Acknowledgments

First of all, I would like to thank my Supervisor Prof. Marco Maratea. I would like to tank my tutor as well Prof. Fulvio Mastrogiovanni. They have been an inspired figure for me during the whole course of this "adventure". His advices and suggestions helped me become the person I actually am. Their knowledge inspired me and I can just consider myself lucky and honored for his consideration towards myself.

Among all of the people I've worked with, I would like to thank Alessio Capitanelli. His patient guidance in all of our works has been of great inspiration for me. Also, I would like to thank Prof. Mauro Vallati who embraced my research visiting period in Huddersfield. Furthermore, I would also to thank Ch.mo Prof. Nicola Leone, Coordinator of the Ph.D. programme in Mathematics and Computer Science.

~

Quando queste parole abiteranno una città di carta, questa tesi sarà indelebilmente terminata e degli occhi, adesso, staranno navigando queste parole. Qui, i miei ringraziamenti informali. Per iniziare, ringrazio la mia famiglia, incommensurabile costante della mia vita, matrice della mia esistenza. Ringrazio Nicolas, che conosco da sempre e che, essendo mio coinquilino mi ha sempre sopportato. Ringrazio Sofia, Vyshack, Francesco, Scott, Massimiliano, Andrea e Fabiana i miei amici conosciuti durante il mio Erasmus in francia e sempre pronti a sostenermi. Ringrazio Andrea, Alessio e Aleks cari amici sempre pronti ad essere un ottima fonte di distrazione. Ringrazio, fra tutti, Aldo, Francesco, Cinzia, Alessio, Pierangela, Roberta, Francesco e Jessica e tutte le altre persone che probabilmente, in questa rocambolesca valanga di emozioni, ho dimenticato. Infine, senza fine alcuna, ringrazio Lidia, per sostenuto e sopportato durante questi anni senza mai vacillare.

"Chi non dà nulla non ha nulla. La disgrazia più grande non è non essere amati, ma non amare."

— Albert Camus, "Taccuini 1935–1959"

Abstract

The manipulation of flexible object is of primary importance in industry 4.0 and in home environments scenarios. Traditionally, this problem has been tackled by developing ad-hoc approaches, that lack of flexibility and portability. We propose an approach in which a flexible object is modelled as an articulated object, or rather, a set of links connect via joints.

In this thesis we present a framework based on Answer Set Programming (ASP) for the automated manipulation of articulated objects in a robot architecture. In particular, ASP is employed for representing the configuration of the articulated object, for checking the consistency of the knowledge base, and for generating the sequence of manipulation actions. The framework is exemplified and validated on the Baxter dual-arm manipulator on a simple reference scenario in which we carried out different experiments analysing the behaviours of different strategies for the action planning module. Our aim is to have an understanding of the performances of these approaches with respect to planning time and execution time as well.

Then, we extend such scenario for having a higher accuracy of the setup, and to introduce a few constraints in robot actions execution to improve their feasibility. Given the extended scenario entails a high number of possible actions that can be fruitfully combined, we exploit macro actions from automated planning in the module that generates the sequence of actions, in order to deal with this extended scenario in a more effective way. Moreover, we analyse the possibilities of mixed encodings with both simple actions and macro actions from automated planning in different "concentrations".
We finally validate the framework also on this extended scenario, confirming the applicability of ASP also in this complex context, and showing the usefulness of macro actions in this robotics application.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contest and motivation. The manipulation of articulated objects is of primary importance in robotics and is one of the most complex robotics tasks [54, 64]. Traditionally, this problem has been tackled by developing ad-hoc approaches, that lack of flexibility and portability. The development of new software, algorithm and strategies, together with the improvements in the mechanical design for grippers and robotic hands, for autonomous robots with robust manipulation skills, can lead to breakthroughs in various applications, such as humanoid robots, horticulture harvesting grasping, human-robot interaction, planetary exploration, flexible manufacturing and much more. This gives the possibility of addressing some of the issues related to robotized work, such as mechanical design issues, control issues, modelling achievements and issues, applications in the industrial field and non-conventional applications (including, for example, service robotics and agriculture)[55, 64]. We view autonomous manipulation as the purposeful and deliberate change of the configuration of an object. In the past years, 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 but the issue is still not being fully addressed. In the literature, the problem of determining the two-dimensional (2D) configuration of articulated or flexible objects has received much attention in the past

few years [13, 79, 98] whereas the problem of obtaining a target configuration via manipulation has been explored in motion planning [9, 95, 100]. A limitation of such manipulation strategies is that they are often crafted specifically for the problem at hand, with the relevant characteristics of the object and robot capabilities being either hardcoded or assumed; thus, in these contexts generalisation property and scalability are somehow limited. Instead, the introduction of the Industry 4.0 paradigm is expected to redefine the nature of shop-floor environments along with many directions, including the role played by robots in the manufacturing process: one of the main tenets considered in Industry 4.0 is the increased customer satisfaction via a high degree of product personalization and just-in-time delivery. For this reason, a higher level of flexibility in manufacturing processes is needed to cope with such diversified demands, especially in low-automation tasks. Our approach automatizes the manipulation of articulated objects without any assumption neither on the characteristic of the articulated object (such as length of the links or stiffness of the joints) neither on the model of dual-arm robot used to manipulate the object.

State of the Art. The manipulation of articulated objects plays an important role in real-world robot tasks, both in-home and industrial environments [55, 64]. Attention has been paid to the development of approaches and algorithms for generating the sequence of movements a robot has to perform to manipulate an articulated object. In the literature, the problem of determining the two-dimensional configuration of articulated or flexible objects has received much attention in the past few years: [13] in which they introduce two PDDL formulations of the task, which rely on conceptually different representations of the orientation of the objects. Moreover, they present experiments involving several planners and increasing size objects demonstrate the effectiveness of the proposed models. [14] in which they propose an action planning and execution framework based on PDDL formulations and a knowledge representation module based on Web Ontology Language (OWL). In [79] they present a learning-based system where a robot takes as input a sequence of images of a human manipulating a rope from an initial to goal configuration and

outputs a sequence of actions that can reproduce the human demonstration. The human demonstration provides a high-level plan of what to do and the low-level inverse model is used to execute the plan. They show that by combining the high and low-level plans, the robot can successfully manipulate a rope into a variety of target shapes using only a sequence of human-provided images for direction. [98] propose a planning method for knotting/unknotting of deformable linear objects; their proposed method indicates that it is theoretically possible for any knotting manipulation of a linear object placed on a table to be realized by a one-handed robot with three translational degrees of freedom (DOF) and one rotational DOF. Degrees of freedom, in a mechanics context, are specific, defined modes in which a mechanical device or system can move. The number of degrees of freedom is equal to the total number of independent displacements or aspects of motion.

The problem of obtaining a target configuration via manipulation has been explored in motion planning as well: [9] describes an adaptable system which can perform manipulation operations, such as Peg-in-Hole or Laying-Down actions, with flexible objects that integrate visual tracking and shape reconstruction with a physical modelling of the materials and their deformations. [95], instead, in which they present a method, which they call trajectory transfer, for adapting a demonstrated trajectory from the geometry at training time to the geometry at test time. Trajectory transfer is based on non-rigid registration, which computes a smooth transformation from the training scene onto the testing scene. [100] based their strategy on manipulating the object at high-speed: by moving the robot at high-speed, we can assume that the dynamic behaviour of the flexible rope can be obtained by performing algebraic calculations of the high-speed robot motion.

A limitation of such manipulation strategies is that they are often crafted specifically for the problem at hand, with the relevant characteristics of the object and robot capabilities being either hardcoded or assumed; thus, in these contexts generalisation property and scalability are somehow limited.

Personal Contribution. In this thesis we present a framework based exclusively on Answer Set Programming (ASP) [3, 11, 82] for the automated manipulation of articulated objects in a robot 2D workspace. ASP is a general, prominent knowledge representation and reasoning language with roots in logic programming and non-monotonic reasoning [41]. A system composed by a Microsoft Kinect camera, QR codes and the ROS package Ar_Track_Alvar, an open-source AR tag tracking library, it is used to gather information about the articulated object. With the pose and orientation fo each link of the object we then employ ASP to represent the configuration of the articulated object. With an unsynchronised process, using the proprieties of ROS nodes, we use another ASP encoding to check the consistency of the knowledge base. Once the consistency has been ensured, another ASP encoding is used to generate the optimal sequence of manipulation actions that change the current object configuration to the desired one. Then, a low-level planner, in our case MoveIt!, it is used to perform the actions on the links. The framework is first exemplified and validated, on a Baxter dual-arm manipulator, on a simple reference scenario, that involves the manipulation of the articulated object on a 2D workspace with the possibility of performing simple operations, like rotating a joint. During this step we investigate different possible strategies to give a small overview of the ASP possibilities. Then, we extend the simple scenario to allow higher representation accuracy of the setup. Moreover, we introduce some constraints to improve the robot reliability while executing actions(i.e. the robot can now grasp an object link only if is placed in the centre of the robot workspace). Indeed, in this scenario, we include more information in the knowledge base representing better the robot and its workspace and we improve the pool of selectable actions as well. Since this extended scenario entails a high number of possible actions to be possibly executed, that can be fruitfully combined, we exploit macro actions [17, 18, 43] from automated planning in the module that generates the sequence of actions, to deal with this extended scenario more effectively. We aimed to reduce both planning and execution time removing idle robot time in between the actions. Roughly, macro actions are

sequences of "elementary" actions that, on the application viewpoint, would be useful to be performed in such a sequence and considered as a "single" action. In general, in robotics applications, this may be useful given that performing a sequence of multiple (low-level) actions usually takes more time than executing the relative macro. In our application scenario, in particular, we have defined macro actions considering sequences of actions that are often performed in sequence by the robot, thus meeting such desirable property. During this step, we explored different strategies involving simple actions, macro actions and various combination of the two. This gives an overview of ASP planning possibilities on more complicated and diverse environments. We finally validate the framework also on this extended scenario, confirming the applicability of the ASP methodology also in the more complex context, at the same time showing the utility of macro actions in this robotics application.

In Chapter 6, we exploit the language PDDL+ to operate on an articulated object in a three-dimensional space. PDDL+ is an extension of PDDL2.1 and it provides a more flexible model of continuous change through the use of autonomous processes and events. In our application, we model the articulated object with 3 DOF joints and links with variable length. We then exploited event and processes to model gravity with different formulations considering different trade-offs between modelling accuracy, planning performances and between human-readability and parsability by planners. Finally, we validate our application in simulation with a dual-arm Baxter robot on two sets of PDDL+ models for the task of automated, robot-based manipulation of articulated objects in a 3D workspace. Then, we give an overview analysis of the performances of several domain-independent PDDL+ planners on realistic articulated object manipulation tasks.

Related Works

Manipulation of Flexible/Articulated Objects The manipulation of articulated objects plays an important role in real-world robot tasks, both in-home and industrial envi-

ronments [55, 64]. Attention has been paid to the development of approaches and algorithms for generating the sequence of movements a robot has to perform to manipulate an articulated object. In the literature, the problem of determining the two-dimensional (2D) configuration of articulated or flexible objects has received much attention in the past few years. In [13, 14] a similar framework based on automated reasoning methodologies have been presented. Such framework employs PDDL language and automated planning engines for the planning module, and Description Logic (DL) solvers in the configuration module, where data are explicitly stored in an ontology, while we use a uniform language and approach (ASP-based) for the whole framework. Moreover, differently from most of our approaches, encodings and solvers employed in [13, 14] are not currently able to return shortest plans, which is otherwise important, given that in this context executing the actions can be expensive. In [61], instead, a custom-designed multi-robot platform is presented, focused on HRI in indoor service robot for understanding natural language requests. Planning is specified using the action language BC [65]. In [97] they propose a planning method for knotting/unknotting of deformable linear objects. They propose a topological description of the state of a linear object with four transitions operations. They demonstrate that it is theoretically possible to knot a linear object placed on a table with a one-handed robot with three translational DOF and one rotational DOF. Furthermore, the approach described in [97] plans using hard encoded states of the linear object. This means that it is no possible to get to configurations of the object that are not reachable using the encoded states. Our architecture gives up on the flexibility of the object to implement a more general approach that does not relay predetermined states. in [78] they propose a learning-based approach to associate the behaviour of a deformable object with a robot's actions, using self-supervision from large amounts of data gathered autonomously by the robot. Their method uses human-provided demonstrations as higher-level guidance: the demonstrations tell the robot what to do, while the learned model tells it how to do it. Differently from our architecture, there is no planning involved since it is a human

operator that gives instructions to the actions to perform and in which sequence. In [94] it is present a method for adapting a demonstrated trajectory from the geometry at training time to the geometry at test time. Trajectory transfer is based on non-rigid registration, which computes a smooth transformation from the training scene onto the testing scene. In this work the use a multi-step task by repeatedly looking up the nearest demonstration and then applying for trajectory transfer. Furthermore, The strategy described in [100] involves manipulating the object at high speed. By moving the robot at high-speed, they assume that the dynamic behaviour of the flexible rope can be obtained by performing algebraic calculations of the highspeed robot motion. Based on this assumption, a dynamic deformation model of the flexible rope it is derived. [7] present a method to manipulate deformable objects that do not require modelling and simulating deformation. This method is based on the concept of diminishing rigidity, which we use to quickly compute an approximation to the Jacobian of the deformable object. This Jacobian is used to drive the points within the deformable object towards a set of targets. In describing a technique to models the bending of a sheet of paper and the paper crease lines which allows the monitoring of the deformations. Moreover, they enabled an anthropomorphic robot to fold paper using a set of tactile- and vision-based closed loop controllers. However, in all these cases, manipulation actions are directly grounded on perceptual cues, such as the peculiar geometry of the object to deal with, assumed to be easy to identify robustly, or based on a priori known or learned information about the object to manipulate, e.g., its stiffness or other physical features [31, 37]. As a result, every time either the element that has to be manipulated or the manipulator changes, a new reasoner has to be developed from scratch. In [59] they present a feature representation based on a histogram of oriented wrinkles, to describe the shape variation of a highly deformable object like clothing. A precomputed visual feedback dictionary using an offline training phase that stores a mapping between these visual features and the velocity of the end-effector.

Manipulation in Motion Planning It is possible to find examples in which robots exhibit the capability of manipulating and operating on mobile parts of the environment, such as handles of different shapes [21], home furniture [62] or valves in search and rescue settings [80]. In [21] they design a task descriptor which encapsulates important elements of a task. They propose a method that enables a robot to decompose a demonstrated task into sequential manipulation primitives and construct a task descriptor and then they show how to transfer a task descriptor learned from one object to similar objects. [62] they èresent an automated assembly system that directs the actions of a team of heterogeneous robots in the completion of an assembly task. From an initial user-supplied geometric specification, the system applies reasoning about the geometry of individual parts to deduce how they fit together. In [80] an integrated valve-turning skill is presented. It only requires an operator to issue a supervisory command to launch valve identification, motion planning, biped locomotion and valve manipulation. However, the employed manipulation strategies are often crafted specifically for the problem at hand, with the relevant characteristics of the object and robot capabilities being either hardcoded or assumed, thus undermining generalisation and scalability. A structured approach to perception, representation and reasoning, as well as execution, seems beneficial: on the one hand, we can decouple perception and representation issues, thus not being tied to specific perception approaches or ad hoc solutions; on the other hand, domain knowledge and reasoning logic can be separated, with the advantages of an increased maintainability, and the possibility to interchange reasoners and models in a modular way.

Reasoning About Actions Actions when executed often change the state of the world. Reasoning about actions helps us to predict if a sequence of actions is indeed going to achieve some goal that we may have; it allows us to plan or come up with a sequence of actions that would achieve a particular goal and maintain particular trajectories; it allows us to explain observations in terms of what actions may have taken place, and it allows us to diagnose faults in a system in terms of finding what actions may have taken place to result in

the faults [5]. [72] points out that, in ontologies, semantic constraints abstract from the way some fact about the case at hand may be represented properly. These semantic constraints can be computed from stored data, in the process implementation. In [6, 48] ASP can be used for verification of properties with Bounded Model Checking This is true if the actions are expressed in an extension of Linear Temporal Logic, of an action domain modelled in terms of fluents, action laws providing direct effects of actions, and causal laws. In [45] they combine Answer Set Programming with Dynamic Linear Time Temporal Logic to define a temporal logic programming language for reasoning about complex actions and infinite computations. Moreover, in [46] they present a framework that can be used for reasoning on business processes. In this work, they show that ASP can be used for verifying process properties in temporal logic. In [47] it is showed that reasoning about actions performed in ASP can rely on domain knowledge in a low-complexity Description Logic. In [44] they describe an approach to process modelling and semantic analysis that can exploit terminological knowledge in relying upon process activities to semantic constraints, via the definition of effects and preconditions of activities, and domain knowledge that relates such effects to the terms used in semantic constraints

Macro Operators An important line of research in AI planning focuses on increasing efficiency of the planning process by reformulating the domain knowledge, to obtain models that are more amenable for automated reasoners. Significant work has been done in the area of the reformulation for improving the performance of domain-independent planners. *Macro-operators* [16, 63, 76, 81] are one of the best-known types of reformulation in classical planning; they encapsulate in a single planning operator a sequence of "original" operators. Technically, an instance of a macro is applicable in a state if and only if a corresponding sequence of operators' instances is applicable in that state and the result of the application of the macro's instance is the same as the result of the application in the corresponding sequence of operators' instances. Informally speaking, macros can be understood as shortcuts in the search space allowing planning engines to generate plans in fewer steps.

In automated planning, macros have been proven to be effective in domains where some actions are likely to be always executed in the same sequence, or in cases where critical sections can be identified, i.e., where there are activities that need to use a limited resource [17]. Notably, the notion of macros can also be exploited by specifically enhanced planning reasoners. This is the case for MacroFF SOL-EP version [10] which can exploit offline extracted and ranked macros, and Marvin [19] that generates macros online by combining sequences of actions previously used for escaping plateaus. Such systems can efficiently deal with drawbacks of specific planning engines, in this case, the FF planner [56]; however, their adaptability for different planning engines might below. In this work, we aimed at exploiting macros in ASP following the more traditional solver-independent approach, i.e., by modifying the encoding, replacing simple actions with macros.

Thesis structure. This document is structured as follows. First, Chapter 2 gives an overview of the *Answer Set Programming* language and of its employment in automated planning. Moreover, we explain in this Chapter the concept of macros both in general and with respect to Answer Set Planning. After, Chapter 3 explain in detail the scenarios models we have been working on, both the simple and the extended scenario, while Chapter 4 gives an insight of all the software tools we used to implement our architecture with all the hardware that was necessary as well. Moreover, explains in deep each module composing our robot-based architecture. In Chapter 5 we present and discuss the results of our experiments. In this Chapter, inside Section 5.1, we also discuss the validation on both the scenarios: the simple reference scenario and the extended reference scenario using a Baxter dual-arm manipulator. Then, Chapter 6 introduced the implementation and results of our PDDL+ implementation on the manipulation of articulated objects in a 3D space and it explains how we modelled gravity. Finally, Chapter 8 summarize our conclusions and discuss future works with a focus on the extendability of our strategy to other robots.

Chapter 2

Answer Set Programming

2.1 Introduction

Answer Set Programming (ASP),[4, 74] referred to also as Disjunctive Logic Programming under the stable model semantics (DLP), is a powerful formalism for Knowledge Representation and Reasoning. Bloomed from the work of Gelfond, Lifschitz [29, 42] and Minker [15, 35] in the 1980s, it has enjoyed a continuously increasing interest within the scientific community. One of the main reasons for the success of ASP is the high expressive power of its language: ASP programs, indeed, allow us to express, in a precise mathematical sense, every property of finite structures over a function-free first-order structure that is decidable in nondeterministic polynomial time with an oracle in NP [22, 30] (i.e., ASP captures the complexity class ΣP 2 = NPNP). Thus, ASP allows us to encode also programs which cannot be translated to SAT in polynomial time.
Moreover, ASP is fully declarative (the ordering of literals and rules is immaterial), and the ASP encoding of a large variety of problems is very concise, simple, and elegant [29].

Example 1. Consider the 3-Colourability problem, a well-known NP-complete problem. Given a graph, the problem is to decide whether there exists an assignment of one of

three colours (say, red, green, or blue) to each node such that adjacent nodes always have different colours. Suppose that the graph is represented by a set of facts F using a unary predicate node(X) and a binary predicate arc(X, Y). Then, the following ASP program (in combination with F) computes all 3-Colorings (as stable models) of that graph.

$$\begin{aligned} r_1\textbf{:}\quad & color(X, red) \vee color(X, green) \vee color(X, blue) : -node(X). \\ r_2\textbf{:}\quad & : -color(X_1, C), color(X_2, C), arc(X_1, X_2). \end{aligned} \tag{2.1}$$

Rule r_1 expresses that each node must either be coloured red, green, or blue; due to minimality of the answer sets models, a node cannot be assigned more than one colour. The subsequent integrity constraint checks that no pair of adjacent nodes (connected by an arc) is assigned the same colour d the answer sets of $F \cup \{r_1, r_2\}$. The graph is 3-colourable if and only if $F \cup \{r_1, r_2\}$ has some answer set.

Unfortunately, the high expressiveness of ASP comes at the price of a high computational cost in the worst case, which implements efficient systems a difficult task. Nevertheless, starting from the second half of the 1990s, and even more in the latest years, several efficient ASP systems have been released [28, 67], that encouraged a number of applications in many real-world and industrial contexts [57, 68]. These applications have confirmed the viability of the ASP exploitation for advanced knowledge-based tasks and stimulated further research in this field. The Italian research community-produced, in the latest 30 years, a significant contribution in the area, addressing the whole spectrum of issues cited above; this contribution ranged from theoretical results and characterizations [51, 69], to practical applications [57, 68, 90], stepping through language extensions [20, 67], evaluation algorithms and optimization techniques [73, 85]. Several of the achieved results are widely recognized as milestones on the road to the current state of the art; this is, for instance, the case of the DLV project [67], that produced one of the world-leading ASP systems.

2.2 The Language

2.2.1 Syntax

Following as standard convention the Prolog syntax, strings starting with uppercase letters denote logical variables, while strings starting with lower case letters denote constants. A term is either a variable or a constant. An atom is an expression $p(t_1, ..., t_n)$, where p is a predicate of arity n and $t_1, ..., t_n$ are terms. A literal l is either an atom p (positive literal) or its negation $not\ p$ (negative literal). Two literals are said to be complementary if they are of the form p and $not\ p$ for some atom p. Given a literal l, $not.l$ denotes its complementary literal. Accordingly, given a set L of literals, $not.L$ denotes the set $\{not.l | l \in \mathrm{L}\}$. A set L of literals is said to be consistent if, for every literal $l \in L$, its complementary literal is not contained in L. A disjunctive rule (rule, for short) r is a construct:

$$a_1 \vee \cdots \vee a_n :- b_1, \ldots, b_k, not\ b_{k+1}, \ldots, not\ b_m. \tag{2.2}$$

where $a_1, \ldots, a_n$ and $b_1, \ldots, b_m$ are literals for $n \geqslant 0$, $m \geqslant 0$ and $k \geqslant 0$. The disjunction $a1 \vee \cdots \vee an$ is called the head of r, while the conjunction $b_1 \ldots b_k, not\ b_{k+1} \ldots not\ b_m$ is referred to as the body of r. A rule without head literals (i.e. $n = 0$) is usually referred to as an integrity constraint. A rule having precisely one head literal (i.e. $n = 1$) is called a normal rule. If the body is empty (i.e. $k = m = 0$), it is called a fact, and in this case the " :- " sign is usually omitted. The following notation will be useful for further discussion. If r is a rule of form (1), then $H(r) = \{a_1, \ldots, a_n\}$ is the set of literals in the head and $B(r) = B^+(r) \cup B^-(r)$ is the set of the body literals, where $B^+(r)$ (the positive body) is $\{b1, \ldots, bk\}$ and $B^-(r)$ (the negative body) is $\{bk + 1, \ldots, bm\}$. An ASP program (also called Disjunctive Logic Program or DLP program) P is a finite set of rules.

A not-free program P (i.e., such that $\forall r \in P : B^-(r) = \emptyset$) is called *positive or Horn*: in positive programs negation as failure (not) does not occur. Instead a v-free program P (i.e., such that $\forall r \in P : |H(r)| \leqslant 1$) is called normal logic program.

It is important to notice that in ASP rules in programs are usually required to be safe: this safety request comes from the databases field (for a detailed discussion, we refer to [77]). A rule r is safe if each variable in r also appears in at least one positive literal in the body of r. An ASP program is safe if each of its rules is safe; in the following, we will only consider safe programs.
A term, an atom, a rule or a program, is called ground if no variable appears in it.

2.2.2 Semantics

We next describe the semantics of ASP programs, which is based on the answer set semantics originally defined in [29]. However, different from [29] only consistent answer sets are considered, as it is now standard practice. In ASP the availability of some pre-interpreted predicates is assumed, such as $=$, $<$, $>$. However, it would also be possible to define them explicitly as facts, so that they are not treated differently.

Herbrand Universe and Literal Base. For any program P, the Herbrand universe, denoted by U_P, is the set of all constants occurring in P. If no constant occurs in P, U_P consists of one arbitrary constant. The Herbrand literal base B_P is the set of all ground literals constructible from predicate symbols appearing in P and constants in U_P.

Ground Instantiation. For any rule r, $Ground(r)$ denotes the set of rules obtained by replacing each variable in r by constants in U_P in all possible ways. For any program P, its ground instantiation is the set $grnd(P) = \Cup_{r \in P} Ground(r)$. It is important to notice that for propositional programs, $P = grnd(P)$ holds.

Answer Sets. For every program P, its answer sets are defined by using its ground instantiation $grnd(P)$ in two steps: first the answer sets of positive disjunctive programs are defined, then the answer sets of general programs are defined by a reduction to positive disjunctive programs and a stability condition. An interpretation I is a consistent set of ground literals $I \subseteq B_P$ w.r.t. a program P.

Given a program P and a (consistent) interpretation $I \subseteq B_p$, a rule $h_1 | \ldots | h_m :- b_1, \ldots, b_n.$ in $grnd(P)$ is satisfied w.r.t. I if some $h \in \{h_1, \ldots, h_m\}$ is true w.r.t. I when $b_1, \ldots, b_n$ are true w.r.t. I; I is a model of P if every rule in $grnd(P)$ is satisfied w.r.t. I. The `reduct` of P w.r.t. I, denoted by P^I, consists of the rules $h_1 | \ldots | h_m :- b_1, \ldots, b_n.$ in $grnd(P)$ such that $b_1, \ldots, b_n$ are true w.r.t. I; I is an answer set of P if I is a $\subseteq$ minimal model of P^I. In other words, an answer set I of P is a model of P such that no proper subset of I is a model of P^I.The semantics of P is given by the collection of its answer sets, denoted by $AS(P)$.

A consistent interpretation $X \subseteq B_P$ is called closed under P (where P is a positive disjunctive datalog program), if, for every $r \in grnd(P)$, $H(r) \cap X \neq \emptyset$ whenever $B(r) \subseteq X$. An interpretation which is closed under P is also called model of P. An interpretation $X \subseteq B_P$ is an answer set for a positive disjunctive program P, if it is minimal (under set inclusion) among all (consistent) interpretations that are closed under P.

Example 2. The positive program $P_1 = \{a \vee b \vee c.\}$ has the answer sets $\{a\}$, $\{b\}$, and $\{c\}$; they are minimal and correspond to the multiple ways of satisfying the disjunction. Its extension $P_2 = P_1 \cup \{: -a.\}$ has the answer sets $\{b\}$ and $\{c\}$: comparing P_2 with P_1, the additional constraint is not satisfied by interpretation $\{a\}$. Moreover, the positive program $P_3 = P_2 \cup \{b : -c.,\ c : -b.\}$ has the single answer set $\{b, c\}$ (indeed, the remaining consistent closed interpretation $\{a, b, c\}$ is not minimal). Finally, it is easy to see that, $P_4 = P_3 \cup \{: -c.\}$ has no answer set.

Example 3. For the negative ground program $P_5 = \{a : -not\ b.\}, A = \{a\}$ is the only answer set, as $P_5^A = \{a.\}$. For example for $B = b$, $P_5^B = \emptyset$, and so B is not an answer set.

Choice rules, normal rules, and integrity constraint To ease the use of ASP in practice, several extensions have been developed. First of all, rules with variables are viewed as shorthands for the set of their ground instances. For $a \in$ A, where A is the set of all literals, a choice rule r is of the form $\{a\} \leftarrow B(r)$ and stands for the pair of rules $a \leftarrow B(r)$, *not* a' and $a' \leftarrow B(r)$, *not* a where a' is a new symbol associated with a. We define $H(r) = \{a\}$ for a choice rule as r. Further language constructs include conditional literals and cardinality constraints. The former are of the form $a : b1, \ldots, bm$, the latter can be written as $s\ \{d1; \ldots; dn\}\ t$, where a and bi are possibly negated (regular) literals and each dj is a conditional literal; s and t provide optional lower and upper bounds on the number of satisfied literals in the cardinality constraint. We refer to $b1, \ldots, bm$ as a condition. The practical value of both constructs becomes apparent when used with variables. For instance, a conditional literal like $a(X) : b(X)$ in a rule's antecedent expands to the conjunction of all instances of $a(X)$ for which the corresponding instance of $b(X)$ holds. Similarly, $2\ \{a(X) : b(X)\}\ 4$ is true whenever at least two and at most four instances of $a(X)$, subject to $b(X)$, are true. Finally, assignments are of form $t = d1; \ldots; dn$. E.g., $Y = a(X) : b(X)$ is true if Y equals the number of satisfied instances of $a(X) : b(X)$. We identify the choice rules, normal rules, and integrity constraint in a logic program P by means of $c(P)$, $n(P)$, and $i(P)$, respectively. We define a program P as normalized, if it can be partitioned into $c(P)$, $n(P)$, and $i(P)$, such that $B(r) = \emptyset$ for each $r \in c(P)$ and $n(P)$ is stratified [87].

Heuristic In computer science, artificial intelligence, and mathematical optimization, a heuristic is a technique designed for solving a problem more quickly when classic methods are too slow, or for finding an approximate solution when classic methods fail to find any

exact solution. This is achieved by trading optimality, completeness, accuracy, or precision for speed. In a way, it can be considered a shortcut. In our case, it is used to reduce the size of the state search in order to find faster the answer set. A heuristic function, also called simply a heuristic, is a function that ranks alternatives in search algorithms at each branching step based on available information to decide which branch to follow. For example, it may approximate the exact solution.

The objective of a heuristic is to produce a solution in a reasonable time frame that is good enough for solving the problem at hand. This solution may not be the best of all the solutions to this problem, or it may simply approximate the exact solution. But it is still valuable because finding it does not require a prohibitively long time. Heuristics may produce results by themselves, or they may be used in conjunction with optimization algorithms to improve their efficiency. Results about NP-hardness in theoretical computer science make heuristics the only viable option for a variety of complex optimization problems that need to be routinely solved in real-world applications. Heuristics underlie the whole field of Artificial Intelligence and the computer simulation of thinking, as they may be used in situations where there are no known algorithms.

Early static heuristics (e.g. Jeroslaw-Wang [58], Literal Count [75]) picked the next variable based on the number of appearances (scores) of different variables in unsatisfied clauses. A major drawback of such an approach is that score calculation requires visiting all the clauses at each node of the search three built from the solver, which implies a very significant overhead. Another disadvantage of static heuristics is that they do not consider information that can be retrieved after a conflict during implication graph analysis. Heuristics based upon such analyses were found to be several orders of magnitude faster [50, 77]. The first dynamic heuristic is called Variable State Independent Decaying Sum (VSIDS) [77]. According to VSIDS, each literal is associated with a counter $cl(p)$, whose value is increased once a new clause containing p is added to the database. Counters are initialized to 0. Every once in a while, all counters are halved. The next literal to be picked is the

one with the largest counter. Ties are broken randomly. Two major advantages of VSIDS over the previous heuristics are that: (1) VSIDS is characterized by a negligible computational cost; (2) VSIDS gives preference to literals that participate in recent conflicts, i.e. it is dynamic. Another well-known decision heuristic, which proved to be even more successful than VSIDS, is MiniSat SAT solver. It implements a variant of VSIDS that, instead of infrequent halving of the scores, MiniSat multiplies the scores after each conflict by 0.95 making the heuristic more dynamic. Based on MiniSat it exists *nOPTSAT optsat* [24, 49]: it allows for constraints, expressed as preferences, to be partially ordered and in many applications, the number of preferences is relatively low. It extends DPLL to compute one optimal model of a set of clauses with respect to a qualitative preference on literals. Therefore, it allows for inconsistent sets of preferences and a partial order on the preferred literals.

2.3 Answer Set Programming in Planning

2.3.1 Introduction

The term planning is commonly referred to as the explicit deliberation process that chooses and organizes actions by anticipating their outcomes and that aims at achieving some prestated objectives. In particular, planning in artificial intelligence is the computational study of this deliberation process and it is referred to as automated planning. Automated planning is that branch of artificial intelligence that concerns the realization of strategies or action sequences, typically for execution by intelligent agents, autonomous robots and unmanned vehicles. Unlike classical control and classification problems, the solutions are complex and must be discovered and optimized in multidimensional space. It is important to underline that AI planning is also related to decision theory.

In known environments with available models, planning can be done offline and the solutions can be found and evaluated prior to execution. In dynamically unknown environ-

ments, the strategy often needs to be revised online, indeed, models and policies must be adapted. Solutions usually resort to iterative trial and error processes commonly seen in artificial intelligence. These include dynamic programming, reinforcement learning and combinatorial optimization. Languages used to describe planning are often called action languages. Given a description of the possible initial states of the world, a description of the desired goals, and a description of a set of possible actions, the planning problem is to synthesise a plan that is guaranteed (when applied to any of the initial states) to generate a state which contains the desired goals (such a state is called a goal state). An action language is a language for specifying state transition systems and is commonly used to create formal models of the effects of actions on the world. Action languages are commonly used in the artificial intelligence and robotics domains, where they describe how actions affect the states of systems over time, and may be used for automated planning. Action languages fall into two classes: action description languages and action query languages. Examples of the former include STRIPS, PDDL, Language A (a generalization of STRIPS), Language B (an extension of A) and Language C. There are also the Action Query Languages P, Q and R and, finally, Answer Set Programming (ASP). Indeed, since modern answer-set solvers make use of boolean SAT algorithms to very rapidly ascertain satisfiability, this implies that action languages can also take advantage of the progress being made in the domain of boolean SAT solving.

Answer Set Planning Answer Set Planning was one of the first challenging applications of Answer Set Programming (ASP). However, when putting plans into practice, their execution and monitoring lead to an increased complexity. Foremost, this is due to the inherent incompleteness of information faced in real-world scenarios. This fundamental problem has already been addressed in various ways, as in conformant, assertional, and assumption-based planning, and often combined with additional sensing actions.

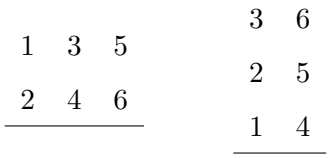


Figure 2-1: A planning problem in the blocks world

In a planning problem, one looks for a sequence of actions that leads from a given initial state to a given goal state. Turning one configuration of blocks into another is, indeed, a planning problem. We will use this example to illustrate some ideas of answer set planning. To specify a planning problem completely, it is necessary to state which actions are allowed in a plan. When a block is "clear", or rather there is nothing on top of it, it is assumed that it can be placed on top of any tower of blocks or on the table. Thus moving a block located in the middle of a tower, along with everything on top of it, is prohibited. To make the blocks world more interesting, let us assume that the robot that will act on the blocks has two grippers that can move blocks independently. For instance, in the initial state shown in Figure 2.3.1, blocks 1 and 3 can be moved concurrently. Since a block can be moved only when it is clear, moving blocks 1 and 2 concurrently is impossible. Besides, we assume that the robot is unable to move b onto b' if b' is being moved also. Even with these restrictions, the planning problem above can be solved in 3 steps:

1. Place blocks 1 and 3 on the table.

2. Place block 2 on block 1 and block 5 on block 4.

3. Place block 3 on block 2 and block 6 on block 5.

Research on planning has two components: representation (design of declarative languages for specifying planning problems) and search (design of planning algorithms). An important class of planning algorithms is based on the idea of reducing a planning problem to the problem of computing a satisfying interpretation for a set of propositional formulas.

This is known as satisfiability planning.

An important advantage of answer set planning is that the representation of properties of actions is easier when logic programs are used instead of classical logic, given the non-monotonic character of negation as failure. The idea of answer set planning is due to Subrahmanian and Zaniolo [96], and the results of computational experiments that use smodels to compute answer sets are reported in [26, 83].

The key element of answer set planning is the representation of a dynamic domain such as the blocks world in the form of a "history program" a program whose answer sets represent possible "histories", or evolutions of the system, over a fixed time interval [70].

The answer sets for that program represent possible evolutions of the blocks world over the time interval $0, \ldots, T$ for a fixed positive integer T. A history of the blocks world is characterized by the truth values of atoms of two kinds: $on(b, l, t)$ (block b is on location l at time t) and $move(b, l, t)$ (block b is moved to location l between times t and $t + 1$). Here b ranges over blocks and l ranges over locations; the time variable t ranges over the time instants $0, \ldots, T$, except that the atoms $move(b, l, t)$ are introduced only for $t \neq T$. When such a program is available, we can compute a plan of length T that solves a given planning problem, or establish that there is no such plan, in the following way. The history program is extended by the constraints representing the initial state and the goal state of the problem. In the example above, $T = 3$ and the constraints are:

$$
\begin{array}{cc}
\leftarrow not\ on(1,2,0) & \leftarrow not\ on(1,table,3) \\
\leftarrow not\ on(2,table,0) & \leftarrow not\ on(2,1,3) \\
\leftarrow not\ on(3,4,0) & \leftarrow not\ on(3,2,3) \\
\ldots & \ldots
\end{array}
$$

Figure 2-2: Initial and Goal configurations

The answer sets for the extended program correspond to the plans of length T that lead from the initial state to the goal state. A planner would invoke a system for computing

answer sets to find an answer set X for the extended program, and then return the list of all atoms in X that represent actions (in the case of the blocks world, the atoms beginning with move). To find a plan consisting of the smallest possible number of steps, the planner would invoke a system for computing answer sets several times, with different values of T, using binary search if desired. (A similar process is used in satisfiability planning.)

2.3.2 Macros in Automated Planning

A macro-action, or macro in short, is a group of actions selected for application at one time like a single action. Macros could represent high-level tasks comprising low-level details. From a broader perspective, macros are like subroutines or procedures in the programming paradigm. However, macros are a promising means by which significant knowledge could be conveyed. Combining several steps in the state space, macros provide extended visibility of the search space to the planner. Carefully chosen macros could help find nodes that are better than the current nodes especially when the goodness of the immediate search neighbourhood cannot be measured appropriately.

Macros encapsulate sequences of (ordinary) planning operators. Advantageously, they can be encoded in the same form as planning operators; macros can, therefore, be added into a domain model and can be exploited in a *solver-independent* way (e.g. encoded in PDDL). As we said, from a search perspective, macros can be seen as "short-cuts" in the state space: they can reduce the number of steps needed to reach the goal, however, at the cost of an increased branching factor.

Let us provide a more formal description of planning operators, and of macros in automated planning. The classical (STRIPS) representation considers static and fully observable environment, and deterministic and instantaneous action effects. The environment is described by first-order logic *predicates* defined as $p = pred\_name(x_1, \ldots, x_n)$, where *pred_name* is a unique predicate name and $x_1, \ldots x_n$ are variable symbols. *States* are defined as sets of *atoms* (grounded predicates whose variable symbols are substituted with

constants - problem-specific objects).

We say that an operator $o = (name(o), pre(o), del(o), add(o))$ is a *planning operator*, where $name(o) = op\_name(x_1, \ldots, x_k)$ (op_name is an unique operator name and $x_1, \ldots x_k$ are variable symbols (arguments) appearing in the operator) and $pre(o)$, $del(o)$ and $add(o)$ are sets of (ungrounded) predicates with variables taken only from $x_1, \ldots x_k$ representing o's precondition, delete, and add effects, respectively. *Actions* are grounded instances of planning operators. An action a is *applicable* in a state s if and only if $pre(a) \subseteq s$. Application of a in s (if possible) results in a state $(s \setminus del(a)) \cup add(a)$.

A *planning domain model* $D = (P, O)$ is specified by a set of predicates (P) and a set of planning operators(O). A *planning task* $\Pi = (D, I, G)$ is specified via a domain model (D), initial state (I) and set of goal atoms (G). Given a planning problem, a *plan* is a sequence of actions such that their consecutive application starting in the initial state results in a state containing all the goal atoms.

Given a planning task Π, we say that a state s' is *reachable* from a state s if and only if there exists a sequence of actions such that their consecutive application starting in s results in s'.

A *Substitution* is a set of mappings from variable symbols to terms that are used to determine which arguments (variable symbols) operators share. Hereinafter, we will assume that different operators have distinct parameters as arguments unless stated otherwise. In the set operations over sets of ungrounded predicates we assume that only predicates having the same name and the same arguments (i.e. the same parameter symbols ordered in the same way) are equivalent.

Naturally, actions influence each other in plans. We say that an action a_i is an *achiever* for an action a_j if an only if $add(a_i) \cap pre(a_j) \neq \emptyset$. We also say that actions a_i and a_j are *independent* if and only if $del(a_i) \cap (pre(a_j) \cup add(a_j)) = \emptyset$ and $del(a_j) \cap (pre(a_i) \cup add(a_i)) = \emptyset$.

Formally, a macro $o_{i,j}$ is constructed by assembling planning operators o_i and o_j (in

that order) as follows. Let Φ and Ψ be mappings between variable symbols (we possibly need to appropriately rename variable symbols of o_i and o_j to construct $o_{i,j}$).

- $pre(o_{i,j}) = pre(\Phi(o_i)) \cup (pre(\Psi(o_j)) \setminus add(\Phi(o_i)))$
- $del(o_{i,j}) = (del(\Phi(o_i)) \setminus add(\Psi(o_j))) \cup del(\Psi(o_j))$
- $add(o_{i,j}) = (add(\Phi(o_i)) \setminus del(\Psi(o_j))) \cup add(\Psi(o_j))$

Longer macros, i.e., those encapsulating longer sequences of original planning operators, can be constructed iteratively by the above approach.

For a macro to be *sound*, no instance of $\Phi(o_i)$ can delete an atom required by a corresponding instance of $\Psi(o_j)$, otherwise they cannot be applied consecutively. Whereas it is obvious that if a predicate deleted by $\Phi(o_i)$ (and not added back) is the same (both name and variable symbols) as a predicate in the precondition of $\Psi(o_j)$ then the macro $o_{i,j}$ is unsound.

Representing Macros in ASP In ASP, a macro is encoded by a single choice rule, which implicitly represents multiple actions, and by several normal rules needed to model their effects. The head of the choice rule contains a single fresh atom, whose variables are all the one appearing in the body of the choice rule, whereas the body of the choice rule is built as described in the following. Given a rule r_i representing an action, $pre(r_i)$ denotes the body of the rule. Intuitively, it represents the conditions that must hold in order to activate the action represented by the rule. Moreover, $del(r_i)$ (resp. $add(r_i)$) represent all the atoms that are set as false (resp. true) whenever the conditions denoted by $pre(r_i)$ hold. Then, a macro $r_{i,j}$ is constructed by assembling the rules representing single actions and by generating $pre(r_{i,j})$, $del(r_{i,j})$, and $add(r_{i,j})$, as follows:

- $pre(r_{i,j}) = pre(r_i) \cup (pre(r_j) \setminus add(r_i))$
- $del(r_{i,j}) = (del(r_i) \setminus add(r_j)) \cup del(r_j)$

- $add(r_{i,j}) = (add(r_i) \setminus del(r_j)) \cup add(r_j)$

where r_i and r_j are two distinct rules. Then, for a macro $r_{i,j}$, the body of the choice rule is represented by $pre(r_{i,j})$.

```
{linkToCentral_take(L1,L2,J1,J2,G1,G2,T)} :- link(L1), link(L2),
    joint(J1), joint(J2), gripper(G1), gripper(G2), time(T),
    not in_centre(J1,T), connected(J1,L1), connected(J1,L2),
    free(G1,T), free(G2,T), L1<>L2, G1<>G2.
```

Figure 2-3: Example of a Macro action in ASP.

Chapter 3

Problem Statement and Reference Scenarios

3.1 Introduction

Our goal is to present (i) an efficient ASP-based planning and execution architecture for the manipulation of articulated objects in terms of perceptual features, their representation and the planning of manipulation actions, which maximises the likelihood of being successfully executed by robots, and (ii) given a specific object's goal configuration, determine a plan to attain it, in which each step involves one or more manipulation actions to be executed by a dual-arm robot. Our working assumptions are:

A_1 flexible objects can be appropriately modelled as articulated objects with a high number of links and joints, as it is customary [100];

A_2 an object's configuration is only affected by robot manipulation actions, or possibly by humans, and the effects of external forces such as gravity are not considered;

A_3 we do not consider possible issues related to grasping or dexterity during the manipulation task;

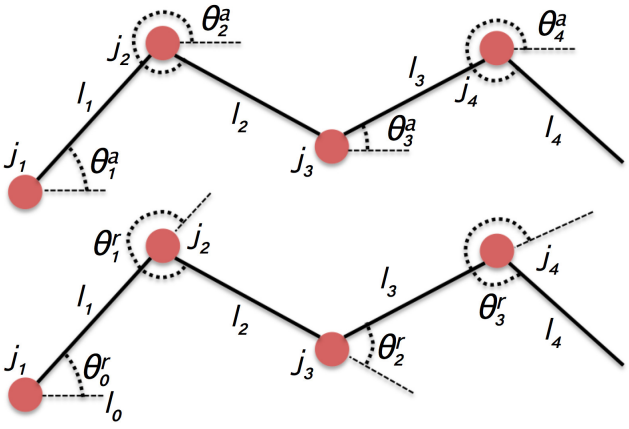


Figure 3-1: Two possible representations: absolute (top) and relative (bottom).

A_4 sensing is affected by noise, but the *symbol grounding problem*, i.e., the association between perceptual features and the corresponding symbols [52], is assumed to be solved.

Based on assumption A_1, we focus on articulated objects only.
In this section, we present, in two separate sub-sections, the simple scenario and the extended scenario we have set up.

3.1.1 Articulated Object

We define an articulated object as a pair $\alpha = (\mathcal{L}, \mathcal{J})$, where $\mathcal{L}$ is the ordered set of its $|\mathcal{L}|$ links and $\mathcal{J}$ is the ordered set of its $|\mathcal{J}|$ joints. Each link $l \in \mathcal{L}$ is characterised by two parameters, namely a length λ_l and an orientation θ_l. We allow only for a limited number of possible orientations, which induces a finite set of allowed angle orientations. If α is represented using absolute angles (Figure 3-1 on the top), then its configuration is a $|L|$-ple:

$$C_{\alpha,a} = \left(\theta^a_1, \ldots, \theta^a_{|L|}\right). \tag{3.1}$$

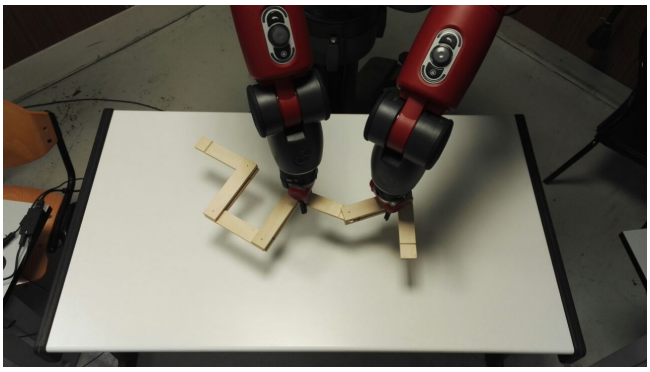


Figure 3-2: The experimental scenario.

Otherwise, if relative angles are used (Figure 3-1 on the bottom), then the configuration must be *augmented* with an initial *hidden* link l_0 in order to define a reference frame:

$$C_{\alpha,r} = \left(\theta_1^r, \theta_2^r, \ldots, \theta_{|L|}^r\right). \tag{3.2}$$

In fact, while in principle the relative approach could represent the configuration of an articulated object with one joint less compared to the absolute one, the resulting representation would not be unique, since the object maintains relative orientations among its parts even when rotated *as a whole*.

3.2 Simple Model

In order to comply with assumption "an object's configuration is only affected by robot manipulation actions, or possibly by humans, and the effects of external forces such as gravity are not considered", which allows us to focus on the manipulation process, we set up a scenario in which a dual-arm Baxter robot manipulates an articulated object that is conveniently located on a table in front of it. The table sustains the object while it is being manipulated by the robot, and it is assumed to be large enough to accommodate the whole object itself, see Figure 3-2. As a consequence, link rotations occur only around

axes centred on specific object joints, but always perpendicular to the table surface. We have crafted two wooden articulated objects of different size: the first, which is simpler, has three 40 *cm* long links (which are connected by two in-between joints), whereas the second is made up of five 20 *cm* long links (connected by four joints). For both objects, links are 2 *cm* wide and 3 *cm* thick. The two objects have been designed to reduce the likelihood of manipulation-specific issues when using Baxter's standard grippers, to comply with assumption A_3. Baxter's *head* is equipped with a camera pointing downward to the table and able to acquire images of the relevant robot manipulation workspace. AR tags are attached to each object link l, which reduces perception errors and is aimed at meeting assumption A_4. Each AR-tag provides an overall link pose, which directly maps to an absolute link orientation θ_l^a. Finally, if relative orientations are chosen, we compute them by performing an algebraic sum between the two absolute poses of two consequent links, e.g., $\theta_1^r = \theta_2^a - \theta_1^a$. After this general scenario introduction, in the next section, we detail the architecture.

3.3 Extended Model

Observing the performance during the execution of our framework, we noticed that the robot was not always able to perform the requested action due to the inappropriate position of the links to manipulate; for example, links that moved outside the working space and could not be reached by the robot. For these reasons we introduce an extended scenario. This scenario does not modify any physical characteristics with respect to the setup introduced in Section 3.2, but represents with higher accuracy the setup, and introduces a few constraints in robot actions execution to improve their feasibility.

Herewith we briefly describe such modelling, and whenever relevant we highlight the main modifications we introduced to the initial scenario.

- *The robot grippers are now explicitly modelled.* Each gripper should be considered

as a resource that can be *occupied* (i.e., keeping a link firmly, or rotating a link) or *free*. The new scenario takes this into account. In this case, it is possible to explicitly represent which gripper can manipulate a given link.

- For several reasons related to the articulated object's configuration while being manipulated, or the specific sequence of manipulation actions, or because of issues concerning gripper-related motion planning and execution, in the simple scenario it may happen that a link could not be reached or grasped, or it may be placed to a part of the robot's workspace where manipulation could be difficult. In the extended scenario, *each time a manipulation action is carried out on a given link, it is assured that the link is centred in the robot workspace*. If this is not the case, the link is moved towards the central part of the table. This maximises the likelihood of a relevant link to be well-centred in the robot workspace.

- *Now grasping and release actions by the two grippers are explicitly modelled.* In the scenario described in Section 3.3, these two actions were not modelled. They were assumed to be properly carried out during the execution phase, as part of each action execution. With an explicit modelling, we can represent grippers' occupancy, and we better characterise the semantics associated with each action, since now grasping, manipulation, and release are distinct.

It is noteworthy that the above-mentioned features allow for several improvements. We can envisage two main advantages: on the one hand, the encoding is expected to be able to better manage the explicitly modelled robot resources (i.e., the grippers); on the other hand, manipulation actions are characterised by a more precise semantics, which does not make any implicit assumption about actual robot behaviour.

Chapter 4

Architecture

4.1 Software

4.1.1 Robot Operating System

The middleware on which this work has been developed is the Indigo version of the Robot Operating System (ROS)Quigley et al. [88][1], which is a modular architecture used in many robotic applications. Each ROS software module is called *"node"*. Several nodes can communicate via topics using the publish/subscribe protocol. Every node can publish and/or subscribe to as many topics as needed. Every time something is published to a topic, every node subscribing to it runs a callback function which takes as input the incoming message. Such architecture is very convenient when many software modules have to run independently.

The publish-subscribe model is a very flexible communication paradigm, but it implements a many-to-many one-way transport that is not appropriate for RPC request-reply interactions. Request-reply communication is done via a *"service"*, which is defined by a pair

[1] http://www.ros.org/

of messages: one for the request and one for the reply. A providing ROS node offers a service under a string name, and a client calls the service by sending the request message and awaiting the reply. Thanks to these components module replacement are easier and the software can be easily extended. Additionally, it is easy to manage the frequency at which messages are sent, which is of fundamental importance when working under limited computational resources and real-time applications, such as ours.
Despite the name, ROS is not a full-fledged Operating System, but it is rather a framework based on a collection of open-source libraries, some of which have a life of their own and have a very lively community. Because of its nature, ROS can run on most operating systems. The preferred development environment is Ubuntu Linux and it is also the one adopted for this project (version 16.04 LTS).
At the time of writing, the last available version of the software is ROS Melodic, but the work presented here is based on the previous version, ROS Kinetic. This comes from the fact that Melodic not yet available at the beginning of this project while Kinetic was already mature and stable at that time. However, the tools used to communicate with the robot we are using, a Baxter robot, are developed for ROS Indigo. That required us to modify slightly the Baxter robot SDK in order to make them work with ROS Kinetic,

4.1.2 Ar_Track_Alvarr

This package is a ROS wrapper for Alvar, an open-source AR-tag tracking library. ALVAR is "a library for virtual and augmented reality." The ALVAR SDK allows you to create AR applications with the most accurate, efficient and robust implementation for marker-, 2D image-, and 3D point cloud-based tracking. ALVAR provides a low-level C++ API to its tracking algorithms and includes several tools that assist in the creation of AR applications. Its low-level interface makes it possible to develop custom solutions that can be integrated with existing products and services. the ar_track_alvar node has three main functionalities:

- Generating AR tags of varying size, resolution, and data/ID encoding

- Identifying and tracking the pose of individual AR tags, optionally integrating Kinect depth data (when a Kinect is available) for better pose estimates.
- Identifying and tracking the pose of "bundles" consisting of multiple tags. This allows for more stable pose estimates, robustness to occlusions, and tracking of multi-sided objects.

Alvar is significantly newer and more advanced than the ARToolkit, which has been the basis for several other ROS AR-tag packages. Alvar features adaptive thresholding to handle a variety of lighting conditions, optical flow based tracking for more stable pose estimation, and an improved tag identification method that does not significantly slow down as the number of tags increases.

Here the presentation video

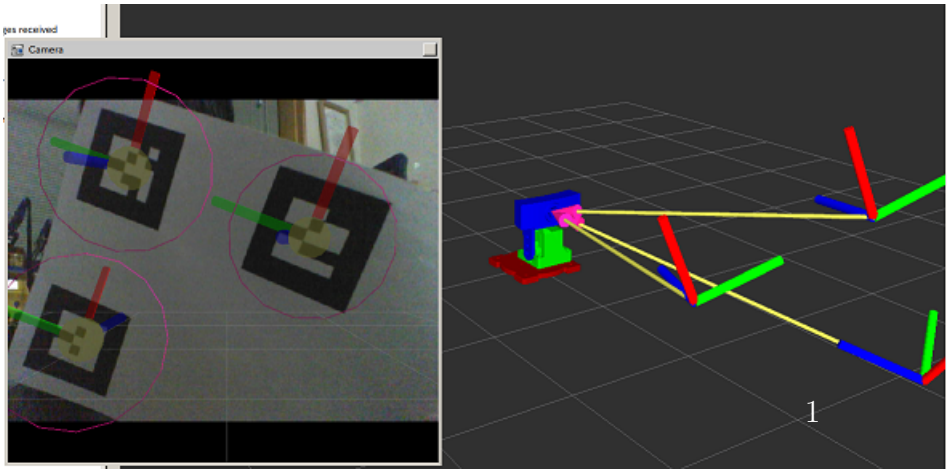


Figure 4-1: Ar_track_alvar algorithm tracking 3 markers

4.1.3 MoveIt!

MoveIt! is a state of the art software aimed at robotic manipulation that offers an accessible, all-in-one solution, to most manipulation related tasks. The list of features includes up-to-date planning algorithms, 3D perception support, kinematic modelling, control and navigation. The software natively relies on ROS and integrates itself in ROS debugging and visualization tools. The aforementioned features make MoveIt! the perfect tool to design and evaluate new robot designs and to build integrated robotics products for industrial, commercial and R&D applications. In this case, MoveIt! has been chosen to easily introduce an execution layer into this project, being the Baxter Robot one of the supported machines. This allowed us to reach a tangible result (i.e. the robot acting in the world and not just in a simulation) and also to stress some important theoretical points, like the effective reliability of the sensing stage.
It is important to note though, that out-of-the-box MoveIt! integration in the Baxter SDK is a little bit below the level what one would expect from an officially supported feature. This required me to take some important design decision when integrating MoveIt! into this framework. I report here two crucial points:

- Robot Model and Collision Avoidance. By default, the Baxter SDK and MoveIt! Packages provide only one 3D model to be used for collision avoidance of the electric gripper fingers. This model describes an electric gripper with a long finger in a very light configuration. Due to that, we had to rewrite the URDF file describing the joints bounding boxes to modify the grippers and adapt them to reality. This allows me to have a proper grasping position but also introduced problems during the grasping phase due to the obstacle avoidance feature provided by MoveIt!. For this reason, this latter feature has been disabled.

- Determining allowed Grasping Pose. The Baxter Development Kit and MoveIt! do not provide any utility to determine the allowed grasping poses for a defined object. For that reason, I opted for an unofficial method available on-line. This method allows us to compute the grasping poses just giving as input a centroid so without any knowledge about the object. Due to the lack of information, this algorithm computes all the grasping poses around the point.

4.1.4 Potassco

Potassco (Potsdam Answer Set Solving Collection) is the bundle tools for Answer Set Programming developed at the University of Potsdam. This collection is composed by: **gringo**, the grounder, **clasp**, the solver and **clingo** that combines the two.

Current answer set solvers work on variable-free programs. Hence, a grounder is needed that, given an input program with first-order variables, computes an equivalent ground (variable-free) program. **gringo** is such a grounder. Its output can be processed further with **clasp**.

clasp is an answer set solver for extended normal and disjunctive logic programs. It combines the high-level modelling capacities of ASP with state-of-the-art techniques from the area of Boolean constraint solving. The primary clasp algorithm relies on conflict-driven nogood learning, a technique that proved very successful for satisfiability checking (SAT). **clasp** has been genuinely developed for answer set solving based on conflict-driven nogood learning. Clasp can be applied as an ASP solver (on aspif or smodels format, as output by gringo), as a SAT solver (on a simplified version of dimacs/CNF format), as a PB solver (on OPB format), or as a C++ library in another program.

clingo combines both gringo and clasp into a monolithic system. This way it offers more control over the grounding and solving process than gringo and clasp can offer individually

- e.g., incremental grounding and solving.

4.2 Hardware

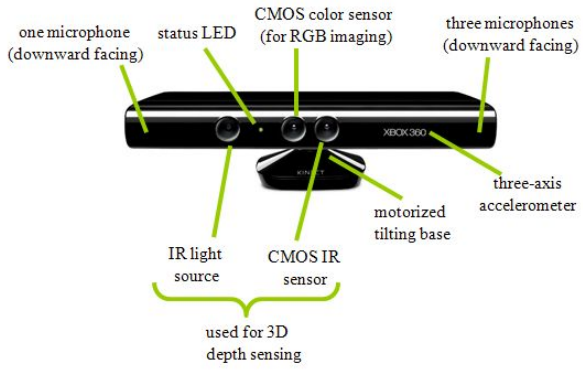
4.2.1 Baxter robot

Figure 4-2: Baxter Robot bust from Rethink Robotics

The Baxter Robot (Rethink Robotics, Last accessed 20/08/2015) is an industrial robot developed by Rethink Robotics and presented in September 2012. An academic edition is available on the market and is very popular among universities and research institution because of the hardware and software specification of the robot. When mounted on its base, the Baxter robot is between 1.78m and 1.92m tall, and weighs 139kg. It comes equipped with two anthropomorphic 7 Degrees Of Freedom arms, an animated face screen and three cameras, two hand-mounted and an head-mounted one, even though it can run only two at time. The robot is also equipped with a large number of sensors, like proximity sensors, accelerometers and more. These sensors are used to make the Baxter robot aware of himself and the surrounding environment, thus enabling the robot to act reactively to external stimuli. This leads to the main feature of this robot: safety. It is possible to work in direct contact with Baxter without need for protection cages and other safety measures.

The robot constantly checks for self-collision and stops whenever it detects an unexpected collision with a third object before causing any harm. This makes the Baxter robot a good candidate for high-level robotic projects, since it is possible to work directly with the robot and iterate quicker. On the other side, outstanding safety features and a low 22.000e price tag are balanced by a below-average dynamical performance and scarce precision, which limits the industrial use of the robot to simple light-duty operations such as sorting and palletizing.

4.2.2 The Microsoft Kinect



KINECT™
for Windows®

Figure 4-3: Kinect from Microsoft

The Microsoft Kinect (Zhang, 2012) is a motion sensing input device mainly designed to provide a natural interface for video games and video game consoles. It consists of an infrared projector and a monochrome CMOS sensor able to provide 3D data under any light condition, as opposed to more traditional Computer Vision Systems. The sensing apparatus is paired with a special microchip to track gestures and objects in three dimensions, a multi-array microphone, a RGB camera and an actuator to tilt the device head. Despite its original use, the Microsoft Kinect has become increasingly popular in research institutions as a cheap yet reliable sensor for 3D Imaging and Motion Analysis, being able

to process 48 skeletal points at a frequency of 30Hz and to provide point clouds with much higher point count than that supported by modern computers. A large part of Kinect success among researchers and enthusiast is the large availability of development tools and the strong community that gathered around it. Not only Microsoft provides its own software development kit for Windows system, but an open-source library is also available for all majors operating systems under the name of OpenKinect (Blake et al., 2011). In both cases the programming language to work with the Kinect is C++.

Parameter	typical Value
Optical Format	1/6-inch (4:3)
Active Imager Size	2.30mm(H) x 1.73mm(V) 2.88mm Diagonal
Active Pixels	640H x 480V
Pixel Size	3.6µm x 3.6µm
Color Filter Array	RGB Bayer Pattern
Shutter type	Electronic Rolling Shutter (ERS)
Maximum Data Rate/ Master Clock	12 MPS–13.5 MPS/ 24 MHz–27 MHz
Frame Rate (VGA 640H x 480V)	30 fps at 27 MHz
ADC Resolution	10-bit, on-chip
Responsivity	1.0V/lux-sec (550nm)
Dynamic Range	71dB
SNR_{MAX}	44dB

Parameter		Value
Optical format		1/2-inch (5:4)
Active imager size		6.66mm(H) x 5.32mm(V)
Active pixels		1,280H x 1,024V
Pixel size		5.2µm x 5.2µm
Shutter type		Electronic rolling shutter (ERS)
Maximum data rate/ master ciock		48 MPS/48 MHz
Frame rate	SXGA (1280 x 1024)	30 fps progressive scan; programmable
ADC resolution		10-bit, on-chip
Responsivity		2.1 V/lux-sec
Dynamic range		68.2dB
SNR_{MAX}		45dB

Figure 4-4: Kinect data sheet

The reasons for such sensor choice are:

- High reliability. Depth information is produced precisely within the sensor range, that is up to 5 meters distance from it;

- Low computational time required to extract depth information of obstacles. Indeed the Kinect output rate is up to 30Hz, which is enough for our needs;

- Perfectly suitable for indoor environment, where there is no need for long-range obstacle detection and there is no sunlight which strongly disturbs, if not prevents, the sensing;

- Not influenced by artificial light to contrarily to classical computer vision methods based on 2D-images

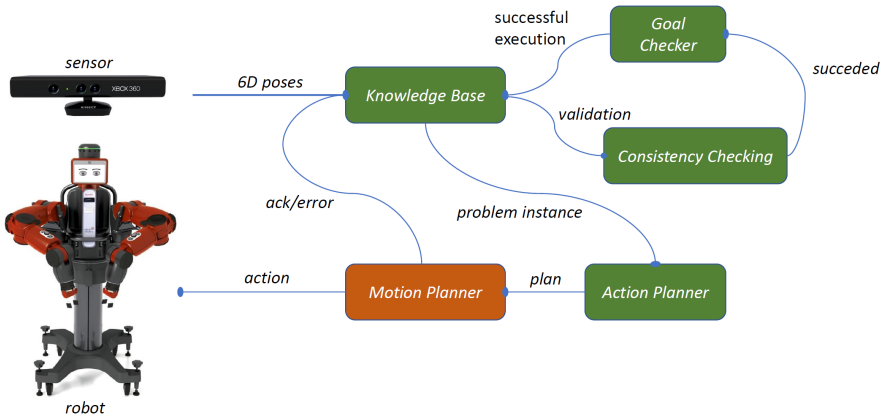


Figure 4-5: The robot's architecture: in *green* the ASP-based modules, in *orange* the ROSPlan-based module.

4.3 Modules and Functioning of the ASP-based Architecture

The architecture of the Baxter from Rethink Robotics is shown in Figure 4-5. It is noteworthy that, in principle, the architecture can be adapted to other robot platforms as well, either in simulation or in real-world conditions, as long as appropriate perception, low-level motion planning tools, and manipulation strategies are adopted.

In the current implementation, perception is managed using a camera sensor located on top of the robot's *head* and pointing downward, which provides 6D poses for each link, which update corresponding ASP-based representation structures in the *Knowledge Base* module. The *Consistency Checking* module performs a check for knowledge base validation. In case the check succeeds, the *Goal Checker* module is notified and relevant parts of the current ASP *Knowledge Base* are processed by the rules encoded in the *Goal Checker* module, aimed at detecting whether the (already computed) plan can be successfully executed, also in response to a possible human intervention. Whenever checks that influence the knowledge base status are performed, a problem instance may be generated, which depends on the target articulated object configuration and the current configuration maintained in the *Knowledge Base* module. The *Action Planner* module receives such problem instance and generates a plan in the form of a suitable sequence of actions to be performed. Once a plan is generated, its actions are processed sequentially to drive the overall behaviour of the robot *Motion Planner* module, which is responsible for the execution. For the use case described in this paper, each action involves one rotation operation on the target link. Rotations occur only around axes centred on the object's joints. Any action may be either successful or not, depending on several reasons related to noise and errors in perception, grasping, and manipulation in a real-world environment. If an action is successful, the *Motion Planner* module proceeds with the one that follows until the plan ends and the *Knowledge Base* module is notified about successful execution. Otherwise,

an issue is raised and re-planning occurs, thereby reiterating the whole work-flow described above.

Note that all modules except *Motion Planner* (i.e., all *green* modules in Figure 4-5) are based on ASP. The *Motion Planner* is the module that has to interact with the robot, and has therefore to follow the constraints posed by the actual machine.

It is important to underline that each model must be modified accordingly to the scenario and the approach that it will be used. In the following section, we will use as reference the codes regarding the Simple Action Scenario with only forward propagation.

4.4 Modules

4.4.1 Knowledge Base

The knowledge base consists of facts over atoms of the form `joint(J)`, `angle(A)`, `isLinked(J1,J2)`, `time(T)`, `hasAngle(J,A,T)`, and `goal(J,A)`, and the constants `granularity` and `timemax`. Atoms over the predicate `joint` represent the joints of the articulated object. Atoms over the predicate `angle` represent the possible angles reachable from the joints and they can range from 0 to 359. Actually, the atom `angle(0)` must be always part of the knowledge base and admissible angles are the ones that can be obtained by rotating a joint by the degrees specified by the constant `granularity`, e.g., if the granularity is 90 degrees, then the admissible angles are 0, 90, 180, and 270. Atoms over the predicate `isLinked` represent links between joints `J1` and `J2`. Atoms over the predicate `time` represent the possible time steps, and they range from 0, which represents the initial state, to `timemax`. Atoms over the predicate `hasAngle` represent the angle `A` of the joint `J` at time `T`. Actually, knowledge base only contains the initial state of each joint, i.e., its angle at time 0. Finally, atoms over the predicate `goal` represent the angle `A` that must be reached by the joint `J` at the time step specified by `timemax`.
An example of the input is represented by the facts and constants reported in Figure 4-6.

```
joint(1..5). angle(0). angle(90). angle(180). angle(270).
isLinked(1,2). isLinked(2,3). isLinked(3,4). isLinked(4,5).
hasAngle(1,90,0). hasAngle(2,180,0). hasAngle(3,180,0).
hasAngle(4,270,0). hasAngle(5,270,0). time(0..timemax).
goal(1,270). goal(2,270). goal(3,180). goal(4,270).
goal(5,270). #const granularity = 90.
```

Figure 4-6: An example of an ASP knowledge base.

4.4.2 Consistency Checking

The module performs some consistency checking on the knowledge base by using some ASP rules, for example:

c_{1a} `:- isLinked(J1,J2), not joint(J1).`

c_{1b} `:- isLinked(J1,J2), not joint(J2).`

c_{2} `:- isLinked(J,J).`

c_{3a} `:- hasAngle(J,A,T), not joint(J).`

c_{3b} `:- hasAngle(J,A,T), not angle(A).`

c_{3c} `:- hasAngle(J,A,T), not time(T).`

c_{3d} `:- hasAngle(J,A,T), not possibleAngle(A).`

c_{4a} `:- goal(J,A), not joint(J).`

c_{4b} `:- goal(J,A), not angle(A).`

c_{5} `moreThanOneGoal(J) :- joint(J), #count{A:goal(J,A)}>1.`

c_{6} `:- joint(J), moreThanOneGoal(J).`

c_{7} `oneStartingAngle(J) :- joint(J), #count{A:hasAngle(J,A,0)}=1.`

c_{8} `:- joint(J), not oneStartingAngle(J).`

c_{9} `:- not time(0).`

c_{10} `:- not angle(0).`

c_{11} `possibleAngle(0).`

c_{12} `possibleAngle(X) :- possibleAngle(Y), X=Y+granularity, X<360.`

c_{13} `:- not angle(X), possibleAngle(X).`

c_{14} `:- angle(X), not possibleAngle(X).`

In particular, rules c_{1a} and c_{1b} check whether atoms over the predicate `isLinked` represent the links between two joints, while c_2 checks whether there is no link between the same joint. Rules c_{3a}, c_{3b}, and c_{3c} check the correctness of the predicate `hasAngle`, whereas

c_{4a} and c_{4b} check the correctness of the predicate `goal`. Rules c_5 and c_6 check whether at most one goal is specified for each joint, whereas rules c_7 and c_8 verify if each joint is in exactly one angle at time step 0. Rules c_9 and c_{10} simply check the existence of the first time step and angle 0, respectively. Finally, rules from c_{11} to c_{14} check whether atoms over the predicate `angle` represent the possible angles.

4.4.3 Goal Checker

During the execution of a plan an external agent may interact with the articulated object, e.g., a human may change the angle of some joints (see, e.g., [**?**]). In such a case, the system must react to the changes if they are not compatible with the plan executed by the robot. This is accomplished by asynchronously creating a new input configuration according to the current status of the object so that the configuration is ready as soon as it is needed. The role of Goal Checker module is to check when there is no need to create a new configuration, that is when all goals have been reached. This is done by using rule r_{15} from the encoding in Figure 4-7.

4.4.4 Action Planning

Simple Scenario

Inside the context of the simple scenario, see Section 3.2 we explored different strategies in order to investigate the pros and drawbacks of each approach. Each strategy is tested out firstly without backward propagation and then including it; the three strategies we investigate are:

- Standard Strategy: the strategy is based on the number of maximum steps allowed: it starts from 1 and it is increased by one unit iff the plan is not found. This guarantees us that the found solution is optimal in terms of the number of action. This comes with a sacrifice in terms of time performances.

- Heuristic Strategy: a *nOPTSAT* based on *optsat* [24, 49] heuristic function it is added to the encoding: it is requested that the maximum steps allowed to reach the goal are as small as possible. This does ensure an optimal solution in term of number of actions and allows us to remove the incremental process of the maximum number of actions.

- Not Optimal Strategy: The maximum steps number is randomly selected and therefore it does not ensure an optimal solution.

Standard Strategy

ASP is not a planning-specific language, but it can be also used to specify encoding for planning domains [71], like our target problem. We have defined several encodings variants, for what concerns either the manipulation modes and the strategy for computing plans. The encoding described in this section is embedded into a classical iterative deepening approach in the spirit of SAT-based planning [60], where `timemax` is initially set to 1 and then increased by 1 if a plan is not found, which guarantees to return the shortest possible plans for a sequential encoding, i.e., when the robot performs only one action for each step (see Section 7.1 for some details about the other strategies).

Figure 4-7 reports our base encoding. Note that it uses operations $\backslash$ and $|\cdots|$, which are not defined in the ASP-Core-2 standard but supported by Clingo [39], and compute the remainder of the division and the absolute value, respectively.

Since we employ an absolute representation, r_1, r_2 and r_3 add to the knowledge base the `joint(0)`, its angle and link to joint 1. This joint will not be moved and it is used only to have a fixed reference between the robot and articulated object frames. Rule r_4 enforces that bidirectionality of linked joints, i.e., if `joint(1)` is linked to `joint(2)` then `joint(2)` is also linked to `joint(1)`. Rule r_5 selects an atom of the form `changeAngle(J1,J2,A,Ai,T)`, where `J1` is the joint to move, `J2` is the joint to keep steady, `A` is the desired angle, `Ai` is the current angle of `J1` and `T` is the cur-

```
r1   joint(0).
r2   hasAngle(0,0,0).
r3   isLinked(0,1).
r4   isLinked(J1,J2) :- isLinked(J2,J1).
r5   {changeAngle(J1,J2,A,Ai,T) : joint(J1), joint(J2), J1>J2, angle(A),
         hasAngle(J1,Ai,T), A<>Ai, isLinked(J1,J2)} <= 1
             :- time(T), T < timemax, T > 0.
r6   ok(J1,J2,A,Ai,T) :- changeAngle(J1,J2,A,Ai,T),
         F1=(A+granularity)\360, F2=(Ai\360), F1=F2, A < Ai.
r7   ok(J1,J2,A,Ai,T) :- changeAngle(J1,J2,A,Ai,T),
         F1=(Ai+granularity)\360, F2=(A\360), F1=F2, A > Ai.
r8   ok(J1,J2,A,0,T) :- changeAngle(J1,J2,A,0,T), A=360-granularity.
r9   ok(J1,J2,0,A,T) :- changeAngle(J1,J2,0,A,T), A=360-granularity.
r10  :- changeAngle(J1,J2,A,Ai,T), not ok(J1,J2,A,Ai,T).
r11  affected(J1,An,Ac,T) :- changeAngle(J2,_,A,Ap,T), hasAngle(J1,Ac,T),
          J1>J2, angle(An), An=|(Ac + (A-Ap)) + 360|\360, time(T).
r12  hasAngle(J1,A,T+1) :- changeAngle(J1,_,A,_,T).
r13  hasAngle(J1,A,T+1) :- affected(J1,A,_,T).
r14  hasAngle(J1,A,T+1) :- hasAngle(J1,A,T), not changeAngle(J1,_,_,_,T),
         not affected(J1,_,_,T), T <= timemax.
r15  :- goal(J,A), not hasAngle(J,A,timemax).
```

Figure 4-7: Base encoding: it allows for forward manipulations only.

rent step. Rule r_{10} ensures the validity of the configuration represented by the atom `changeAngle(J1,J2,A,Ai,T)`, that is when each action has a desired angle `A` that can be reached in one step (rules r_6, r_7, r_8, and r_9). Rule r_{11} is used to identify which joints are affected from the atom selected in r_5. Rules r_{12} and r_{13} are used to update the joints angles for the next step, while r_{14} states that if neither r_{12} nor r_{13} have affected a joint then its angle remains unchanged. Finally, r_{15} states the the goal must be reached.

Moreover, we tested each encoding with also the possibility to perform the movement in both directions, i.e., contrarily to the encoding in Figure 4-7, that from here on we call *SAS* (Simple Action Scenario), it is possible to move the first joint holding the second one. This is accomplished by slightly modifying the encoding to allow the propagation in

```
r5'  changeAngle(L1,L2,J,A1,A2,G1,G2,T):- link(L1), link(L2), joint(J),
       angles(A1), angles(A2), gripper(G1), gripper(G2), time(T),
       in_centre(J,T), grasped(G1,L1,T), grasped(G2,L2,T),
       in_hand(L1,T), in_hand(L2,T), not free(G1,T),
       not free(G2,T), connected(J,L1), connected(J,L2),
       hasAngle(J,A2,T), L1<>L2, G1<>G2
```

Figure 4-8: Rule r_5 with its preconditions updated.

both directions: forward and backward. In particular, we removed $J1 > J2$ in rule r_5 and we changed the preconditions of r_5. This change of the preconditions was necessary to fit the new model: two links must be grasped and those links must be in the center of the workspace. For sake of readability we will, from now on, call the modified version of rule r_5, rule $r_{5'}$ (shown in Figure 4-8).

Heuristic Strategy

In order to ensure an optimal solution in term of number of actions and can remove the incremental process of the maximum number of actions, we developed an encoding by employing a strategy based on the algorithm *optsat* [24, 49], where the heuristic of the solver is modified in order to prefer plans with increasing length, and (*ii*) by using a choice rule to select the time step and by letting the solver find a plan, of course possibly losing optimality (see also [25]).
Only a few changes to the encoding shown in Figure 4-7 were necessary:

- two literals had to be added to the encoding:

```
r_h1  1<=maxspan(X): time(X), X>0<=1.
r_h2  #heuristic maxspan(X). [timemax-X@1,true]
```

  where r_{h1} is a choice rule that ensure that the variable X, that represent the max-

imum time (or rather the maximum number of steps), is > 0 and that it has only one value, while r_{h2} ensure that the variable X is a small as possible using the above mentioned heuristic function;

- every constant *timemax* had to be replaced with the variable X an, therefore, the literal $maxspan(X)$ had to be added in each one of these rules;
- the goal literals in the knowledge base (see Figure 4-6) had to be changed from

```
goal(A,B).  to  goal(A,B,X), maxspan(X).
```
(4.1)

Not Optimal Strategy

With this encoding we wanted to investigate if dropping the certainty of a optimal solution could lead us to better performances with respect to the execution time, Moreover we wanted to investigate if such approach could solve complex problems with an higher number of joints and possible angles.
The strategy we decided to test choose a random number of maximum steps. Similarly to the heuristic case we had to slightly change the encoding:

- only one literal had to be added

r_{no1} `maxspan(X) : X=1..timemax = 1.`

where r_{no1} state that the variable X must be chosen between 1 and *timemax*;

- *timemax* had to be fixed to a value, in our case $timemax = 50$;
- as in the heuristic strategy case Every constant *timemax* had to be replaced with the variable X an, therefore, the literal $maxspan(X)$ had to be added in each one of these rules and the goal literals in the knowledge base (see Figure 4-6) had to be changed as showed in equation 4.1.

4.4.5 Extended Scenario

Inside the context of the extended scenario, see Section 3.3, we explored two different strategies in order to investigate the pros and drawbacks of each approach. Moreover, in this scenario, we investigate the propriety of the macros (see sec 2.3.2) and both the strategies are tested out directly including backward propagation. The two strategies are:

- Simple Actions Extended Scenario (SAES) (Figure 4-9): This encoding models the same actions as the Standard Strategy in the simple scenario but it includes also the robot resources that can be occupied at each time step (i.e. the robot gripper);
- Macro Actions Extended Scenario (MAES) (Figure 4-10): Here we have modelled the same scenario as in the SAES. The difference consists of the modelled actions: sets of simple actions are gathered inside just one atom.

In this sub-section, we focus on the action planning module in its first paragraph, which is the module that undertakes the major changes and then comment in a second paragraph about how the other modules have been updated to accommodate the changes.

Simple Action Extended Scenario

ASP Encoding for the Action Planning Module. It is important to notice that, in order to achieve the bi-directionality of the propagation, as we said in the preceding section, it is sufficient to remove the constraint $J1 > J2$ from the choice rule r_5. It is important to emphasize here that, aside from such changes, the idea behind the base encoding is maintained: the general structure and the method the encoding deals with movements that concern the articulated object configuration changes are the same as in Figure 4-7.

Figure 4-9 reports the further rules needed to deal with the extended scenario. Rule r_{16}, r_{18} and r_{23} are the selection of the possible actions in this model; r_{16} puts the joint that has to be moved in the center, r_{18} selects an atom of the form $take\_links\_to\_move(L1, L2, J, G1, G2, T)$ where $L1$ and $L2$ are the links that have to be grasped, while r_{23} selects at atom of the

```
r16 {move_link_to_central(L1,J1,G2,T)} :- link(L1), joint(J1),
       gripper(G2), time(T), connected(J1,L1), free(G2,T),
       not in_hand(L1,T), not in_centre(J1,T).
r17 in_centre(J,T+1) :- move_link_to_central(_,J_,T), T < timemax+1.
r18 {take_links_to_move(L1,L2,J,G1,G2,T)} :- link(L1), link(L2),
       joint(J), gripper(G1), gripper(G2), in_centre(J,T)
       free(G1,T), free(G2,T), not in_hand(L1,T), not in_hand(L2,T),
       connected(J,L1), connected(J,L2), L1<>L2, G1<>G2.
r19 in_hand(L,T+1) :- take_links_to_move(L,_,_,_,_,T), T < timemax+1.
r20 in_hand(L,T+1) :- take_links_to_move(_,L,_,_,_,T), T < timemax+1.
r21 grasped(G,L,T+1) :- take_links_to_move(L,_,_,G,_,T), T < timemax+1.
r22 grasped(G,L,T+1) :- take_links_to_move(_,L,_,_,G,T), T < timemax+1.
r23 {release_links(L1,L2,J,G1,G2,T)} :- link(L1), link(L2), joint(J),
       gripper(G1), gripper(G2), time(T), grasped(G2,L2,T),
       grasped(G1,L1,T), in_hand(L1,T), in_hand(L2,T),
       not free(G1,T), not free(G2,T), connected(J,L1),
       connected(J,L2), L1<>L2,G1<>G2.
r24 free(G,T) :- release_links(_,_,_,_,G,T),T<timemax+1.
r25 free(G,T) :- release_links(_,_,_,G,_,T),T<timemax+1.
r26 action(T,move_link_to_central(L1,J1,G2,T)) :-
       move_link_to_central(L1,J1,G2,T).
r27 action(T,take_links_to_move(L1,L2,J,G1,G2,T)) :-
       take_links_to_move(L1,L2,J,G1,G2,T).
r28 action(T,changeAngle(L1,L2,J,A1,A2,G1,G2,T)) :-
       changeAngle(L1,L2,J,A1,A2,G1,G2,T).
r29 action(T,release_links(L1,L2,J,G1,G2,T)) :-
       release_links(L1,L2,J,G1,G2,T).
r30 :- time(T), #count{Z : action(T,Z)} != 1.
r31 in_hand(L,T+1) :- in_hand(L,T), not release_links(L,_,_,_,_,T),
       not release_links(_,L,_,_,_,T), T < timemax+1.
r32 free(G,T+1) :- free(G,T), not take_links_to_move(_,_,_,_,G,T),
       not take_links_to_move(_,_,_,G,_,T), T < timemax+1.
r33 grasped(G,L,T+1) :- grasped(G,L,T), T < timemax+1
       not release_links(L,_,_,G,_,T), not release_links(_,L,_,_,G,T).
```

Figure 4-9: Simple Action Extended Scenario *(SAES)* encoding.

form $release\_links(L1, L2, J, G1, G2, T)$ that signals to the robot to release the links it was acting on, respectively. Rules r_{17} is used to signal which link is on the centre as effects of the action $move\_link\_to\_central(L1, J1, G2, T)$. Rules r_{19} and r_{20} are used to identify the grasped joints, while r_{21} and r_{22} are used to identify which gripper is occupied and which is free. Rules r_{24} and r_{25} are used to notify that the robot's hands are free and they can be used again to grasp. Furthermore, rules from r_{26} to r_{30} ensure that only one action (among the ones represented by rules r_{16}, r_{18}, r_{23} and $r_{5'}$) is selected for each time step. Instead, the rules from r_{31} to r_{33} are used to propagate the information that has not been changed in the current time step to the next one. Overall, the encoding of the Action Planning module for the extended scenario, that we call Simple Action Extended Scenario (*SEAS*), is composed of all the rules of *SAS* (from r_1 to r_{15}), with the necessary modification of r_5 to $r_{5'}$, plus the rules shown in Figure 4-9 (from r_{16} to r_{33}). Note that that rule involving the goal is already present in the *SAS* encoding (i.e., r_{15}).

ASP Encodings for the other ASP Modules. As it is clear from the description of the Action Planning encoding above, to have a working architecture we had to update also other modules to make them coherent with the encoding. With regard to the knowledge base, we had to add some atoms: the two grippers `gripper(1)` and `gripper(2)`, in order to model the presence of the robot grippers, the atoms `free(1,0)` and `free(2,0)`, in order to signal that at time `0` the grippers are always free and ready to grasp and, eventually, an atom of the form `in_center(J,0)`, in order to signal that the joint `J` is at the center of the workspace at time step 0. Moreover, this last change led us to the necessity to add, to the knowledge base, both links and joints: the robot must act on the links but we want that a certain joint is in the middle of the workspace. For this reason the atoms of the form `isLinked(L1,L2)`, where `L1` and `L2` are two links, became `connected(J,L)`, where `J` is a joint and `L` a link.

With regards to the consistency checking module, it has to be updated in order to keep consistency with the rest of the architecture. For this reason, a check on each new atom

that we have added to the knowledge base and encoding must be added.

Macros in the extended scenario

The following macros have been considered:

- `linkToCentral_take`: it is the composition of the action *move_link_to_central*, that moves the articulated object so that the joint in between the links that have to be manipulated is in the centre of the workspace, and *takes_links_to_move*, that grasps the links to be manipulated. As links cannot be grasped by the robot if they are not in the centre of the workspace, this macro aims at providing a single rule for cases where links are not in the right position.

- `changeAngle_release`: it is the composition of *changeAngle*, that changes the angle of a link, and *release_links*, that releases the links currently grasped. This macro aims at providing a single rule for cases where it is necessary to act on a link and then releasing it.

- `take_changeAngle_release`: represents the composition of *takes_links_to_move*, *changeAngle*, and *release_links*. This macro aims at providing a single action for cases where it is necessary to act on a link that was already in the centre of the workspace.

Macros Encoding for the Extended Scenario. A macro action is a set of simpler actions regrouped in one atom. A macro rule is valid only if a combination of the literals composing the bodies of the simpler rules replaced by the macro are satisfied. Moreover, its effects will be a set of the effects of the same replaced actions. The encoding presented in the following is an improved version of the encoding *SAES*, and is based on similar principles of the encodings *SAS* and *SAES*: the general structure and the method the encoding deals with movements are the same shown in Figure 4-7. Figure 4-10 partially

m_1 `{linkToCentral_take(L1,L2,J1,J2,G1,G2,T)} :- link(L1), link(L2), joint(J1), joint(J2), gripper(G1), gripper(G2), time(T), not in_centre(J1,T), connected(J1,L1), connected(J1,L2), free(G1,T), free(G2,T), L1<>L2, G1<>G2.`

m_2 `{changeAngle_release(L1,L2,J,G1,G2,A1,A2,T)} :- link(L1),link(L2), joint(J), gripper(G1), gripper(G2), angles(A1), angles(A2), in_centre(J,T), not free(G1,T), not free(G2,T), in_hand(L1,T), in_hand(L2,T), connected(J,L1), connected(J,L2), grasped(G1,L1,T), hasAngle(L1,A2,T), grasped(G2,L2,T), L1<>L2, G1<>G2.`

m_3 `{take_changeAngle_release(L1,L2,J,A1,A2,G1,G2,T)} :- link(L1), link(L2), joint(J), angles(A1), angles(A2), gripper(G1), gripper(G2), time(T), in_centre(J,T), free(G1,T), free(G2,T), connected(J,L1), connected(J,L2), in_centre(J,T), hasAngle(L1,A2,T), time(T), L1<>L2, G1<>G2.`

Figure 4-10: Macros encoding.

reports how the three macros mentioned in the previous paragraph are encoded in ASP, considering only preconditions and choice rules, and not showing the effects.

Macro m_1 is the composition of the two simple action r_{16} and r_{19}. It requires that only the preconditions of r_{16} are satisfied, i.e., the body of r_{16} corresponds to the body of the macro. The effects of m_1 are represented by r_{17}, r_{20}, r_{21}, r_{22} and r_{23}. Indeed, r_{17} is the effect of r_{16}, whereas the other rules are the effects of r_{19}. It is important to notice that in the *SAES*, r_{17} is necessary to be true to have a r_{19} action. Macro m_2 is the composition of the two simple actions $r_{5'}$, as shown in Figure 4-8, and r_{24}. Its body is composed only from the preconditions of $r_{5'}$ since the preconditions of r_{24} are a subset of them. Instead, its effect are represented by r_{25} and r_{26} (as effects of r_{24}), and by the rules from r_6 to r_{14} (as effects of the rule r_5). Finally, m_3 is the largest macro and it is the composition of the three simple actions r_{19}, $r_{5'}$ and r_{24}. Its body is composed by the literals of r_{19} only since its

effects are preconditions to $r_{5'}$ and r_{24}. Its effects are equivalent to the m_2 ones. Moreover, each macro is supposed to last for only one time-step since the time steps are used only to label the actions sequence. Indeed, the time step are not used as a reference to the real execution time: different simple actions have different execution times and therefore each macro has a different execution time. However, we use time steps in our encoding to ensure that two simple actions are not performed at the same time and, since the action composing the macros are performed in sequence, each macro can be supposed to last for one time-step.

The *MAES* encoding is composed from these three macros and their effects plus (modified versions of) the rules from r_{27} to r_{31}, necessary to ensure that only one action is selected at each time step, the rules from r_{32} to r_{34}, and eventually r_{15} that ensure that the goal is reached. Rules from r_{27} to r_{34} were to be modified according to the rest of the encoding, e.g., rules from r_{27} to r_{30} are reduced to just three rules, one for each macro.

Mixed encodigs for the extended scenario

We developed some mixed encodings, composed by both simple rules and macro rules, to investigate if a combination of the two approaches could lead to better results, with respect to both planning time and the number of action, than the already presented approaches. Our aim is to include the advantages of both the approaches `SAES`, that allows working on the same link without releasing it after each action. and `MAES`, which improves the planning time performances. In order to do so and to exploit the potentiality of the mixed encodings we developed a fourth macro:
`grasp_changeAngle`: represents the composition of *takes_links_to_move* and *changeAngle*. This macro aims at providing a single action for cases where it is necessary to act on a link that was already in the centre of the workspace. Moreover, it allows to not release the link after each action and, therefore, to keep working in the same link.

m_4 {grasp_changeAngle(L1,L2,J,A1,A2,G1,G2,T)} :- link(L1), link(L2), joint(J), angles(A1), angles(A2), gripper(G1), gripper(G2),time(T), in_centre(J,T), free(G1,T), free(G2,T), connected(J,L1), connected(J,L2), grasped(G1,L1,T), grasped(G2,L2,T), hasAngle(L1,A2,T), L1<>L2, G1<>G2.

Since they are just a composition of the rules composing SAES (see figure 4-9)and MAES (see Paragraph 4-10) plus the bove explained macro we will just list the rules that compose the new encodings. We developed three mixed encoding:

- **Macros Action Prevalent (MAP)-** it is composed by: $r5'$ from SAES, m_1, m_2, m_3 and m_4 from MAES;
- **Simple action prevalent (SAP) -** it is composed by: $r5'$, r_{19} and r_{24} from SAES, m_1 and m_2 from MAES;
- **Equally Distributed (ED) -** it is composed by: all the simple actions from SAES ($r5'$, r_{16}, r_{19} and r_{24}) plus all the macros from MAES (m_1, m_2, m_3 and m_4).

Chapter 5

Validation of the Framework and Performances Analysis

5.1 Validation of the Framework

5.1.1 Simple Scenario

A validation scenario where a robot has to manipulate a 5-link articulated object has been set up both in simulation and in real-world using the Baxter dual-arm manipulator. Objects composed by 5 links provide a very valuable ground for testing our approach, as they are not so long to make the manipulation difficult for the robot, and at the same time they are articulated enough to require to plan movements in order to reach a goal configuration. The use of Baxter is justified by its widespread adoption as a research platform and by the necessity to employ a robot with two arms to manipulate the object, i.e., the robot should be able to keep a link of an object with one arm while it rotates an adjacent one.

Simulations have some practical advantages in this scenario. Indeed, they allow running a greater number of planning-execution cycles with minimal human supervision and shorter

```
a1 : changeAngle(2,1,90,180,1)      a2 : changeAngle(1,0,180,90,2)
a3 : changeAngle(3,2,90,180,3)      a4 : changeAngle(1,0,270,180,4)
```

Figure 5-1: The planning and execution process on the sample scenario: an excerpt of the answer set returned by Clingo ($a_1 \ldots a_4$ are compact references for the ground actions).

execution times. Moreover, they are less susceptible to uncertainty and low-level motion planning failures, which are outside of the scope of this work. Nevertheless, we also test with the real robot to provide a more robust proof-of-concept of the proposed architecture. A video showing the Baxter in operation, via the introduced framework, can be found at https://tinyurl.com/yd6kqgjn.

In our setting, we employed (i) ALVAR, an AR tag tracking library, to detect the absolute pose of the object's links using a head-mounted camera; and (ii) MoveIt!, as the de facto standard for motion planning and execution in the robotic community. The system was implemented in the Robot Operating System (ROS, Indigo release) framework, while Gazebo 2 was used as a simulation environment for the relevant part. The system has been tested on a machine with an Intel i7-4790 CPU and 16 GB of RAM. All the results of the evaluation are available at https://tinyurl.com/ydzyefux.

The evaluation procedure unfolds as follows. First, the object is set up in a random configuration coherent with the specified granularity and within an acceptable margin of error. The initial and goal configurations are then represented in terms of the ASP atoms reported in Section 4.4.1, and processed by the state-of-the-art ASP system Clingo [39] together with the encoding in Section 4.4.4 in order to generate a (valid) plan. Actions of the plan are then executed through the low-level motion planning layer, where an action consists of rotations around the object's joints perpendicular axes.

```
link(1..5). joint(1..4).
#const granularity = 60. time(0..timemax).
angle(0). angle(60). angle(120).
angle(180). angle(240). angle(300).
connected(1,1). connected(1,2). connected(2,2). connected(2,3).
connected(3,3). connected(3,4). connected(4,5). connected(4,5).
hasAngle(1,0,0). hasAngle(2,90,0). hasAngle(3,0,0).
hasAngle(4,60,0). hasAngle(5,120,0).
goal(1,0). goal(2,90). goal(3,0). goal(4,300). goal(5,300).
```

a_{s1} : take_links_to_move(4,3,3,0,1,0)
a_{s2} : changeAngle(4,3,3,0,60,0,1,1)
a_{s3} : changeAngle(4,3,3,300,0,0,1,2)
a_{s4} : release_links(3,4,3,1,0,3)
a_{s5} : move_link_to_central(4,4,0,4)
a_{s6} : take_links_to_move(5,4,4,1,0,5)
a_{s7} : changeAngle(5,4,4,300,0,1,0,6)

a_{m1} : take_changeAngle_release(4,3,3,0,60,0,1,0)
a_{m2} : take_changeAngle_release(4,3,3,300,0,0,1,1)
a_{m3} : linkToCentral_take(5,4,4,4,0,1,2)
a_{m4} : changeAngle_release(5,4,4,0,1,300,0,3)

Figure 5-2: mple on the extended scenario. The knowledge base of the problem solved with both *SAES* and *MAES* (top). Then, two excerpts of the answer set returned by Clingo: when *SAES* is employed (middle), and when *MAES* is employed (bottom).

5.1.2 Extended Scenario

We validated the extended scenario in real-world using the Baxter dual-arm manipulator, employing the same experimental setting already described in Section 5.1. For both the two new approaches our aim is to compare and underline advantages and drawbacks of the two encodings; objects composed by 5 links provide a very valuable ground for this purpose, as they are not so long to make the manipulation difficult for the robot, and at the same time they are articulated enough to require to plan movements in order to reach a goal configuration.

Figure 5-2 shows the knowledge base related to the considered sample instance, as well as two excerpts of the answer set produced by Clingo when using the *SAES* and *MAES* encoding, respectively. It is possible to appreciate the differences in the knowledge base with regards to the simple scenario, as explained in Section 4.4.5. Furthermore, we can compare the solutions generated by the two extended encodings:

- *SAES* plan: a_{s1} signals that the robot will take the links 3 and 4 with the left and right gripper, respectively; a_{s2} signals that the angle of link 4 will change from 60 degrees to 0; a_{s3} signals that the angle of link 4 will change from 0 degrees to 300; a_{s4} signals that the two links will be released; a_{s5} signal that joint 4 (between link 4 and 5) will be placed at the centre of the workspace; a_{s6} signals that the robot will take links 4 and 5 with the left and right gripper, respectively; and finally a_{s7} signals that the angle of the link 5 will change from 0 degrees to 300.

- *MAES* plan: a_{m1} signals that the robot will take links 3 and 4 with the left and right gripper, respectively, and that the angle of link 4 will change from 60 degrees to 0. After, the robot will release the links; a_{m2} signals that the robot will take links 3 and 4 with the left and right gripper, respectively, and that the angle of link 4 will change from 0 degrees to 300. Then the robot will release the links; a_{m3} signals that the two links will be released; finally a_{m4} signals that the angle of the link 5 will change from 0 degrees to 300.

While the two solutions share several similarities, there are nonetheless a few remarkable differences. It is possible to notice that with the macros it is not possible to act on a link without releasing it just after: this leads to a small unnecessary idle time if the robot has to move a link two times in a row since it has to release it and grasp it again just after. We avoid that problem in *SAES* since the two actions *changeAngle* and *releaseLinks* are not encapsulated in a macro. Nonetheless, it is evident, from the tables shown in Figure 5.3, that the macros improve considerably the planning performances and the success

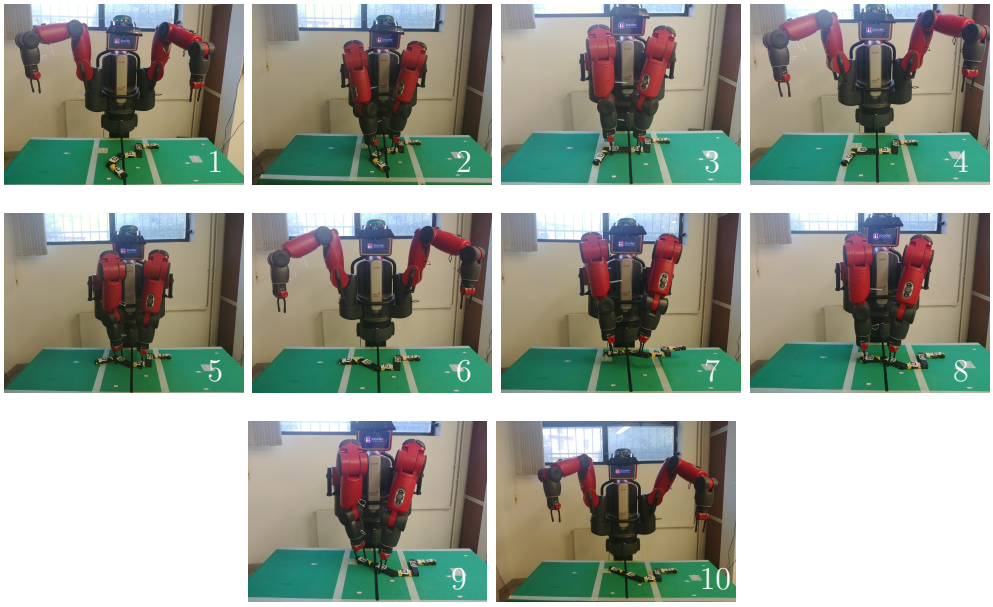


Figure 5-3: The planning and execution process on the extended scenario: the robot actions and (intermediate) states induced by the computed plan.

rate. To summarize, introducing the macros leads to better runtime performance, but at the cost of reintroducing some idle robot time in certain specific cases. This is aligned with the results achieved in automated planning when exploiting macros: runtime and coverage performance of planners tend to improve, but the quality of generated plans may be negatively affected due to repetitions and suboptimalities [17].

Figure 5-3 illustrates the execution of the solution generated using the *MAES* encoding, on the sample instance at the top of Figure 5-2, which produces the (partial) answer set at the bottom of the same figure. Starting from the initial configuration of the articulated object (Figure 5-3.1), Figures 5-3.2, 5-3.3 and 5-3.4 represent the three steps of the macro action a_{m1}, i.e.,grasping, changing the angle and releasing the links, respectively (see Figure 5-2). Figures 5-3.5, 5-3.6 represent the action a_{m2} that is almost equivalent to the action a_{m1}. Figure 5-3.7 and Figure 5-3.8 represent two steps of the action a_{m3} that bring the fourth joint to the center of the workspace, while Figure 5-3.9 represent the changing angle

step of the action a_{m4}. Finally, Figure 5-3.10 shows the final configuration that corresponds to the required goal configuration of the 5-link articulated object. It is important to notice that, for sake of brevity, we skipped some of the steps of the actions a_{m2} and a_{m3} since we already showed the general behaviour of the actions of the form *take_changeAngle_release* in Figure 5-3.2, 5-3.3 and 5-3.4. It is interesting to notice that, apart from the steps shown in Figure 5-3 that would have been skipped by the *SAES* plan (see Figure 5-2) reducing the robot idle time, the flow of the actions during the execution process of *MAES* and *SAES* is almost equivalent even if the organization of the plan is different.

5.2 Performances and Data Analysis

We compared the planning performance of the considered encodings: execution time on a real environment or in the simulation are not taken into account since we want to evaluate the ASP language effectiveness on this planning task without independently from the used robot or simulation tool. To give an overview using coverage (percentage of solved instances) and PAR10. Penalised Average Runtime (PAR10) score is a metric usually exploited in machine learning and algorithm configuration techniques. This metric trades off coverage and runtime for solved problems: if an encoding e allows the solver to solve an instance Π in time $t \leq T$ (T = 300s in our case), then $PAR10(e, \Pi) = t$, otherwise $PAR10(e, \Pi) = 10 \times T$ (i.e., 3000s in our case). This allows us to have an overview of the performances of the strategies under exam.

5.2.1 Simple Scenario

In order to obtain an overview of the capabilities of the encodings presented in the previous chapter in Section 4.4.4, we needed to generate different instances for the simple scenario. The two main factors changing from instance to instance are the `maximum number of angles that a link can be positioned to ("#na")` and the `number of links composing the articulated object ("#nl")`. For each couple `#na`, `#nl`

we generated 10 instances, with different initial states and goal configurations, and we assumed `#na ∈ [2,14]` and `#nl ∈ [3,12]`. This brought us to a total of 400 instances. This allows us to have a comparison between two fairly enough similar architectures developed in different languages opening to us the possibility to analyse the advantaged and drawbacks of our architecture.
Therefore, each result table contains different set-ups: **Standard** in which results obtained with the ASP solver Clingo using the Standard strategy; **Heuristic** in which results obtained with the ASP solver Clingo using the Heuristic strategy and **Not Optimal** in which results obtained with the ASP solver Clingo using the Not Optimal strategy. Each table will contain experiment for a fixed `#na` with one row for each `#nl`.

Experimental Data Analysis

Here we will show the results we obtained from the tests with the different encodings. Moreover, we analyse the results with a small paragraph for each strategy.

Standard Strategy It ensures an optimal solution but at the cost of longer execution time to compute the plan. This makes this strategy more suitable for offline planning, or rather it is more suitable for problems that do not often need replanning in real-time. However, for smaller or medium plans, we ensure always an optimal solution in term of the number of action. It is important to notice that a robot, e.g. a dual-arm Baxter robot, requires a not negligible time to perform an action, therefore, even if the planning time is longer the execution time of the plan can be much shorter. Indeed, the trade-off `planning time - execution time` it is not always trivial to compute: often it is better to lose time during the planning stage in order to reduce the execution of the plan itself. Moreover, the optimality of the solution brings another advantage: the less action is in the plan the less is the probability that the low-level planner, `Mooveit!` in our case (see Chapter 4), will fail.

Heuristic Strategy It ensures an optimal solution as the standard strategy. With this approach we avoid the need for external script and, as shown from the tables, it usually performs worst to the standard strategy on a problem involving a small/medium amount of links. However, on a problem with a higher number of links and possible orientations, it is usually able to find more solutions than the standard strategy.

Not Optimal As we expected the computed solutions are way larger on the solutions computed on a smaller problem by the two previous approaches. We expected a not optimal solution but we also expected to have a much faster execution time on some problem due to its randomicity. Indeed, as the tables show, this algorithm is able to find solutions for some of the problems involving a high number of links with suitable planning time, i.e., this is the algorithm that performs best with 12 possible orientations. The main problem of the found solutions is their length: since the number of allowed steps is random, often the found plan is composed of a high number of actions. This leads to a short planning time but a really long execution time for the robot making these solutions not exploitable for a real environment.

5.2.2 Extended Scenario

In order to obtain an overview of the capabilities of the considered encodings, we used as test problems the same problems, with the due adaptations, of the Simple Scenario. Eventually, we had 320 instances with the number of links varying from 4 up to 12 and with granularity values of 4, 6, 8 or 12 possible angles, 10 instances for each pair (number of links, granularity). For each testing instance, time limit of 300 seconds and the memory limit of 16 GB was applied. Clingo was used to solve the ASP-encoded instances. All the experiments were conducted on Intel i7-4790 CPU and Linux OS.

We compared the performance of the considered encodings using coverage (percentage of solved instances) and PAR10.

Number Angles: 4

PAR10

	Standard	Heuristic	Not Optimal
3	0.0	0.0	0.09
4	0.0	0.13	0.37
5	0.04	0.2	0.9
6	2.8	1.93	0.0
7	10.38	3.44	0.09
8	1235.98	327.74	0.12
10	1339.36	1547.04	1878.5
11	2105.99	1814.01	2412.34
12	2704.99	2420.69	3000.0
14	3000.0	2721.45	2704.5

Coverage

	Standard	Heuristic	Not Optimal
3	100	100	120
4	100	100	100
5	100	100	100
6	100	100	100
7	100	100	100
8	60	90	100
10	50	50	40
11	30	40	20
12	10	20	0
14	0	10	10

Number Angles: 6

PAR10

	Standard	Heuristic	Not Optimal
3	0.0	0.13	0.22
4	0.08	0.73	1.38
5	1.52	2.05	6.75
6	636.69	326.72	5.24
7	3000.0	2139.18	11.74
8	2403.78	1822.12	39.71
10	2697.74	3000.0	2719.62
11	2719.29	3000.0	3000.0
12	3000.0	3000.0	3000.0
14	3000.0	3000.0	3000.0

Coverage

	Standard	Heuristic	Not Optimal
3	100	100	100
4	100	100	100
5	100	100	100
6	80	90	100
7	0	30	100
8	20	40	100
10	10	0	10
11	10	0	0
12	0	0	0
14	0	0	0

Number Angles: 8

PAR10

	Standard	Heuristic	Not Optimal
3	0.0	0.31	1.11
4	1.93	10.03	6.17
5	331.61	54.22	42.46
6	1545.12	1837.03	42.77
7	2723.41	2720.7	1003.58
8	3000.0	2712.66	2707.24

Coverage

	Standard	Heuristic	Not Optimal
3	100	100	100
4	100	100	100
5	90	100	100
6	50	40	100
7	10	10	70
8	0	10	10

Number Angles: 12

PAR10

	Standard	Heuristic	Not Optimal
3	0.29	2.83	2.92
4	335.42	1237.28	26.8
5	1553.26	2136.08	2.95
6	3000.0	3000.0	1261.75
7	3000.0	3000.0	2418.21
8	2429.32	3000.0	3000.0

Coverage

	Standard	Heuristic	Not Optimal
3	100	100	100
4	90	60	100
5	50	30	100
6	0	0	60
7	0	0	20
8	20	0	0

Table 5.1: Results, in terms of PAR10 and coverage, achieved by the considered encodings on the testing instances. Instances are grouped according to the number of links of the articulated object to be manipulated (rows) and the granularity of the angular values, with 10 instances for each pair (number of links, granularity). Cases not solved by any considered encoding are omitted.

Experimental Data Analysis

Here we will show the results we obtained from tests with the different encodings. Moreover, we make a comparative analysis between each strategy.

The above tables summarises the results achieved by Clingo for solving instances encoded in the *Standard Strategy*, *SAES*, and *MAES*. It is worth reminding that the *Standard Strategy*, showed in Section 4.4.4, encoding is much more simplistic than the others, as it ignores the position of the links to be manipulated, and considers only one action that has to be broken down into a large number of low-level primitives from the low-level planner, MoveIt! in our case. Therefore, the planning result to be superficial and it leads to failures during the execution. On the contrary, *SAES* and *MAES* encodings provide a more detailed and rich description of the problem, that allows to generate plans that are easier to be put in place by the manipulator. So, a direct comparison between the *Standard Strategy* and *SAES/MAES* is not possible, but it is nonetheless interesting to have also the results obtained by the *Standard Strategy*. Unsurprisingly, the *Standard Strategy* encoding allows to solve a larger number of instances and is generally the fastest among the considered encodings. Due to the above-mentioned issues, the excellent planning performance of *Standard Strategy* comes at the cost of a potentially large number of failures in execution, since the encoding does not take into consideration the boundaries of the robot workspace: during the execution of the computed solutions the planner may ask the robot to perform actions on joints that are at the limit of the robot workspace, leading the arms to singularity positions and consecutively stopping the execution. It is trivial to notice that the more joints the articulated object has, the easier it is to have these failures.

The comparison of the performance achieved by Clingo when using the *SAES* and *MAES* encodings can shed some light on the usefulness of the macros. It is easy to notice that the use of macros allows Clingo to solve a larger number of instances and that macros are generally helpful in improving the runtime. This is true regardless of the granularity considered for the manipulation of the angles, and the number of considered

Number Angles: 4

PAR10

	Standard	SAES	MAES
4	0.0	3.82	0.66
5	0.04	929.23	9.9
6	2.8	2104.76	665.19
7	10.38	3000.0	1265.75
8	1235.98	3000.0	2448.22
10	1339.36	3000.0	3000.0
11	2105.99	3000.0	3000.0
12	2704.99	3000.0	3000.0

Coverage

	Standard	SAES	MAES
4	100	100	100
5	100	70	100
6	100	30	80
7	100	0	60
8	60	0	20
10	50	0	0
11	30	0	0
12	10	0	0

Number Angles: 6

PAR10

	Standard	SAES	MAES
4	0.08	16.36	4.43
5	1.52	2402.65	427.06
6	636.69	3000.0	2701.3
7	3000.0	3000.0	2439.51
8	2403.78	3000.0	3000.0
10	2697.74	3000.0	3000.0
11	2719.29	3000.0	3000.0
12	3000.0	3000.0	3000.0

Coverage

	Standard	SAES	MAES
4	100	100	100
5	100	20	90
6	80	0	10
7	0	0	20
8	20	0	0
10	10	0	0
11	10	0	0
12	0	0	0

Number Angles: 8

PAR10

	Standard	SAES	MAES
4	1.93	940.82	64.16
5	331.61	2726.58	2423.15
6	1545.12	3000.0	3000.0
7	2723.41	3000.0	3000.0
8	3000.0	3000.0	3000.0

Coverage

	Standard	SAES	MAES
4	100	70	100
5	90	10	20
6	50	0	0
7	10	0	0
8	0	0	0

Number Angles: 12

PAR10

	Standard	SAES	MAES
4	335.42	1824.43	1252.49
5	1553.26	3000.0	3000.0
6	3000.0	3000.0	3000.0
7	3000.0	3000.0	3000.0
8	2429.32	3000.0	3000.0

Coverage

	Standard	SAES	MAES
4	90	40	60
5	50	0	0
6	0	0	0
7	0	0	0
8	20	0	0

Table 5.2: Results, in terms of PAR10 and coverage, achieved by the considered encodings on the testing instances. Instances are grouped according to the number of links of the articulated object to be manipulated (rows) and the granularity of the angular values, with 10 instances for each pair (number of links, granularity). Cases not solved by any considered encoding are omitted.

links. Remarkably, with a granularity value of 12 (i.e., joints' angle can be modified by 30 degrees per movement), the use of macros allows Clingo to solve 60% of instances of size 4; 40% of instances can be solved using *SAES*, instead.

Mixed Encodings In order to obtain an overview of the capabilities of the considered encodings we used as test problems the same problems as we did for the two encodings *SAES* and *MAES*. For each testing instance, time limit of 300 seconds and the memory limit of 16 GB was applied. Clingo was used to solve the ASP-encoded instances. All the experiments were conducted on Intel i7-4790 CPU and Linux OS.

As for the other strategies we compared the performance of the considered encodings using coverage (percentage of solved instances) and PAR10.

The two encodings *SAP* and *MAP* do not show any improvement, on the contrary, they show a deterioration of the performances with respect to the *MAES encoding* solving fewer instances and with worst planning times. However, the *ED* encoding shows unexpected improvement in both the number of solved instances and planning time performances. It is possible to notice from the tables that for a problem with a medium/high number of links and possible orientation the *ED* encoding has not only a higher solving capability but also better performances. However, this is not true with a low number of possible orientations: under this condition the *MAES* encoding still perform better. From these results we can deduce that giving to the solver high degree of freedom of choice does not lead to worst performances but, on the contrary, it gives the possibility to the planner to find solutions faster. These results, therefore, suggest that the encoding composed by both simple and macro actions can improve the performances of the planner but this is not the only advantage: a plan composed by both the action typology may be able to avoid idle robot time in a more efficient way than a plan composed by only one of the two. Indeed, this could lead to better execution performances as well.

Number Angles: 4

PAR10

	SAP	MAP	ED
4	1.76	3.03	2.37
5	24.36	34.61	16.04
6	940.29	1537.17	616.56
7	2714.4	3000.0	1215.37
8	3000.0	3000.0	1864.42
10	3000.0	3000.0	2705.43
11	3000.0	3000.0	2716.35

Coverage

	SAP	MAP	ED
4	100	100	100
5	100	100	100
6	70	50	80
7	10	0	60
8	0	0	40
10	0	0	10
11	0	0	10

Number Angles: 6

PAR10

	SAP	MAP	ED
4	8.18	11.73	11.66
5	2400.76	2400.97	26.11
6	2706.21	2707.48	2701.66
7	3000.0	3000.0	2122.22

Coverage

	SAP	MAP	ED
4	100	100	100
5	20	20	100
6	10	10	10
7	0	0	30

Number Angles: 8

PAR10

	SAP	MAP	ED
4	949.48	1236.45	67.27
5	2717.16	2426.75	1530.97
6	3000.0	3000.0	2418.39

Coverage

	SAP	MAP	ED
4	70	60	100
5	10	20	50
6	0	0	20

Number Angles: 12

PAR10

	SAP	MAP	ED
4	1833.73	2115.91	936.48
5	2726.73	3000.0	2712.49

Coverage

	SAP	MAP	ED
4	40	30	70
5	10	0	10

Table 5.3: Results, in terms of PAR10 and coverage, achieved by the considered encodings on the testing instances. Instances are grouped according to the number of links of the articulated object to be manipulated (rows) and the granularity of the angular values, with 10 instances for each pair (number of links, granularity). Cases not solved by any considered encoding are omitted.

Chapter 6

Automated Planning Encodings for the Manipulation of Articulated Objects in 3D with Gravity via PDDL+

In this chapter, we introduce the languages **PDDL** and **PDDL+** with a set of formulations for performing automated manipulation of articulated objects in a three-dimensional workspace by a dual-arm robot. We present some formulations that differ in terms of how gravity is modelled, considering different trade-offs between modelling accuracy and planning performance, and between human-readability and parsability by planners. Our experimental analysis compares the formulations on a range of domain-independent planners, that aim at generating plans for allowing a dual-arm robot to manipulate articulated objects of different sizes. Validation is performed in simulation on a Baxter robot. As we previously underline, the manipulation of articulated objects plays an important role in real-world robot tasks, both in-home and industrial environments [54, 64]. In liter-

ature, the problem of determining the two- or three-dimensional (2D or 3D) configuration of articulated or flexible objects has received much attention in the past few years [8, 13, 14, 78, 97], whereas the problem of obtaining a target configuration via manipulation has been explored in motion planning [9, 94]. However, the employed manipulation strategies are often crafted specifically for the problem at hand, with the relevant characteristics of the object and robot capabilities being either hardcoded or assumed, thus undermining generalisation and scalability. More general solutions [1, 14] are limited to 2D configuration, with a partial exception for the work in [1], where the notion of *overlap* between different parts of a cable is explicitly considered.
To sum up, in this section we present:

- Two sets of PDDL+ models for the task of automated, robot-based manipulation of articulated objects in a 3D workspace. Each set of models considers three different levels of complexity for representing the effect of gravity.
- We analyse the performances of a number of domain-independent PDDL+ planners on realistic articulated object manipulation tasks.

We also validate generated plans using a robot control architecture for a dual-arm robot manipulator in simulation. As a side effect of our work, we provide a challenging domain, and its PDDL+ models, to the planning community.

6.1 PDDL

PDDL (Planning Domain Description Language) is a standard encoding language for "classical" planning. The components of PDDL files are:

- **Requirement**s: defining levels of abstraction in the language, e.g., "STRIPS", temporal, probabilistic effects etc.
- **Types**: sets of the things of interest in the world,

- **Objects**: instances of types,
- **Predicates**: Facts about objects that can be true or false,
- **Initial state of the world**: before starting the planning process,
- **Goal**: properties of the world true in goal states and achieved after the planning process,
- **Actions/Operators**: ways of changing states of the world and going from the initial state to goal states.

A planning task in PDDL is specified in two text files:

- A domain file for requirements, types, predicates and actions,
- A problem file for objects, initial state and goal specification.

Domain Files. Domain files are as follows:

$$
\begin{aligned}
&(define\ (domain\ <domain\ name>)\\
&<PDDL\ code\ for\ requirements>\\
&<PDDL\ code\ for\ types>\\
&<PDDL\ code\ for\ predicates>\\
&<PDDL\ code\ for\ first\ action>\\
&\ldots\\
&<PDDL\ code\ for\ last\ action>\\
&)
\end{aligned}
$$

Where ¡domain name¿ is a string that identifies the planning domain. Examples are available in these repositories: Logistics, Depots, Gripper, Blocksworld [84].

Problem Files. Problem files are as follows:

$$
\begin{aligned}
&(define\ (problem\ < problem\ name >)\\
&(:\ domain\ < domain\ name >)\\
&< PDDL\ code\ for\ objects >\\
&< PDDL\ codefor\ initial\ state >\\
&< PDDL\ code\ for\ goal\ specification >\\
&)
\end{aligned}
$$

Where ¡problem name¿ is the string that identifies the planning task, e.g. gripper with 4 balls to move. ‘ is the planning domain name corresponding to problem file. Examples are available in these repositories:Logistics, Depots , Gripper, Blocksworld 4[84].

6.2 Problem Statement

Among the tasks typically carried out in shop-floor environments, the manipulation of flexible objects, e.g., cables [53, 91], is particularly challenging. On the one hand, it is beneficial to plan the target cable configuration in advance; on the other hand, it is often necessary to keep a cable firmly using one grasping point to be able to manipulate other parts. A robot capable of manipulating flexible objects in its 3D workspace must be able to: (i) represent object configurations adopting suitable modelling assumptions, and then segment the whole manipulation problem in simpler actions to be sequenced and performed, each action operating in-between two intermediate 3D object configurations; and (ii) represent the actions to carry out using a formalism which allows for robust plan execution and modelling inaccuracies.

These requirements lead to a robot perception and control architecture characterized by the following features: (a) similarly to the approach described in [1], the robot plans an appropriate sequence of actions to determine relevant 3D intermediate configurations for articulated objects (i.e., a suitable *simplified* model for a flexible object like a cable) in order

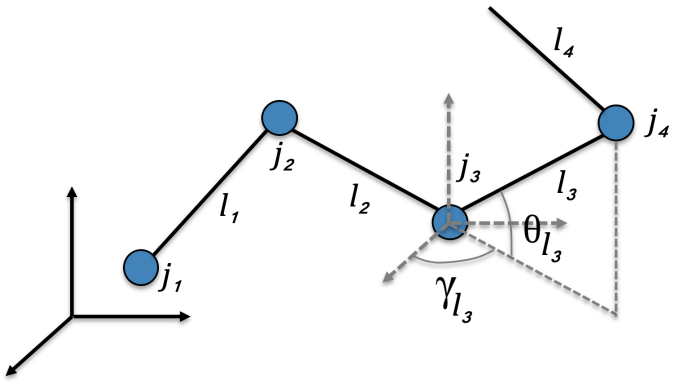


Figure 6-1: A 3D articulated object configuration.

to determine a target 3D configuration; and (b) during plan execution, the robot monitors the outcome of each action, and compares it with the intermediate target configuration to achieve. The problem we consider in this paper can be defined as follows: given a target object configuration in 3D space, determining a plan $\mathcal{P}$ to obtain it as an ordered set of actions $\mathcal{P} = \{a_1, \ldots, a_i, \ldots, a_N; \prec\}$, where each action a_i involves one or more 3D manipulation operations to be executed by a dual-arm robot. We pose a number of assumptions described as follows:

1. the effects of gravity on all articulated object's 3D configurations are explicitly considered;

2. we do not assume any specific grasping or manipulation strategy to obtain a target 3D object configuration starting from another configuration;

3. the perception of articulated objects, although affected by noise, is considered *perfect*, i.e., data association is given.

We define an articulated object in 3D as a 2-ple $\alpha = \langle \mathcal{L}, \mathcal{J} \rangle$, where $\mathcal{L}$ is the ordered set of its L links, i.e.,

$$\mathcal{L} = \{l_1, \ldots, l_j, \ldots, l_L; \prec\}, \tag{6.1}$$

and $\mathcal{J}$ is the ordered set of its $J = L - 1$ joints, i.e.,

$$\mathcal{J} = \{j_1, \ldots, j_k, \ldots, j_J; \prec\}. \tag{6.2}$$

Each link l is characterised by three parameters, namely a length, and two orientations θ_l and γ_l, expressed with respect to a robot-centred reference frame (Figure 6-1). We allow only for a limited number of discrete orientation values, i.e., θ_l and γ_l can take values from a pre-determined set of possible values. Given a link l_j, upstream links are those from l_1 to l_{j-1}, whereas downstream links range from l_{j+1} to l_L. Such absolute representation leads to the direct perception of links and their orientations. When a sequence of manipulation actions is planned, changing one absolute orientation requires, in principle, the propagation of such change upstream or downstream the object via joint connections. Given an articulated object α, its configuration is a L-ple:

$$\mathcal{C}_\alpha = \{(\theta, \gamma)_1, \ldots, (\theta, \gamma)_l, \ldots, (\theta, \gamma)_L\}, \tag{6.3}$$

where it is intended that the orientations θ and γ are expressed with respect to an *absolute*, robot-centred, reference frame.

6.3 Formulation

In order to address the problem introduced above, we exploited PDDL+ to formulate three different domain models, corresponding to three different levels of abstraction of the impact of gravity on the articulated object.

PDDL+ [36] is an extension of the standard planning domain modelling language, PDDL, to model mixed discrete-continuous domains. In addition to instantaneous and durative actions, PDDL+ introduces *continuous processes* and *exogenous events*, that are triggered by changes in the environment. Processes are used to model continuous change,

and therefore are well suited in this context to model the impact of gravity on articulated objects.

The absolute representation of angles we employ, on the one hand, reduces the burden on the robotic framework, because the orientations of links are directly observable by the robot perception system and do not require any additional calculation. On the other hand, the complexity of the planning process is increased, due to the fact that any manipulation action has to be propagated to all the upstream or downstream link orientations. The interested reader is referred to [14] for an extensive comparison of different joint angles representation techniques in a 2D setting.

In the proposed PDDL+ models, a `connected` predicate is used to describe the fact that two links are jointed. Joints are not explicitly modelled: the `connected` predicate indicates the presence of a joint between the two involved links, and the orientation is given via the `angle` function, which indicates the absolute orientation of the link l_i with regards to a plane j. The value of angles ranges between 0 and 359 degrees. The effect of the manipulation of two `connected` links are propagated via a corresponding `affects` predicate. In order to reduce the computational complexity, we fixed the way in which the robot can manipulate two connected links so that propagation can only happen upstream. In other words, given two consecutive links, we allow the robot to move only the upstream link, while the other one is kept fixed. It should be noted that, if needed, the model can be easily extended to deal with both up and downstream manipulation by adding the appropriate predicates. The number of planes that can be represented is not fixed and can be easily modified: in our evaluation, we considered 2 planes, vertical and horizontal, corresponding to a 3D space.

The planner can modify the orientation of links using the following constructs:

- An operator `start-increase(l1,l2,plane)` is used by the planner to manipulate the orientation of the link l_2 on the plane *plane*, by using a gripper for keeping l_1 still, and another gripper for moving l_2.

- A process `move-increase(l2, plane)` is used for modelling the continuous movement performed by the robot to increase the absolute angle related to l_2 on the corresponding *plane*. This process is activated by the above operator.
- An operator `stop-increase(l1,l2,plane)` is activated by the planner to stop the modification of the orientation of the l_1 and l_2 links. The robot is therefore releasing the two links.
- The events `back-to-zero(l, plane)` and `back-to-360(l, plane)` are triggered when the value of the angle of link l on *plane* reaches, respectively, 360 or 0. In the former (latter) case, the value of the angle is reset to 0 (359).
- A process `propagate-increase(l1,l2,plane)` is activated when a process `move-increase(l2, plane)` is ongoing, and it allows to propagate the effects of the current manipulation on all the affected upstream angles.

In a nutshell, the planning engine can modify the angle between two connected links via the operator `start-increase(l1,l2, plane)`: this starts the movement process, that can be stopped by the engine using another dedicated operator. The movement process is also impacting a (potentially long) cascade of processes that models the propagation of the manipulation to affected upstream angles.

The above-listed constructs are in charge of performing and modelling manipulations aimed at increasing angles. A corresponding set of constructs is used to allow the planner to decrease some specified angles. Figure 6-2 shows the PDDL+ encoding of the `start-increase` operator and the `back-to-zero` event. Notably, the predicate `in-use` is exploited to avoid parallel manipulations of the articulated object by the robot. This is because the robot's grippers are not explicitly modelled, therefore many different actions could potentially be planned in parallel by the planning engine. The `freeToMove` predicates are used to indicate if a link is currently being manipulated or not; these predicates are a sort of token for grasping a specific link.

```
(:action start-increase
:parameters (?l1 -link ?l2 -link ?x -plane)
:precondition (and  (connected ?l1 ?l2)
(not (in-use)))
:effect (and  (in-use)
(not (freeToMove ?l2))
(not (freeToMove ?l1))
(increasing_angle-robot ?l2 ?x)))

(:process move-increase
:parameters (?l2 -link ?x -plane)
:precondition
(increasing_angle-robot ?l2 ?x)
:effect
(increase (angle ?l2 ?x)(* #t (speed-i))))

(:event back-to-zero
:parameters (?l3 -link ?x -plane)
:precondition
(>= (angle ?l3 ?x) 360)
:effect
(assign (angle ?l3 ?x) 0))
```

Figure 6-2: Part of the proposed PDDL+ formulation.

6.3.1 Modelling Gravity

Gravity is one of the main reasons for encoding a model in PDDL+, as gravity effects are (i) continuous in nature, and (ii) not under the direct control of the planning engine. For these reasons, PDDL+ constructs such as continuous processes and events are extremely handy for describing the impact of gravity on the 3D manipulation of an articulated object.

The representation of the effects of gravity on an articulated object can be cumbersome and may prevent the generation of valid plans in a reasonable amount of time. Because of that, we introduce three different levels of complexity that can be implemented in the proposed PDDL+ model. It is worth noting that the typical articulated object, in order to support the manipulation via a robot, has quite stiff joints, which are therefore resisting –up to some degrees– to the gravity effect.

No Gravity (NoG). The most trivial way to reduce the complexity burden due to the computation of the effects of gravity on the articulated object is, of course, to completely ignore gravity. In cases where the joints of the articulated objects are extremely stiff, this model can still give some useful information to the robot. Notably, the reduced complexity may allow to quickly re-plan in cases where the robot observes that gravity has significantly modified the configuration of the object.

Uniform Circular Motion (MCU). A more sophisticated way of modelling the impact of gravity on an articulated object can be obtained by taking a joint-by-joint perspective. As links are connected by joints, they can not fall to the ground but are bound to each other by the joints. The impact of gravity on a joint angle can be modelled as a uniform circular motion that moves the angle towards a value of 360 (if we consider a 180-degree angle to be on the z axis). In this encoding, the angular speed is constant. The impact of gravity on an angle is modelled using a pair of dedicated processes (according to the fact that the initial angle is lower or higher than 180) and, due to the fact that angles are absolute, such motion is also propagated to all the affected joints via a different PDDL+ process. The effects of gravity on a joint can be stopped for two reasons: (i) the angle has reached the rest position (360/0 degrees), or (ii) the corresponding link has been grabbed by the gripper of the robot.

Uniformly Accelerated Circular Motion (MCUA). Building on top of the MCU formalisation, we introduce a more advanced representation of the impact of gravity by modelling it as a uniformly accelerated circular motion. As before, all joints angle tend to return to a 360 degree position on the z axis. However, their initial angular speed is 0, but it is uniformly accelerated. The acceleration is encoded in PDDL+ employing an additional process, that is in charge of increasing the angular speed while the gravity effect is active on a specific joint, and an appropriate event that "resets" the speed value when the effect of gravity ends.

```
(:process gravity-increase
:parameters (?l1 - link)
:precondition (and  (freeToMove ?l1)
(> (angle ?l1 ZAXES) 180)
(< (angle ?l1 ZAXES) 360))
:effect (and
(increase
(angle ?l1 ZAXES) (* #t (speed-g)))
(increasing_angle-gravity ?l1)))
```

Figure 6-3: The process used in the MCU formulation to model the effect of gravity on angles between 180 and 359 degrees.

As for the manipulation of angles performed by the robot, also in the MCU and MCUA formulations, the effect of gravity on an angle is propagated to all the affected joints by a set of dedicated processes and events. An example of the process exploited in the MCU formulation for modelling gravity is provided in Figure 6-3.

6.3.2 Alternative Formulations

The process presented in Figure 6-3, as the dual `gravity-decrease` and those exploited in the MCUA formulation, has been modelled in the most human-readable way, due to the fact that robotics experts have to be involved in the modelling process. For this reason, the process keeps true the Boolean predicate `(increasing_angle-gravity ?l1)`, that is used to represent the fact that gravity is impacting the corresponding link. Semantically, this implies that every time step in which the process is active, the corresponding predicate is set to true. Therefore, the angle assigned to a link affected by gravity is the increase of the constant value `speed-g` at each discrete time steps `t`. While this can be interpreted as an abuse of PDDL+ language features, it is supported by some state-of-the-art planning engines and provides a good ground for describing and discussing the model with robotics experts.

Furthermore, in a preliminary set of experiments, we observed that some aspects of the modelling of processes and events of the presented PDDL+ models were not accepted by

some of the planning engines at the state of the art. With regards to events, an example of an unaccepted event is provided in Figure 6-2. The `back-to-zero` event is used to reset an angle to 0 degrees as soon as the value of 360 degrees is reached. While it is easy to see that the effect of the event is making the precondition false, preventing the event to be re-applied immediately, some planning engines do not accept this formulation. Instead, they require that a Boolean precondition is falsified by the list of effects. We, therefore, modified the formulations with an additional predicate: it is initially set to true to allow events to be triggered. As soon as an event is triggered, its effects falsify the predicate, that is then reset to true by a subsequent `reset` event.

For the sake of completeness, and to exploit the opportunity to investigate how planning engines performance are affected by different models, we consider in our experimental analysis both formulations. We will refer to the "Original" formulation as the formulation that is not well-supported by planning engines but maximises human readability. The other formulation will be referred to as "Modified", and it aims at maximising the parsability by planning engines. Both versions of the models and the corresponding problem instances can be found at https://github.com/Flaudia/AMAO.

6.4 Experimental Analysis

This section presents the PDDL+ benchmarks and the planners employed in our analysis, as well as the results of the experiments we conducted. The main aim of this analysis is to test whether our overall PDDL+ solution can solve tasks that model practical, real-world applications, i.e., in terms of robot workspace and the number of links and physical features characterising the articulated object. For each set of formulations, we have generated planning instances by varying the following parameters: (i) number of links of the articulated object: 3, 4, 5, 6, 7, 8, 10, 12; (ii) in MCU: angular speed of 0.1, 0.5, and 1.0 grades per second; and (iii) in MCUA: acceleration of 0.1 and 0.5 grades per second.

In order to guarantee a fair assessment of the performance of planners according to

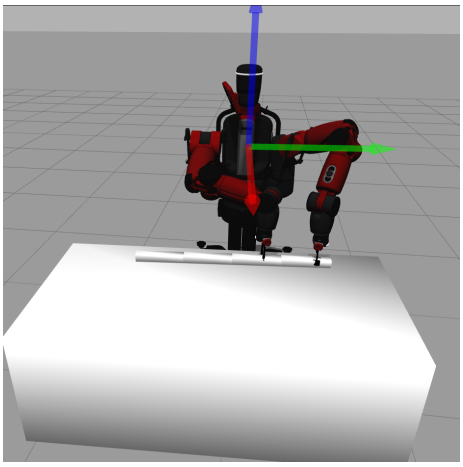


Figure 6-4: A simulated Baxter robot manipulating a four link articulated object. The robot-centred reference frame is highlighted with different colours for the three reference axes.

the level of complexity used for encoding the impact of gravity, for each size of the articulated object 5 manipulation tasks were created by randomly generating initial and final configurations (while ensuring that a plan is possible). Those instances are then encoded in PDDL+ according to the complexity level and to the value of the corresponding MCU or MCUA parameter. Besides the size of the object, no additional parameters have to be set for the NoG formulation. Therefore, for each size of the object, there are 5 tasks, encoded in 30 different problem models. The total number of problem models considered in our experimental analysis is 240.

As a reference robot, we considered a Baxter dual-arm manipulator, which is widely used for research purposes and for performing manipulation tasks. This type of robot is directly supported by the presented PDDL+ formulation and has been used –in simulation– to validate the generated plans. Moreover, it takes care also of the motion planning part. An example is shown in Figure 6-4.

In our analysis we have employed the following state-of-the-art PDDL+ solvers:

- UPMurphi [23] is possibly the most popular domain-independent PDDL+ planner,

Table 6.1: Results achieved by DiNo on the considered benchmarks. Sizes greater than 7 are omitted, as the planner did not solve any benchmark of these sizes. For each dimension of the articulated object, results are presented in terms of average runtime (percentage of solved instances). The average is calculated by considering solved instances only.

		Size – Number of links of the articulated object				
		3	4	5	6	7
NoG		0.5 (40)	88.1 (40)	8.9 (20)	22.0 (40)	– (0)
MCU	0.1	0.6 (40)	130.1 (40)	13.4 (20)	27.2 (40)	– (0)
	0.5	0.6 (40)	130.2 (40)	13.5 (20)	26.8 (40)	– (0)
	1.0	0.6 (40)	125.5 (40)	13.4 (20)	26.7 (40)	– (0)
MCUA	0.1	0.6 (40)	0.9 (20)	15.4 (20)	36.2 (40)	– (0)
	0.5	0.6 (40)	1.1 (20)	15.4 (20)	36.0 (40)	– (0)

and is based on a model-checker adapted to deal with PDDL+;

- DiNo [86], which adds heuristics to the UPMurphi approach;
- ENHSP [89, 92], a numeric planner with heuristics extended to process PDDL+ problems.

Those planners have been selected due to their widespread use in literature and in applications of PDDL+ planning.

Experiments have been run on a machine equipped with i7-6900K 3.20 Ghz CPU, 32 GB of RAM, running Ubuntu 16.04.3.LTS OS. 8 GB of memory were made available for each planner run, and a 5 CPU-time minutes cut-off time limit was enforced. All such planners do not utilize multiple cores. The experiments on the Original formulation –i.e. the formulation presented in the corresponding section of this paper– failed on two of the selected planners, i.e., UPMurphi and DiNo. A detailed analysis of logs highlighted that some aspects of the modelling of processes and events of the presented PDDL+ models were not accepted by the mentioned planning engines. With regards to events, an example of an unaccepted event is provided in Figure 6-2. The `back-to-zero` event is used to reset an angle to 0 degrees as soon as the value of 360 degrees is reached. As we state in

Table 6.2: Results achieved by ENHSP on the considered benchmarks. For each dimension of the articulated object, results are presented in terms of average runtime (percentage of solved instances). The average is calculated by considering solved instances only.

		Size – Number of links of the articulated object							
		3	4	5	6	7	8	10	12
NoG		0.4 (100)	0.6 (100)	0.7 (80)	7.3 (80)	15.5 (40)	3.8 (20)	108.4 (60)	4.5 (20)
MCU	0.1	0.5 (100)	1.9 (100)	1.3 (80)	4.4 (60)	63.9 (40)	36.0 (20)	– (0)	198.5 (20)
	0.5	0.5 (100)	1.9 (100)	1.3 (80)	14.9 (80)	14.5 (40)	60.7 (20)	– (0)	– (0)
	1.0	0.5 (100)	1.7 (100)	40.0 (80)	3.2 (80)	15.4 (40)	55.3 (20)	– (0)	– (0)
MCUA	0.1	0.5 (100)	1.2 (100)	1.3 (80)	4.4 (60)	19.3 (20)	42.5 (20)	50.3 (20)	– (0)
	0.5	0.5 (100)	1.2 (100)	1.3 (80)	4.8 (60)	19.4 (20)	50.0 (20)	55.3 (20)	– (0)

Section 6.3.2, it is easy to see that the effect of the event is making the precondition false, preventing the event to be re-applied immediately, DiNo and UPMurphi do not accept this formulation. For this reason, we added adopted the precautions described in the above-mentioned Section

In terms of processes, Figure 6-3 shows an example of a process formulation that is not accepted by DiNo and UPMurphi.

In such process, the Boolean predicate `(increasing_angle-gravity ?l1)` is listed as an effect. This issue was quite expected, and it has been discussed also in the Formulation section that having predicates as effects of processes can be controversial. However, interestingly, ENHSP does support this modelling approach and provides valid plans nevertheless. For the sake of providing a model that can be parsed by all the considered planners, we removed the Boolean predicate from the effects of processes: such predicates were used for making it easier for a human expert to model events. In the modified PDDL+ models, events have a longer list of preconditions.

To be as inclusive as possible in terms of planning engines, in the rest of this section we will discuss results obtained with the modified version of the PDDL+ models.

Tables 6.1-6.2-6.3 present the results of DiNo, ENHSP, and UPMurphi, respectively. All tables are organised as follows. The columns report the various number of links, while

Table 6.3: Results achieved by UPMurphi on the considered benchmarks. Sizes greater than 5 are omitted, as the planner did not solve any benchmark as well. For each dimension of the articulated object, results are presented in terms of average runtime (percentage of solved instances). Average is calculated by considering solved instances only.

		Size – Number of links		
		3	4	5
NoG		29.2 (40)	170.2 (20)	– (0)
MCU	0.1	33.1 (40)	180.3 (20)	– (0)
	0.5	32.5 (40)	178.9 (20)	– (0)
	1.0	32.1 (40)	175.6 (20)	– (0)
MCUA	0.1	2.4 (20)	– (0)	– (0)
	0.5	45.6 (40)	214.8 (20)	– (0)

in the rows there are the different formulations, with their variants. For each introduced gravity encoding and the number of links, it is reported the average runtime for solved instances, while in parenthesis it is reported the percentage of solved instances. In general, it is easy to derive from the results presented in the tables that, of course, the difficulty in solving the instances increases with the number of links. With regards to the level of complexity of the gravity, the NoG model –which completely ignores gravity– seems to be the easiest (relatively) to solve, followed by MCUA, while MCU seems to be the hardest. Intuitively, the fact that MCUA is easier than MCU can be due to the fact that in the MCUA model the effect of gravity slowly builds up, while in the MCU formulation the impact of gravity starts immediately at full speed. In other words, gravity has a much more significant impact on the search process in the MCU model, rather than in the MCUA.

Considering the MCU and MCUA formulations separately, the performance of the planning engines are not significantly affected by the employed parameters for MCUA, while for MCU the situation looks different: DiNo and UPMurphi do not look to be affected by the employed parameters, while the performance of ENHSP can significantly differ, also in the percentage of solved instances (see, e.g., analysis for 6 links). Our analysis suggests that this is due to the fact that DiNo and UPMurphi tend to solve only the easiest

instances, for each considered size of the object, that requires short and quick to execute plans to be solved. In that, the impact of using the MCU or MCUA model is limited. Instead, as ENHSP can solve also some more complicated instances, taking into account gravity becomes pivotal. Moreover, we can see that ENHSP can solve instances up to 12 links, while DiNo and UPMurphi stop at 6 and 2, respectively. ENHSP is also the only solver able to solve all instances up to 4 links, and the majority of the instances up to 6 links.

About the plans returned by solvers, we noticed no significant difference in terms of both quality and structure for the three approaches. It may be the case that DiNo is able to take better into account the upstream propagation of the effects of the manipulation on angles. While ENHSP tends to provide plans where the robot operates directly on the links of the joint that needs to be modified for reaching the goal position, DiNo seems to prefer the robot to work on different links, and to exploit the use of propagation processes to reach the goal position. However, given the small number of instances solved by DiNo, it is hard to assess whether this behaviour emerged by chance, or it is due to the characteristics of the planning approach exploited by the engine.

Validation. The validation performed by simulating the execution of generated plans on a Baxter robot suggests that the MCU representation provides a reasonably accurate way to encode the gravity effect, as the generated plans can be executed in the simulation environment without further adjustments.

6.4.1 Comparison of PDDL+ Models

Out of the considered planners, only ENHSP does support the Original formulation encodings and provides valid plans. In this section, we, therefore, exploit this planner to compare the impact of the two formulations on its performance. For the sake of providing a model that can be parsed by all the considered planners, we removed the Boolean predicate from

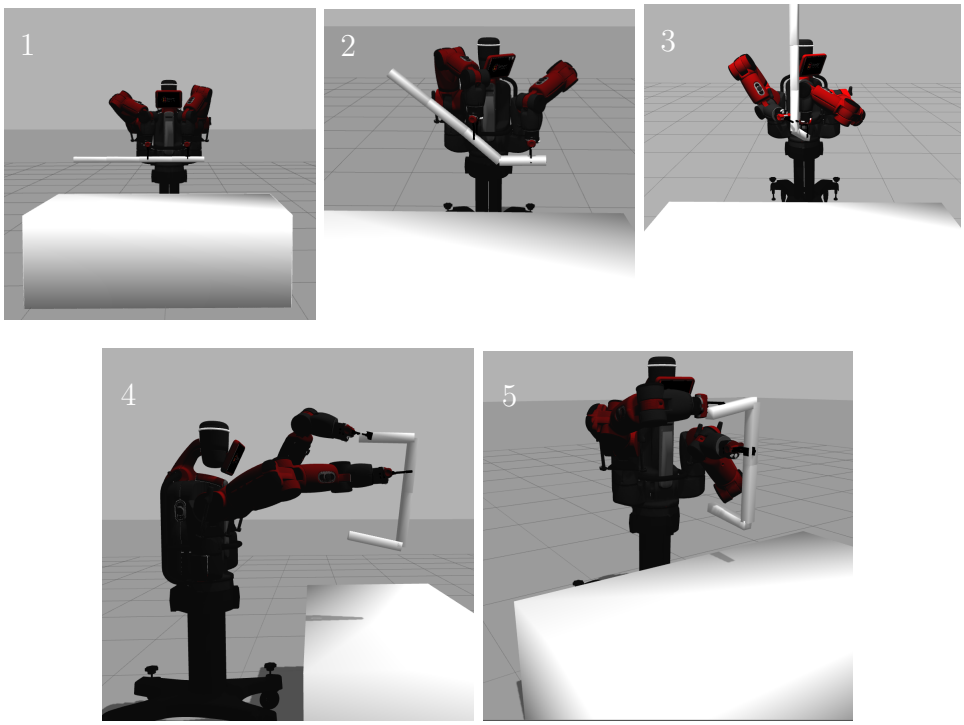


Figure 6-5: The planning and execution process on the extended scenario: the robot actions and (intermediate) states induced by the computed plan.

the effects of processes: such predicates were used for making it easier for a human expert to model events.

Figure 6-6 shows how the performance of ENHSP are affected by the two sets of PDDL+ models, i.e., the Original and Modified models introduced in Section 6.3.

In Figure 6-6, performance is compared in terms of Penalised Average Runtime 10 (PAR10). PAR10 is the average runtime where unsolved instances count as 10*cutoff time. PAR10 is a metric usually exploited in machine learning and algorithm configuration techniques, as it allows to consider coverage and runtime at the same time.

In our analysis, PAR10 is calculated considering all instances of the same size, in terms of the number of links, regardless of the way in which gravity is modelled. In other words, each point of the graph corresponds to the PAR10 score obtained by the planner on the

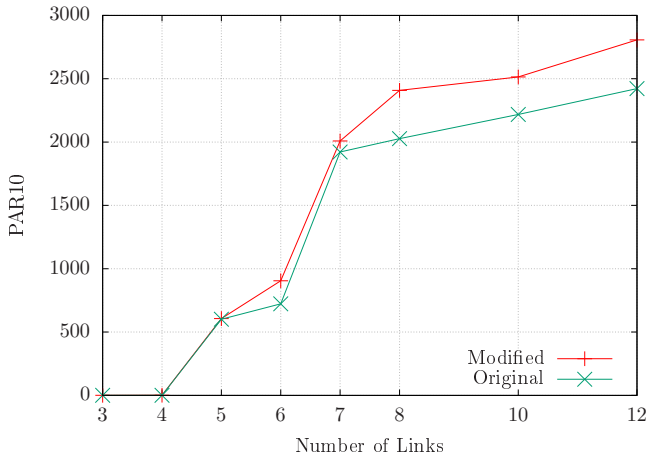


Figure 6-6: Comparison of ENHSP performance, in terms of PAR10, when run on the Original formulation and on the Modified models.

30 planning instances generated for the considered size of the object. It can be noted that when considering objects with more than 5 links, the use of the Original formulation allows improving the performance of ENHSP. According to the Wilcoxon signed-rank test [99], performed by considering all the instances, the performance improvement obtained by using the Original models is substantial and statistically significant ($p < 0.05$). In terms of quality of the generated plans, there is still no significant difference between plans generated using the Original or the Modified models.

While this may be due to the way in which the specific planner has been designed, and it is not easy to generalise, it is still interesting to note that the most human-friendly encoding is the one that allows to deliver the best performance.

Chapter 7

Related and Future works

7.1 Related works

Manipulation of Flexible/Articulated Objects The manipulation of articulated objects plays an important role in real-world robot tasks, both in home and industrial environments [55, 64]. 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. In the literature, the problem of determining the two-dimensional (2D) configuration of articulated or flexible objects has received much attention in the past few years. In [13**?**] a similar framework based on automated reasoning methodologies has been presented. Such framework employs PDDL language and automated planning engines for the planning module, and Description Logic (DL) solvers in the configuration module, where data are explicitly stored in an ontology, while we use a uniform language and approach (ASP-based) for the whole framework. Moreover, differently from most of our approaches, encodings and solvers employed in [13**?**] are not currently able to return shortest plans, which is otherwise important, given that in this context executing the actions can be expensive. In [61], instead, a custom-designed multi-robot platform is presented, focused on HRI in indoor service robot for understanding natural language

requests. Planning is specified using the action language BC [65]. In [97] they propose a planning method for knotting/unknotting of deformable linear objects. They propose a topological description of the state of a linear object with four transitions operations. They demonstrate that it is theoretically possible to knot a linear object placed on a table with a one-handed robot with three translational DOF and one rotational DOF. Furthermore, the approached described in [97] plans using hard encoded states of the linear object. This mean that it is no possible to get to configurations of the object that are not reachable using the encoded states. Our architecture give up on the flexibility of the object in order to implement a more general approach that does not relay predetermined states. in [78] they propose a learning-based approach to associate the behaviour of a deformable object with a robot's actions, using self-supervision from large amounts of data gathered autonomously by the robot. Their method uses human-provided demonstrations as higher level guidance: the demonstrations tell the robot what to do, while the learned model tells it how to do it. Differently from our architecture, there is no planning involved since it is a human operator that gives instructions to the actions to perform and in which sequence. In [94] it is present a method for adapting a demonstrated trajectory from the geometry at training time to the geometry at test time. Trajectory transfer is based on non rigid registration, which computes a smooth transformation from the training scene onto the testing scene.In this work the use a multi-step task by repeatedly looking up the nearest demonstration and then applying trajectory transfer. [9] describes an adaptable system which is able to perform manipulation operations (such as Peg-in-Hole or Laying-Down actions) with flexible objects. They integrate visual tracking and shape reconstruction with a physical modeling of the materials and their deformations as well as action learning techniques. Furthermore, The strategy described in [100] involves manipulating the object at highspeed. By moving the robot at high-speed, they assume that the dynamic behaviour of the flexible rope can be obtained by performing algebraic calculations of the highspeed robot motion. Based on this assumption, a dynamic deformation model of the flexible rope it is derived. [7] present

a method to manipulate deformable objects that does not require modeling and simulating deformation. This method is based on the concept of diminishing rigidity, which we use to quickly compute an approximation to the Jacobian of the deformable object. This Jacobian is used to drive the points within the deformable object towards a set of targets. In describe a technique to models the bending of a sheet of paper abd the paper crease lines which allows the monitoring of the deformations.Moreover, they enabled an anthropomorphic robot to fold paper using a set of tactile- and vision-based closed loop controllers. However, in all these cases, manipulation actions are directly grounded on perceptual cues, such as the peculiar geometry of the object to deal with, assumed to be easy to identify in a robust way, or based on a priori known or learned information about the object to manipulate, e.g., its stiffness or other physical features [31, 37]. As a result, every time either the element that has to be manipulated or the manipulator changes, a new reasoner has to be developed from scratch. In [59] they present a feature representation based on a histogram of oriented wrinkles, to describe the shape variation of a highly deformable object like clothing. A precomputed visual feedback dictionary using an offline training phase that stores a mapping between these visual features and the velocity of the end-effector.

Manipulation in Motion Planning It is possible to find examples in which robots exhibit the capability of manipulating and operating on mobile parts of the environment, such as handles of different shapes [21], home furniture [62] or valves in search and rescue settings [80]. In [21] they design a task descriptor which encapsulates important elements of a task.They propose a method that enables a robot to decompose a demonstrated task into sequential manipulation primitives and construct a task descriptor and then they show how to transfer a task descriptor learned from one object to similar objects. [62] they èresent an automated assembly system that directs the actions of a team of heterogeneous robots in the completion of an assembly task. From an initial user-supplied geometric specification, the system applies reasoning about the geometry of individual parts in order to deduce how they fit together. In [80] an integrated valve-turning skill is

presented. It only requires an operator to issue a supervisory command to launch valve identification, motion planning, biped locomotion and valve manipulation. However, the employed manipulation strategies are often crafted specifically for the problem at hand, with the relevant characteristics of the object and robot capabilities being either hard coded or assumed, thus undermining generalisation and scalability. A structured approach to perception, representation and reasoning, as well as execution, seems beneficial: on the one hand, we can decouple perception and representation issues, thus not being tied to specific perception approaches or ad hoc solutions; on the other hand, domain knowledge and reasoning logic can be separated, with the advantages of an increased maintainability, and the possibility to interchange reasoners and models in a modular way.

Reasoning About Actions Actions when executed often change the state of the world. Reasoning about actions helps us to predict if a sequence of actions is indeed going to achieve some goal that we may have; it allows us to plan or come up with a sequence of actions that would achieve a particular goal and maintain particular trajectories; it allows us to explain observations in terms of what actions may have taken place; and it allows us to diagnose faults in a system in terms of finding what actions may have taken place to result in the faults [5]. [72] points out that, in ontologies, semantic constraints abstract from the way some fact about the case at hand may be actually represented in a proper way. These semantic constraints can be computed from stored data, in the process implementation. In [6, 48] ASP can be used for verification of properties with Bounded Model Checking This is true if the actions are expressed in an extension of Linear Temporal Logic, of an action domain modeled in terms of fluents, action laws providing direct effects of actions, and causal laws. In [45] they combine Answer Set Programming with Dynamic Linear Time Temporal Logic to define a temporal logic programming language for reasoning about complex actions and infinite computations. Moreover, in [46] they present a framework that can be used for reasoning on business processes. In this work they show that ASP can be used for verifying process properties in temporal logic. In [47]

it is showed that reasoning about actions performed in ASP can rely on domain knowledge in a low-complexity Description Logic. In [44] they describe an approach to process modelling and semantic analysis that is able to exploit terminological knowledge in relying process activities to semantic constraints, via the definition of effects and preconditions of activities, and domain knowledge that relates such effects to the terms used in semantic constraints

Macro Operators An important line of research in AI planning focuses on increasing efficiency of the planning process by reformulating the domain knowledge, to obtain models that are more amenable for automated reasoners. Significant work has been done in the area of reformulation for improving the performance of domain-independent planners. *Macro-operators* [16, 63, 76, 81] are one of the best known types of reformulation in classical planning; they encapsulate in a single planning operator a sequence of "original" operators. Technically, an instance of a macro is applicable in a state if and only if a corresponding sequence of operators' instances is applicable in that state and the result of the application of the macro's instance is the same as the result of application in the corresponding sequence of operators' instances. Informally speaking, macros can be understood as shortcuts in the search space allowing planning engines to generate plans in fewer steps. In automated planning, macros have been proven to be effective in domains where some actions are likely to be always executed in the same sequence, or in cases where critical sections can be identified, i.e., where there are activities that need to use a limited resource [17]. Notably, the notion of macros can also be exploited by specifically enhanced planning reasoners. This is the case for MacroFF SOL-EP version [10] which is able to exploit offline extracted and ranked macros, and Marvin [19] that generates macros online by combining sequences of actions previously used for escaping plateaus. Such systems can efficiently deal with drawbacks of specific planning engines, in this case the FF planner [56]; however, their adaptability for different planning engines might be low. In this work, we aimed at exploiting macros in ASP following the more traditional solver-independent approach, i.e.,

by modifying the encoding, replacing simple actions with macros.

7.2 Future Works

The ASP architecture described in this thesis can surely be improved and integrated with different systems. It can be integrated with ROSClingo [2], which is a system that combines the ASP solver Clingo (version 4) with the ROS middleware. In particular, it provides a high-level ASP-based interface to control the behaviour of a robot and to process the results of the execution of the actions. In our framework the interaction with ROS is handled by a custom script. Unfortunately it does not receive any maintenance since a couple of years. Morever, the ASP architecture presented in this paper can be integrated with telingo [12], an extension of the ASP system Clingo with temporal operators over finite linear time, which automatically uses the multi-shot interface of Clingo, and therefore it might be useful to further simplify our architecture and improve performance. Furthermore, ASP has been employed in different domains, including robotic, e.g., [2, 33, 34, 38, 93]. These consider logistic and ricochet robots domain, as well as cooperative robots, whose ultimate goal is not the validation and exploitation of the techniques on a real robot, as in our case. For a recent overview, the interested reader is referred to [32]. Focusing on planning encodings, recently the Plasp system [25] has been further extended with both SAT-inspired and genuine encodings. Some of them have helped to reduce the (still existing) gap with automated planning techniques. Our aim in the design of the encoding was to obtain a devoted and working solution for the problem at hand, rather than the fastest possible one. Nonetheless, results in [25] could be employed to further speed-up our Action Planning module.

For what regard Answer Set programming different combination of grounders and solvers could be tested:

- **grouder:** $I - DLV$ + **solver:** $WASP$

- **grouder:** $GRINGO$ + **solver:** $WASP$
- **grouder:** $I - DLV$ + **solver:** $CLASP$

Where I-DLV [66] and WASP [27] are respectively a grounder and a solver developed by University of Calabria and Vienna University of Technology.

Finally, the proprieties of macros in ASP can be further explored. New macros can be implemented joining different basic actions or developing new simple actions to be then included in the already developed macros or in new ones. Furthermore, new mixed encodings can be modeled using new mix of simple and macros actions.

Chapter 8

Extendability of the framework and Conclusions

8.1 Extendability fo the framework

The architecture presented in this paper it is composed by different models as shown in Chap 4. However, the communication between these models is independent of the language used to perform the module task. The only necessary requirement is that the output of the model remains with the same syntax. Indeed, both the consistency checking module and the goal checker module, that we developed using ASP, can be developed using other knowledge representation languages such as OWL, CycL or KL-ONE. The action planning module can be developed using other planning languages such as PDDL. Moreover, our architecture could be tested on other dual-arm robots, such as YuMi - IRB 14000, TIAGo++ for pal robotics, AMIGO from Eindhoven University of Technology, Romeo is a humanoid robot built by Aldebaran and with any other dual-arm robot. However, our architecture could be used to find a solution for the manipulation of articulated objects using single-arm robots. This can be done with, mainly, two approaches; the first one, would not require any change in the high-level planning system. It would be sufficient to add to the set-up

an object, i.e. a hook, used to keep steady one of the link as if it was grasped by one of the robot grippers. This would lead to some changes in the low-level planning module since the action to grasp a certain link will have to be modified in order to put a link inside the "hook". However, the planning strategy we develop to move the articulated object from an initial configuration to a desired one would be valid. Instead, the second approach would require to modify the encoding adding to the knowledge base the hook used to keep steady one of the link. Moreover, at least one simple action would have to be added to the encoding. This action would state that the robot has to grasp a certain link to then put it inside the hook. This would lead to a small increase in the complexity of the problem.

8.2 Future Works

The ASP architecture described in this thesis can surely be improved and integrated with different systems. It can be integrated with ROSClingo [2], which is a system that combines the ASP solver Clingo (version 4) with the ROS middleware. In particular, it provides a high-level ASP-based interface to control the behaviour of a robot and to process the results of the execution of the actions. In our framework, the interaction with ROS is handled by a custom script. Unfortunately, it does not receive any maintenance since a couple of years. Moreover, the ASP architecture presented in this paper can be integrated with telingo [12], an extension of the ASP system Clingo with temporal operators over finite linear time, which automatically uses the multi-shot interface of Clingo, and therefore it might be useful to further simplify our architecture and improve performance. Furthermore, ASP has been employed in different domains, including robotic, e.g., [2, 33, 34, 38, 93]. These consider logistic and ricochet robots domain, as well as cooperative robots, whose ultimate goal is not the validation and exploitation of the techniques on a real robot, as in our case. For a recent overview, the interested reader is referred to [32]. Focusing on planning encodings, recently the Plasp system [25] has been further extended with both SAT-inspired and genuine encodings. Some of them have helped to reduce the (still existing) gap with

automated planning techniques. Our aim is the design of the encoding was to obtain a devoted and working solution for the problem at hand, rather than the fastest possible one. Nonetheless, results in [25] could be employed to further speed-up our Action Planning module.

For what regards Answer Set programming different combination of grounders and solvers could be tested:

- **grouder:** $I-DLV$ + **solver:** $WASP$
- **grouder:** $GRINGO$ + **solver:** $WASP$
- **grouder:** $I-DLV$ + **solver:** $CLASP$

Where I-DLV [66] and WASP [27] are respectively a grounder and a solver developed by University of Calabria and Vienna University of Technology.

Finally, the proprieties of macros in ASP can be further explored. New macros can be implemented joining different basic actions or developing new simple actions to be then included in the already developed macros or in new ones. Furthermore, new mixed encodings can be modeled using new mix of simple and macros actions.

8.3 Conclusion

With this work, we presented an ASP framework for the automated manipulation of articulated objects in a robot 2D workspace. We demonstrated the validity and usefulness of the proposed approach by running real-world experiments with a Baxter, on two scenarios. In the second, extended scenario, we imported in ASP the concept of macros in order to deal with the task more efficiently. Our analysis shows the effectiveness of our proposed ASP-based approach, using Clingo as a solver, and the usefulness of employing macros.

We see several avenues for future work. First, we are interested in validating the framework on different dual-arm robots, possibly manipulating different articulated objects: given the nature of the approach, we expect it to generalise with a reasonably

limited effort. We also plan to integrate our approach with telingo, to simplify the interaction with robots. Then, we plan to develop a mixed encoding composed of simple and macro actions. Finally, our instances could be an interesting benchmark domain for ASP Competitions (see, e.g., [40]).

All materials presented in this thesis, including instances, results and validations, can be found at https://tinyurl.com/ycbp798j .

Bibliography

[1] Agostini, A., Torras, C., and Worgotter, F. (2011). Integrating task planning and interactive learning for robots to work in human environments. In *Proc. of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pages 2386–2391. IJCAI/AAAI. 86, 88

[2] Andres, B., Rajaratnam, D., Sabuncu, O., and Schaub, T. (2015). Integrating ASP into ROS for reasoning in robots. In *Proceedings of the International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 69–82. Springer. 110, 114

[3] Baral, C. (2003a). *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press. 16

[4] Baral, C. (2003b). *Knowledge representation, reasoning and declarative problem solving*. Cambridge university press. 23

[5] Baral, C. (2010). Reasoning about actions and change: from single agent actions to multi-agent actions. In *Twelfth International Conference on the Principles of Knowledge Representation and Reasoning*. 21, 108

[6] Baral, C. and Gelfond, M. (2000). Reasoning agents in dynamic domains. In *Logic-based artificial intelligence*, pages 257–279. Springer. 21, 108

[7] Berenson, D. (2013). Manipulation of deformable objects without modeling and simu-

lating deformation. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4525–4532. IEEE. 19, 106

[8] Bertolucci, R., Capitanelli, A., Dodaro, C., Leone, N., Maratea, M., Mastrogiovanni, F., and Vallati, M. (2019). An asp-based framework for the manipulation of articulated objects using dual-arm robots. In *Proc. of the 15th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2019)*, volume 11481 of *Lecture Notes in Computer Science*, pages 32–44. Springer. 86

[9] Bodenhagen, L., Fugl, A. R., Jordt, A., Willatzen, M., Andersen, K. A., Olsen, M. M., Koch, R., Petersen, H. G., and Krüger, N. (2014). An adaptable robot vision system performing manipulation actions with flexible objects. *IEEE Transactions on Automation Science and Engineering*, 11(3):749–765. 14, 15, 86, 106

[10] Botea, A., Enzenberger, M., Müller, M., and Schaeffer, J. (2005). Macro-FF: improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research (JAIR)*, 24:581–621. 22, 109

[11] Brewka, G., Eiter, T., and Truszczynski, M. (2011). Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103. 16

[12] Cabalar, P., Kaminski, R., Morkisch, P., and Schaub, T. (2019). telingo = ASP + time. In *Proceedings of the International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, volume 11481 of *Lecture Notes in Computer Science*, pages 256–269. Springer. 110, 114

[13] Capitanelli, A., Maratea, M., Mastrogiovanni, F., and Vallati, M. (2017). Automated planning techniques for robot manipulation tasks involving articulated objects. In *Proceedings of the International Conference of the Italian Association for Artificial Intelligence (AI*IA)*, pages 483–497. Springer. 14, 18, 86, 105

[14] 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. 14, 18, 86, 91

[15] Chakravarthy, U. S., Grant, J., and Minker, J. (1988). Foundations of semantic query optimization for deductive databases. In *Foundations of deductive databases and logic programming*, pages 243–273. Elsevier. 23

[16] Chrpa, L. (2010). Generation of macro-operators via investigation of action dependencies in plans. *Knowledge Engineering Review*, 25(3):281–297. 21, 109

[17] Chrpa, L. and Vallati, M. (2019). Improving domain-independent planning via critical section macro-operators. In *Proceedings of the the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI 2019)*, pages 7546–7553. AAAI Press. 16, 22, 75, 109

[18] Chrpa, L., Vallati, M., and McCluskey, T. L. (2015). On the online generation of effective macro-operators. In Yang, Q. and Wooldridge, M. J., editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)*, pages 1544–1550. AAAI Press. 16

[19] Coles, A., Fox, M., and Smith, A. (2007). Online identification of useful macro-actions for planning. In *Proceedings of ICAPS*, pages 97–104. 22, 109

[20] Costantini, S. and Formisano, A. (2009). Modeling preferences and conditional preferences on resource consumption and production in asp. *Journal of Algorithms*, 64(1):3–15. 24

[21] Dang, H. and Allen, P. K. (2010). Robot learning of everyday object manipulations via human demonstration. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1284–1289. IEEE. 20, 107

[22] Dantsin, E., Eiter, T., Gottlob, G., and Voronkov, A. (2001). Complexity and expressive power of logic programming. *ACM Computing Surveys (CSUR)*, 33(3):374–425. 23

[23] Della Penna, G., Magazzeni, D., Mercorio, F., and Intrigila, B. (2009). Upmurphi: A tool for universal planning on PDDL+ problems. In *Proc. of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009)*. AAAI. 97

[24] Di Rosa, E., Giunchiglia, E., and Maratea, M. (2010). Solving satisfiability problems with preferences. *Constraints*, 15(4):485–515. 30, 59, 61

[25] Dimopoulos, Y., Gebser, M., Lühne, P., Romero, J., and Schaub, T. (2017). plasp 3: Towards effective ASP planning. In *Proceedings of the International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, volume 10377 of *Lecture Notes in Computer Science*, pages 286–300. Springer. 61, 110, 114, 115

[26] Dimopoulos, Y., Nebel, B., and Koehler, J. (1997). Encoding planning problems in nonmonotonic logic programs. In *Proceedings of the 4th European Conference on Planning: Recent Advances in AI Planning*, ECP '97, pages 169–181, Berlin, Heidelberg. Springer-Verlag. 33

[27] Dodaro, C., Alviano, M., Faber, W., Leone, N., Ricca, F., and Sirianni, M. (2011). The birth of a wasp: Preliminary report on a new asp solver. *CILC*, 810:99–113. 111, 115

[28] Drescher, C., Gebser, M., Grote, T., Kaufmann, B., König, A., Ostrowski, M., and Schaub, T. (2008). Conflict-driven disjunctive answer set solving. *KR*, 8:422–432. 24

[29] Eiter, T., Faber, W., Leone, N., and Pfeifer, G. (2000). Declarative problem-solving using the dlv system. In *Logic-based artificial intelligence*, pages 79–103. Springer. 23, 26

[30] Eiter, T., Gottlob, G., and Mannila, H. (1997). Disjunctive datalog. *ACM Transactions on Database Systems (TODS)*, 22(3):364–418. 23

[31] Elbrechter, C., Haschke, R., and Ritter, H. (2012). Folding paper with anthropomorphic robot hands using real-time physics-based modeling. In *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, pages 210–215. IEEE. 19, 107

[32] Erdem, E. and Patoglu, V. (2018). Applications of ASP in robotics. *Künstliche Intelligenz*, 32(2-3):143–149. 110, 114

[33] Erdem, E., Patoglu, V., and Saribatur, Z. G. (2015). Integrating hybrid diagnostic reasoning in plan execution monitoring for cognitive factories with multiple robots. In *Proceedings of ICRA*, pages 2007–2013. IEEE. 110, 114

[34] Erdem, E., Patoglu, V., Saribatur, Z. G., Schüller, P., and Uras, T. (2013). Finding optimal plans for multiple teams of robots through a mediator: A logic-based approach. *Theory and Practice of Logic Programming*, 13(4-5):831–846. 110, 114

[35] Fernández, J. A. and Minker, J. (1992). Semantics of disjunctive deductive databases. In *International Conference on Database Theory*, pages 21–50. Springer. 23

[36] Fox, M. and Long, D. (2006). Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research*, 27:235–297. 90

[37] Frank, B., Schmedding, R., Stachniss, C., Teschner, M., and Burgard, W. (2010). Learning the elasticity parameters of deformable objects with a manipulation robot. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1877–1883. IEEE. 19, 107

[38] Gebser, M., Jost, H., Kaminski, R., Obermeier, P., Sabuncu, O., Schaub, T., and Schneider, M. (2013). Ricochet robots: A transverse ASP benchmark. In *Proceedings*

of the International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR), pages 348–360. Springer. 110, 114

[39] Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., and Wanko, P. (2016). Theory solving made easy with clingo 5. In *Proceedings of the Technical Communications of the International Conference on Logic Programming (ICLP)*, pages 2:1–2:15. Schloss Dagstuhl. 59, 72

[40] Gebser, M., Maratea, M., and Ricca, F. (2017). The sixth answer set programming competition. *Journal of Artificial Intelligence Research*, 60:41–95. 116

[41] Gelfond, M. and Lifschitz, V. (1988). The stable model semantics for logic programming. In *Proceedings of the International Conference on Logic Programming (ICLP)*, pages 1070–1080. MIT Press. 16

[42] Gelfond, M. and Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New generation computing*, 9(3-4):365–385. 23

[43] Gerevini, A. E., Saetti, A., and Vallati, M. (2015). Exploiting macro-actions and predicting plan length in planning as satisfiability. *AI Communications*, 28(2):323–344. 16

[44] Giordano, L. and Dupré, D. T. (2019). Asp and ontologies for reasoning on business processes. 21, 109

[45] Giordano, L., Martelli, A., and Dupré, D. T. (2013a). Reasoning about actions with temporal answer sets. *Theory and Practice of Logic Programming*, 13(2):201–225. 21, 108

[46] Giordano, L., Martelli, A., Spiotta, M., and Dupré, D. T. (2013b). Business process verification with constraint temporal answer set programming. *Theory and Practice of Logic Programming*, 13(4-5):641–655. 21, 108

[47] Giordano, L., Martelli, A., Spiotta, M., and Dupré, D. T. (2016). Asp for reasoning about actions with an elˆ bot knowledge base. In *CILC*, pages 214–229. 21, 108

[48] Giunchiglia, E. and Lifschitz, V. (1998). An action language based on causal explanation: Preliminary report. In *AAAI/IAAI*, pages 623–630. 21, 108

[49] Giunchiglia, E. and Maratea, M. (2006). Solving optimization problems with DLL. In Brewka, G., Coradeschi, S., Perini, A., and Traverso, P., editors, *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI 2006)*, volume 141 of *Frontiers in Artificial Intelligence and Applications*, pages 377–381. IOS Press. 30, 59, 61

[50] Goldberg, E. (2002). E. and y. novikov. berkmin: A fast and robust sat-solver. In *Proc. Design, Automation, and Test in Europe Conference and Exposition (DATE)*, pages 131–149. 29

[51] Grossi, G., Marchi, M., Pontelli, E., and Provetti, A. (2007). Improving the adjsolver algorithm for asp kernel programs. In *Proc. of ASP2007, 4th International Workshop on Answer Set Programming at ICLP07*. 24

[52] Harnad, S. (1990). The symbol grounding problem. *Physica D*, 42:335–346. 40

[53] Henrich, D. and Worn, H. (2000). *Robot manipulation of deformable objects*. Advanced Manufacturing. Springer. 88

[54] Heyer, C. (2010a). Human-robot interaction and future industrial robotics applications. In *Proc. of IEEE International Conference on Intelligent Robots and Systems (IROS 2010)*, pages 4749–4754. IEEE. 13, 85

[55] Heyer, C. (2010b). Human-robot interaction and future industrial robotics applications. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4749–4754. IEEE. 13, 14, 18, 105

[56] Hoffmann, J. and Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research (JAIR)*, 14:253–302. 22, 109

[57] Ielpa, S. M., Iiritano, S., Leone, N., and Ricca, F. (2009). An asp-based system for e-tourism. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 368–381. Springer. 24

[58] Jeroslow, R. G. and Wang, J. (1990). Solving propositional satisfiability problems. *Annals of mathematics and Artificial Intelligence*, 1(1-4):167–187. 29

[59] Jia, B., Hu, Z., Pan, J., and Manocha, D. (2018). Manipulating highly deformable materials using a visual feedback dictionary. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 239–246. IEEE. 19, 107

[60] Kautz, H. A. and Selman, B. (1992). Planning as satisfiability. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 359–363. 59

[61] Khandelwal, P., Zhang, S., Sinapov, J., Leonetti, M., Thomason, J., Yang, F., Gori, I., Svetlik, M., Khante, P., Lifschitz, V., Aggarwal, J. K., Mooney, R. J., and Stone, P. (2017). Bwibots: A platform for bridging the gap between AI and human-robot interaction research. *International Journal of Robotics Research*, 36(5-7):635–659. 18, 105

[62] Knepper, R. A., Layton, T., Romanishin, J., and Rus, D. (2013). Ikeabot: An autonomous multi-robot coordinated furniture assembly system. In *2013 IEEE International conference on robotics and automation*, pages 855–862. IEEE. 20, 107

[63] Korf, R. (1985). Macro-operators: A weak method for learning. *Artificial Intelligence*, 26(1):35–77. 21, 109

[64] Krüger, J., Lien, T. K., and Verl, A. (2009). Cooperation of human and machines in assembly lines. *CIRP Annals*, 58(2):628 – 646. 13, 14, 18, 85, 105

[65] Lee, J., Lifschitz, V., and Yang, F. (2013). Action language BC: preliminary report. In Rossi, F., editor, *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, pages 983–989. IJCAI/AAAI. 18, 106

[66] Leone, N., Pfeifer, G., Faber, W., Calimeri, F., Dell'Armi, T., Eiter, T., Gottlob, G., Ianni, G., Ielpa, G., Koch, C., et al. (2002). The dlv system. In *European Workshop on Logics in Artificial Intelligence*, pages 537–540. Springer. 111, 115

[67] Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., and Scarcello, F. (2005). The dlv system for knowledge representation and reasoning. In *In ACM Transaction on Computational Logic. To Appear*. Citeseer. 24

[68] Leone, N., Ricca, F., and Terracina, G. (2009). An asp-based data integration system. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 528–534. Springer. 24

[69] Leone, N., Rullo, P., and Scarcello, F. (1997). Disjunctive stable models: Unfounded sets, fixpoint semantics, and computation. *Information and computation*, 135(2):69–112. 24

[70] Lifschitz, V. (1999). Answer set planning. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 373–374. Springer. 33

[71] Lifschitz, V. (2002). Answer set programming and plan generation. *Artificial Intelligence Journal*, 138(1-2):39–54. 59

[72] Ly, L. T., Rinderle-Ma, S., Göser, K., and Dadam, P. (2012). On enabling integrated process compliance with semantic constraints in process management systems. *Information Systems Frontiers*, 14(2):195–219. 21, 108

[73] Maratea, M., Ricca, F., Faber, W., and Leone, N. (2008). Look-back techniques and heuristics in dlv: Implementation, evaluation, and comparison to qbf solvers. *Journal of Algorithms*, 63(1-3):70–89. 24

[74] Marek, V. W. and Truszczyński, M. (1999). Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm*, pages 375–398. Springer. 23

[75] Marques-Silva, J. (1999). The impact of branching heuristics in propositional satisfiability algorithms. In *Portuguese Conference on Artificial Intelligence*, pages 62–74. Springer. 29

[76] McCluskey, T. L. and Porteous, J. M. (1997). Engineering and compiling planning domain models to promote validity and efficiency. *Artificial Intelligence*, 95(1):1–65. 21, 109

[77] Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., and Malik, S. (2001). Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535. ACM. 29

[78] Nair, A., Chen, D., Agraval, P., Isola, P., Abbeel, P., Malik, J., and Levine, S. (2017a). Combining self-supervised learning and imitation for vision-based rope manipulation. In *Proc. of the 2017 IEEE International Conference on Robotics and Automation (ICRA 2017)*, pages 2146–2153. IEEE. 18, 86, 106

[79] Nair, A., Chen, D., Agrawal, P., Isola, P., Abbeel, P., Malik, J., and Levine, S. (2017b). Combining self-supervised learning and imitation for vision-based rope manipulation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2146–2153. IEEE. 14

[80] Newman, W., Chong, Z.-H., Du, C., Hung, R. T., Lee, K.-H., Ma, L., Ng, T. W., Swetenham, C. E., Tjoeng, K. K., and Wang, W. (2014). Autonomous valve turning with

an atlas humanoid robot. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 493–499. IEEE. 20, 107

[81] Newton, M. A. H., Levine, J., Fox, M., and Long, D. (2007). Learning macro-actions for arbitrary planners and domains. In *the International Conference on Automated Planning and Scheduling, ICAPS*, pages 256–263. 21, 109

[82] Niemelä, I. (1999). Logic programs with stable model semantics as a constraint programming paradigm. *AMAI*, 25(3-4):241–273. 16

[83] Niemelä, I. (1999). Logic programs with stable model semantics as a constraint programming paradigm. *Annals of mathematics and Artificial Intelligence*, 25(3-4):241–273. 33

[84] Pellier, D. and Fiorino, H. (2018). Pddl4j: a planning domain description library for java. *Journal of Experimental & Theoretical Artificial Intelligence*, 30(1):143–176. 87, 88

[85] Perri, S., Scarcello, F., Catalano, G., and Leone, N. (2007). Enhancing dlv instantiator by backjumping techniques. *Annals of Mathematics and Artificial Intelligence*, 51(2-4):195. 24

[86] Piotrowski, W. M., Fox, M., Long, D., Magazzeni, D., and Mercorio, F. (2016). Heuristic planning for PDDL+ domains. In *Proc. of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, pages 3213–3219. IJCAI/AAAI Press. 98

[87] Przymusinski, T. C. (1989). On the declarative and procedural semantics of logic programs. *Journal of Automated Reasoning*, 5(2):167–205. 28

[88] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan. 45

[89] Ramírez, M., Papasimeon, M., Lipovetzky, N., Benke, L., Miller, T., Pearce, A. R., Scala, E., and Zamani, M. (2018). Integrated hybrid planning and programmed control for real time UAV maneuvering. In *Proc. of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2018)*, pages 1318–1326. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM. 98

[90] Ricca, F., Gallucci, L., Schindlauer, R., Dell'Armi, T., Grasso, G., and Leone, N. (2008). Ontodlv: an asp-based system for enterprise ontologies. *Journal of Logic and Computation*, 19(4):643–670. 24

[91] Saadat, M. and Nan, P. (2002). Industrial applications of automatic manipulation of flexible materials. *Industrial Robot: an International Journal*, 29(5):434–442. 88

[92] Scala, E., Haslum, P., Thiébaux, S., and Ramírez, M. (2016). Interval-based relaxation for general numeric planning. In *Proc. of the 22nd European Conference on Artificial Intelligence (ECAI 2016)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 655–663. IOS Press. 98

[93] Schäpers, B., Niemueller, T., Lakemeyer, G., Gebser, M., and Schaub, T. (2018). ASP-Based Time-Bounded Planning for Logistics Robots. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 509–517. AAAI Press. 110, 114

[94] Schulman, J., Ho, J., Lee, C., and Abbeel, P. (2016). Learning from demonstrations through the use of non-rigid registration. In *Robotics Research*, pages 339–354. Springer. 19, 86, 106

[95] Schulman, J., Lee, A., Ho, J., and Abbeel, P. (2013). Tracking deformable objects with point clouds. In *2013 IEEE International Conference on Robotics and Automation*, pages 1130–1137. IEEE. 14, 15

[96] Subrahmanian, V. S. and Zaniolo, C. (1995). Relating stable models and ai planning domains. In *In Proc. ICLP-95*, pages 233–247. MIT Press. 33

[97] Wakamatsu, H., Arai, E., and Hirai, S. (2006a). Knotting and unknotting manipulation of deformable linear objects. *International Journal of Robotic Research*, 25(4):371–395. 18, 86, 106

[98] Wakamatsu, H., Arai, E., and Hirai, S. (2006b). Knotting/unknotting manipulation of deformable linear objects. *International Journal of Robotic Research*, 25(4):371–395. 14, 15

[99] Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83. 103

[100] Yamakawa, Y., Namiki, A., and Ishikawa, M. (2013). Dynamic high-speed knotting of a rope by a manipulator. *IJARS*, 10:1–12. 14, 15, 19, 39, 106