. This holds for other solutions as well.

As anticipated above, PLANHRC has been deployed on a dual-arm Baxter manipulator to enable the robot to autonomously manipulate articulated objects. The Baxter operates on a 3-link articulated object, assuming that the angle resolution is 90 *deg*, i.e., a $3-4$ problem according to the definition introduced above. Figure 2.14 shows a sequence of configurations, including the initial one in the top-leftmost position, and the goal one in

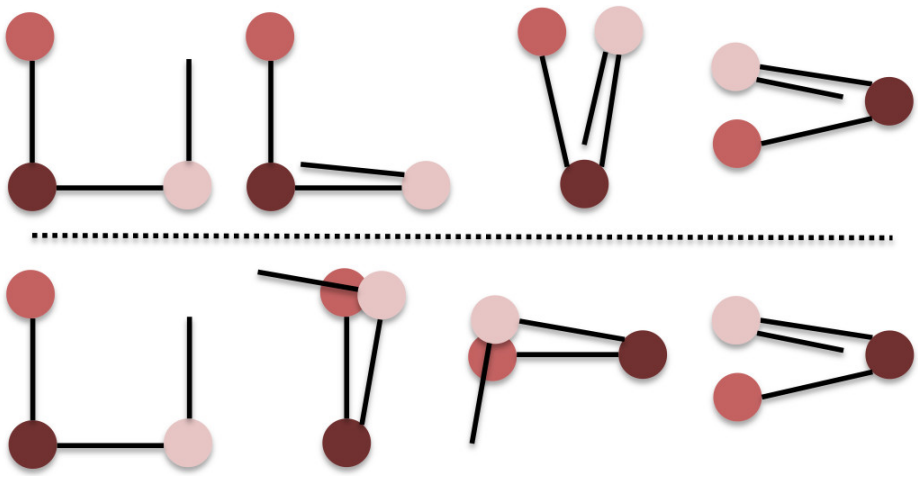

Figure 2.13 A sequence of configurations for a $3-3$ problem, using the relative formulation with Probe (first row, from left to right) and Madagascar (second row, from left to right).

the bottom-rightmost position, from left to right and top to bottom, whereas Figure 2.15 shows the corresponding relevant instants during the execution of the plan by the robot. It is worth noting that, each time a RotateClockwise or RotateAntiClockwise action is executed, the actual robot behaviour is made up of three steps:

1. Firmly *grasp* the link associated with the interested joint that must be kept still;
2. *Grasp* the link that must be rotated;
3. Perform a *rotation* of the proper amount.

In PLANHRC, this can be done indifferently by the left or right robot arms, according to a simple heuristics related to which arm is closer to the link to operate on. Grasping actions in Figure 2.14 are indicated with grasping signs close to the interested link, plus an *R* sign to indicate that the action is performed with the right arm, or *L* otherwise. We decided not to model grasping actions at the planning level for two reasons: on the one hand, they would have increased the burden of the planning process; on the other hand, each rotation must be preceded by a grasping operation, and therefore this sequence can be easily serialized in the execution phase.

Figure 2.16 and Figure 2.17 show two examples of plans where human intervention occurs to successfully accomplish the whole cooperation process. In the figures, the two sequences must be analysed from top to bottom and left to right.

In Figure 2.16, it is possible to see that the human operator performs an action *while* the robot is executing a rotation action on other links (top-right and mid-left snapshot). The

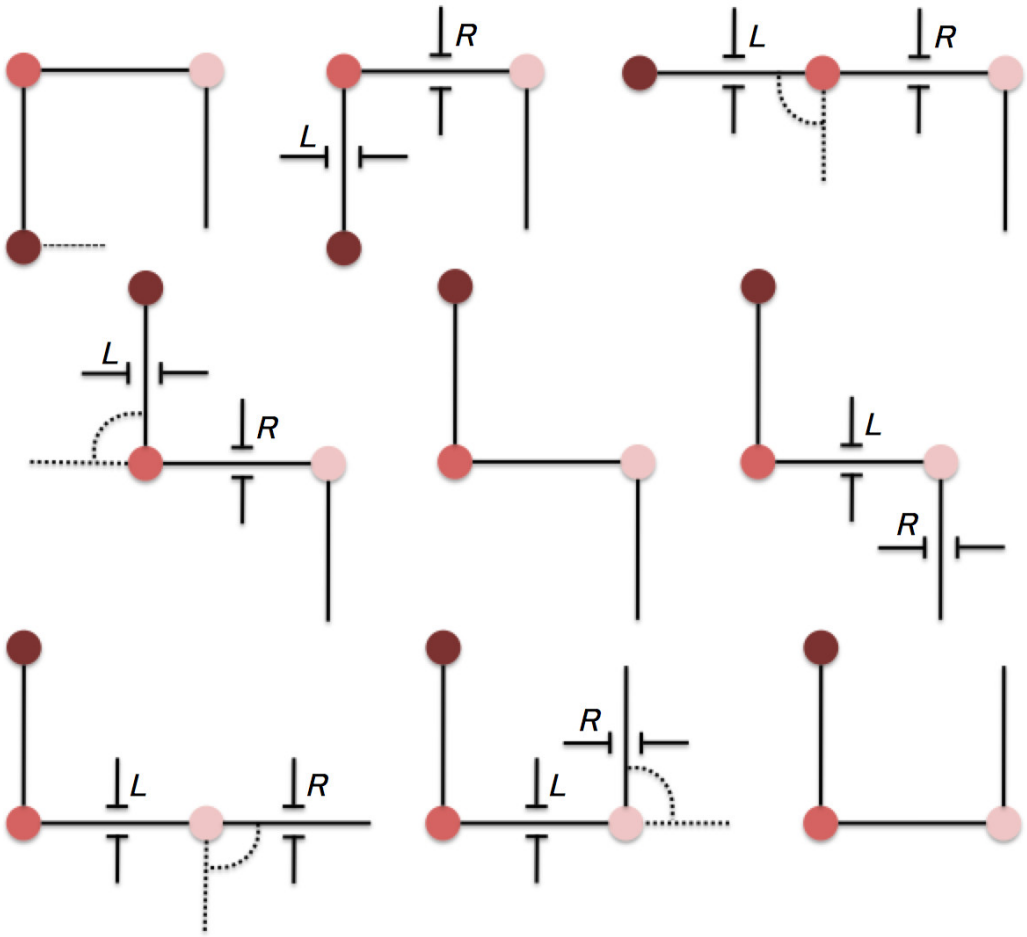

Figure 2.14 A sequence of configurations for a $3-4$ problem, from left to right and top to bottom, as seen from the robot's perspective.

action performed by the human operator leads to a situation compatible with the object's target configuration. As a consequence, the final configuration is reached in snapshot mid-centre. Afterwards, the operator modifies again the status of the first link (mid-right snapshot), thereby leading to a configuration not compatible with the goal one. As a consequence, the robot intervenes to restore it (bottom-centre and bottom-right snapshots). This sequence demonstrates two important features of PLANHRC: first, the freedom human operators have in performing actions asynchronously with robot actions; second, the robot capabilities in keeping the cooperation *on track* coping with possible human mistakes.

Figure 2.17 shows an example where a human operator helps the robot complete an action, which was not performed in its entirety. The robot starts executing a plan (top-left and top-centre snapshots). However, a rotation action is not completed, leading the object's configuration to a state not compatible with the expected one (mid-right snapshot). Then,

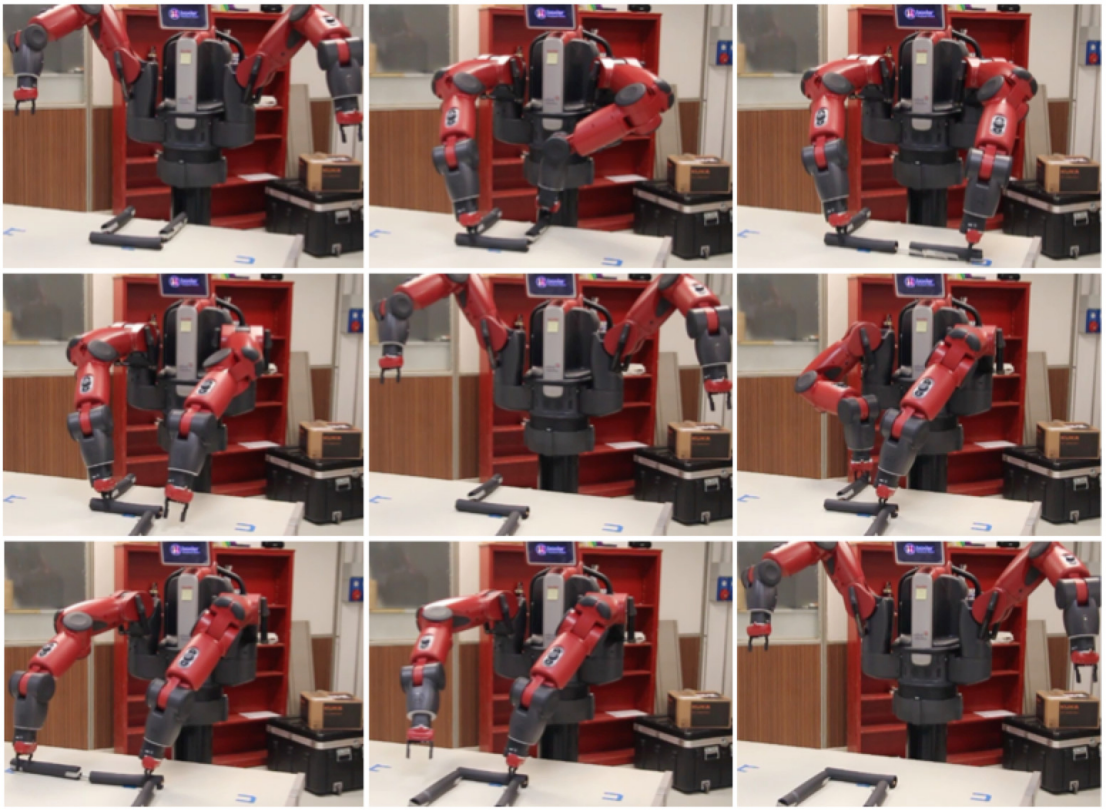

Figure 2.15 The sequence of Figure 2.14 as executed by Baxter without human intervention.

the human operator intervenes with an action aimed at completing the intended rotation and, at the same time, performing an additional rotation on the last link in the chain (mid-right snapshot). From that moment on, the robot autonomously completes the plan. This sequence shows how a plan can be successfully recovered by human intervention, and the fact that the robot can seamlessly continue plan execution.

2.6.4 Discussion

On the basis of the requirements outlined above and the experimental analysis carried out to evaluate the whole PLANHRC architecture, it is possible to make a few interesting remarks, perform a comparison with other approaches in the literature, and draw some conclusions. In particular, the discussion that follows is focused on three aspects, namely planning performance, the generation of *natural* sequences of manipulation actions and the resulting cooperation process according to which human operators interact with the robot.

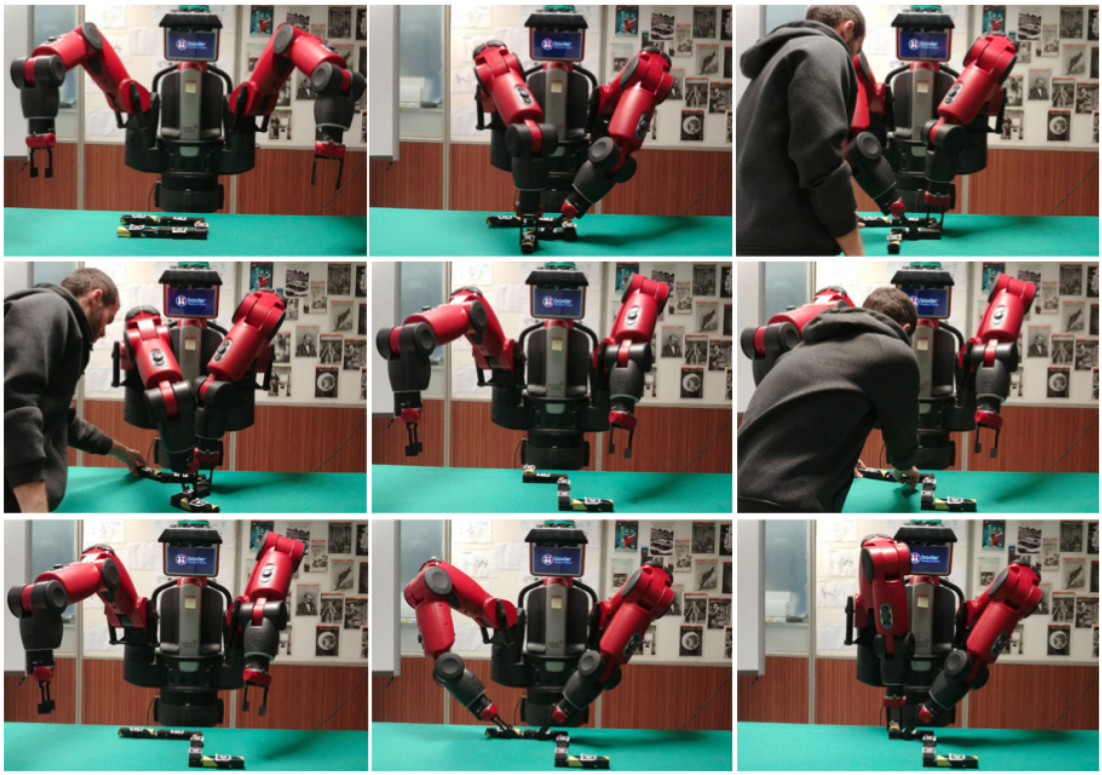

Figure 2.16 A series of manipulation actions executed with the help of a human operator.

Planning performance

The absolute and the relative formulations are characterized by different performance results.

When using the absolute formulation, both Probe and Madagascar are capable of solving problem instances with a limited number of links and orientations in less than 1 s, which is a reasonable upper bound for the reasoning time of a collaborative robot interacting with a human operator, with Probe outperforming Madagascar on bigger problem instances. With around 10 links, the time required to obtain a plan (if it exists) significantly increases, due to the large number of possible orientations, with solution times up to an average of 100 s and beyond. When using Probe, solution times for the same problem instance have a certain variance, which is almost uniform for different numbers of links and possible orientations. If Madagascar is used, such variance generally decreases, but sometimes it may become significantly large, as shown for example in the problem instance $8-4$. By carefully analyzing cases where Madagascar shows significantly high runtimes, we observed that the planner finds problem instances where subsequent connected joints need to be rotated in opposite ways (e.g., the angle of one joint has to be decreased, while the angle of the other joint has to be increased) particularly challenging to solve. In that cases, the planner keeps

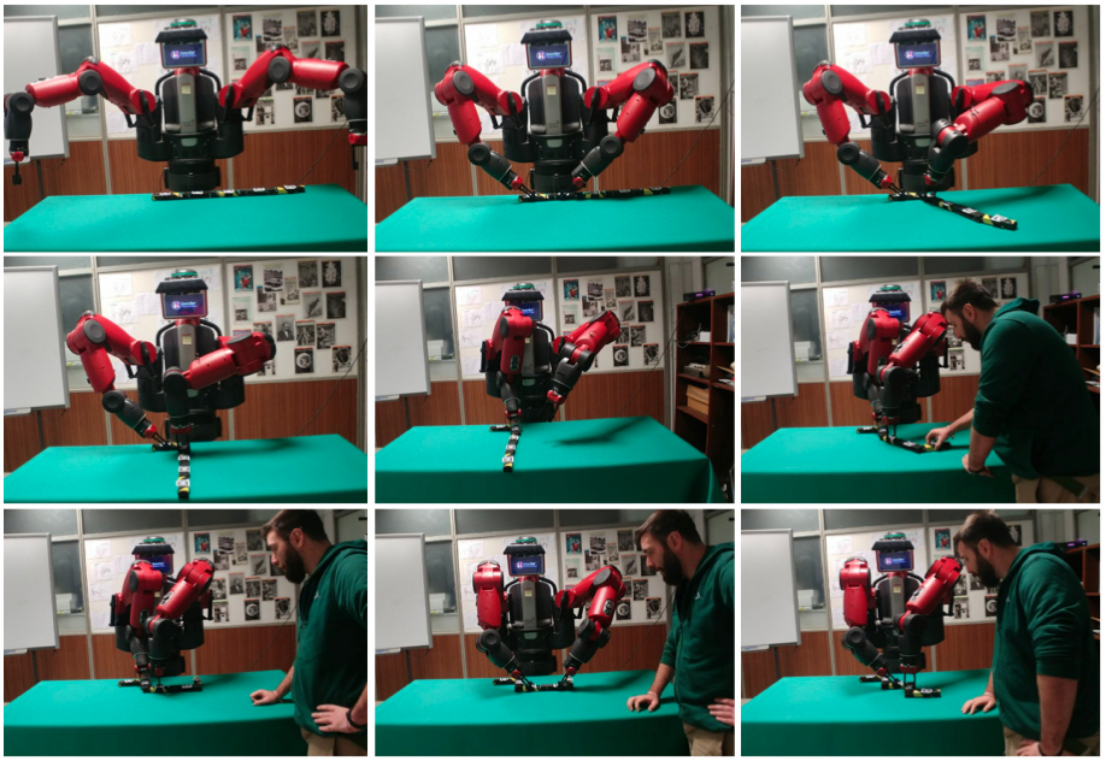

Figure 2.17 Another series of manipulation actions executed with the help of a human operator.

looping between a very small number of configurations, trying to fix the orientation of a joint at a time, ignoring the effect of such actions on the rest of the articulated object. As far as human-robot cooperation processes are concerned, if an absolute formalization were used, then Probe would represent the best trade-off between complexity and solution times. In principle, Madagascar would be a better choice for problems with a reduced number of links and possible orientations, but the occasional presence of large variances in solution times would seriously jeopardize the human-robot cooperation process. The two planners behave differently when using a relative formulation. Both Probe and Madagascar prove capable of solving large problem instances (i.e., with up to 20 links and up to 12 possible orientations) in less than 0.2 *s*. Solution times increase also in this case, but the very low time scale makes such trend relevant only to a limited extent. Differently from the case with the absolute formulation, Probe behaves quite deterministically, and the same holds for Madagascar. When dealing with human-robot cooperation, both planners are suitable to be used if a relative formulation is adopted, with a slight preference for Probe.

The relative formulation proves to be essential when the robot must deal with the directive D_2 discussed in the Introduction, and in particular to allow for a fast action re-planning when needed, as required by R_3.

Differently from those approaches encoding human operator preferences in the planning model, typically using heuristics Gombolay et al. (2015), when adopting a relative formulation PLANHRC tends to find minimum-length plans (in terms of number of actions), i.e., the plan as devised by the robot is *efficient*. As a matter of fact, interventions of human operators are treated by PLANHRC as *perturbations* with respect to the execution of the efficient plan. However, sometimes these perturbations may be helpful (i.e., the human operator helps the robot perform an action), whereas in other cases they constitute *detours* with respect to the original plan, which is tolerated because such detours express human operator preferences. Differently from the approaches presented in Cirillo et al. (2010); Roncone et al. (2017); Wilcox et al. (2012), PLANHRC does not model human preferences in the planning models, but accommodates for them on-line. Only to a limited extent, the approach presented in Darvish et al. (2017) goes in the direction pursued by PLANHRC. The use of AND/OR graphs to model a limited number of alternative cooperation models allows human operators to select on the fly which one they want to follow. However, the AND/OR graph encodes models which have been *a priori* defined, and this is different from the approach of PLANHRC where (i) there is no need for such an encoding, and (ii) in principle, the cooperation is not limited to a given number of alternatives.

Natural action sequences

In general, the two formulations lead to qualitatively different plans, i.e., plans with different actions.

Independently of the employed planner, the absolute formulation originates plans longer than those obtained using the relative formulation. In the absolute case, the solution may contain apparently unnecessary actions, as well as repeated sequences of actions. This is due to the fact that when working on orientations of links located downstream in the chain, such orientations may be later modified as a side-effect when the algorithm operates on links upstream, therefore requiring reworking on downstream links. Such plans are the result of certain planner heuristics. However, they are often unnatural for humans to understand, which is of the utmost importance in human-robot cooperation processes.

Plans obtained starting from the relative formulation are shorter and – in a generic sense – more understandable by humans. Since the representation of orientations is relative for pairwise links, the planner does not need to modify orientations of downstream links

multiple times, and solutions tend to include sequences of actions operating on the same link. This makes plans easy to follow, irrespectively whether they are generated using Probe or Madagascar.

Thus, as far as naturalness is concerned, the relative formulation must be preferred over the absolute formulation. Shorter and easy-to-understand plans are supposed to strengthen a human operator's ability to supervise robot actions in compliance with directive D_2 and to intervene when required, as prescribed by requirement R_5. However, it is noteworthy that PLANHRC has not been tested in real-world conditions yet. As a consequence, there are still to-be-validated hypotheses requiring us to conduct a specifically designed study, also related to the role of context-aware planning in human-robot cooperation Mastrogiovanni et al. (2013).

According to the studies discussed in Gombolay et al. (2014, 2013), human operators tend to prefer a partial control on the cooperation process, with the aim of maximising the overall human-robot team's performance. The approach pursued by PLANHRC goes in this direction in that it enables the robot to generate an efficient plan, but it allows humans to intervene when required. If compared to those approaches explicitly or implicitly encoding human preferences in the cooperation process Cirillo et al. (2010); Darvish et al. (2017); Johannsmeier and Haddadin (2017); Roncone et al. (2017); Wilcox et al. (2012), PLANHRC does not offer any formal guarantee about the naturalness of the generated plan, that is to say in terms of an easy understanding of the sequence of basic manipulation actions by human operators. However, when a relative formulation is adopted, the planner tends to produce natural, easy-to-understand plans without *prior* knowledge being encoded in the system, which is a clear advantage should the system be extended to other use cases.

The cooperation process

In absence of errors related to action execution, once a plan is available PLANHRC should be able to carry it out in its entirety. This is in agreement with directive D_1 discussed in the Introduction. However, when either one action is not executed successfully or it has been carried out only partially, a human operator can intervene to obtain an object configuration that the robot can operate upon. These two facts support requirement R_2.

As described above, before any action is executed, the robot checks whether a number of expected normative predicates hold in the current planning state. Implicitly, this means that any error in action execution or human intervention is synchronously assessed before the next planned action can start. Obviously enough, this represents a limiting factor for PLANHRC, and originates from the focus on planning sequences of *states* to be reached

rather than actions. A more flexible reactive system may make use of human actions to determine causes of faults on the fly, instead of being limited in assessing their outcomes at discrete intervals. However, it also enforces the fact that humans are in control at any time: the robot simply waits for human intervention to finish and then plans a course of actions from that moment on.

This approach makes PLANHRC different from a number of human-robot cooperation frameworks described in the literature Agostini et al. (2011); Caccavale and Finzi (2017); Karpas et al. (2015); Koppula et al. (2013); Kwon and Suh (2014); Liu and Fisac (2015). While in Agostini et al. (2011) a predefined set of possible cause-effect events are considered, PLANHRC consider each predicate in the ontology as normative information that must be validated on-line, independently of the cause that may have generated a norm violation. PLANHRC does not explicitly detect human operator actions Koppula et al. (2013); Kwon and Suh (2014); Liu and Fisac (2015), and therefore it is not able to perform action-dependent behaviour, but only state-dependent behaviour. In virtue of this, PLANHRC may be employed to perform anticipative behaviors like done in Liu and Fisac (2015), where a Bayesian network is employed to that aim, but using only the current cooperation state (i.e., adopting a sort of Markov assumption). It is noteworthy that PLANHRC explicitly does not consider temporal aspects in planning execution. Whilst – in principle – temporal PDDL-based planners may be used to generate plans adhering with well defined temporal constraints, at run time PLANHRC may support the use of temporal-based constraints validation as done, for instance, in Karpas et al. (2015).

2.7 We need a better way to plan in the loop

This Chapter describes a state-of-the-art a hybrid reactive/deliberative architecture for collaborative robots in industrial scenarios, and it shows a use case where a human and a robot collaboratively manipulate articulated objects. In particular:

- It shows how two different representation and planning models for articulated objects impact on planning performance and plan quality, in terms of number of actions and simplicity of the plan;
- It demonstrates the feasibility of an approach to human-robot cooperation where actions by human operators are automatically managed in virtue of their effects as perceived by the robot.

The developed architecture is evaluated on the basis of a number of functional and non functional requirements: the possibility for the system to implicitly recognize the effects of human actions, the robot's capabilities in adapting to those actions, and a fast (re-)planning process when needed, just to name the most important ones.

From the evaluation, one result is clear. Plan-HRC presents some highly desirable characteristics in the context of Human-Robot cooperation, such as allowing the human operator to lead the interaction and act when and as he sees fit, while simultaneously striving towards an efficient execution. Nevertheless, re-planning is a very frequent occurrence, potentially happening as often as every single cycle of the architecture loop, in the limit case where the robot and the human act as two agents of a turn-based process. Hence, PLANHRC's efficiency and the quality of the cooperation itself, depends strongly on the time needed to re-plan.

In the work presented in this pages, a solution was found in the relative formulation of the model, that while not as intuitive as the absolute one, lead to acceptable performance and even more understandable final plans compared to the alternative. This fact proves that a clever formulation can be a powerful tool in order to maintain planning time in check: we will discuss this approach further in the next Chapter. Despite that, not all problem might have a simple formulation, or simply the number of objects needed to plan upon in a real-world scenario might be so big that a planner could struggle even with an otherwise relatively simple formulation. After all, traditional heuristic-search based planners have always been more suitable for offline use, and the extreme online re-planning requirements of the PLANHRC architecture might be an unfair benchmark to judge them. For these reasons, in Chapters 4 to 6 we will explore a radically different way to generate PDDL-compliant plans using Large Language Models (LLM) and show how this method can help us overcome some of the limitations we observed here.

It must also be noted that, while the absolute problem formulation has shown poor performance in these experiments, it also proved to be an extremely challenging benchmark for traditional heuristic-search based planners. Coincidentally, the same holds true, albeit for different reasons, also for LLM. For this reason, we will discuss and expand this formulation further in the next Chapters.

Chapter 3

Optimizing total planning and execution time using Macros

3.1 What is the right level of abstraction?

As we have discussed in the previous Chapter, many AI techniques used in Robotics and Ambient Intelligence for symbolic reasoning and task planning have historically suffered from the well-known combinatorial explosion issue: that means that as the number of symbols and relationships among them increase, the computation time increases as well, in a dramatic way. This phenomenon heavily limits the application of symbolic AI techniques in the real world, as it is often impossible to keep such AI model in a loop unless the problem at hand is very simple.

At the moment, there is no real solution to this problem, even though a number of practical workarounds can be implemented to alleviate the issue. In literature, a traditional workaround to this problem is to increase the level of abstraction of the model, in order to reduce the number of symbols that must be considered by the solver. This approach is common in automatic planning, and, especially in PDDL-based planning, it is usually implemented through the use of *macros*. Macros are actions that an agent can perform that are composed as a sequence of multiple, elementary actions. Albeit macros can reduce planning time significantly, the resulting plan may be suboptimal at execution time compared to a plan using exclusively elementary actions, as every time the planner selects a macro, there is a chance that the macro involves the execution of an otherwise unnecessary elementary action. Of course, such downside can be reduced, but not completely eliminated, by defining macros intelligently, that is by bundling together actions that are usually executed in sequence anyways. This task usually falls into the designers' hands and heavily depends on their intuition, albeit in

literature some approaches have been proposed to generate macros automatically according to several heuristics.

The literature on macros heavily focuses on their effect on planning performance, but little work has been done to assess their true impact on real world performance in robotic applications. First of all, considering how fast planning times can grow in complex planning domains, in many cases the occasional execution of an unnecessary action might be a cost worth paying for consistently lower planning times. Secondly, during their execution, macros are usually decomposed back into their primitive elementary actions for them to be executed in sequence. Yet, it is also possible to ground macros to their own specific implementations instead of decomposing them again into their original constituent actions. In this way, additional optimizations are possible that can mitigate the occasional unnecessary action.

Let us take as an example the manipulation of articulated objects discussed in the previous Chapter and imagine that we want to rotate clockwise a link of our object by 180 degrees. If our arbitrary elementary actions are 90 degrees rotations, we can thus define a macro constituted by two successive clockwise rotations. We now have two options at execution time: either execute the two original 90 degrees rotation; or execute a new procedure especially designed to ground the macro, and that performs a 180 degrees rotation. Intuitively, this new procedure can save time over executing the two elementary actions' by the simple fact that the robot could plan a single smooth path and possibly avoid releasing and grasping the link again in between the two actions.

The combination of this two factors should lead us to question the common knowledge that macros are only a workaround for faster planning times that is detrimental to execution. Instead, in this Chapter we hypothesize that certain level of abstraction through macros can have a largely positive effect on total planning and execution time.

Hence, the major contribution of the work presented in this Chapter, is to provide, to the best of our knowledge, a first-of-its-kind analysis of PDDL macros impact on both planning and execution time in a robotics use case, taking as reference scenario the manipulation of an articulated object.

In order to prove our hypothesis, we extended the absolute formulation domain introduced in Section 2.5.5 and define five additional planning domains, each characterized by an increasing level of abstraction. At lower levels, few macros are introduced and they coexist with some of the original elementary actions. Vice versa, at higher levels all actions are macros. We then compared the original model and the five additional models including macros both in term of planning and execution time. Execution time has been measured in a simulation environment making sure that each macro has its own optimized implementation.

Results on our reference planning scenario show that with an adequate choice of macros it is possible to reduce average planning time by 85% and its variance by 99% without a major impact on execution efficiency. Indeed, even execution time can be reduced, albeit slightly, by up to 17%.

The limitations of the approach discussed in these pages is that it is obviously dependent on the domain taken into consideration, the macros' definition and the implementation of the software modules that ground said macros. Despite that, our results prove that in scenarios where planning time is critical, such as when high level planning is a component of a larger robotic architecture that must adapt to unforeseen events and human intervention, macro optimization is a powerful tool that not necessarily jeopardize execution times.

3.2 Macros: definition and usage in task planning

The manipulation of articulated objects plays an important role in real-world robot tasks, both in home and industrial environments (Heyer, 2010a). In Chapter 2, much attention has been paid to the development of approaches and algorithms for generating the sequence of movements a robot has to perform in order to manipulate an articulated object, therefore we will keep this Section short and focus instead on macros and their use in this specific operational scenario.

The manipulation of an articulated object in a 2D workspace involves the possibility of performing actions like rotating one of its link with respect to one another around their shared joint, with some constraints in robot actions execution to enforce their feasibility. Macros are actions that an agent can perform that are composed as a sequence of multiple, elementary actions. Therefore, we can efficiently exploit macro actions in order to generate more compact and effective plans. Macros can be considered as sequences of elementary actions that, on an application viewpoint, would be useful to be performed in such a sequence, and be considered as a single action. The concept of macro is as old as STRIP based planning (Fikes et al., 1972), but given that the definition of a set of macro is arbitrary, there has been a significant research efforts over the years to generate macros automatically according to some beneficial heuristic. In particular, macros can be defined through knowledge automatically learned offline from both the domain taken into consideration and the planner to be used to solve it (Gerevini et al., 2011), but online (Chrpa et al., 2015) and domain-independent (Chrpa and Vallati, 2019) approaches have been proposed too. Most of these works focus on learning macros for general planning domains, but there are a few examples that take specifically into consideration domains employed in robotics, and in particular for mobile

Table 3.1 Summary of models that uses macro actions discussed in this Chapter.

Model	Description
`m_0`	No use of macros
`m_25`	25% macro
`m_25_B`	Alternative 25% macro
`m_50`	50% macro
`m_50_B`	Alternative 25% macro
`m_75`	75% macro

robots (Hofmann et al., 2017). Surprisingly, macros have already been successfully employed in automated planning for articulated objects' manipulation (Bertolucci et al., 2021), but only in systems based on the Answer Set Programming (ASP) framework (Lifschitz, 2019). In this Chapter, we will discuss a similar approach based on PDDL instead, in order to extend the models defined in the previous Chapter and maintain compatibility with a wide range of PDDL-based tools in robotics.

The lack of focus on robotics applications in the literature on macros is part of the reason why most of these work focus mainly on their effect on planning time, and are instead less interested in their effects on execution time and quality. Nevertheless, insights about the right level of abstraction to adopt while keeping into consideration execution, would not only be useful in robotics applications, but could be themselves useful to improve existing automated macro-generation systems with heuristics which are more aware of the real world.

3.3 Updated planning models with macros

In this Section we will broadly describe five additional planning models for the manipulation of articulated objects using macros. Table 3.1 summarizes the models taken into consideration in this Chapter.

A sixth model is included in the Table, `m_0`, which represents our baseline model without any use macros. This model corresponds to the absolute formulation model outlined in Section 2.5.5, with a small modification. Whereas before we have considered that the robot can keep a link still and rotate the other, or vice versa, causing effects to propagate both upstream or downstream accordingly, we will now enforce only downstream propagation. This is achieved by introducing a new predicate, `link-before`, that models the order of the links, and updating the rotation actions preconditions accordingly. In practical terms, the

robot can keep the first link still and rotate the second, but not the opposite. This modification mitigates the issues discussed in Section 2.6.4, making the behavior of the conditional model more intelligible to the human operator and thus simplifying the qualitative analysis of the resulting plans.

All successive models employ the same definitions of object types and predicates, and only differ in the action set which is made available to the planner to solve problems on that given domain. As a consequence, problem instances are intercompatible among the proposed domains, which in turn allows a simpler and fairer comparison of their performance. In the naming convention for the models, numbers ranging from 0 to 75 represent a rough percentage of how much the the model relies on macros, which we will refer to as *macro-ness*. Hence, the base model is called `m_0` as it does not employ macros at all, while `m_75` represents our most abstraction-aggressive model. It must be noted that `m_75` already uses exclusively macro, but we assigned this name to it to reflect the fact that the macros employed are relatively close to the elementary actions and that an higher level of abstraction is theoretically possible. A hypothetical `m_100` would obviously not make sense, as it would imply an action that literally solves the problem. For similar reasons, we did not include in our comparison any model with a macro-ness level above `m_75`, as we believe that at that point the use of macros would make the solution of the relative problem instances trivial.

Obviously, in our macro models we only allow macros composed of a combination of actions defined in `m_0`, which are considered as elementary action and are indeed very close to basic physical activities, such as grasp, release move and rotate. As the level of macro-ness increases, there are several arbitrary ways to define macros as a sequence of the aforementioned elementary actions. Based on the physicality of the operative scenario taken into consideration, two obvious candidates can be proposed:

1. **Type A** macros can be defined as a sequence of heterogeneous elementary actions that collectively achieve an evident sub-goal and possibly free up robot resources for successive actions e.g., a `grasp-rotate` or `grasp-rotate-release` action;

2. **Type B** macros can be defined as a repetition of a single action that must be often repeated multiple times in order to achieve a specific sub-goal, but do not guarantee that the sub-goal is achieved, neither free up resources for successive actions, as they assume that the elementary action might need to be repeated additional times in order to achieve the desired sub-goal, e.g., a `rotate-rotate-rotate` action.

The first approach is clearly more conservative, as it presents a limited risk of performing unnecessary elementary actions. This is especially true in domain like `m_25` where macros

and elementary actions coexist, thus the planner can easily choose between using a macro like `grasp-rotate-release` when a single joint angle increment is required, or using elementary actions for larger increments. Nevertheless, occasional inefficiencies can still occur, for example when a link is released before continuing to act on the same joint, causing two unnecessary actions, a release and a grasp.

Conversely, the second approach can intuitively provide large time savings by consolidating rotations of the same joint together, but at a high risk of *overshooting*, thus forcing the planner to correct the joint configuration with one or more elementary rotations in the opposite direction. In this case there is an obvious trade-off between how many elementary actions are grouped together and the expected statistical cost of a mistake; in simpler terms, if a macro is defined as two consecutive rotation, the worst case scenario is that an elementary action must be spent to correct the eventual mistake; if it is defined by three rotations, the worst case is two actions; and so on.

In our naming convention, models which include a `B` in their name include macros of the Type B, while the other models exclusively include macros of Type A. Domains `m_0` and `m_25_B` are integrally reported for reference in Appendix A.

3.4 Experimental setup

As we have discussed at the beginning of this Chapter, our intention is to analyze the performance of the models in Table 3.1 both in therms of planning and execution times. Instead of integrating the whole pipeline in an architecture like `planHRC`, we have decided to keep these two evaluations separated. Nevertheless, great attention has been dedicated to ensure that such evaluations are realistic and compliant with the manipulation of an articulated object in a real-world HRC scenario. Details are provided in the next two subsections.

3.4.1 Planning setup and benchmark composition

In order to test the planning performance of the models in Table 3.1, we developed a custom, highly configurable software to generate random problem instances. In order to obtain problem instances that are reasonably easy to solve but that are also representative of a real-world articulated object manipulation scenario, including the limitations of the robotic platform of choice, the following settings have been adopted:

- The object to be manipulated is composed of 4 links;

- Angle resolution has been set to 24, thus the minimum angle rotation is 15 degrees;
- A joint angle configuration can only range between 0 and -90 degrees (rotate towards the robot) in order to be compliant with the robot allowed workspace;
- All problems start with the object straight, i.e., with all joint angle configurations set to 0 degrees, this is meant to represent a realistic scenario where a single initial, unprocessed object, must be customized in an Industry 4.0 flexible production scenario.

All problem instances were solved by the Probe planner, which ran on an Ubuntu 20.04.4 Windows Linux Subsystem (WSL) environment, deployed on a machine equipped with an 8-core Ryzen 3700X 3600 Mhz CPU and 16GB@3600 Mhz RAM. We recorded planning times as reported by the planner itself together with each generated plan. As customary, for each candidate model we also measured minimum and maximum planning times, standard deviation of the planning time, and the average number of steps (i.e., actions) used to solve the problem.

As an additional metric, for each model we also marked as *failures* every time the planner failed to generate the plan in less than 1s. A low number of failures for a given candidate model, can be used an additional metric to measure the *fluency* allowed by the model in HRC scenarios where frequent replanning is required, as per Hoffman (2019), who show that the delay experienced by a human immediately after completing an action, as incurred by their teammate, has a strong correlation with subjective fluency perception.

3.4.2 Execution in a simulated environment

For our assessment, we generated three additional random problem instances. We have used each of the 6 planning domains that we are considering to plan against these three problems. The planning step has been repeated 10 times and the results have been averaged to account for the stochastic nature of the heuristic implemented by the planner.

As the planner is not optimal and implements a stochastic heuristic, we checked whether for each problem-domain pair we have obtained the same plan over the aforementioned 10 planning attempts. Whenever this was not the case, the first result from the planner was selected for execution, as this is what would happen in the real world.

Experiments to assess and compare the execution time performance have been carried out on the Tiago[1] robotic platform in a simulated environment, which was deployed on a Ubuntu 20.04.5 LTS machine running ROS Noetic, MoveIt and the Gazebo simulator. As

[1] `https://pal-robotics.com/robots/tiago/`

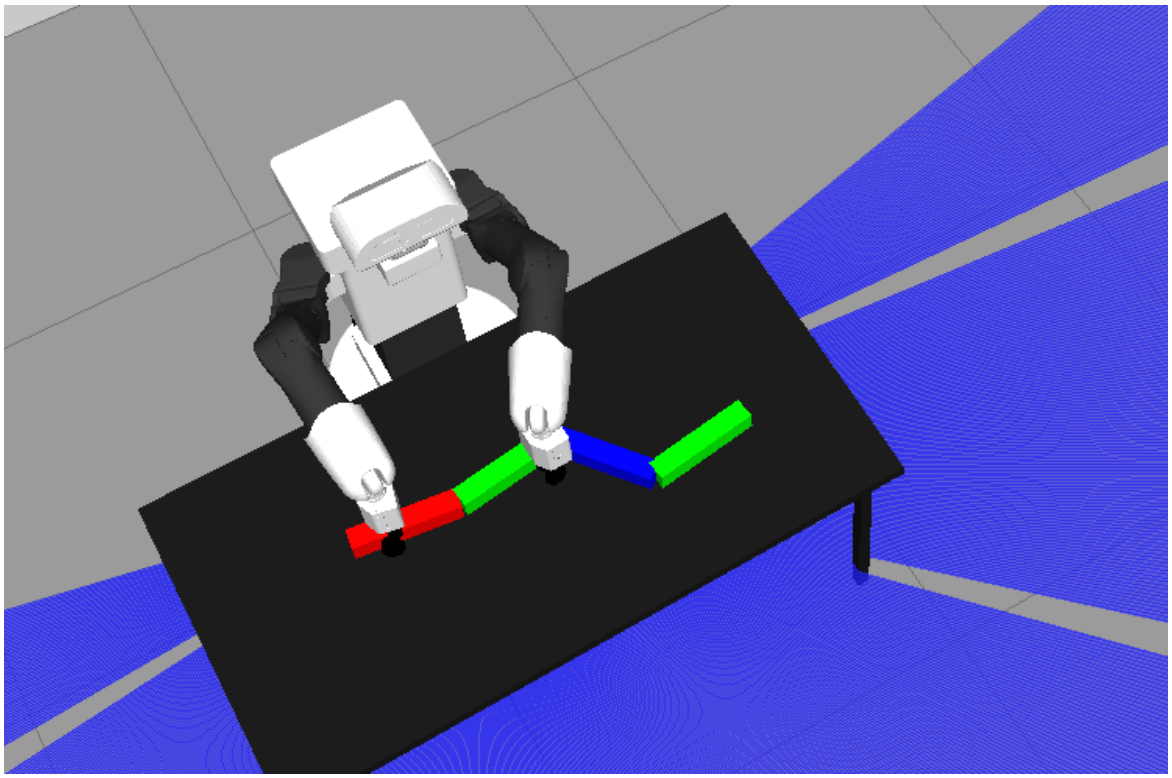

Figure 3.1 The Tiago robot acting on the articulated object in the simulated environment.

we are only interested in measuring the execution time of the plan, there is no complex architecture to control the robot. Instead, a simple parser reads the actions in the plans and triggers the execution of a corresponding motion control routine. Everything is executed in open loop and perception is not considered. Again, in order to make sure that the results are not influenced by any stochastic process, we repeated the simulation multiple times, even though, in this case, we obtained essentially the same results in every run. Figure 3.1 depicts our experimental setup.

3.5 Models comparison and discussion

Figure 3.2 and Table 3.2 report the results of our analysis on planning times. Let us start by discussing the boxplots in Figure 3.2, and in particular the first four: `m_0`, `m_25`, `m_50` and `m_75`. Unsurprisingly, the use of macros leads to a reduction of both median planning times and the variance of planning times, albeit we can observe a diminishing return as we increase the level of macron-ness of the model, as the median planning time of `m_50` and `m_75` is essentially the same.

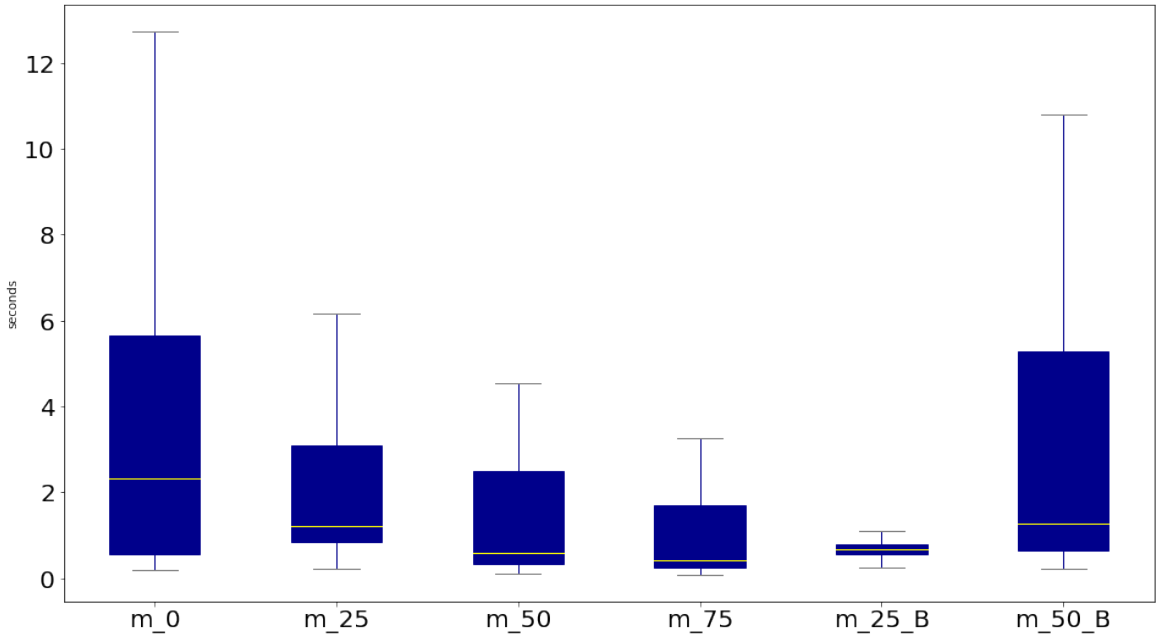

Figure 3.2 The planning times of the Probe planner over the 6 domains presented in Table 3.1 and 100 randomly generated problem instances.

Results for `m_25_B` and `m_50_B` are more interesting. We remember to the reader that models labeled with a `B` use Type B macros as defined in Section 3.4.1, which is a more aggressive choice of macros. Indeed, despite the limited number of macros included in its action set, `m_25_B` results being our overall top performer. It has a slightly higher median planning time compared to `m_75` and is essentially on par with `m_50`, but it has such a small variance that it is clearly the best choice from a planning performance point of view.

On the contrary, `m_50_B` not only presents much worse performance despite having a higher degree of macro-ness, but it is our overall worse performer among all the domains that include macros. By qualitatively observing the plans produced by `m_50_B`, we could confirm that the issue lies in the large number of unnecessary actions induced by the macros. In particular, this model includes the macro `increase_angle_first_child_45`, which corresponds to three consecutive rotations of the same link. We observe that the planner aggressively uses this macro, but often overshoots and then wastes two actions to fix the error. Interestingly, the same macro is included also in `m_25_B`, but this model also includes some elementary actions that provide an alternative choice to the planner. The planner cleverly takes advantage of this possibility and thus avoids wasteful actions.

Table 3.2 Timing performance of the Probe planner over the 6 domains considered.

Domain	**avg_steps**	**t_min**	**t_max**	**t_avg**	σ	**fails %**
m_0	20.787	0.167	53.071	4.928	48.578	68
m_25	13.270	0.186	24.436	2.354	14.989	45
m_50	15.100	0.082	25.949	1.684	11.634	39
m_75	15.860	0.074	43.050	1.436	20.230	31
m_25_B	8.930	0.201	3.524	0.735	0.367	14
m_50_B	13.404	0.164	18.068	2.725	11.371	50

The main takeaway of these first results is that a certain degree of macro-ness is usually highly beneficial, but higher levels can lead to a sudden and catastrophic loss of performance. Another important aspect is that the quality of the macro has higher impact than their number.

Considering also the diminishing returns of higher levels of macro-ness, it seems that the best course of action is to:

- Include macros of Type B whenever possible;
- Limit the use of macros to a few essential ones;
- Combine the use of macros with elementary actions, in order to limit the chance of the planner being forced to select a wasteful macro.

We can have a more in depth look at the timing performance of the 6 domains considered in the Chapter in Table 3.2. Again, m_25_B is superior in all metrics except minimum planning time, where its outperformed by m_75. In the Table we can also observe why m_25_B might be the preferable choice in HRC scenarios: while using this planning domain, Probe exceeded our self-imposed limit of 1s on planning time only 14% of the times, which is less than half than the second best contender, m_75. Even better, the worse planning time for m_25_B is about 3.5s, with the second best m_50_B reaching about 18s. While 3.5s is almost 4 times higher than our self-imposed limit of 1s on planning times, it is still very acceptable in the context of HRC.

Let us now consider the results of our experiments at execution time. As discussed in the previous Section, we have selected three problems to be solved by probe using all 6 domains taken into consideration. Table 3.3 reports the average planning time over 10 attempts. We immediately observe that from a planning point of view, the first problem is particularly

Table 3.3 Average planning times for the 18 plans to be executed. We compare each of the 6 domains considered in this chapter over 3 sample problem instances.

Problem	m_0	m_25	m_50	m_75	m_25_B	m_50_B
1	20.693	2.719	0.789	0.516	0.737	0.665
2	0.530	0.527	0.211	0.165	0.599	0.410
3	0.678	0.604	0.214	0.187	0.628	0.374

challenging, especially for our baseline model m_0. In contrast, the other two problems are relatively easy to solve for Probe regardless of the choice of domain.

Figure 3.3 shows the execution time of the plans generated from the three test problem instances by the six domains taken into consideration. In the first problem, we observe execution times growing with the increasing level of macro-ness, as expected. Nevertheless, m_25 performs just as well as m_0, and m_25_B only adds 2s. Interestingly, 2s is roughly the difference between m_25 and m_25_B planning times, resulting in a very similar total planning and execution time. On the other side, we observe a sharp rise in execution times at m_50 and above levels, which all result in very similar plans and an execution time about 25% longer than m_0, m_25 and m_25_B.

Problem 2 highlights the situation that we initially hypothesized, where optimizations in the execution routines that ground the macro actions, actually more than compensate the loss of time due to a suboptimal plan, compared to to the one generated with m_0. As a consequence, the execution time of the problem obtained by m_25_B and m_25 are respectively 17% and 6% shorter than the m_0 execution time. Again, we observe that higher level of macro-ness are instead highly detrimental, with m_50_B taking about 90% more time than m_25_B.

Finally, problem 3 well represents the not-so-rare possibility, that the macros actually do not introduce any unnecessary actions compared to the baseline plan generated with m_0, thus all models exactly take the same time.

Overall, the results presented so far confirm our hypothesis that macros are ultimately beneficial to total planning and execution time. As expected from a planning point of view, models employing macros tend to lead to a significant reduction in planning times and their variance, which ultimately support shorter waiting times and improved fluency in the context of a HRC robotics architecture. More importantly, we have shown that the use of macro does not always imply an abrupt decrease in execution time performance, at least at lower levels of macro-ness. A careful choice of macro and ad-hoc implementations for the functions that

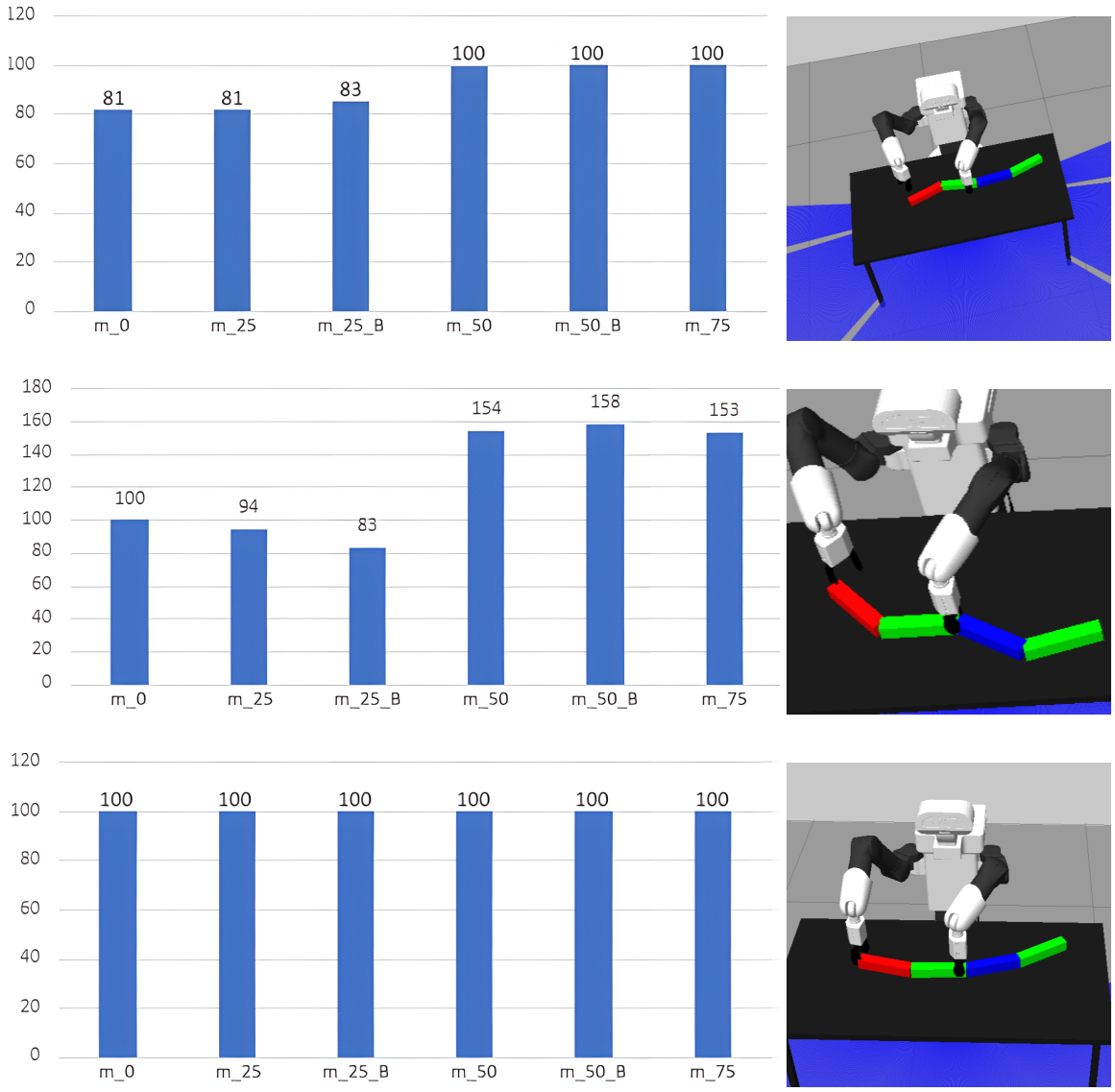

Figure 3.3 Execution times of plans generated by by the six planning domains taken into consideration for the 3 test problems. Planning times are expressed in seconds.

ground said macros, hugely limits the negative effect of macros, and can even lead to shorter execution times in a few cases, albeit by a small margin.

Obviously, the main limitation of our work is that it is highly dependent on our choice of operative scenario, planning domains definitions, and implementation of actions in the execution stack. Hence, while we identified a clear best performer in the `m_25_B` domain in terms of both planning and execution performance, such result cannot be straightforwardly generalized. Nevertheless, our results highlight a more general trade-off between planning

and execution times. Let us take as an example `m_75` which is our second top performer in terms of average planning times. This model leads to longer execution times with respect to our baseline domain `m_0` by more than what it can save in planning times. Yet, there are HRC applications where fluency is more important than efficiency, thus a significant reduction in planning times and their variance might be well worth an increase in execution times.

Chapter 4

Beyond heuristic-search planning: Neurosymbolic approaches

4.1 The potential benefits of Neurosymbolic Planning

As we have seen in Chapter 2, we are witnessing the emergence of scenarios where robots are expected to operate with an increased autonomy, versatility, and robustness, especially in tasks to be carried out jointly, or in close coordination with humans. In fact, in human-robot collaboration tasks, humans could act as simple *beneficiaries* of robot behaviors, or they could act as *teammates*, actually partaking in robot actions as they unfold. This state of affairs not only holds in assistive and service robotics, but is becoming particularly relevant in settings inspired by the Industry 4.0 paradigm, which is believed to profoundly shape the nature of next-generation manufacturing operations (Heyer, 2010b). These scenarios require robots to exhibit quite dynamic and reliable capabilities in sustaining a purposeful collaboration with human operators: while robots may be characterized by a continuum range in autonomy, also depending on the specific operation at hand, human operators may supervise robot actions and intervene if and when necessary. Therefore, robots must not only operate reliably in line with the interaction with human operators, but also to face unforeseen events very likely to happen when the work to be done is poorly structured, and human loosely predictable behavior is involved (Darvish et al., 2018). This paradigm shift requires robots to validate or re-plan their future actions continuously, ideally to predict or at least to react to changing conditions or the outcomes of unforeseen human behavior. As a consequence, it turns out that planning performance is of critical importance to ensure both an efficient robot operation and an effective human-robot collaboration.

In scenarios characterized by such requirements, conventional, symbolic task planners have limitations that hinder their efficacy. In current practice, they use search heuristics to explore the range of possible states that the modeled environment can assume until they find a sequence of actions in-between states leading to a state compatible with the desired goal situation. As a result, they tend to exhibit planning times' combinatorial explosion as the number of symbols (representing classes of entities, instances, modeled fragments of reality as predicates, or actions, and their combinations) involved in the planning process increases. As we have shown in Chapter 2, this phenomenon makes frequent re-planning extremely costly in terms of needed planning time, specifically considering the fact that a plan synthesized by a symbolic task planner is an *atomic* entity, that is, it is either generated in its entirety, or it is not available. Although approaches for hierarchical task planning have been proposed in the literature, whereby high-level (generic) plans may be created quickly, and low-level (specific) sub-plans can be generated in chunks, the basic issue remains, and it is also necessary to carefully engineer which part of the plan could be synthesized at the high-level, and which part (in terms of actions actually executable by a robot) should be made available in sub-plans (Darvish et al., 2021; Mastrogiovanni et al., 2004; Murali et al., 2020).

When human operators and robots collaborate on a task, they can achieve a high standard of coordination, leading to a synchronized blending of their actions with precise and efficient timing (Carfí et al., 2019). This feature is referred to as *fluency* as per Hoffman (2019), and it has been demonstrated that the delay experienced by a human immediately after completing an action, as incurred by their teammate, has a strong correlation with subjective fluency perception. Obviously enough, the aforementioned combinatorial explosion of planning times negatively affects the quality of human-robot collaboration, and specifically its fluency. It has been argued that fluency in human-robot collaboration could be improved by introducing intuitive communication channels able to make clear which action a robot will carry out next Macció et al. (2022). In fact, that works while the cooperation unfolds, whereas in case of re-planning it would be necessary to put the collaboration process on hold.

In the next Chapter, we introduce a framework, which we call Teriyaki[1], to train and invoke Large Language Models (LLMs) (Hatcher and Yu, 2018) to behave as task planners. LLMs such as OpenAI's GPT-3 (Brown et al., 2020) are characterized by the possibility of:

- Linearly scaling in computational complexity with the total length of prompt and completion;

[1]Teriyaki: `https://github.com/alessiocpt/teriyaki`

- Generating partial results while iteratively predicting the next set of symbols in a sequence.

We argue that these two features could be leveraged to design a task planner with a reduced run-time complexity, and capable of making the next action available ahead of full plan synthesis, which would allow a robot using such planner to begin action execution before the entire plan is generated, in a full iterative fashion. If the robot architecture allowed it, this may unlock concurrent plan generation and execution. Moreover, if such a model could be trained to receive input and return output using the Planning Domain Definition Language (PDDL) (Aeronautiques et al., 1998), it would maintain full compatibility with existing software frameworks for symbolic task planning such as ROSPlan (Cashmore et al., 2015b).

We refer to Teriyaki as *neurosymbolic* planning, as it combines neural network based learning as *substrate* with symbolic knowledge representation and reasoning as *logic* (Garcez and Lamb, 2020). Unlike traditional symbolic task planners, our approach is domain-dependent and requires training, but it is trivial to generate large data sets for training on a given domain using *traditional* task planners.

To summarize, the main contributions of this work, which constitutes the core of this Thesis, are the following:

- We designed, developed, and released to the community Teriyaki, a framework to train and use GPT-3 to solve PDDL-compatible problems for a given domain;
- Compared to similar methods in literature, Teriyaki has been tested on plans 80% longer and including conditional effects;
- We demonstrate that LLMs can outperform non-optimal planners in terms of plan length;
- We propose to exploit LLMs capability to stream the output plan as it is being generated, for simultaneous planning and execution, leading to a significantly shorter waiting time for execution to begin in real-world human-robot collaboration scenarios.

Despite the positive results, we want to highlight that solvers generated with Teriyaki should not be considered at the moment as a complete alternative to traditional, symbolic planners. Instead, we recommend considering them as a proof-of-concept of the planning capabilities of LLMs and their possible applications as the technology matures, especially regarding to human-robot collaboration scenarios.

In this Chapter, we will lay down some necessary groundwork on Large Language Models and how their logical reasoning capabilities have been assessed so far in the literature. In particular, we will discuss the limitations of most of the approaches in the literature and justify why instead we decided to base Teriyaki on a completely different assumption.

4.2 Language models are (unreliable) few-shot learners

Neurosymbolic approaches foresee that:

- Their knowledge is encoded in vector-based representations supporting neural networks which maximize an efficient learning from data;
- Discrete symbols *become available* as a result of querying and extracting knowledge from the trained network (Garcez and Lamb, 2020).

For a long time, it has been theorized that such approaches may provide an alternative to the problem of combinatorial explosion in reasoning by leveraging the reasoning mechanisms encoded in the learning process. However, it is only recently that they started to gain traction, mainly for Natural Language Processing (Dale, 2021). Most notably, GPT-3 and its successors GPT-3.5 and GPT-4, are famous LLMs released by the company OpenAI[2] that achieved astonishing results in generating human-level complex text in the form of structured sentences (Brown et al., 2020). Other popular models are LLama 2 (Touvron et al., 2023), LaMDA (Thoppilan et al., 2022), PALM (Chowdhery et al., 2022), Megatron-Turing NLG (Smith et al., 2022) or BLOOM (Scao et al., 2022). A few of the most notable applications of these models are text summary generation and completion, sentiment analysis, and, of particular interest to our application, code generation (Chen et al., 2021).

LLMs are one of the most promising applications of *Deep Generative Models* (Oussidi and Elhassouny, 2018), that is, a category of unsupervised Deep Learning algorithms capable of capturing the probabilistic distribution that can underlie the generation of a class of data. Once estimated, it is possible to generate synthetic data compatible with such probabilistic distribution. LLMs in particular are capable of estimating the probability of a *sentence* represented as the product of each discrete symbol's probability given the symbols preceding it. For instance, given a few words as a prompt, they can easily suggest the most likely way to complete the sentence. Usually, such symbols are referred to as *tokens* and represent short sequences of characters.

[2]Web: https://openai.com/blog/gpt-3-apps

Common ways to implement LLMs are Recurrent Neural Networks (Mikolov et al., 2011), Long Short-Term Memory (LSTM) networks (Sundermeyer et al., 2012) and, most recently, Transformers (Vaswani et al., 2017), of which GPT models are the most notable example. Transformers are models based on an encoder-decoder architecture, and adopt a *self-attention* mechanism that allow them to weigh each part of the input data differently. While in the current state of affairs there are no guarantees about the long-term logical coherence of the answers given by a Transformer-based architecture, especially for long sequences of symbols, such models exhibit surprising capabilities in generating plausible and coherent lists of instructions, from cooking recipes to functioning code.

Since their popularization, most research works have tried to assess the extent of LLMs' general logical capabilities, especially in zero-shot to few-shot attempts. The seminal paper from Brown et al. Brown et al. (2020), *"Language Models are Few-Shot Learners"*, defines zero-shot learning as the case when questions implying a certain level of logic capabilities are posed directly to the model, whereas one-shot and few-shot approaches provide a very limited amount of examples to the model as part of the request. In all such cases, the model is just prompted, without further training with specific examples, a procedure now commonly referred to as *fine-tuning*.

The results obtained by Brown et al. are certainly impressive, and sparked a vast amount of research trying to assess how far reaching the zero-shot, or even few-shot, reasoning capabilities of LLMs really are. Unfortunately, in the specific case of planning, the results of the following studies were mixed. On one side, it is impressive in absolute terms that a neural network can respond in natural language to a logical question it has not been trained specifically for, and get the answer right even 40% of times, as it hints at a glimpse of intelligence beyond what we would expect from simple luck or parroting humans. On the other side, such a low accuracy makes these tools unsuitable for many real-world applications, especially in robotics, where precision and reliability is essential.

For example, the work by Valmeekam et al. (2022) proposes a benchmark for LLM-based planning and reasoning in a few-shot scenario. The authors employed PDDL, but only to automatically generate, in natural language, several logical problems never seen before by the model, and compared the performance of popular LLMs under several metrics. The best performer could solve only 25 out of 500 problems, that is, 5%. Better results were obtained by Wang et al. (2023) using few-shot learning and *grammar prompting*. In this approach, GPT-3.5 and GPT-4 are used to predict a specialized grammar given a test input, and then generate the output according to the rules of the grammar itself. Among others, the authors tested inputs in PDDL format extracted from popular but rather simple PDDL

domains. While the benefits of grammar prompting are evident, in absolute terms they obtained mixed results depending on the domain, with success rates ranging from 40% to 100%. LLM-Planner (Song et al., 2023) is another few-shot method based on GPT-3 and designed specifically for embodied reasoning, that is, techniques aimed at being used in agents dealing with real-world, physical environments. This approach uses near-natural language and is characterized by a dynamic re-planning mechanism grounded on what an agent could observe in the environment at each step. High-level planning accuracy in a variety of tasks ranges from 26% to 51%. Similar approaches are discussed also by Logeswaran et al. (2022) and Wake et al. (2023). Singh et al. (2023) proposed instead Progprompt, a method that exploits the strong performance of GPT-3 in code generation by prompting the model to generate executable Python code to solve a given problem. Silver et al. (2023) further improved on this concept by proposing a method for generalized planning in PDDL domains by also including an automated debugging mechanism.

As perfectly stated by Huang et al. (2022), the main issue with almost all these approaches is that plans produced naively by LLMs *cannot map precisely to admissible actions*, thus most of them attempt to increase their success rate in achieving the desired goal state by clever prompting approaches. Silver et al. (2022) face similar challenges while trying to solve actual PDDL problems, maintaining the PDDL format both in input and output. As an alternative solution, they propose to use the good-but-imperfect LLM-generated output to support a traditional (that is, based on heuristic search), PDDL-based planner. While this approach mitigates most of the issues described before, especially regarding mapping admissible actions precisely, and often provides planning time improvements above 10% compared to traditional planning, the ability to generate a plan action-by-action is lost. As we will discuss further in the next Chapters, at the moment this is the key advantage of Neurosymbolic planning compared to traditional planning, especially in HRC scenario. The approach in Silver et al. (2022) should then be considered more as an optimization of traditional planning rather than a paradigm shift towards Neurosymbolic planning.

4.3 Fine-tuning LLMs to approximate a search heuristic

From the literature that we have analyzed in the previous Section, it appears that, at least with currently available LLMs, it is not possible to solve task planning problems reliably, that is, with a success rate comparable to that of traditional, PDDL-based planners, with clever prompting only. We are then left with only another option that received much less attention in literature: *fine-tuning*. Such an approach involves the further training of a base model

with specific examples, namely problem-plan pairs, related to the planning domain at hand. Plansformer is an approach of this kind proposed by Pallagani et al. (2023). The authors trained five different models by fine-tuning CodeT5 (Wang et al., 2021), a LLM specialized in code generation, to solve just as many simple, well-known, PDDL-based domains such as the *Tower of Hanoi*, where Plansformer reached 97% valid plans. The fine-tuning dataset has been generated by solving 18000 randomly generated problems with the Fast-Downward planner (Helmert, 2006), and then translating both problems and plans into a more compact form. PDDL-based problems are augmented by listing the actions in the domain to explicitly teach the model about their preconditions and effects.

A possible interpretation of these results is that there are obvious similarities between the capability of LLMs to learn the probability of a symbol following the preceding ones, and the search heuristics that power traditional PDDL-based planners. Hence, we are not just extracting knowledge from the model and exploiting its general reasoning capabilities, but rather we are *training the model to approximate the search heuristic of the planner generating the dataset*.

Teriyaki, that was developed in parallel with Plansformer, takes a similar approach with a few key differences:

- We assume that LLMs are capable of handling unaltered PDDL-based problem and plan descriptions, which in turn makes the model easier to use in software architectures for robots;
- With the aim of reducing the length of the prompt, which is one of the main LLMs' constraints, we instead exclude some information, on the assumption that the model can learn it and, for example, no explicit knowledge of the action preconditions and effects is given to Teriyaki during fine-tuning;
- It is trained on a dataset of non-optimal plans, which allowed us to observe that LLMs are capable of outperforming the traditional planner they are trained upon in terms of average plan length in selected domains;
- We propose to use Teriyaki to *stream* plans as they are generated, that is, action-by-action, to enable simultaneous planning and execution in robot tasks, specifically in human-robot collaboration.

Chapter 5

The Teriyaki framework: PDDL planning using LLM

5.1 An introduction to Teriyaki

In Chapter 4, we have briefly introduced Teriyaki, a framework to generate Neurosymbolic planners based on LLMs, designed to overcome the biggest shortcomings of traditional planners in robotic architectures that require frequent re-planning, especially in HRC scenarios. In this Chapter we will describe in detail the design of Teriyaki and the resulting training process.

Teriyaki, being based on Neural Networks, requires to be trained on problem instances belonging to the planning domain which it will later be called to solve. Referring back to Chapters 2 and 3, we have first trained, and later evaluated and tested the model, on two planning domains for the collaborative manipulation of articulated objects by human operators and robots. A challenging task that has been widely discussed in literature (Bertolucci et al., 2021, 2019; Capitanelli et al., 2018). The planning domains selected for this evaluation employs an *absolute formulation*, as defined in Section 2.5.5, because of the challenges it poses on the planning process:

- For symbolic task planners, because the manipulation of an articulated object based on said formulation scales very poorly with the number of its links, joints, and the allowed angle configurations, thus constituting an ideal benchmark to verify potential scalability benefits of the proposed alternative method;
- For LLMs, since the domains include *conditional effects*, that is, an action could imply modifying the state of the representation in ways that are not immediately

apparent from the arguments of the action itself, for example, if a link were rotated around a joint, all downward links in the chain should be also *implicitly* rotated with respect to an absolute reference frame. We hypothesize that conditional effect will significantly challenge the capability of the LLM to maintain the long-term coherence of its output. Demonstrating that LLM could not only generate PDDL plans, but also support conditional planning, is by itself a contribution to the state-of-the-art.

The training process, which is detailed in the following Sections of this Chapter, has been performed on a large data set of 9000 *problem-plan* pairs generated automatically and solved by an existing, state-of-the-art, traditional, symbolic, PDDL-based planner, namely Probe (Lipovetzky and Geffner, 2011b). The resulting models have been rigorously tested on 1000 pairs not previously used for training, and evaluated in terms of planning accuracy, plan length, and planning times. The details of the evaluation process are presented in Chapter 6. During training, data related to validation and planning accuracy, defined as the percentage of plans that can be formally proved to be correct, have been collected to investigate their evolution with the growing number of training samples and to assess whether transfer learning between similar planning domains is possible. Results show near identical planning accuracy between Probe and the solvers generated with Teriyaki. However, Teriyaki outperforms Probe in terms of shorter plan length by up to 13.5% in one of the domains. Regarding planning times, it is noteworthy that an objective comparison is not possible due to the different computing architectures on which the two planners run, that is, Teriyaki is based on a proprietary cloud-based service, namely GPT-3, whose response time was not guaranteed nor predictable during test sessions, whereas Probe can run on a fully accessible and controlled workstation. In fact, in the current experimental architecture, Probe remains faster in generating a complete plan, that is, in a situation where we require Teriyaki to synthesize a full PDDL-compatible plan end-to-end. However, when the planner as a module is integrated into a robot architecture, the use of Probe forces us to adopt a pure sense-plan-act approach, that is, the plan must be synthesized in its entirety before its first action can be executed, whereas Teriyaki generates and makes available for execution the first action as it is produced. In our experiments, this reduces the overall waiting time by 61.4% on average, as it exploits simultaneous planning and execution. Finally, we also experimentally observe that Teriyaki solvers scale with the input and output length rather than with the problem and domain complexity, hinting that there might exist more complex domains where Teriyaki could outperform Probe, an in general a traditional symbolic planner.

5.2 Planning Domains

Figure 5.1 shows a Baxter robot executing a plan from the two domains, and a possible interaction with a human. In our work, and with the aim of challenging both the chosen, traditional, PDDL-based planner as well as Teriyaki, we have selected two PDDL domains, which we refer to from now on as MACRO and NO-MACRO. These domains correspond respectively to the `m_25_B` and `m_0` domains outlined in Section 3.1 and provided in Appendix A. Both domains employs the *absolute formulation* defined in Section 2.5.5, that is, actions can have implicit effects on joint angles not directly affected by the outcomes of the action itself. If managed by a traditional, symbolic action planner, this requires propagating the (conditional) effects of each action to modify the value of state variables not directly affected by the effects. Likewise, we argue that in the case of LLMs like GPT-3, conditional effects could stress the model because it should be harder to maintain coherence in the plans given the generative process.

Regarding the MACRO domain, this model makes available to the planners some additional macro actions. As we have discussed in Chapter 3, Macros are ordered, compound actions that bundle together a number of basic actions, for example, a `grasp-rotate-release` action instead of three *atomic* ones, that is `grasp`, `rotate`, and `release`. Their use is an effective way to reduce the planning time in traditional action planners at the cost of accepting *less optimal* plans. In fact, the use of macros could lead a planner to enforce the use of an actions sequence, that is, a given macro, at the cost of introducing possibly spurious actions. For example, if the goal state assumed the articulated object to be in a given configuration, but with the robot gripper still maintaining its hold on the object, the planner could prefer using a macro `grasp-rotate-release` followed by another `grasp` action (partially compensating the effects of the macro) instead of two atomic actions `grasp` and `rotate`. Macros are also supposed to facilitate the generative process of GPT-3 since the use of macros is expected to shorten the resulting plan significantly.

More specifically, the reason why we chose exactly this domain for comparison is that it was the top performer in the comparison presented in Section 3.5, and yet it presents only minimal differences with the NO-MACRO domain, limited exclusively to its action set. Most interestingly, the MACRO domain leads to plans which are, on average, about half the length of those produced by the NO-MACRO domain. Since the two domains are fundamentally similar but lead to plans of different lengths, they are ideal candidates to test how Teriyaki-based solvers scale with the cumulative input and output length.

Figure 5.1 A Baxter robot executing actions in two domains involving the manipulation of an articulated object. A human can act on an articulated object's joint at any time, forcing the robot to re-plan.

5.3 PDDL version and planner

As both domains taken into consideration here used conditional effects, we used PDDL 2.1 and a compatible PDDL-based planner, namely Probe (Lipovetzky and Geffner, 2011b), one of the planners used in (Capitanelli et al., 2018) and that we already discussed in Chapter 2. It must be noted that the choice of Probe has an impact on the quality of the results, and in

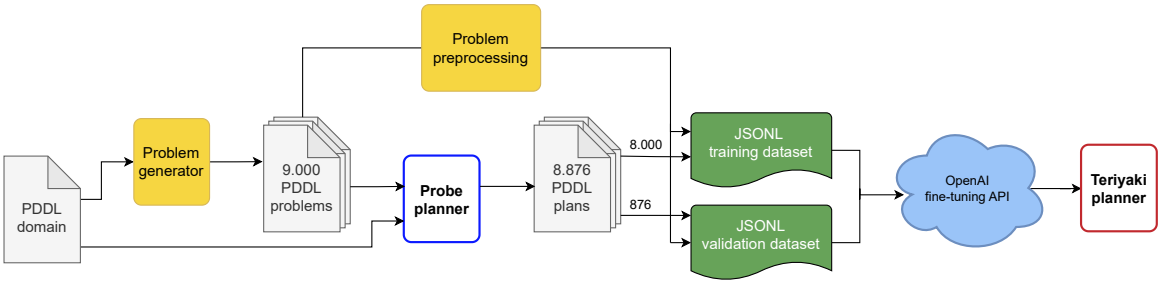

Figure 5.2 A diagram of Teriyaki fine-tuning process. Blocks in yellow represent custom code developed for data generation and processing as described in section 5.4.

the future, an optimal planner might be used instead, like in the Plansformer (Pallagani et al., 2023).

5.4 Choice of the LLM: GPT-3

As we anticipated, in this work we leverage GPT-3, a closed-source model and commercial product that at the time of this writing can only be accessed as a cloud-based service through OpenAI's API. Our results are not limited to the specific features of GPT-3 and could in theory be replicated with any available LLM. In practice, since our first experiments with GPT-3, we have tested several smaller models that can be run locally, such as LLama 2 7B, 13B, and 30B, and none of them could reliably solve problems in the two selected domains. We will discuss this subject further in Chapter 7. Forced to employ a commercial model as a service, we decided to opt for GPT-3 due to its size and speed compared to its more recent versions.

5.5 Dataset generation and composition

In the case of GPT-3, the training can be performed by providing a structured file where each line is a *prompt-completion* pair. We assumed that the GPT-3 model can implicitly learn by example the rules usually defined in a PDDL domain file, such as the allowed (modeled) actions, and their respective preconditions and effects. Hence, we conducted training sessions using only *problem-plan* pairs, whereas we use the problem as a prompt and the plan as completion. We remark here that only part of the PDDL problem is used as a prompt. As we have discussed in Chapter 2 and can be observed in Figure 2.7, many of the predicates are actually *static*, that is, they are used to encode general properties of the problem, such as the

relative order of joint angle configurations, an example being: a joint at 45 *deg* can be rotated clockwise to 60 *deg*, and counter-clockwise to 30 *deg* in 15 *deg* increments. These predicates remain the same in all problems and are never modified by action effects. Therefore, we assume that the model can learn them from the implicit information encoded in the plans, in the same way as the domain. We thus removed them from the prompt to reduce its length, as prompt length has an impact on response time and the total prompt plus completion length of each query cannot exceed 2048 tokens, or circa 8000 characters, for the selected model.

It must be noted that such optimization in terms of prompt length, is not only beneficial to performance, as planning times depends on the cumulative input and output length, but it is necessary as the NO-MACRO domain average total input and output length barely fits in the maximum length allowed by the model. Newer models such as GPT-3.5 Turbo and GPT-4 allows much longer sequences of tokens to be processed and might not require such processing, but nevertheless may benefit from such optimization when possible.

At the time the experiments were performed, OpenAI's documentation suggested using the largest and most powerful version of the model, called `davinci`, for a conditional generation, that is, generating original text based on input. For this use case, the documentation recommends training the model using at least 500 training samples and aiming for 8000. One of the advantages of Teriyaki is that we can easily generate large training datasets using randomly generated problems and a PDDL-based action planner. Therefore, using the same custom generator introduced in Section 3.4.1, we generated 9000 problems, out of which 8000 were reserved for training and 1000 for validation. It must be noted that out of the 9000 planning attempts performed by the planner 124 failed, so the validation set has been reduced to 876 samples. Finally, we added another 1000 unique problem-plan pairs as a test set, in this case, we ensured that all the chosen samples could be solved by Probe.

The next step requires validating the completeness of the plans generated by Probe to ensure that we train our model only on correct data. To do so we use the widely known VAL validation suite (Howey et al., 2004), which requires plans to be formatted in the International Planning Competition (IPC) standard[1]. As Probe does not return plans compliant with this standard, we developed custom code to reformat the plan files to be compatible with VAL. This is also necessary to ensure that Teriyaki will be trained on IPC-compatible standard plans, and will thus later reply to our queries in the same way, allowing us to easily benchmark its planning accuracy. Running VAL over the dataset resulted in all plans passing the test, meaning that they reached the desired goal state, even though it must be reminded that Probe

[1]Web: https://ipc2023.github.io/

previously failed to generate a plan *at all* for 124/9000 problems, that is, about 1.37% of the total.

Finally, we compiled the training and validation datasets in `jsonl` format. Each line of a dataset file is composed of a *prompt*, that is, the problem, and a *completion*, that is, the plan. We added to the end of each prompt and completion the respective termination sequences, namely `\n\n###\n\n` and `END`. As the name suggests, the termination sequences signal to the network the end of the message. This is especially important for the completion as the termination sequence can then be used as a stopping condition when querying the model. A sample line is provided in Listing 5.1, edited for the sake of brevity and clarity. The whole data generation and dataset preparation process is outlined in Figure 5.2.

```
{"prompt":
"(:init (angle_joint angle315 joint1)
(angle_joint angle300 joint2)
(angle_joint angle285 joint3)
(in-centre joint2)
(free gleft) (free gright))
(:goal (and
(angle_joint angle0 joint1)
(angle_joint angle300 joint2)
(angle_joint angle285 joint3)))
\n\n###\n\n",
"completion":
" 0.00100: (link-to-central-grasp ...)
0.00300: (increase_angle_first_child_45 ...)
0.00500: (release-links ...)
0.00700: (link-to-central-grasp ...)
0.00900: (decrease_angle_first_child_45 ...)
END"}
```

Listing 5.1 An example of training sample.

5.6 Training

As far as the training process is concerned, we decided to run a complete 8000-sample training on the MACRO domain, since plans in its dataset are shorter on average, and therefore *supposedly* easier to be learned by GPT-3. We call this model Teriyaki-MACRO.

When we train a LLM to perform action planning, the cost function rewards linguistic coherence rather than planning accuracy. *This means that we are hypothesizing linguistic coherence as a proxy for logical coherence*, and therefore we must assume that validation accuracy during the training process differs from the planning accuracy of the resulting model, whereas the planning accuracy is defined as the percentage of plans that are

- Formally correct;
- Able to reach the desired goal state, or a state compatible with it.

By *"formally correct"* we mean that each action's arguments are consistent both with the specifications given in the planning domain and the state of the modeled reality when that action is carried out, that is, the action preconditions are not violated. To be able to measure how planning accuracy increases with the size of the provided training set, we decided to perform the training process in steps. At each step, we provided samples to double the total size of the training set. Starting from the minimum amount of 500 samples, we then trained the system with 1000, 2000, 4000, and 8000 total samples, and saved a snapshot of the trained model at each step.

As anticipated above, the base model chosen for the training process is GPT-3 and specifically `text-davinci-002`. Regarding the hyper-parameters, the number of training epochs was set to 2, while the batch size and the learning rate multiplier were left to their default values. The batch size defaults to 0.2% of the number of examples in the training set, while the default learning rate is automatically determined in a range from 0.05 to 0.2 depending on the final batch size. We also highlight that since the model was effectively fine-tuned five times, the learning rate was reset to the default value at the beginning of each session, with an effect similar to gradient descent with warm restarts (Loshchilov and Hutter, 2016). We did provide the optional validation set and enabled the `compute_classification_metrics` options to obtain training statistics for further analysis. The total training cost for this procedure on a single planning domain at the time of training (01/12/2022) was around 250$.

5.7 Transfer Learning

For the NO-MACRO domain, we used the same methodology, but we decided to generate two candidate models. The first model is trained starting from the `text-davinci-002` model as before, whereas the second is trained starting from the MACRO model obtained previously. We call these models Teriyaki-NO-MACRO (davinci) and Teriyaki-NO-MACRO (MACRO). The hypothesis is that as domains share many concepts and the MACRO model has already been primed for them, the second candidate should reach a higher planning accuracy with a smaller amount of training samples.

Results shown in Section 6.2 confirmed this hypothesis, so we interrupted training when Teriyaki-NO-MACRO (MACRO) reached comparable results to its parent model and discarded entirely Teriyaki-NO-MACRO (davinci). Regarding the training dataset, the only difference with the example provided in Listing 5.1 is that the prompt part is preceded by the `\n--NO-MACRO` tag. This tag was introduced to test whether the model could be used to solve problems for both the MACRO and the NO-MACRO domains, by simply adding the tag in our queries to the system. Unfortunately, the NO-MACRO models loses the ability to solve problems in the MACRO domain, suggesting that to generate models that can solve multiple domains, *training should be performed including examples from all the domains of interest*.

5.8 Invoking the LLM and Plan Streaming

After the training phase, it is possible to query the model through an API call by providing as a prompt the PDDL predicates describing the initial and goal states for the robot and the articulated object, as one would do with a traditional, symbolic PDDL-compatible action planner. Several parameters can be configured, and below is a list of those that can impact the overall quality of the resulting plan.

- `temperature` is the most important as it controls how *deterministic* the reply (and therefore the plan) will be. Keeping it at 0 corresponds to an `argmax` operation on the next token probability and ensures that we always get the most likely result. While in a regular text generation some level of *creativity* is desirable, we strongly recommend keeping it at 0 for plan generation, especially when robot actions are planned in the context of a human-robot collaboration scenario.

- `presence_penalty` and `frequency_penalty` can assume values between -2 and 2, and can be used to reward or penalize repetitions in the generated text. We have

observed that setting the former to 2 seems to improve planning accuracy by $1-2\%$ and *vice versa*, but at the moment we have not investigated enough this effect, so we decided to set the default value to 0 for both in our tests.

- `stop` can be used to set a string as a stopping condition, meaning that when that string is generated by the model, the generation immediately terminates. Coherently with our training dataset, we set this value to `END`.

- `max_tokens` controls the maximum response length, thus we recommend setting it as high as possible to minimize the chance of a plan being truncated. Since each model has a maximum total prompt/completion length, and the worst-case prompt length depends on the planning domain at hand, this value should be assessed case by case. In our case, 1900 appears to be the highest value for robust operation.

To stream the plan we set the parameter `stream` to `true`. In this way, the model starts returning tokens as soon as they are available, instead of waiting for the full sequence of tokens to be generated. This has no impact on the time needed to receive the full answer, but it does reduce significantly the time needed to receive the first token.

5.9 Why streaming? Interleaved planning and execution

A careful reader might argue that it is also possible to act on the stopping condition in order to get only the first action of a plan. That would be possible by setting `stop` to `\n` instead of `END`. In this way it would be possible to get an action to execute, observe the results (possibly including also external perturbations as it happen in the `planHRC` architecture described in Section 2.5), and then submit a new query to the LLM. We call this process *Interleaved Planning and Execution* (IPE).

The objection is valid and we partially explored it, but we would recommend streaming as superior alternative in general. The obvious inefficiency introduced by IPE is that at each query we are asking the LLM to process again our prompt, whereas with streaming the prompt is processed only once. As we have seen the length of the prompt has an impact on planning time performance, and that is without considering other unforeseeable factor such as queuing time to the cloud service that provides the LLM, which also compounds with each call. Moreover, processing the prompt multiple times does not only have a negative impact on performance, but also on cost, as the pricing structure of cloud LLMs is usually *pay per use* computed on the number of processed tokens. In this regard, an architecture that

supports streaming is strictly better, as we are always free to monitor execution and, in case a re-planning is needed, close the streaming and submit an updated request. With IPE we are forced instead to perform this operation for every action, regardless of whether the plan is unfolding as expected or not.

The less obvious reason to avoid IPE, is that in the naïve implementation proposed here, it is susceptible to the *Sussman Anomaly* (Sussman, 1973). To understand why Plan Streaming is not affected by this issue, but IPE is, we provide the following possible interpretation. The transformer architecture on which most modern LLM are built is a recurrent network. At each iteration, the LLM guesses the next token based on the list of previous tokens, not just in the initial prompt, but also those previously generated. As such, the LLM has knowledge of previous actions and, being trained on plans that are not affected by the anomaly, it has learned how to correctly sequence actions avoiding getting stuck in a loop. When we use IPE, the knowledge of previous actions is lost, hence the LLM just picks the most probable next action based on the current state. This often results in loops where the robot infinitely re-centers the objects back and forth between two link.

Considering that IPE is anyways less efficient, and that trying to work around the Anomaly would make this approach even more complex, we have not tried to propose a solution to this problem yet. Nevertheless, we decided to include this note about IPE for two reasons:

- The first is the fact that the Anomaly appears in IPE might be trivial, but the fact that it does not appear when we ask the LLM to generate a full plan is not. We should appreciate that the LLM has learned from the heuristic search planner to converge towards a given goal and that, for it to happen, the context of previous actions is fundamental.

- There might be cases where IPE could be useful in the future, for example with smaller models which might lose logical coherence after a few actions in complex planning domains. In this case, assuming one could work around the Anomaly, IPE could be used to enforce long-term coherence at an architectural level.

Chapter 6

Teriyaki in action: Parallel Planning and Execution

6.1 Relation between Token and Planning Accuracy

In Section 5.6, we pointed out that we use linguistic coherence as a proxy for logical coherence. Figure 6.1 compares the evolution of the validation token accuracy and the planning accuracy for the MACRO model, against the number of examples used to train the model itself. On the one hand, the validation token accuracy measures the accuracy at predicting the next token, that is, approximately the next 4 characters of the response, in our case coinciding with the plan, on the validation set. On the other hand, we define validation planning accuracy as the percentage of plans in the 876-sample validation set that are both formally correct and reach the desired goal state, that is, which one passes the VAL validation utility test. For the former, data are retrieved from the classification metrics reported by GPT-3 itself after training, where this information is available every 8 training steps. For the latter, we used the snapshots of the model taken after training with 500, 1000, 2000, 4000, and 8000 examples. Such snapshots are used to plan against all 876 problems in the validation set in the conditions described in Section 5.8. VAL is run on the obtained plans and finally, the validation planning accuracy can be computed. In Figure 6.1, the evolution of planning accuracy is represented by the orange bars to highlight the fact that the value is measured at each snapshot. It must also be noted that the parameter `elapsed_examples` does not correspond to the number of unique examples in the training set, but it is scaled by a factor of 2 because we used two epochs for training, thus each example was used twice.

In Figure 6.1 the orange bars report the validation accuracy of the Teriyaki-MACRO model. It can be observed that GPT-3 reaches a very high validation token accuracy after the

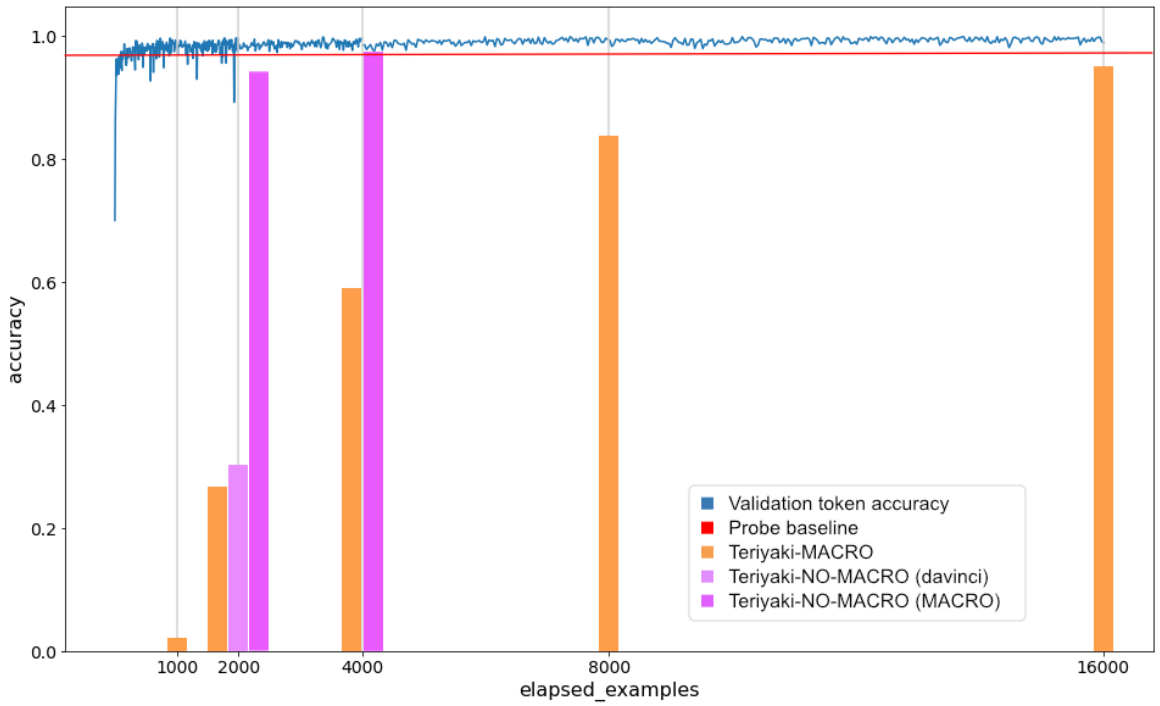

Figure 6.1 Evolution of the token validation accuracy and the planning accuracy as the number of training examples increases. The blue line represents the evolution of the validation token accuracy during learning as reported by GPT-3 classification metrics. The bars represent the planning accuracy of the Teriyaki-MACRO, Teriyaki-NO-MACRO (davinci) and Teriyaki-NO-MACRO (MACRO) models. The red line represents our baseline, that is, the percentage of plans solved by Probe.

first 500 samples, as expected by a model well-known for its few-shot learning capabilities. Nevertheless, planning accuracy rises at a much slower pace. Like in the few-shot methods discussed in Section 4.2, the LLM cannot yet map precisely to admissible actions and even a single mistaken token could break a plan that would be otherwise *linguistically* coherent. A very common mistake in plans generated by the model trained with only 500 samples is that the model ignored the conditional effects of actions. As we correctly hypothesized, conditional effects can be quite problematic as they imply propagating their effects downstream in the plan to keep the overall semantic coherence of the plan. In this case, actions are correct and reasonably parameterized but they do not meet the necessary preconditions as they ignore that a given joint is in a state different from the one expected by the model, due to the *indirect* effects of a previously planned action. Eventually, the model reaches a very high 95% planning accuracy on the validation set after training over 8000

unique samples. Despite this, the model does not seem to overfit to the problems used for training, as shown by the results on the test set presented in Section 6.3.

Results of validation planning accuracy are strikingly similar to those obtained by the Plansformer, yet they have been achieved with a training set of 8000 instead of 18000, and 2 training epochs instead of 3. This could point to GPT-3 having stronger performance in this task than CodeT5, but the phenomenon requires further investigation to be clearly ascertained.

During this experiment, we also measured the number of planning attempts that failed because of their excessive length. While this number was always small compared with the validation set size, the number of failures decreased from 24 in the first training step, down to 1 in the final model, further suggesting that the model becomes better at generating shorter plans and avoiding unnecessary actions.

6.2 Transfer Learning

As previously discussed in Section 5.7, the Teriyaki solver for the NO-MACRO planning domain has been chosen starting from two base candidates, namely Teriyaki-NO-MACRO (davinci) and Teriyaki-NO-MACRO (MACRO). As we trained the two models, we kept track of the planning accuracy as described in Section 5.7. The light pink and dark pink bars in Figure 6.1 report results in planning accuracy for the Teriyaki-NO-MACRO (davinci) and Teriyaki-NO-MACRO (MACRO) models, respectively. After 1000 samples, the Teriyaki-NO-MACRO (davinci) model reached a planning accuracy of 32.5%, while Teriyaki-NO-MACRO (MACRO) reached 95.2%. Because of this result, we immediately dropped the former model, while we further trained the latter using 2000 samples. At this stage the Teriyaki-NO-MACRO model reached a validation planning accuracy of 98.8%, exceeding the 95% validation planning accuracy achieved by the MACRO model after 8000 samples. Because of this result, we decided not to proceed with further training.

It is noteworthy that the Teriyaki-NO-MACRO (davinci) model still reached a higher validation planning accuracy at 1000 samples than Teriyaki-MACRO. This result might suggest that against our initial assumption, the NO-MACRO model is easier to learn for GPT-3 than the MACRO one. The phenomenon must be explored more in-depth, but it could be related to the number and quality of the actions in the planning domain. From these considerations, from now on when we simply refer to Teriyaki-NO-MACRO we actually mean Teriyaki-NO-MACRO (MACRO).

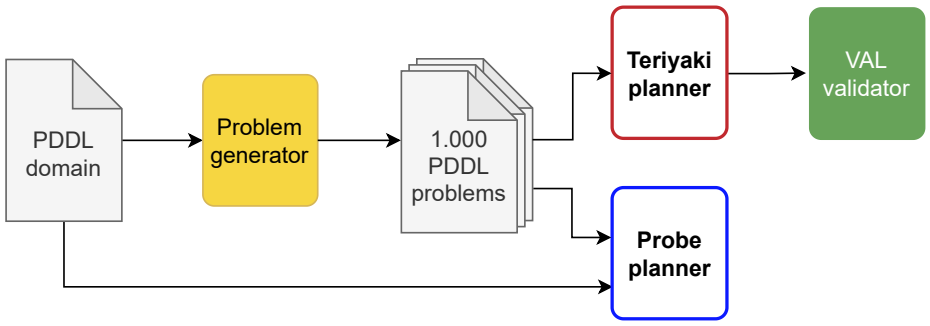

Figure 6.2 A diagram of Teriyaki testing process as described in section 6.3.

6.3 Comparison of Solvers

We tested the performance of both Teriyaki-MACRO and Teriyaki-NO-MACRO models in terms of planning accuracy, plan length, and planning times on a test set of 1000 problem-plan pairs not previously used for training and validation, and we compared the results to the performance of the traditional action planner Probe. The process is summarized in Figure 6.2.

Before proceeding, we want to remark that, due to the significantly different computing workflows used to run Teriyaki and Probe (basically, a cloud-based architecture accessed via proprietary APIs and software installed on a local machine, respectivelly), a fair comparison between them in terms of planning time is not possible. Therefore, results about planning times in this Section are only meant at providing a baseline for future work and to show how planning times scale differently for each Teriyaki planner in the two different planning domains taken into consideration in this work.

Probe ran on an Ubuntu 20.04.4 Windows Linux Subsystem (WSL) environment, deployed on a machine equipped with an 8-core Ryzen 3700X 3600 Mhz CPU and 16GB@3600 Mhz RAM. We recorded planning times as reported by the planner itself together with each generated plan. As far as planning accuracy is concerned, all plans generated were valid, but we considered the instances in which the planner failed to generate a plan as failures.

Regarding Teriyaki models, we prompted them using the settings described in Section 5.8, then verified the validity of the obtained plans using the same VAL validation tool employed at the data set generation phase. As the OpenAI API does not provide a method to log the exact execution time of a call, planning times of Teriyaki solvers have been measured by recording the time it took each API call to return a response to the client application. For this reason, it must be noted that it is impossible to discern how long each call has been queued by the cloud application before execution. We assessed that the effect of queuing is

Table 6.1 Summary of the tested models and date of testing

Model	Base model	Test date	Start	Finish
Teriyaki-MACRO	`davinci`	18/02/2023	11:17:56	13:47:41
Teriyaki-NO-MACRO	MACRO	19/02/2023	17:22:24	21:25:58

Table 6.2 Comparison of Teriyaki models against Probe in their respective domains on the test dataset

Solver	Domain	Acc.	steps	t_max	t_avg	t_std
Teriyaki-MACRO	MACRO	95.5	10.953	54.32	8.99	4.77
Probe	MACRO	98.6	11.111	36.71	2.11	3.47
Teriyaki-NO-MACRO	NO-MACRO	94.0	19.158	54.71	14.61	7.16
Probe	NO-MACRO	98.6	22.137	43.77	2.79	3.30

not negligible as running tests after the official release of ChatGPT, another popular GPT-3 derived product by OpenAI, led to longer planning times than previously recorded, possibly due to the increased traffic to the servers. In order to partly mitigate this effect, the tests presented here were performed during the weekend, preferably in the morning CET time. In Table 6.1 we also include the date and the starting and finishing time of each test session for reference.

For all solvers and models, plan length has been computed simply as the number of actions in the plan.

Table 6.2 compares the Teriyaki-MACRO and Teriyaki-NO-MACRO models against Probe in the respective domains, in terms of accuracy (in percentage), average plan length, maximum and average planning times, as well as the standard deviation of the planning times (in seconds).

As anticipated, Probe is faster than the trained Teriyaki solvers to generate plans end-to-end, in a traditional sense-plan-act workflow. Yet, Teriyaki solvers still offer decent performance. Despite being trained on plans generated by Probe, Teriyaki models are capable of solving problems that Probe failed to process, and even generate *shorter* plans. The difference in plan length is only 1.5% for the Teriyaki-MACRO, but it raises to 13.5% for Teriyaki-NO-MACRO, which in general leads to plans almost twice as long. This seems to suggest that the training procedure rewards shorter completions and that the effect might

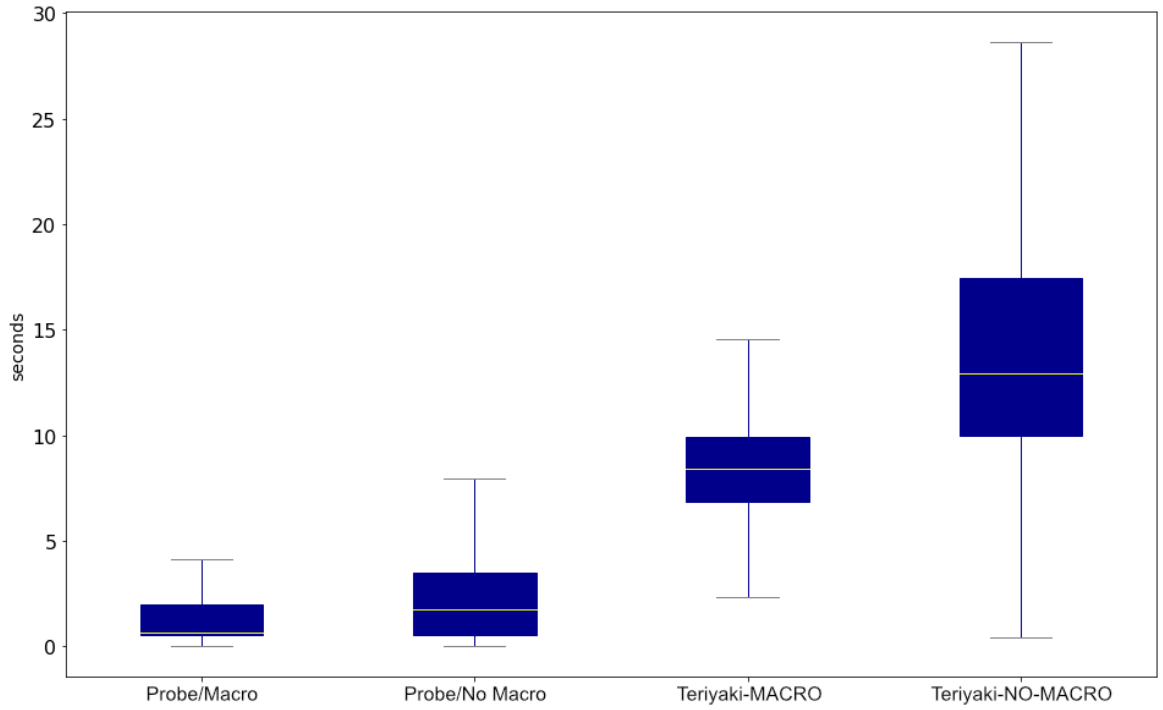

Figure 6.3 Comparison of the Teriyaki MACRO and NO-MACRO models' planning times against Probe in their respective planning domains.

be stronger the longer the supposed completion gets. Nevertheless, this phenomenon requires a more systematic investigation.

Figure 6.3 allows for a better assessment of the timing performance of the proposed models, and it compares them to those generated by Probe. It can be observed that Teriyaki models do actually scale with the combined length of the input and the output, as hypothesized. Planning times of the Teriyaki-NO-MACRO model are approximately twice of the Teriyaki-MACRO model, as expected considering that the plans of the former are approximately twice longer than those of the latter. Box plots in the Figure associated with Teriyaki models have a very distinct shape when compared to those of Probe on both domains, which hints at an almost Gaussian distribution of planning times. This is coherent with the fact that the plans, which are generated from randomly initialized problems, can assume any length.

A note on Probe results

A careful reader will notice that the Probe's timing results reported in Section 6.3 and in Section 3.5 differs significantly despite being carried out on the same hardware and on the same models (MACRO/`m_25_B` and NO-MACRO/`m_0`). In particular, average plan length

and planning time increased by about 6,5% and 76,5% respectively, albeit median planning time remained similar.

The cause of these apparently discording results is the composition of the respective test datasets on which the timing performance have been computed. As outlined in Section 3.4, the results in Section 3.5 have been obtained from a test dataset of 100 problem instances that mimicked realistic conditions in the cable routing scenario, i.e., (i) in all problem instances the object was initialized straight; (ii) joint angle configurations could range from 270 to 360 degrees; and, (iii) each joint had to be rotated by at least 15 degrees to reach the desired goal configuration. In other words, there were no *trivial* problem instances that could be solved by just one or two actions. On the contrary, training, validation and test dataset for Teriyaki have been generated randomly and without said limitations, thus they include many problem leading to extremely short plans. This was obviously intended, as Teriyaki-generated planner should not overfit to a specific subclass of problems and ensure reliable operations in all possible problem instances. This is particularly important as we plan to integrate Neurosymbolic planners in architectures like `planHRC`, where the planner might be invoked at any time, even to solve trivial problems, in order to respond to unforeseen events or human interactions.

6.4 Action-by-action Plan Streaming

One of the major strengths of neurosymbolic action planning using LLMs is that the plan can be streamed as it is being generated. If properly leveraged, this feature can support action-by-action generation instead of adopting an end-to-end approach. In many applications involving the use of robots with a required flexible behavior, and especially when frequent re-planning is expected due to changing conditions, simultaneous planning and execution by means of concurrent (sets of) processes could greatly reduce the waiting time for a plan to be available, and therefore executed. As we have previously pointed out, this may nicely correlate with a perceived interaction fluency in human-robot collaboration.

To test the performance of Teriyaki in this regard, we set the Teriyaki-MACRO model to plan against our test dataset until it generated 1000 actions. Figure 6.4 compares Teriyaki-MACRO single action timings against Probe timings to generate 1000 whole plans. Not only do we observe that wait times are reduced by 61.2% on average, but also the response time standard deviation is reduced from 3.47 to just 0.15. This corresponds to a 95.6% decrease and makes Teriyaki much more predictable than the traditional heuristic-search solver. The result of the variance could be extremely important for applications that require a

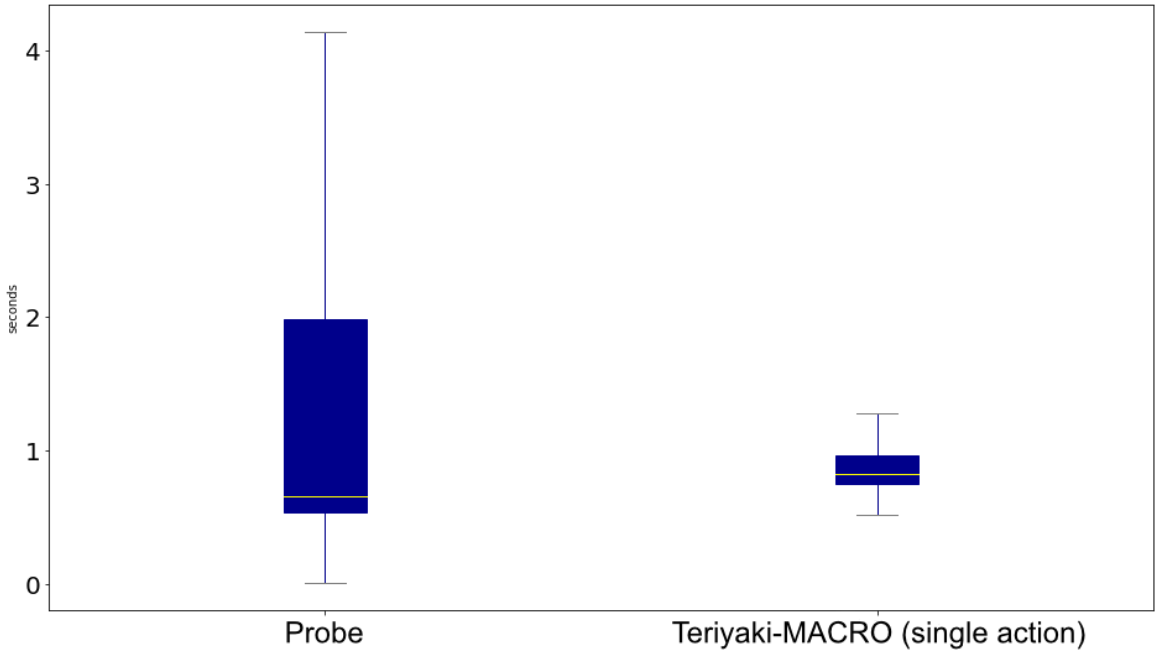

Figure 6.4 Comparison of the time it takes Teriyaki-MACRO to generate a single action against Probe on the MACRO domain.

decision to be taken withing a predetermined time frame. It also supports nicely our claim that Teriyaki solvers are ideal for HRC applications, as it should make the robotic agent much more predictable and consistent in the eyes of the operator.

6.5 Limitations

We previously pointed out that, although our results are quite promising in so far as they unlock the possibility of simultaneous planning and execution, the data-driven synthesis of a solver cannot be considered a complete alternative to traditional action planning at this stage. As a matter of fact, a symbolic action planner is needed to obtain the problem-pair data used for training the models. Herewith, we mention three main current limitations.

1. The first drawback in the current implementation is that planning times are slower than the baseline symbolic planner Probe, *at least* using GPT-3 as base LLM to implement the overall architecture.

2. The second is related to the need to consider planning domains *small enough* to fit into the 2048 tokens context characterizing GPT-3.

3. The third concerns advanced capabilities exhibited by some PDDL-compatible planners.While we included conditional planning among the features supported by Teriyaki, other relevant capabilities such as numerical and temporal planning have not been considered. Their inclusion could prove challenging due to difficulties of LLMs in generating mathematically accurate results.

In spite of these limitations, it is noteworthy that there are a number of LLMs and Transformer-based learning architectures that are being made available at the writing of this Thesis. In view of the favorable performance scaling discussed in this Chapter, we consider LLM-based neurosymbolic planning as a promising approach worth considering for future investigation.

6.6 Results summary

In this Chapter we introduced Teriyaki, a framework to generate neurosymbolic, PDDL-compliant planners based on GPT-3, that is, a Transformer-based Large Language Model architecture recently made available worldwide. In its current implementation, Teriyaki relies on a pragmatic and inexpensive procedure for the generation of a training dataset, which is based on the use of an existing action planner able to generate plans on the basis of randomly defined inputs. Training can leverage high-performance computing machinery in the cloud, and the resulting model can be deployed to any software architecture for robots adopting standard PDDL-compatible syntax and interfaces.

The major contribution of this work is the empirical evidence that this approach can be a viable solution for action planning. In particular, we showed that: (i) planning accuracy is on par with that of a traditional, state-of-the-art, PDDL-compliant planner, (ii) the average plan length is up to 13.5% shorter, (iii) it is possible to better use Teriyaki in scenarios where robots must re-plan frequently, and in particular due to its ability to generate plans action-by-action, thus reducing average waiting time for a plan by 61.2%, with standard deviation by 95.6%.

Overall, these results hints at a scenario in which the approach can fairly scale up in terms of the number of potentially supported planning domains, even though this remains one of the points to further investigate, and where fluency in human-robot interactions can greatly benefit of the increased reaction times of the robot action planning capabilities.

Chapter 7

Towards Neurosymbolic planning using local LLM

7.1 The benefit and challanges of local LLMs

In the previous three Chapters we have described Teriyaki as a framework to generate neurosymbolic planners by fine-tuning GPT-3. While GPT-3 has been an amazing tool to validate the feasibility of our proposed method, and its performance has proven competitive with similar approaches in literature, it does suffer from a number of practical limitations:

1. **Only in the cloud.** As discussed in Section 6.3, the cloud nature of GPT-3 posed significant challenges when we tried to benchmark the planning times of Teriyaki solvers. Such issues could be even worse in all real-world applications that might require the service to always be available and/or respect specific timing requirements.

2. **Platform control.** Being a cloud-exclusive service, users have no control over the GPT-3 lifecycle. Indeed, the version of GPT-3 used as a base model for Teriyaki will be retired by OpenAI in Q1 2024. This will make our models inaccessible and force us to run a new fine-tuning on a different model which might also be retired later on.

3. **Cost.** While not expensive in absolute terms, the *pay per token* pricing policy of OpenAI might limit the economic sustainability of certain projects, both in academia and in the industry, and in particular its usage by students.

4. **Closed source.** GPT-3 is a closed source model and as such it is incompatible with any future research on the topic that might benefit from modifications to the network architecture.

For all these reasons, it would be highly beneficial to any future work on the subject, to first replicate the Teriyaki results on a *local* LLM. In this way, future research effort could benefit from complete control over the platform and unlimited availability, and new exciting research scenarios could open, where, for example, the LLM can run locally alongside other components of the control architecture.

In recent years, a number of local LLM have been made available with different usage licenses to the public, such as the aforementioned LLama 2 (Touvron et al., 2023) and a plethora of derivative models fine-tuned for specific use cases. Compared to OpenAI's models like GPT-3, local LLM are usually smaller, in the sense that their network is characterized by a significantly reduced number of parameters in order to fit on consumer or professional workstation hardware. This limitation poses another interesting scientific question: *do we really need an LLM as large as GPT-3 to apply a framework like Teriyaki, or working neurosymbolic planners can also be generated by fine-tuning smaller models?*

In Section 6.1, we highlighted that, when training an LLM to solve a logical problem, we are using linguistic coherence as a proxy for logical coherence. Undoubtedly, a LLM as large as GPT-3 needs billions of parameter to encode the grammar, syntax and vocabulary of dozens of languages and even programming languages, as well as store a large corpus of knowledge that has acquired from the data in the training set. We can hypothesize that, when we fine-tuned GPT-3 in order to approximate the heuristic of the Probe planner, only a small part of those parameters is significantly updated. Hence, a smaller model might be sufficient to replicate the results we obtained with GPT-3. Indeed, this question has already partially proven by Pallagani et al. (2023) with their Plansformer based on CodeT5, albeit on simpler problems and at the cost of a much more extensive training. For reference, GPT-3, which has now been surpassed in size by far larger models, counts 175 billion parameters, while CodeT5 is much smaller, counting only 220 million parameter. Another added benefit of a smaller model is that, on the same hardware, it would run faster. Again, the work by Pallagani et al. (2023) has shown timing performance better than their reference planner FastDownward (Helmert, 2006), unlike Teriyaki that in this regard has lost against Probe. As usual in this field, fair comparisons between models are difficult and not particularly meaningful, but the takeaway of this result is that smaller models might be preferable in many logical reasoning applications of LLM, whenever possible.

Nevertheless, despite the relatively small size of CodeT5, it must also be noted that The Plansformer runs on server grade hardware, namely, 9 (Dual P-100) and 44 (Dual V100) nodes (albeit it must also be noted that training was performed on a single GPU node). Such configuration is far from what is commonly understood as a *local* LLM model, and is

achievable only by setting up an High-Performance Computing facility or, more commonly, by renting the machines in the cloud. This solution can solve the limitations 2 and 4 that we listed at the beginning of this Section, but leaves our first complaint unanswered and may actually be more costly than using a service like GPT-3 on-demand, especially when the ease of setup is taken into consideration. In other words, while using smaller LLM models has been proven feasible, we are still far from them being an efficient or practical alternative to OpenAI's solutions, and more research is needed towards that end.

For this reason, in this Chapter we will explore a different approach. Based again on the assumption that a reduced number of parameters might be sufficient for the kind of logical reasoning that has been demonstrated by solutions like Teriyaki and the Plansformer, we will attempt to fine-tune a LLM through a particular fine-tuning technique called Low-Rank Adaptation (LoRA) (Hu et al., 2021), which should allow us to to fine-tune a compact LLM on a local workstation. LoRA leverages low-rank matrix factorization to reduce the model's parameter space, allowing a much more efficient fine-tuning process. In other words, LoRA allows to update only a subset of highly meaningful parameter during the fine-tuning process, instead of the whole network. When used to fine-tune a model on a specific task, LoRA often results in only a minimal loss of answers' quality compared to a regular fine-tuning process. Preliminary results confirm the feasibility of the approach from a training performance point view, but the resulting models have failed so far to correctly solve any of the problems on Teriyaki's test set, as they tend to lose logical coherence after the first few actions. Nevertheless, training metrics have shown a token validation loss of about 0.3 in our best candidate model, against the 0.2 achieved by Teriyaki when it started to manifest robust planning performance. This result hints to the fact that a much larger training dataset might be needed to compensate for the smaller model being employed and the reduced number parameters being fine-tuned, and it is coherent with the differences in training performances that we observed between Teriyaki and the Plansformer, despite the different fine-tuning methodology. Hence, the main contribution of this Chapter is to inform the reader on the challenges imposed by smaller LLM and provide preliminary guidelines and insights about their fine-tuning process for Neurosymbolic planning.

In the next Sections, we will provide some background about LoRA, describe our approach in terms of models used and experimental setup, and provide a comparative analysis of the training performance over different model candidates, characterized by different base models' size and choice of training hyperparameters.

7.2 Low Rank Adaptation of Large Language Models

Low Rank Adaptation (LoRA) is an innovative fine-tuning method primarily applied in large language models (Hu et al., 2021). It addresses the issue of substantial computational requirement for standard fine-tuning by LoRA identifying, isolating and subsequently adjusting a low-rank subset of the model parameters, while keeping the rest unchanged.

Mathematically, given a pre-trained model parameter θ_0, LoRA modifies it by Δ, where $\Delta = UDV^T$ with UDV^T being the singular value decomposition (SVD) of Δ. Here, the rank of Δ is restricted to be less than or equal to a small number k (i.e., LoRA Rank). Then, the fine-tuned model parameters θ can be presented as:

$$\theta = \theta_0 + \alpha\Delta$$

where α is a scaling factor to prevent overfitting and help the model stay close to the initial state.

Low Rank Adaptation (LoRA) is thus an efficient strategy for fine-tuning large language models which focuses on reducing the number of trainable parameters and GPU memory requirements, thereby allowing for fine-tuning tasks even with less powerful hardware. LoRA proposes a unique method wherein the majority of pretrained model weights are frozen, while low-rank decomposition matrices are allowed to be trainable. This not only simplifies the scalability of large language models, but also helps skim down the GPU memory need by roughly a factor of three. Therefore, LoRA can cut down the number of trainable parameters by a factor of 10,000 as compared to GPT-3 175B.

The efficacy of LoRA is evident in its performance results that are on par with, if not better than, traditional fine-tuning models. Furthermore, it exhibits faster training throughput and does not incur additional inference latency as compared to other adapter methods. Combining LoRA with existing prefix-based approaches has demonstrated improved performance and higher model quality.

Large-scale pretraining on general domain data coupled with adaptation to specific tasks is viable with LoRA. This efficient adaptation strategy maintains high model quality while allowing for rapid task-switching and sharing of model parameters. Despite the reduced computational resources, LoRA makes it more tenable to comprehend the transformation of features learned during pre-training for downstream tasks.

The fine-tuning of large language models may be resource-intense concerning storage and hardware. Yet, it outperforms few-shot learning on GPT-3. LoRA is a promising solution to this challenge, presenting the potential for more principled ways to select weight matrices.

In practical use, key hyperparameters when fine-tuning with LoRAs include:

- **Batch size**: The total number of training examples used in a single iteration. The choice of batch size influences both training dynamics and computational efficiency.
- **Micro batch size**: Refers to the number of training examples that are processed simultaneously on the GPU. This parameter must be carefully balanced to maximize utilization of computational resources while avoiding out-of-memory errors.
- **LoRA Rank (k)**: The maximum rank of the low-rank component Δ in the model. A higher rank allows more flexibility in adaptations but has a higher computational cost.
- **LoRA Alpha**: The scaling factor applied to Δ, determining how much modification it can impose during fine-tuning. This helps balance task-specific adaptation with the ability to retain the original pre-trained model's knowledge.
- **Dropout**: Dropout is a regularization technique which randomly sets a proportion of input units to 0 at each update during training time to prevent overfitting. This parameter controls the chance of an input unit to be set to 0.
- **Cutoff length**: The maximum sequence length during the back-propagation process. Keeping sequences below this length helps manage the computational requirements of LoRA, making it feasible to perform fine-tuning even on less resourceful hardware.

7.3 Model selection and training methodology

In order to fine-tune LLMs with LoRAs and test whether it is possible to achieve adequate planning performances with on-site workstations, we set up a local training and inference environment. Hardware-wise, the environment was deployed on an Intel Core i9-11900 2.50 GHz, 64Gb RAM machine mounting an A6000 NVidia GPU with 48GB of VRAM.

The software stack was based on the popular *Text Generation Webui* utility[1], which automatically sets up a CUDA-accelerated PyTorch environment, and provides convenient OpenAI standard-compliant APIs and a graphical interface to interact with the models. The utility also provides streamlined access to open source models by allowing to download models from the HuggingFace platform [2] and by providing reasonable default configurations for each model.

[1] https://github.com/oobabooga/text-generation-webui
[2] https://huggingface.co

At the time the experiments were carried out, the most powerful family of local models was the LLama 2 family by Meta[3], which includes models with 7, 13, 30 and 70 billions of parameters. While a plethora of models derived from Llama 2 were available at the time of testing, which could outperform LLama 2 in specific benchmarks, we decided to focus on the models that were relatively close to the original Llama 2 models, as none of the more specialized models available appeared as it could offer an actual benefit to our application. On the contrary, we feared that excessive fine-tuning to beat specific benchmarks could be detrimental to our specific task. For this reason, we eventually opted for Llama 2 family models provided by Nous Research[4]. For context, Llama 2 was pretrained on 2 trillion tokens of data from publicly available sources. Nous Research versions were fine tuned with additional publicly available instruction datasets, as well as over one million new human-annotated examples. At the time of testing, Nous Research version of Llama 2 models were widely considered by the community the best performing all-round models, and were available with all models' sizes variants, i.e., 7, 13, 30 and 70 billions of parameters.

It must be noted that 7B and 13B models can be loaded in their native form in the 48GB VRAM of our A6000 GPU, while at the time of testing the 30B and 70B could not. For this reason, 30B and 70B models had to loaded with a parameters' precision reduced from 16 to 8 bit which usually leads to a modest loss in accuracy. For the sake of simplicity, we have thus decided to load all models with 8 bit precision. All models have been loaded with the `Transformers` model loader and default settings.

For the training stage, we decided to employ the same dataset used to train Teriyaki and described in Section 5.5. On one side, our local training environment requires the dataset to be provided in JSON format instead of JSONL; on the other, our local environment is much more flexible and allows to specify in details how the prompt and completion should be formatted. Hence, we first defined a custom, planning specific, dataset format template, and then used a custom software to translate the Teriyaki dataset into the new format.

[3] https://ai.meta.com/llama/

[4] https://huggingface.co/NousResearch/Nous-Hermes-Llama2-13b

```
{
    "problem,plan":
    "PDDL planning

    ### Problem:
    %problem%

    ###
    Plan:
    %plan%END"
}
```

Listing 7.1 The proposed custom planning format.

```
[
    {"problem":
    "(angle_joint angle270 joint1)
    (angle_joint angle285 joint2)
    (angle_joint angle330 joint3)
    (in-centre joint3)\n(free gleft)
    (free gright))
    (:goal
    (and
    (angle_joint angle315 joint1)
    (angle_joint angle285 joint2)
    (angle_joint angle0 joint3)
    )
    )",
    "plan":
    "0.00100: (link-to-central-grasp ...)
    0.00300: (increase_angle_first_child_45 ...)
    0.00500: (release-links ..)
    ...
    0.01700: (increase_angle_first_child ...)\n\n"},

]
```

Listing 7.2 An example of a training sample in the new JSON format.

Listing 7.1 shows our format template. In practice, requests to the model will have to be provided according to this template, substituting `%problem%` and `%plan%` respectively with the prompt and completion as defined in Section 5.5. Note that now there is no need anymore to add a stopping condition to the problem/prompt and plan/completion as this can be specified in the template (i.e., `###` and `END`).

Accordingly, our dataset will be a JSON file with a list of problem/plan pairs which will be automatically substituted in the template that we have specified. A training sample as defined in the new format is provided in Listing 7.2.

7.4 Results and discussion

As preciously anticipated, the method outlined in the previous section failed to generate planners capable of solving plans with the reliability demonstrated by Teriyaki in Chapter 6. Nevertheless, the training metrics collected during our experiments highlight some interesting phenomena and can provide some guidance to the readers in the choice of training hyperparameters, if they are interested in pursuing this line of research. In particular, we believe it is interesting to discuss the impact of the LoRA rank, batch size, and model size on training performance. We will conclude this Section with our currently best performing model and a discussion on the possible way forward.

7.4.1 Impact of the LoRA rank

In section 7.2, we have seen the importance of the rank of a LoRA, as it determines how many of the parameters of the initial model will be allowed to be updated during the fine-tuning process. At the beginning of this chapter, we have hypothesized that only a small part of the model parameters need to be significantly updated in order to learn to simulate a search heuristic. Nevertheless, two questions arise: firstly, whether the maximum rank allowed by our experimental setup, i.e., 1024, is sufficient for fine tuning Teriyaki-like solvers; and, secondly, if so, whether we can go even lower and what impact it might have on the model's performance. For reference, a rank between 64 and 128 is usually used to teach the model a specific writing style, whereas a rank between 128 and 256 is used to teach a new skill. Ranks between 512 and 1024 are used rarely, and only in particular scenarios where fine details are important. For this reason we decided to test ranks from 128 and up, but we soon noticed that exceeding 512 lead only to negligible improvements.

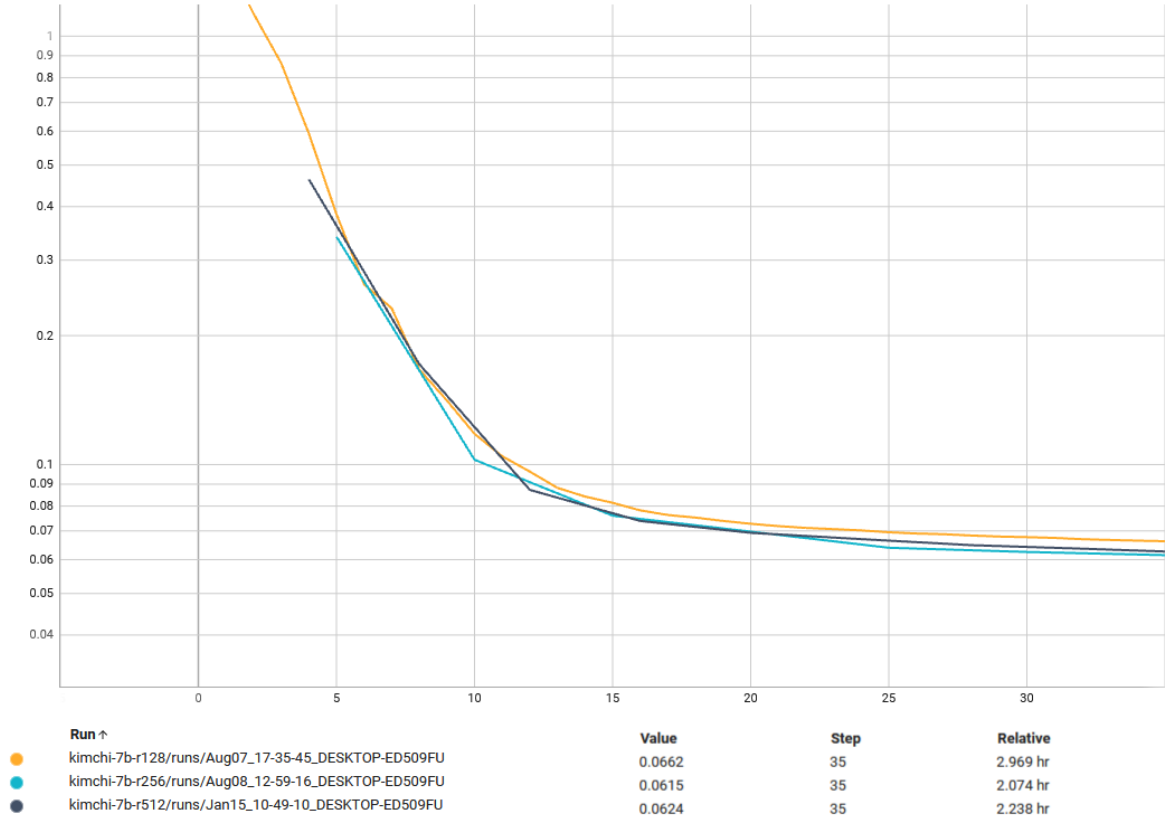

Figure 7.1 Evaluation loss evolution with LoRA rank 128, 256 and 512 while fine-tuning NousResearch LLama 2 7B with the Teriyaki dataset. Evaluation loss is plotted against training steps, where each training step corresponds to 400 training samples. Hence, the graph shows the evolution of the evaluation loss over almost 2 full epochs.

Figure 7.1 compares the token evaluation loss evolution with LoRA rank 128, 256 and 512. The base model being fine-tuned is NousResearch LLama 2 7B. The LoRA alpha has been always left to its recommended value, i.e. twice the LoRA rank. Learning rate is set for all three sessions to the default $3e-4$ and controlled by a linear scheduler. LoRA dropout is also set to the default value 0.05. Batch size is set to 400 and micro batch size to 4.

Overall, we can observe that against our initial hypothesis, LoRA rank does not have a very strong impact over the capability of the model to learn this specific skill. We do observe a perceivable improvement between rank 128 and 256, as expected considering the complexity of the skill that we are trying to teach. Increasing the rank further to 512 leads to near identical performance, and actually a very slight decrease, albeit within the margin of error. This result is surprising considering that a rank above 256 and up to 1024 is usually be beneficial in scenarios that require attention to fine details, which is definitely true for

planning. As we have already seen in Chapter 4, precisely mapping admissible actions is one of the biggest challenges when exploiting LLMs for task planning.

As a training step involves processing a full batch of training samples, and given that batch size is set to 400, the graph in Figure 7.1 shows the resulting token evaluation loss after 14.000 samples, or almost 2 full epochs over the Teriyaki dataset. This enables us to compare these results with Figure 6.1, which similarly shows token evaluation loss evolution while fine-tuning GPT-3 for 2 epochs. The capability of GPT-3 to learn from the given dataset is massively superior: the model quickly reaches a token evaluation loss value around 0.02, and while it oscillates around this value over thousands of training samples, we see planning accuracy slowly rising as well. Conversely, these models based on NousResearch LLama 2 7B reach at best 0.0615. While for typical LLM task related to NLP this is an outstanding result for such a small model, the value appears to be insufficient for planning tasks. On the positive side, after 14.000 samples token evaluation loss still appears to be decreasing, hinting that a larger dataset might allow to overcome the model shortcomings.

7.4.2 Impact of the Batch Size

In this Section we will discuss the impact of batch size on training performance. In Section 5.6 we have seen that OpenAI's fine-tuning API automatically sets batch size to 0.2% of the training dataset. Teriyaki training dataset is composed of 8.000 training sample, thus resulting in a batch size of 16. Nevertheless, the reader might have noticed that in the previous Section batch size was set to 400. This steep increase is motivated by the recent trend to use increasingly large batch sizes both for LLM training and fine-tuning. For reference, LLama 2 was trained with a batch size of $4M$ tokens while fine-tuning is usually performed with a batch size of 64. The reason for this trend is that a large batch size reduces the fine-tuning noise, albeit at the cost of a significantly higher memory consumption.

Figure 7.2 shows the evolution of token evaluation loss against training steps when batch size is set to 16, 64 and 128. Again, the base model being fine-tuned is NousResearch LLama 2 7B. LoRA rank is set to 512 for all three models. Indeed, rank 512 performed as well as 256 in the comparison presented in previous section, but we opted for 512 to reduce the risk of the LoRA rank bottlenecking our subsequent benchmarks. All other training hyperparameters are left unchanged compared to the example in the previous Section. For a fairer comparison, all three models have been fine-tuned with 12.800 training samples (i.e., roughly 1.5 epoch over the Teriyaki dataset), but since batch size influences the number of training steps, the final number of steps appears different between the three fine-tuning runs being compared.

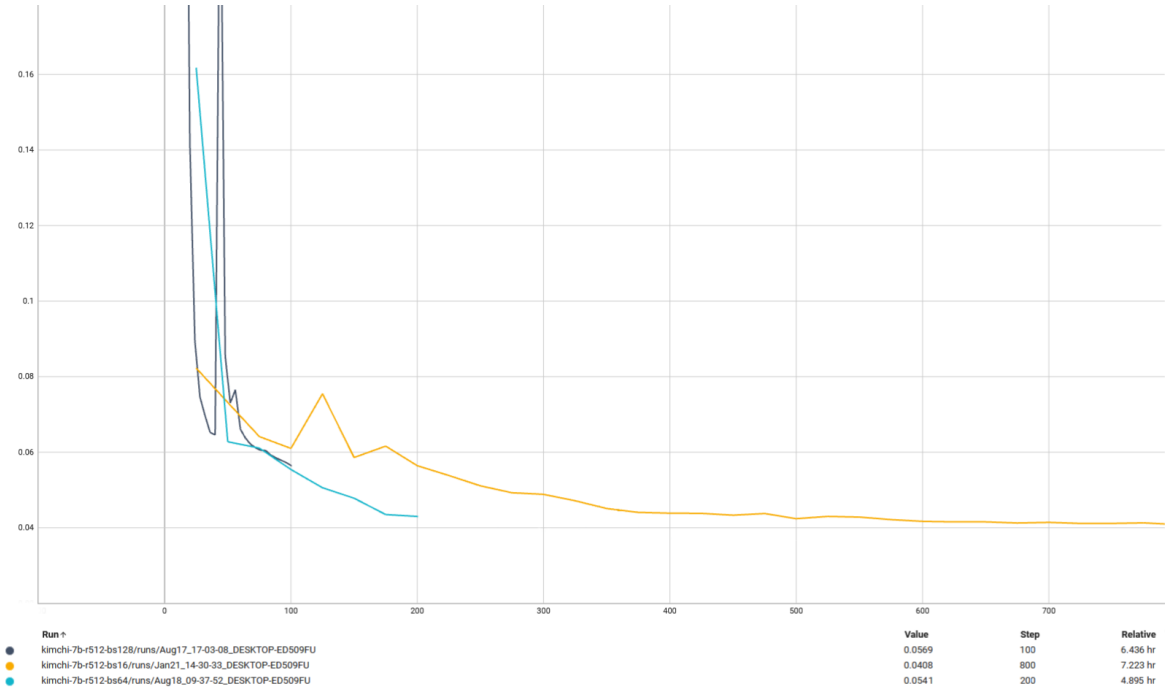

Figure 7.2 Evaluation loss evolution with batch size 16, 64, 128 while fine-tuning Nous-Research LLama 2 7B with the Teriyaki dataset. Rank size is set to 512. Evaluation loss is plotted against training steps, which depend on the batch size. For a fairer comparison, all three models have been fine-tuned with 12.800 training samples (i.e., roughly 1.5 epochs over the Teriyaki dataset), corresponding respectively to 800, 200 and 100 training steps.

The first aspect that we notice is that a batch size of 16, which supposedly corresponds to the value that OpenAI's system has automatically selected during Teriyaki planners' fine-tuning, is the best performer of the three. This result is slightly unexpected given the general consensus that a larger batch size is usually beneficial and it is often common practice to set the batch size as high as allowed by the limitations of the hardware used for fine-tuning. To further corroborate this result, we observe that also batch size 64 performs significantly better than batch size 128. A possible interpretation of this result that we propose is that, while a higher batch size might reduce noise in the fine-tuning process, it might also make it harder to learn fine details in the dataset. As our dataset requires extreme precision both in sequencing actions and in assigning them the correct parameters, an excessive batch size value might be detrimental. This phenomenon requires further investigation and might have different explanations, but, if confirmed, it might be of interest to the wider LLM community in any fine-tuning aimed at precision sensitive applications.

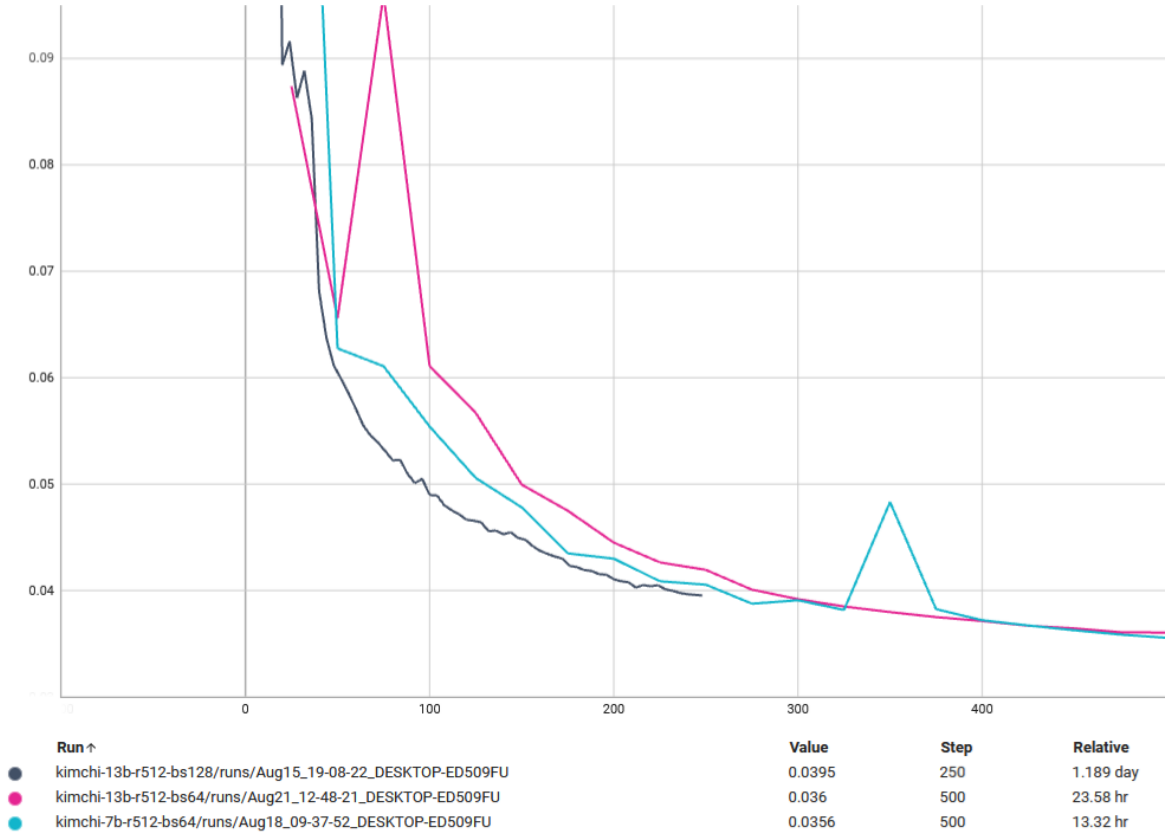

Figure 7.3 Evaluation loss evolution while fine-tuning NousResearch LLama 2 7B and 13B with the Teriyaki dataset. Batch size is set to 64 for both models, but an additional fine-tuning run for the 13B models with batch size set to 128 is included in the comparison. Rank size is set to 512 for all models. Evaluation loss is plotted against training steps, which depend on the batch size. For a fairer comparison, all three models have been fine-tuned with 32.000 training samples (i.e., 4 epochs over the Teriyaki dataset), thus the run with batch size 128 has half the final number of training steps.

7.4.3 Impact of the Base Model Size

So far, we have only considered the smallest of the base models at our disposals, i.e., NousResearch LLama 2 7B. Considering the difficulties that we have experienced so far in obtaining reliable planning performance, it is natural to ask ourselves whether increasing again the base model size could be helpful. Figure 7.3 compares the evolution of token evaluation loss against training steps for both NousResearch LLama 2 7B and 13B. As in the previous Section, we have set LoRA rank to 512 to avoid any possible bottlenecking without negatively impacting performance. Similarly, we used for both models a batch size of 64, but we also included a fine-tuning run for the 13B model with batch size 128, in order to verify whether, for a larger model, a larger batch size might be beneficial. In this experiment we

also decided to increase our training epochs to 4, resulting in 32.000 training samples, or 500 training steps for the runs with batch size set to 54, and 250 for the run with batch size set to 128. All other training hyperparameters are left unchanged compared to the examples in the previous Sections.

At a glance, it appears evident that the 7B and 13B models perform equally well, thus suggesting that adopting a larger model is not a viable strategy to improve planning performance. On the positive side, this result partially supports our hypothesis that only a small portion of the model parameters are truly important for task planning, albeit it must be remembered that we are fine-tuning with LoRA and that with a fully fledged fine-tuning some improvements might be possible. This is definitely an aspect that requires further investigation in the future.

So far we have not paid attention to fine-tuning time. In this experiment, as expected, there is a large difference between the 13B and 7B models training time, with the former taking about twice as much time to complete compared to the latter. Considering the near identical performance, the 7B model is the obvious choice in terms of practicality. This result further suggests that smaller models are an ideal candidate for task planning applications of LLM, at least in the sense that we have adopted in Section 4.3.

Finally, it is also interesting to note that the fine-tuning run of the 13B model using batch size 128 is our worse performer of the three. This result further supports our hypothesis in the previous Section that an unnecessarily big batch size might be detrimental while fine-tuning a model for tasks that require a high level of precision and attention to fine-details..

7.4.4 Increasing dataset and batch size

In the previous three Sections we have presented a number of possible optimizations to better learn to solve task planning problem instances with a local LLM. In particular, we have observed that significant performance improvements are achievable by adequate choice of LoRA rank and batch size, which allowed us to reduce token validation loss by more than 30%. We have also observed that despite the significant size of our training dataset and having increased the number of epochs from 2, as used in Chapter 5, to 4, did not cause any significant overfitting. On the contrary, the graphs presented clearly show that the token evaluation loss is still decreasing at the end of each fine-tuning session, albeit slowly.

For this reason, we include one last experiment where we fine-tuned NousResearch LLama 2 7B with LoRA rank 512, batch size 64 and number of epochs 4, with an extended dataset. For this purpose, we employed the same methodology discussed in Section 5.5 to generate a training and validation dataset composed of 20.000 and 2.000 samples respectively,

which we refer to as Teriyaki-XL dataset. Figure 7.4 shows the result. In this configuration, we achieve a very reasonable token validation loss value, i.e., 0.0324, which is within reach of our target 0.02 and about 50% lower that what we obtained in our first, naïve, experiments. It is interesting to note that also in this experiment the token validation loss is still decreasing at the end of the fine-tuning run, suggesting that an even bigger dataset and an even larger number of epochs could be beneficial. Initially the scientific community feared rapid overfitting with the increasing number of epochs, especially when fine-tuning for conditional generation tasks. This is the reason why GPT-3 recommended to use 2 epochs for conditional generation, and why we adopted the same value to fine-tune Teriyaki planners, as described in Section 5.6. Nevertheless, today up to 6 or even 8 epochs are being regularly used without significant issues.

On the negative side, it must be noted that the Teriyaki-XL dataset now exceeds the dataset used by the Plansformer by 2.000 samples, and yet it is still insufficient to reach reliable planning performance. While it must be noted that the planning domain that we are taking into consideration is much more complex, for example, for the presence of conditional effects, this result should let us consider whether fully fine-tuning a smaller model like CodeT5 is ultimately more convenient than fine-tuning with LoRA a base model like LLama 7B, despite the eventual technical difficulties due to the necessity of high-performance computing infrastructure. This aspect surely requires a more in-depth investigation in future work on the topic.

A possible inspiration for future research directions and for a strategy to overcome the difficulties that we have outlined so far, comes exactly from the Plansformer and the most recent advancements in the state of the art. One difference between the Plansformer and Teriyaki that we have discussed before is that the former's dataset has been generated using an optimal planner. At first sight this difference might look inconsequential to the purpose of generating correct plans, and to only have an effect on the quality of the plans generated. Nevertheless, the work of Gunasekar et al. (2023) on Phi-1 has shown that dataset quality has an outstanding impact on LLM performance, causing a major shift of focus in the community from LLM architectures to datasets composition. For reference, Phi-1 is a small, 1B parameters model that is trained exclusively on textbook quality text, and which has been shown to outperform models several times larger in terms of number of parameters. By extending this concept to or task planning scenario, we can hypothesize that optimal plans have higher quality compared to non-optimal ones, as they are shorter and more consistent among themselves in the way the actions are sequenced. This is similar to the concept that textbooks are, in general, more concise and precise on a given topic than what can be scraped

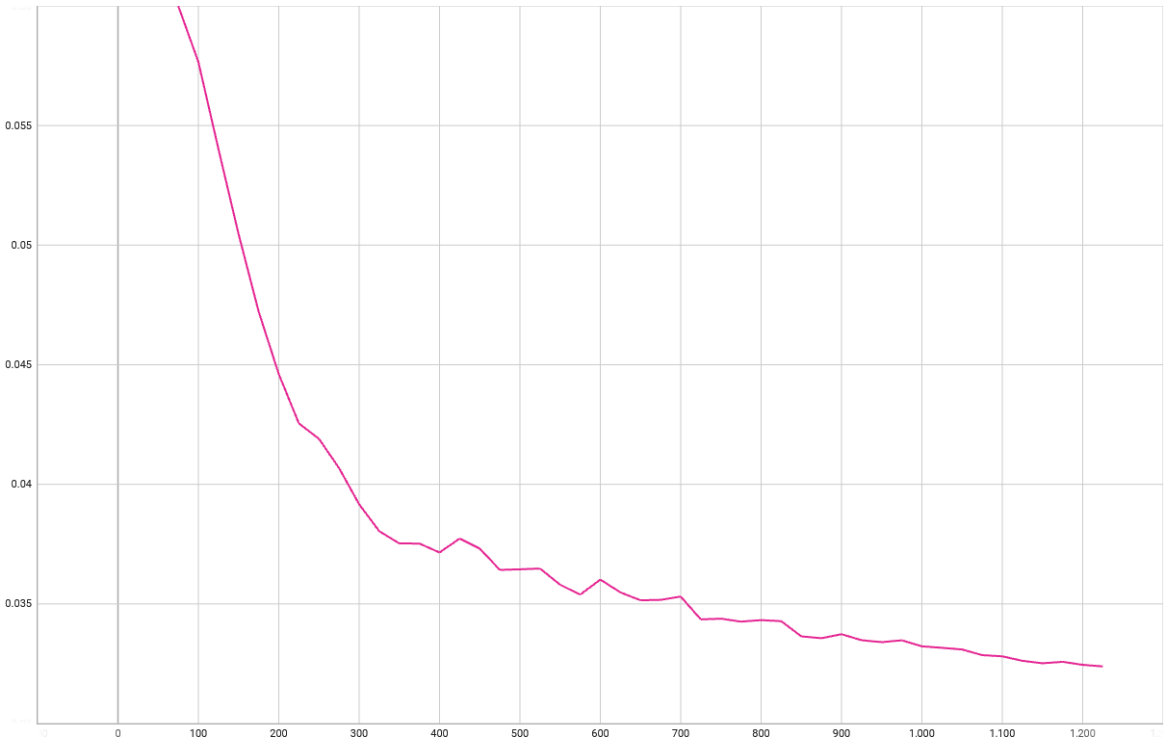

Figure 7.4 Evaluation loss evolution while fine-tuning NousResearch LLama 2 7B with an extended dataset. Batch size is set to 64, while Rank size is set to 512. Evaluation loss is plotted against training steps, which depend on the batch size, hence here 80.000 training samples (i.e., 4 epochs over the Teriyaki-XL dataset), appear as 1250 training steps. The final token validation loss value is 0.0324.

by random websites and users' conversations from the internet. In future work we will try to repeat the last experiment presented in this section with a dataset composed of optimal plans. If our hypothesis is correct, we might finally be able to replicate the performance of Teriyaki with relatively affordable, local hardware, and shed a new light on LLM task planning capabilities.

Chapter 8

Conclusions

We have finally arrived to final chapter of this thesis. In the first Chapter, we have set on a journey to keep the planner in the loop, that is, recognize the value of symbolic task planning as a powerful, easy to deploy, humanly intelligible tool; and overcome the traditional issues that limits its use in the real world, especially when planning must happen online as part of a larger planning and execution architecture. In Chapter 2 we discussed an architecture, `PlanHRC`, that heavily highlights such limitations. As the name suggests, `PlanHRC` is designed to support long-term autonomy, robust operation and human-robot cooperation in the operative scenario of the manipulation of articulated objects. In this Chapter we delved with great detail into the description of the planning models that can be used to describe the problem of articulated object manipulation, and stressed how their performance is crucial not only to robust operation, but also to ensure a fruitful and pleasant cooperation with humans through improved fluency.

Aware of the limitations of said models, we entered Chapter 3, where we proposed a first, novel solution to this problem, albeit based on the traditional tools of symbolic planning, i.e., the use of macros as a tool to reduce planning times. The novelty of our approach in Chapter 3 is to not only focus on planning times, but also analyze the impact of macros on execution times in the real world. By providing ad-hoc execution routines to ground macros, we have shown that is is possible to obtain outstanding reductions in planning times without hurting significantly execution times, despite the risk for macro to introduce spurious, unnecessary elementary actions into the final plan.

In Chapter 4, we have started to completely re-imagine what symbolic planning could be. We made an hypothesis that recent advancements in Large Language Models could be used to bridge the gap between traditional symbolic AI a the latest generation of machine learning-based methods. In particular, we referred to this new class of solvers as Neurosymbolic

Planners, and delved into the state of the art of LLM applications to logical reasoning and planning. The result of this analysis was that a large part of the scientific community is focusing on the general one- or few-shots capabilities of LLM to answer to logical problems, resulting in philosophically exciting but unreliable results. Instead, we have decided to use LLMs as a tool to approximate the search heuristic of a traditional planner by fine-tuning it on a large dataset of problem-plan pairs.

In Chapter 5 we have introduced Teriyaki, a method to generate Neurosymbolic planners by fine-tuning GPT-3. We have shown how to generate an adequate dataset and provided extensive details on the fine-tuning process and choice of training hyperparameters. Chapter 6 follows by presenting our results: Teriyaki-based Neurosymbolic planners are capable of reliably ($> 95\%$) generating plans for the specific domains for which they are trained. Most importantly, we are the first to our knowledge to apply a similar technique to a challenging planning domain with real-world, robotic applications. In such context, while our Neurosymbolic planners are still slower than their traditional counterpart, we have proposed a novel operation mode, i.e., simultaneous planning and execution, which is enabled by the unique characteristics of Neurosymbolic planners, and which can reduce by more than 60% waiting times for a plan.

In Chapter 7, we have hypothesized that a much smaller mode model compared to GPT-3 could be sufficient to implement Teriyaki-like Neurosymolic planners, and explored the possibility to bring Neurosymbolic planners to local hardware through the use of local models and LoRA fine-tuning. Such advancement not only is of great scientific interest, but would also make Neurosymbolic planners much more accessible to a wider audience. While at the moment planners based on local LLMs failed to achieve the same reliability shown by the models in Chapter 5, our preliminary results suggest that this goal is achievable, albeit at the cost of a precise optimization of the training hyperparameters and a much larger training dataset. The Chapter also proposes some directions for future work, in particular regarding the quality of the dataset used to train the models.

Looking back at this journey, we believe that this thesis is a confident step forward towards infusing new life in the field of symbolic task planning, in particular in real world, robotic applications that require the planner to be constantly in the planning and execution loop of a robotic software architecture. Referring back at our initial example of the `PlanHRC` architecture, the combination of the models proposed in Chapter 3 with the simultaneous planning and execution capabilities demonstrated in Chapter 6, can provide a significant improvement in waiting times for a plan, whether the robot is just starting to act, or a re-planning is required, to address an unforeseen event or accommodate for human interventions.

In particular, simultaneous planning and execution can be a real game changer considering that traditional planners can only return a plan once a solution, from initial to goal state, has been identified.

We are still far from the day that Neurosymbolic planners can be a complete alternative to traditional heuristic search planners. Nevertheless, we hope that the results described in this thesis prove without a doubt that this is a new line of research which deserves to be explored in greater depth. In particular, we hope to have convinced the reader that a deeper collaboration between the two communities of cognitivist and emergent AI scientists, could be highly beneficial in unlocking the full potential of LLM for logic reasoning and planning capabilities, especially in robotics applications.

Regarding the shortcoming of Neurosymbolic systems that we have seen in this pages, we are confident that many of them will be naturally addressed as an increasing number of foundation models are being made available to the public, each providing stronger text generation and logical reasoning capabilities. In this regard, and considering our experiences in Chapter 7, we believe that some of the most interesting advancement will not come from models like a hypothetical GPT-5, but rather by its smaller siblings and the truly outstanding community of research and engineers that is flourishing around them.

References

Aeronautiques, C., Howe, A., Knoblock, C., McDermott, I. D., Ram, A., Veloso, M., Weld, D., SRI, D. W., Barrett, A., Christianson, D., et al. (1998). Pddl| the planning domain definition language. *Technical Report, Tech. Rep.*

Agostini, A., Torras, C., and Wörgötter, F. (2011). Integrating task planning and interactive learning for robots to work in human environments. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)*, Barcellona, Spain.

Ahmadzadeh, S., Paikan, A., Mastrogiovanni, F., Natale, L., Kormushev, P., and Caldwell, D. (2015). Learning symbolic representations of actions from human demonstrations. In *Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA 2015)*, Seattle, WA, USA.

Baraglia, J., Cakmak, M., Nagai, Y., Rao, R., and Asada, M. (2016). Initiative in robot assistance during collaborative task execution. In *Proceedings of the 2016 ACM/IEEE International Conference on Human-Robot Interaction (HRI 2016)*, Christchurch, New Zealand.

Berenson, D. (2013). Manipulation of deformable objects without modelling and simulating deformation. In *Proceedings of the 2013 IEEE-RSJ International Conference on Intelligent Robots and Systems (IROS 2013)*, Tokyo, Japan.

Bertolucci, R., Capitanelli, A., Dodaro, C., Leone, N., Maratea, M., Mastrogiovanni, F., and Vallati, M. (2021). Manipulation of articulated objects using dual-arm robots via answer set programming. *Theory and Practice of Logic Programming*, 21(3):372–401.

Bertolucci, R., Capitanelli, A., Maratea, M., Mastrogiovanni, F., and Vallati, M. (2019). Automated planning encodings for the manipulation of articulated objects in 3d with gravity. In *AI* IA 2019–Advances in Artificial Intelligence: XVIIIth International Conference of the Italian Association for Artificial Intelligence, Rende, Italy, November 19–22, 2019, Proceedings 18*, pages 135–150. Springer.

Bodenhagen, L., Fugl, A., Jordt, A., Willatzen, M., Andersen, K., Olsen, M., Koch, R., Petersen, H., and Kruger, N. (2014). An adaptable robot vision system performing manipulation actions with flexible objects. *IEEE Transactions on Automation Science and Engineering*, 11(3):749–765.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Caccavale, R. and Finzi, A. (2017). Flexible task execution and attentional regulations in human-robot interaction. *IEEE Transactions on Cognitive and Developmental Systems*, 9(1):68–79.

Capitanelli, A., Maratea, M., Mastrogiovanni, F., and Vallati, M. (2017). Automated planning techniques for robot manipulation tasks involving articulated objects. In *Proceedings of the 2017 Conference of the Italian Association for Artificial Intelligence (AIxIA 2017)*, Bari, Italy.

Capitanelli, A., Maratea, M., Mastrogiovanni, F., and Vallati, M. (2018). On the manipulation of articulated objects in human–robot cooperation scenarios. *Robotics and Autonomous Systems*, 109:139–155.

Carfí, A., Foglino, F., Bruno, B., and Mastrogiovanni, F. (2019). A multi-sensor dataset for human-human handover. *Data in Brief*, 22:119–117.

Cashmore, M., Fox, M., Long, D., Magazzeni, D., Ridder, B., Carrera, A., Palomeras, N., Hurtos, N., and Carreras, M. (2015a). ROSPlan: planning in the Robot Operating System. In *Proceedings of the 2015 International Conference on Automated Planning and Scheduling (ICAPS 2015)*, Jerusalem, Israel.

Cashmore, M., Fox, M., Long, D., Magazzeni, D., Ridder, B., Carrera, A., Palomeras, N., Hurtos, N., and Carreras, M. (2015b). Rosplan: Planning in the robot operating system. In *Proceedings of the international conference on automated planning and scheduling*, volume 25, pages 333–341.

Chakraborti, T., Kambhampati, S., Scheutz, M., and Zhang, Y. (2017). AI challenges in human-robot cognitive teaming. *arXiv preprint arXiv:1707.04775*.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. (2021). Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. (2022). Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

Chrpa, L. and Vallati, M. (2019). Improving domain-independent planning via critical section macro-operators. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7546–7553.

Chrpa, L., Vallati, M., and McCluskey, T. L. (2015). On the online generation of effective macro-operators. AAAI Press.

Cirillo, M., Karlsson, L., and Saffiotti, A. (2010). Human-aware task planning. *ACM Transactions on Intelligent Systems and Technology*, 1(2):1–26.

Clair, A. and Matarić, M. (2015). How robot verbal feedback can improve team performance in human-robot as collaborations. In *Proceedings of the 2015 ACM/IEEE International Conference on Human-Robot Interaction (HRI 2015)*, Portland, USA.

Coleman, D., Sucan, I., Chitta, S., and Correll, N. (2014). Reducing the barrier to entry of complex robotic software: a moveit! case study. *arXiv preprint arXiv:1404.3785*.

Dale, R. (2021). Gpt-3: What's it good for? *Natural Language Engineering*, 27(1):113–118.

Darvish, K., Simetti, E., Mastrogiovanni, F., and Casalino, G. (2021). A hierarchical architecture for human-robot cooperation processes. *IEEE Transactions on Robotics*, 37(2):567–586.

Darvish, K., Wanderlingh, F., Bruno, B., Simetti, E., Mastrogiovanni, F., and Casalino, G. (2017). Flexible human-robot cooperation models for assisted shop-floor tasks. *arXiv preprint arXiv:1707.02591*.

Darvish, K., Wanderlingh, F., Bruno, B., Simetti, E., Mastrogiovanni, F., and Casalino, G. (2018). Flexible human-robot cooperation models for assisted shop-floor tasks. *Mechatronics*, 51:97–115.

Dautenhahn, K. (2007). Socially intelligent robots: dimensions of human-robot interaction. *Philosophical Transactions of the Royal Society of London, Series B, Biological Sciences*, 362(1480):679–704.

Denei, S., Mastrogiovanni, F., and Cannata, G. (2015). Towards the creation of tactile maps for robots and their use in robot contact motion control. *Robotics and Autonomous Systems*, 63(3):293–308.

Elbrechter, C., Haschke, R., and Ritter, H. (2011). Bi-manual robotic paper manipulation based on real-time marker tracking and physical modelling. In *Proceedings of the 2011 IEEE-RSJ International Conference on Intelligent Robots and Systems (IROS 2012)*, San Francisco, CA, USA.

Elbrechter, C., Haschke, R., and Ritter, H. (2012). Folding paper with anthropomorphic robot hands using real-time physics-based modeling. In *Proceedings of the 2012 IEEE-RAS International Conference on Humanoid Robotics (HUMANOIDS 2012)*, Osaka, Japan.

Fikes, R. E., Hart, P. E., and Nilsson, N. J. (1972). Learning and executing generalized robot plans. *Artificial intelligence*, 3:251–288.

Fox, M., Howey, R., and Long, D. (2005). Validating plans in the context of processes and exogenous events. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI 2005)*, Pittsburgh, Pennsylvania, USA.

Fox, M. and Long, D. (2006). Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research*, 27:235–297.

Frank, B., Schmedding, R., Stachniss, C., Teschner, M., and Burgard, W. (2010). Learning the elasticity parameters of deformable objects with a manipulation robot. In *Proceedings of the 2010 IEEE-RSJ International Conference on Intelligent Robots and Systems (IROS 2010)*, Taipei, Taiwan.

Garcez, A. d. and Lamb, L. C. (2020). Neurosymbolic ai: the 3rd wave. *arXiv preprint arXiv:2012.05876*.

Gerevini, A. E., Saetti, A., and Vallati, M. (2011). Exploiting macro-actions and predicting plan length in planning as satisfiability. In *Congress of the Italian Association for Artificial Intelligence*, pages 189–200. Springer.

Gombolay, M., Gutierrez, R., Starla, G., and Shah, J. (2014). Decision making, authority, team efficiency and human worker satisfaction in mixed human-robot teams. In *Proceedings of Robotics: Science and Systems X (RSS 2014)*, Berkeley, USA.

Gombolay, M., Huang, C., and Shah, J. (2015). Coordination of human-robot teaming with human task preferences. In *Proceedings of the 2015 AAAI Fall Symposium Series*, Palo Alto, CA, USA.

Gombolay, M., Wilcox, R., and Shah, J. (2013). Fast scheduling of multi-robot teams with temporospatial constraints. In *Proceedings of Robotics: Science and Systems IX (RSS 2013)*, Berlin, Germany.

Goodrich, M. and Schultz, A. (2007). Human-robot interaction: a survey. *Foundations and Trends in Human-Computer Interaction*, 1(3):203–275.

Gunasekar, S., Zhang, Y., Aneja, J., Mendes, C. C. T., Del Giorno, A., Gopi, S., Javaheripi, M., Kauffmann, P., de Rosa, G., Saarikivi, O., et al. (2023). Textbooks are all you need. *arXiv preprint arXiv:2306.11644*.

Haddadin, S. and Croft, E. (2016). Physical human-robot interaction. In *B. Siciliano and O. Khatib (Eds.) Springer Handbook of Robotics*. Springer International Publishing.

Harnad, S. (1990). The symbol grounding problem. *Physica D*, 42:335–346.

Hatcher, W. G. and Yu, W. (2018). A survey of deep learning: Platforms, applications and emerging research trends. *IEEE Access*, 6:24411–24432.

Helmert, M. (2006). The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246.

Henrich, D. and Worn, H. (2000). *Robot manipulation of deformable objects*. Advanced Manufacturing. Springer-Verlag, London, Berlin, Heidelberg.

Heyer, C. (2010a). Human-robot interaction and future industrial robotics applications. In *Proceedings of the 2010 IEEE-RSJ International Conference on Intelligent Robots and Systems (IROS 2010)*, Taipei, Taiwan.

Heyer, C. (2010b). Human-robot interaction and future industrial robotics applications. In *2010 ieee/rsj international conference on intelligent robots and systems*, pages 4749–4754. IEEE.

Hoffman, G. (2019). Evaluating fluency in human–robot collaboration. *IEEE Transactions on Human-Machine Systems*, 49(3):209–218.

Hofmann, T., Niemueller, T., and Lakemeyer, G. (2017). Initial results on generating macro actions from a plan database for planning on autonomous mobile robots. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 27, pages 498–503.

Howard, A. and Bekey, G. (1997). Recursive learning for deformable object manipulation. In *Proceedings of the 1997 International Conference on Advanced Robotics (ICAR 1997)*, Monterey, CA, USA.

Howey, R., Long, D., and Fox, M. (2004). Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pages 294–301. IEEE.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. (2021). Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Huang, W., Abbeel, P., Pathak, D., and Mordatch, I. (2022). Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pages 9118–9147. PMLR.

Jimenez, P. (2012). Survey on model-based manipulation planning of deformable objects. *Robotics and Computer-Integrated Manufacturing*, 28(2):154–163.

Johannsmeier, L. and Haddadin, S. (2017). A hierarchical human-robot interaction-planning framework for task allocation in collaborative industrial assembly processes. *IEEE Robotics and Automation Letters*, 2(1):21–48.

Karpas, E., Levine, S., Yu, P., and Williams, B. (2015). Robust execution of plans for human-robot teams. In *Proceedings of the 2015 International Conference on Automated Planning and Scheduling (ICAPS 2015)*, Jerusalem, Israel.

Koppula, H., Jain, A., and Saxena, A. (2013). Anticipatory planning for human-robot teams. In *Proceedings of Robotics: Science and Systems IX (RSS 2013)*, Berlin, Germany.

Krotzsch, M., Simancik, F., and Horrocks, I. (2013). A description logic primer. *arXiv:1201.4089v3*.

Krüger, J., Lien, T., and Verl, A. (2009). Cooperation of humans and machines in the assembly lines. *CIRP Annals - Manufacturing Technology*, 58(2):628–646.

Kwon, W. and Suh, I. (2014). Planning of proactive behaviours for human-robot cooperative tasks under uncertainty. *Knowledge-based Systems*, 72:81–95.

Lifschitz, V. (2019). *Answer set programming*. Springer Heidelberg.

Lipovetzky, N. and Geffner, H. (2011a). Searching for plans with carefully designed probes. In *Proceedings of the 2011 International Conference on Automated Planning and Scheduling (ICAPS 2011)*, Freiburg, Germany.

Lipovetzky, N. and Geffner, H. (2011b). Searching for plans with carefully designed probes. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 21, pages 154–161.

Liu, C. and Fisac, J. (2015). Goal inference improves objective and perceived performance in human-robot collaboration. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015)*, Istanbul, Turkey.

Liu, H. and Wang, L. (2017). Gesture recognition for human-robot collaboration: a review. *International Journal of Industrial Ergonomics*, - in press.

Logeswaran, L., Fu, Y., Lee, M., and Lee, H. (2022). Few-shot subgoal planning with language models. *arXiv preprint arXiv:2205.14288*.

Loshchilov, I. and Hutter, F. (2016). Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*.

Macció, S., Carfí, A., and Mastrogiovanni, F. (2022). A system for hierarchical planning in service mobile robotics. In *2022 IEEE International Conference on Robotics and Automation*.

Mastrogiovanni, F., Paikan, A., and Sgorbissa, A. (2013). Semantic-aware real-time scheduling in robotics. *IEEE Transactions on Robotics*, 29(1):118–135.

Mastrogiovanni, F., Sgorbissa, A., and Zaccaria, R. (2004). A system for hierarchical planning in service mobile robotics. In *Proceedings of the 8th Conference on Intelligent Autonomous Systems (IAS-8)*, Amsterdam, The Netherlands.

McDermott, D. (1998). PDDL – the Planning Domain Definition Language. Technical report, Yale.

Mikolov, T., Kombrink, S., Burget, L., Černockỳ, J., and Khudanpur, S. (2011). Extensions of recurrent neural network language model. In *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5528–5531. IEEE.

Miller, S., van den Berg, J., Fritz, M., Darrell, T., Goldberg, K., and Abbeel, P. (2011). A geometric approach to robotic laundry folding. *International Journal of Robotic Research*, 31(2):249–267.

Munzer, T., Mollard, Y., and Lopes, M. (2017). Impact of robot initiative on human-robot collaboration. In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction (HRI 2017)*, Vienna, Austria.

Murali, P. K., Darvish, K., and Mastrogiovanni, F. (2020). Deployment and evaluation of a flexible human-robot collaboration model based on and/or graphs in a manufacturing environment. *Intelligent Service Robotics*, 13:439–457.

Nair, A., Chen, D., Agraval, P., Isola, P., Abbeel, P., Malik, J., and Levine, S. (2017). Combining self-supervised learning and imitation for vision-based rope manipulation. In *Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA 2017)*, Singapore.

Oussidi, A. and Elhassouny, A. (2018). Deep generative models: Survey. In *2018 International conference on intelligent systems and computer vision (ISCV)*, pages 1–8. IEEE.

Pallagani, V., Muppasani, B., Srivastava, B., Rossi, F., Horesh, L., Murugesan, K., Loreggia, A., Fabiano, F., Joseph, R., Kethepalli, Y., et al. (2023). Plansformer tool: Demonstrating generation of symbolic plans using transformers. In *IJCAI*, volume 2023, pages 7158–7162. International Joint Conferences on Artificial Intelligence.

Prewett, M., Johnson, R., Saboe, K., Elliott, L., and Coovert, M. (2010). Managing workload in human-robot interaction: a review of empirical studies. *Computers in Human Behaviour*, 26(5):840–856.

Rintanen, J. (2014). Madagascar: scalable planning with SAT. In *Proceedings of the 2014 International Planning Competition (IPC 2014)*, Portsmouth, NH, USA.

Roncone, A., Mangin, O., and Scassellati, B. (2017). Transparent role assignment and task allocation in human robot collaboration. In *Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA 2017)*, Singapore.

Saadat, M. and Nan, P. (2002). Industrial applications of automatic manipulation of flexible materials. *Industrial Robot: an International Journal*, 29(5):434–442.

Sanderson, A. C., Peshkin, M. A., and de Mello, L. S. H. (1988). Task planning for robotic manipulation in space applications. *IEEE Transactions on Aerospace and Electronic Systems*, 24(5):619–629.

Scao, T. L., Fan, A., Akiki, C., Pavlick, E., Ilić, S., Hesslow, D., Castagné, R., Luccioni, A. S., Yvon, F., Gallé, M., et al. (2022). Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*.

Schulman, J., Ho, J., Lee, C., and Abbeel, P. (2016). Learning from demonstrations through the use of non-rigid registration. In *M. Inaba and P. Corke (Eds.) Robotics Research*, volume 114 of *Springer Tracts in Advanced Robotics*. Springer International Publishing, Lausanne, Switzerland.

Sebastiani, E., Lallement, R., Alami, R., and Iocchi, L. (2017). Dealing with on-line human-robot negotiations in hierarchical agent-based task planner. In *Proceedings of the 2017 International Conference on Automated Planning and Scheduling (ICAPS 2017)*, Pittsburgh, USA.

Silver, T., Dan, S., Srinivas, K., Tenenbaum, J. B., Kaelbling, L. P., and Katz, M. (2023). Generalized planning in pddl domains with pretrained large language models. *arXiv preprint arXiv:2305.11014*.

Silver, T., Hariprasad, V., Shuttleworth, R. S., Kumar, N., Lozano-Pérez, T., and Kaelbling, L. P. (2022). Pddl planning with pretrained large language models. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*.

Singh, I., Blukis, V., Mousavian, A., Goyal, A., Xu, D., Tremblay, J., Fox, D., Thomason, J., and Garg, A. (2023). Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530. IEEE.

Smith, C., Karayiannidis, Y., Nalpantidis, L., Gratal, X., Qi, P., Dimarogonas, D., and Kragic, D. (2012). Dual arm manipulation: a survey. *Robotics and Autonomous Systems*, 60(10):1340–1353.

Smith, S., Patwary, M., Norick, B., LeGresley, P., Rajbhandari, S., Casper, J., Liu, Z., Prabhumoye, S., Zerveas, G., Korthikanti, V., et al. (2022). Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*.

Song, C. H., Wu, J., Washington, C., Sadler, B. M., Chao, W.-L., and Su, Y. (2023). Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2998–3009.

Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S., and Abbeel, P. (2014). Combined task and motion planning through an extensible planner-independent interface layer. In *Proceedings of the 2014 IEEE International Conference on Robotics and Automation (ICRA 2014)*, Hong Kong, China.

Sundermeyer, M., Schlüter, R., and Ney, H. (2012). Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*.

Sussman, G. J. (1973). A computational model of skill acquisition.

Thoppilan, R., De Freitas, D., Hall, J., Shazeer, N., Kulshreshtha, A., Cheng, H.-T., Jin, A., Bos, T., Baker, L., Du, Y., et al. (2022). Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. (2023). Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Tsarouchi, P., Matthaiakis, A., Makris, S., and Chryssolouris, G. (2016). Human-robot interaction review and challenges on task planning and programming. *International Journal of Computer Integrated Manufacturing*, 29(8):916–931.

Valmeekam, K., Olmo, A., Sreedharan, S., and Kambhampati, S. (2022). Large language models still can't plan (a benchmark for llms on planning and reasoning about change). *arXiv preprint arXiv:2206.10498*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Wakamatsu, H., Arai, E., and Hirai, S. (2006). Knotting and unknotting manipulation of deformable linear objects. *International Journal of Robotic Research*, 25(4):371–395.

Wake, N., Kanehira, A., Sasabuchi, K., Takamatsu, J., and Ikeuchi, K. (2023). Chatgpt empowered long-step robot control in various environments: A case application. *arXiv preprint arXiv:2304.03893*.

Wang, B., Wang, Z., Wang, X., Cao, Y., Saurous, R. A., and Kim, Y. (2023). Grammar prompting for domain-specific language generation with large language models. *arXiv preprint arXiv:2305.19234*.

Wang, Y., Wang, W., Joty, S., and Hoi, S. C. (2021). Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859*.

Wilcox, R., Nikolaidis, S., and Shah, J. (2012). Optimisation of temporal dynamics for adaptive human-robot interaction in assembly manufacturing. In *Proceedings of Robotics: Science and Systems VIII (RSS 2012)*, Sydney, Australia.

Yamakawa, Y., Namiki, A., and Ishikawa, M. (2013). Dynamic high-speed knotting of a rope by a manipulator. *International Journal of Advanced Robotic Systems*, 10:1–12.

Appendix A

Planning domains

A.1 Common definitions

```
(define (domain joint_bar)
(:requirements :strips :equality :typing :adl :conditional-effects)

(:types link joint angle gripper)

(:predicates
    (connected ?j - joint ?l - link)
    (angle-before ?a - angle ?a1 - angle)
    (angle_joint ?a - angle ?j - joint)
    (link-before ?l - link ?l1 - link)
    (grasp ?g - gripper ?l - link)
    (in-hand ?l - link)
    (in-centre ?j - joint)
    (free ?g - gripper)
    (affected ?j -joint ?l - link ?j1 - joint)
)
```

A.2 `m_0` action set

```
(:action move-link-to-central
    :parameters (?link1 - link ?joint1 ?joint2 - joint ?g2 - gripper)
    :precondition (and
        (in-centre ?joint1)
        (connected ?joint2 ?link1)
        (free ?g2)
        (not (in-hand ?link1)))
```

```
    :effect (and
        (not (in-centre ?joint1))
        (in-centre ?joint2)
    )
)

(:action take-links-to-move
:parameters (?link1 ?link2 - link ?joint - joint ?g2 ?g1 - gripper)
:precondition (and
    (in-centre ?joint)
    (free ?g2)
    (free ?g1)
    (not (in-hand ?link1))
    (not (in-hand ?link2))
    (not (= ?g1 ?g2))
    (connected ?joint ?link1)
    (connected ?joint  ?link2)
    (not (= ?link1 ?link2))
)
:effect (and
    (not (free ?g2))
    (not (free ?g1))
    (in-hand ?link1)
    (in-hand ?link2)
    (grasp ?g2 ?link2)
    (grasp ?g1 ?link1))
)


(:action increase_angle_first_child
:parameters (?link1 ?link2 - link ?joint - joint
             ?a1 ?a2 - angle ?g2 ?g1 - gripper)
:precondition (and
    (link-before ?link2 ?link1)
    (in-centre ?joint)
    (grasp ?g2 ?link2)
    (in-hand ?link1)
    (in-hand ?link2)
    (grasp ?g1 ?link1)
    (connected ?joint ?link1)
    (connected ?joint  ?link2)
    (not (= ?link1 ?link2))
```

```
    (angle_joint ?a1 ?joint)
    (angle-before ?a1 ?a2)
    )
:effect
    (and
        (not (angle_joint ?a1 ?joint))
        (angle_joint ?a2 ?joint)
        (forall (?js - joint ?a3 ?a4 - angle )
            (when (and (affected ?js ?link1 ?joint)
                       (not (= ?js ?joint))
                       (angle_joint ?a3 ?js)
                       (angle-before ?a3 ?a4) )
                (and
                    (not (angle_joint ?a3 ?js))
                    (angle_joint ?a4 ?js)
                )
            )
        )
    )
)

(:action decrease_angle_first_child
:parameters (?link1 ?link2 - link ?joint - joint
             ?a1 ?a2 - angle ?g2 ?g1 - gripper)
:precondition (and
    (link-before ?link2 ?link1)
    (in-centre ?joint)
    (grasp ?g2 ?link2)
    (in-hand ?link1)
    (in-hand ?link2)
    (grasp ?g1 ?link1)
    (connected ?joint ?link1)
    (connected ?joint  ?link2)
    (not (= ?link1 ?link2))
    (angle_joint ?a1 ?joint)
    (angle-before ?a2 ?a1)
    )
:effect
    (and
        (not (angle_joint ?a1 ?joint))
        (angle_joint ?a2 ?joint)
        (forall (?js - joint ?a3 ?a4 - angle )
```

```
            (when (and (affected ?js ?link1 ?joint)
                  (not (= ?js ?joint))
                  (angle_joint ?a3 ?js)
                  (angle-before ?a4 ?a3))
                (and
                    (not (angle_joint ?a3 ?js))
                    (angle_joint ?a4 ?js)
                )
            )
        )
    )
)

(:action release-links
:parameters (?link1 ?link2 - link ?joint - joint ?g2 ?g1 - gripper)
:precondition (and
    (grasp ?g2 ?link2)
    (in-hand ?link1)
    (in-hand ?link2)
    (grasp ?g1 ?link1)
    (connected ?joint ?link1)
    (connected ?joint  ?link2)
    (not (= ?link1 ?link2))
    )
:effect (and
    (free ?g2)
    (free ?g1)
    (not (in-hand ?link1))
    (not (in-hand ?link2))
    (not (grasp ?g2 ?link2))
    (not (grasp ?g1 ?link1)))
    )
)
```

A.3 m_25_B action set

```
(:action take-links-to-move
:parameters (?link1 ?link2 - link ?joint - joint ?g2 ?g1 - gripper)
:precondition (and
    (in-centre ?joint)
    (free ?g2)
    (free ?g1)
    (not (in-hand ?link1))
    (not (in-hand ?link2))
    (not (= ?g1 ?g2))
    (connected ?joint ?link1)
    (connected ?joint  ?link2)
    (not (= ?link1 ?link2))
)
:effect (and
    (not (free ?g2))
    (not (free ?g1))
    (in-hand ?link1)
    (in-hand ?link2)
    (grasp ?g2 ?link2)
    (grasp ?g1 ?link1))
)

(:action increase_angle_first_child
:parameters (?link1 ?link2 - link ?joint - joint ?a1 ?a2 - angle ?g2 ?g1 - gripper)
:precondition (and
    (link-before ?link2 ?link1)
    (in-centre ?joint)
    (grasp ?g2 ?link2)
    (in-hand ?link1)
    (in-hand ?link2)
    (grasp ?g1 ?link1)
    (connected ?joint ?link1)
    (connected ?joint  ?link2)
    (not (= ?link1 ?link2))
    (angle_joint ?a1 ?joint)
    (angle-before ?a1 ?a2)
    )
:effect
    (and
        (not (angle_joint ?a1 ?joint))
        (angle_joint ?a2 ?joint)
```

```
        (forall (?js - joint ?a3 ?a4 - angle )
            (when (and (affected ?js ?link1 ?joint)
                       (not (= ?js ?joint))
                       (angle_joint ?a3 ?js)
                       (angle-before ?a3 ?a4))
                (and
                    (not (angle_joint ?a3 ?js))
                    (angle_joint ?a4 ?js)
                )
            )
        )
    )
)

(:action decrease_angle_first_child
:parameters (?link1 ?link2 - link ?joint - joint ?a1 ?a2 - angle ?g2 ?g1 - gripper)
:precondition (and
    (link-before ?link2 ?link1)
    (in-centre ?joint)
    (grasp ?g2 ?link2)
    (in-hand ?link1)
    (in-hand ?link2)
    (grasp ?g1 ?link1)
    (connected ?joint ?link1)
    (connected ?joint  ?link2)
    (not (= ?link1 ?link2))
    (angle_joint ?a1 ?joint)
    (angle-before ?a2 ?a1)
    )
:effect
    (and
        (not (angle_joint ?a1 ?joint))
        (angle_joint ?a2 ?joint)
        (forall (?js - joint ?a3 ?a4 - angle )
            (when (and (affected ?js ?link1 ?joint)
                       (not (= ?js ?joint))
                       (angle_joint ?a3 ?js)
                       (angle-before ?a4 ?a3))
                (and
                    (not (angle_joint ?a3 ?js))
                    (angle_joint ?a4 ?js)
                )
```

```
            )
        )
    )
)


(:action release-links
:parameters (?link1 ?link2 - link ?joint - joint ?g2 ?g1 - gripper)
:precondition (and
    (grasp ?g2 ?link2)
    (in-hand ?link1)
    (in-hand ?link2)
    (grasp ?g1 ?link1)
    (connected ?joint ?link1)
    (connected ?joint  ?link2)
    (not (= ?link1 ?link2))
)
:effect (and
    (free ?g2)
    (free ?g1)
    (not (in-hand ?link1))
    (not (in-hand ?link2))
    (not (grasp ?g2 ?link2))
    (not (grasp ?g1 ?link1)))
)

(:action link-to-central-grasp
:parameters (?link1 ?link2 - link ?joint1 ?joint2 - joint ?g2 ?g1 - gripper)
:precondition (and
    (in-centre ?joint1)
    (connected ?joint2 ?link1)
    (connected ?joint2 ?link2)
    (not (= ?link1 ?link2))
    (free ?g2)
    (free ?g1)
    (not (= ?g1 ?g2))
    (not (in-hand ?link1))
    (not (in-hand ?link2))
    )
:effect (and
    (not (in-centre ?joint1))
    (not (free ?g2))
```

```
    (not (free ?g1))
    (in-centre ?joint2)
    (grasp ?g2 ?link2)
    (grasp ?g1 ?link1)
    (in-hand ?link1)
    (in-hand ?link2))
)

(:action increase_angle_first_child_45
:parameters (?link1 ?link2 - link ?joint - joint ?a1 ?a2 ?a3 ?a4 - angle ?g2 ?g1 - gripper)
:precondition (and
    (link-before ?link2 ?link1)
    (in-centre ?joint)
    (grasp ?g2 ?link2)
    (in-hand ?link1)
    (in-hand ?link2)
    (grasp ?g1 ?link1)
    (connected ?joint ?link1)
    (connected ?joint  ?link2)
    (not (= ?link1 ?link2))
    (angle_joint ?a1 ?joint)
    (angle-before ?a1 ?a2)
    (angle-before ?a2 ?a3)
    (angle-before ?a3 ?a4)
    )
:effect
    (and
        (not (angle_joint ?a1 ?joint))
        (angle_joint ?a4 ?joint)
        (forall (?js - joint ?a5 ?a6 ?a7 ?a8 - angle )
            (when (and (affected ?js ?link1 ?joint)
                       (not (= ?js ?joint))
                       (angle_joint ?a5 ?js)
                       (angle-before ?a5 ?a6)
                       (angle-before ?a6 ?a7)
                       (angle-before ?a7 ?a8))
                (and
                    (not (angle_joint ?a5 ?js))
                    (angle_joint ?a8 ?js)
                )
            )
        )
```

```
    )
)

(:action decrease_angle_first_child_45
:parameters (?link1 ?link2 - link ?joint - joint ?a1 ?a2 ?a3 ?a4 - angle ?g2 ?g1 - gripper)
:precondition (and
    (link-before ?link2 ?link1)
    (in-centre ?joint)
    (grasp ?g2 ?link2)
    (in-hand ?link1)
    (in-hand ?link2)
    (grasp ?g1 ?link1)
    (connected ?joint ?link1)
    (connected ?joint  ?link2)
    (not (= ?link1 ?link2))
    (angle_joint ?a1 ?joint)
    (angle-before ?a2 ?a1)
    (angle-before ?a3 ?a2)
    (angle-before ?a4 ?a3)
    )
:effect
    (and
        (not (angle_joint ?a1 ?joint))
        (angle_joint ?a4 ?joint)
         (forall (?js - joint ?a5 ?a6 ?a7 ?a8 - angle )
            (when (and (affected ?js ?link1 ?joint)
                       (not (= ?js ?joint))
                       (angle_joint ?a5 ?js)
                       (angle-before ?a8 ?a7)
                       (angle-before ?a7 ?a6)
                       (angle-before ?a6 ?a5))
                (and
                    (not (angle_joint ?a5 ?js))
                    (angle_joint ?a8 ?js)
                )
            )
        )
    )
)
```