An ASP-based Framework for the Manipulation of Articulated Objects using Dual-arm Robots

Riccardo Bertolucci[1], Alessio Capitanelli[2], Carmine Dodaro[2], Nicola Leone[1], Marco Maratea[2], Fulvio Mastrogiovanni[2], and Mauro Vallati[3]

[1] DeMaCS, University of Calabria, Italy
{bertolucci,leone}@mat.unical.it
[2] DIBRIS, University of Genova, Genova, Italy
{name.surname}@unige.it
[3] University of Huddersfield, UK
m.vallati@hud.ac.uk

Abstract. The manipulation of articulated objects is of primary importance in robotics, and is one of the most complex robotics tasks. Traditionally, this problem has been tackled by developing ad-hoc approaches, that lack of flexibility and portability.
In this paper 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, as well as for generating the sequence of manipulation actions. The framework is validated both in simulation and on the Baxter dual-arm manipulator, showing the applicability of the ASP methodology in this complex application scenario.

1 Introduction

The manipulation of articulated objects plays an important role in real-world robot tasks, both in home and industrial environments [16,20]. Attention has been put 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 2D configuration of articulated or flexible objects has received much attention in the past few years [5,6,23,27], whereas the problem of obtaining a target configuration via manipulation has been explored in motion planning [2,26,28]. 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 hard coded or assumed; thus, in these contexts generalisation property and scalability are somehow limited.

In this paper we present a framework based on Answer Set Programming (ASP) [3,24] 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 [14].

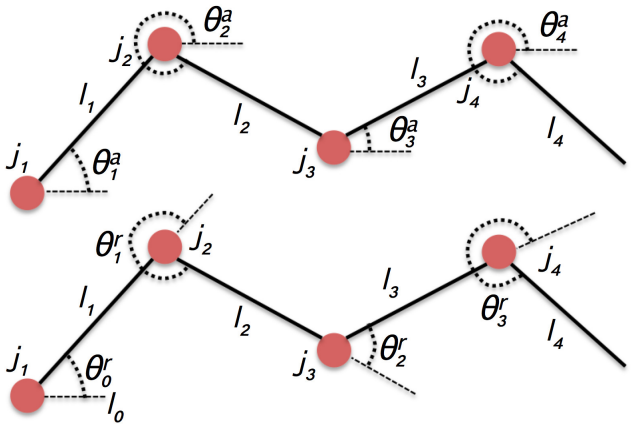

Fig. 1. Two possible representations: absolute (top) and relative (bottom).

In particular, ASP is employed for performing all automated reasoning related tasks, i.e., both for planning actions that the robot has to execute, and for checking the consistency of the configurations of the articulated object as it changes over time. The validity of the framework is finally demonstrated both in simulation and on the Baxter dual-arm manipulator.

2 Problem Statement and the Reference Scenario

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 [28];
- 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;
- A_4 sensing is affected by noise, but the *symbol grounding problem*, i.e., the association between perceptual features and the corresponding symbols [15], is assumed to be solved.

On the basis of assumption A_1, we focus on articulated objects only. We define an articulated object as a 2-ple $\alpha = (\mathcal{L}, \mathcal{J})$, where $\mathcal{L}$ is the ordered set

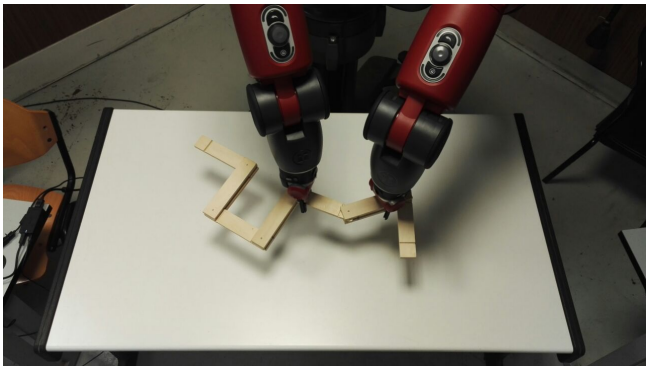

Fig. 2. The experimental scenario.

of its $|L|$ links and $\mathcal{J}$ is the ordered set of its $|J|$ 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. This induces a set of allowed angle orientations A with size $|A|$. If α is represented using absolute angles (Figure 1 on the top), then its configuration is a $|L|$-ple:

$$C_{\alpha,a} = \left(\theta^a_1, \ldots, \theta^a_{|L|}\right). \tag{1}$$

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

$$C_{\alpha,r} = \left(\theta^r_{1,h}, \theta^r_2, \ldots, \theta^r_{|L|}\right). \tag{2}$$

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*.

In order to comply with assumption A_2, we setup a scenario in which a Baxter robot manipulates an articulated object located on a horizontal table in front of it, assumed to be large enough to accommodate the object itself, see Figure 2. Therefore, rotations occur only around an axis perpendicular to the table. We have crafted two wooden articulated objects of different size: the first has three 40 *cm* long links (which are connected by two joints), and the second is made up of seven 20 *cm* long links (connected by six joints). For both objects, links are 3 *cm* thick. The two objects can be easily manipulated by the Baxter's standard grippers, which complies with assumption A_3. The Baxter's *head* is equipped with a camera pointing downward to the table. QR tags are fixed to each object link l, which is aimed at meeting assumption A_4. Each QR code provides an overall link pose, which directly maps to an absolute link orientation θ^a_l. 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.,

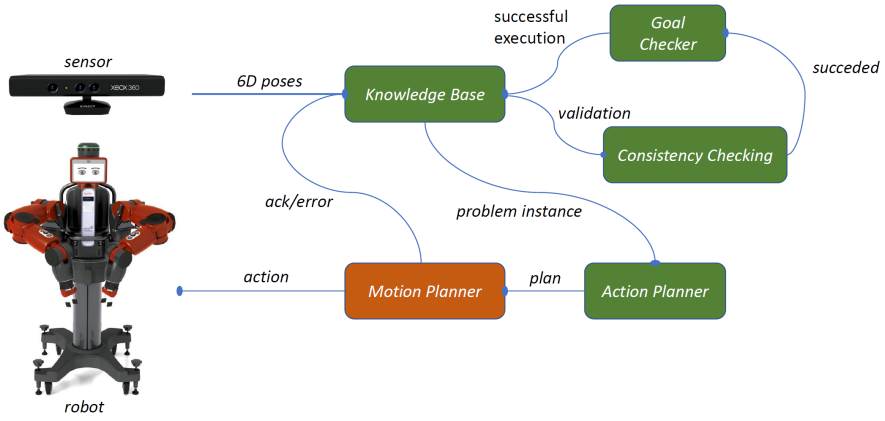

Fig. 3. The robot's architecture: in *green* the ASP-based modules, in *orange* the ROSPlan-based module.

$\theta_1^r = \theta_2^a - \theta_1^a$. After this general scenario introduction, in the next section we detail the architecture.

3 The Robot Architecture

The architecture of the Baxter from Rethink Robotics is shown in Figure 3. 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 ASP current *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 of 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 a number of 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 3) 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.

4 ASP Modules

In this section we describe how ASP is used to implement the modules depicted in Figure 3. In the following, we assume the reader is familiar with ASP and ASP-Core-2 input language specification [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. Note that the constant `timemax` is not included in the example, its usage will be described in Section 4.3.

4.2 Consistency Checking module

The module performs some consistency checking on the knowledge base by using the following ASP encoding:

```
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.
```

Fig. 4. An example of an ASP knowledge base.

```
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_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, rule 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. Rule 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.

It is important to emphasise here that the encoding reported in this section comprises only rules for covering the failure cases that can occur in our setting. The Knowledge Base is created starting from of the camera sensors of the robot, therefore in case of failure it is not possible to repair it by using an ASP encoding. Moreover, according to our experience some of the rules presented in the encoding might be simplified. As an example, c_{1a} and c_{1b} might be merged in

```
c_1 :- isLinked(J1,J2), not joint(J1), not joint(J2).
```

```
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).
```

Fig. 5. Base encoding: it allows for forward manipulations only.

since in our setting if `J1` is not a valid joint also `J2` is not a valid joint. Similar considerations hold for rules c_3 and c_4. However, we decided to use this encoding since it is more general.

4.3 Action Planning Module

ASP is not a planning-specific language, but it can be also used to specify encoding for planning domains [22], 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 [18], where `timemax` is initially set to 1 and then increased by 1 if a plan is not found, which guarantees to return shortest possible plans for a sequential encoding, i.e., when the robot performs only one action for each step (see Section 6 for some details about the other strategies).

Figure 5 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 [13], 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 current 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.

4.4 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., [6]). 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 5.

5 Validation of the Framework

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 in order 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 to run a greater number of planning-execution cycles with minimal human supervision and shorter 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 in order 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

a_1 : `changeAngle(2,1,90,180,1)` a_2 : `changeAngle(1,0,180,90,2)`
a_3 : `changeAngle(3,2,90,180,3)` a_4 : `changeAngle(1,0,270,180,4)`

Fig. 6. 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).

(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 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 4.1, and processed by the state-of-the-art ASP system Clingo [13] together with the encoding in Section 4.3 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.

An example is shown in Figures 4, 6 and 7: Figure 4 reports the ASP representation of the scenario in which the number of joints composing the articulated object, their initial state and the goal to achieve are given, while Figure 6 lists an excerpt of an answer set obtained by Clingo with the encoding in Section 4.3. Each atom of the form `changeAngle` in the answer set represents an action to perform on a joint with the meaning detailed in Section 4.1.

Eventually, Figure 7 illustrates the execution process. In particular, starting from the initial configuration of the articulated object (image7.1), images 7.2, 7.4, 7.5 and 7.7 represent action's execution for a_1, a_2, a_3, and a_4, respectively (see Figure 6), whereas images 7.3, 7.4 and 7.6 represent intermediate configurations. Finally, image 7.8 shows the final state, that corresponds to the required goal configuration of the 5-links articulated object. It is important to note that 7.4 displays both a_3 execution and its resulting intermediate state since it was just a rotation of the whole object.

Other than the sample scenario, we have performed an experimental analysis on the Action Planning module by varying the number of joints (up to 14) and the granularity, and by randomly generating initial and final configurations, for a total of about 400 instances. On the successfully solved instances, Clingo took $1.5s$ average processing time and could solve the problem in around 8 steps on average, with results as low $0.01s/4$ steps and never above $2.2s/9$ steps, which confirms the applicability of ASP reasoning in this context. All plans have been validated with the VAL tool [17]. Albeit the performance deteriorates when both the resolution and the links of the object are increased, they are

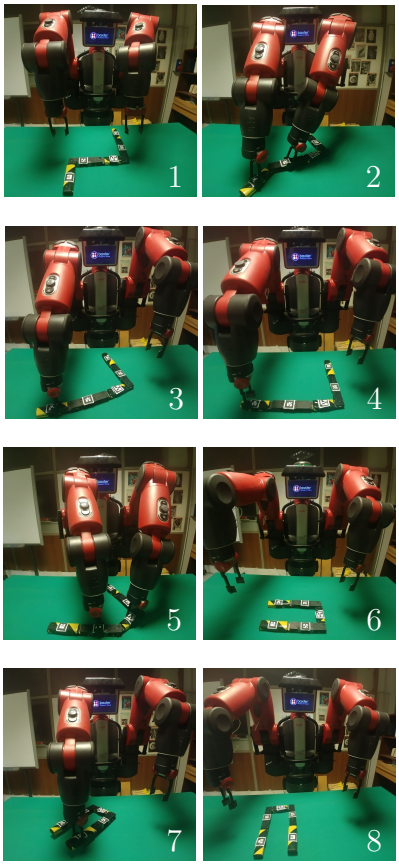

Fig. 7. The planning and execution process on the sample scenario: The robot actions and (intermediate) states induced by the computer plan.

encouraging considering current workspace dimensions and dexterity levels for bi-manual robots, which represent the true bottleneck in this scenario.

Remarkably, the proposed ASP approach is guaranteed to compute the shortest plan, due to the use of an iterative deepening procedure. This is pivotal, as it allows to minimise the actual execution time of the robot, which is the most consuming part.

6 Related work

In Section 4.3 we have shown one of our encoding with one particular manipulation mode and search strategy but, as we already said, we have designed a series of encodings (available at https://tinyurl.com/ycbp798j), including different manipulation modes, i.e., also backward (given our link ordering), and

search strategies. As far as the strategies are concerned, we have designed encoding in which, imposing a reasonable `timemax` (i) by employing a strategy based on the algorithm *optsat* [7], 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 timestep and we let the solver finding a plan, of course possibly loosing optimality (see also [8]).

In [5,6] 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) in the whole framework. Moreover, differently from most of our approaches, encodings and solvers employed in [5,6] are not currently able to return shortest plans, which is otherwise important, given that in this context executing the actions can be expensive. In [19], 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 [21].

The ASP architecture used in this paper can be integrated with ROSoClingo [1], 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.

Moreover, it is worth pointing out that ASP has been already employed in robotic domains, e.g., [1,10,11,12,25]. These consider logistic and ricochet robots domain, as well as on 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 [9].

Focusing on planning encodings, recently the Plasp system [8] 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 [8] could be employed to further speed-up our Action Planning module.

7 Conclusions

In this paper 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 both in simulation and by running real-world experiments with a Baxter, which is widely adopted for research purposes. The experimental results of our validation also indicates that the proposed ASP-based approach, using Clingo as a solver, is capable of generating optimal results,

with regards to the number of actions that the Baxter has to perform, in a very limited amount of time.

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 ROSoClingo, to simplify the interaction with robots. Finally, we plan to extend our approach in order to cope with different types of robots (e.g., those with different number of arms / grippers), and to extend it to model and support the 3D manipulation of articulated objects.

References

1. Andres, B., Rajaratnam, D., Sabuncu, O., Schaub, T.: Integrating ASP into ROS for reasoning in robots. In: Proceedings of the International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR). pp. 69–82. Springer (2015)
2. Bodenhagen, L., Fugl, A.R., Jordt, A., Willatzen, M., Andersen, K.A., Olsen, M.M., Koch, R., Petersen, H.G., Krüger, N.: An adaptable robot vision system performing manipulation actions with flexible objects. IEEE Transactions on Automation Science and Engineering 11(3), 749–765 (2014)
3. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Communications of the ACM 54(12), 92–103 (2011)
4. Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Ricca, F., Schaub, T.: ASP-Core-2 Input Language Format (2013)
5. Capitanelli, A., Maratea, M., Mastrogiovanni, F., Vallati, M.: 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). pp. 483–497. Springer (2017)
6. Capitanelli, A., Maratea, M., Mastrogiovanni, F., Vallati, M.: On the manipulation of articulated objects in human-robot cooperation scenarios. Robotics and Autonomous Systems 109, 139–155 (2018)
7. Di Rosa, E., Giunchiglia, E., Maratea, M.: Solving satisfiability problems with preferences. Constraints 15(4), 485–515 (2010)
8. Dimopoulos, Y., Gebser, M., Lühne, P., Romero, J., Schaub, T.: plasp 3: Towards effective ASP planning. In: Proceedings of the International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR). Lecture Notes in Computer Science, vol. 10377, pp. 286–300. Springer (2017)
9. Erdem, E., Patoglu, V.: Applications of ASP in robotics. Künstliche Intelligenz 32(2-3), 143–149 (2018)
10. Erdem, E., Patoglu, V., Saribatur, Z.G.: Integrating hybrid diagnostic reasoning in plan execution monitoring for cognitive factories with multiple robots. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA). pp. 2007–2013. IEEE (2015)
11. Erdem, E., Patoglu, V., Saribatur, Z.G., Schüller, P., Uras, T.: 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 (2013)
12. Gebser, M., Jost, H., Kaminski, R., Obermeier, P., Sabuncu, O., Schaub, T., Schneider, M.: Ricochet robots: A transverse ASP benchmark. In: Proceedings of

the International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR). pp. 348–360. Springer (2013)
13. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Wanko, P.: Theory solving made easy with clingo 5. In: Proceedings of the Technical Communications of the International Conference on Logic Programming (ICLP). pp. 2:1–2:15. Schloss Dagstuhl (2016)
14. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Proceedings of the International Conference on Logic Programming (ICLP). pp. 1070–1080. MIT Press (1988)
15. Harnad, S.: The symbol grounding problem. Physica D 42, 335–346 (1990)
16. Heyer, C.: Human-robot interaction and future industrial robotics applications. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 4749–4754. IEEE (2010)
17. Howey, R., Long, D., Fox, M.: VAL: automatic plan validation, continuous effects and mixed initiative planning using PDDL. In: Proceedings of the IEEE International Conference on Tools with Artificial Intelligence (ICTAI). pp. 294–301. IEEE Computer Society (2004)
18. Kautz, H.A., Selman, B.: Planning as satisfiability. In: Proceedings of the European Conference on Artificial Intelligence (ECAI). pp. 359–363 (1992)
19. 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., Stone, P.: Bwibots: A platform for bridging the gap between AI and human-robot interaction research. International Journal of Robotics Research 36(5-7), 635–659 (2017)
20. Krüger, J., Lien, T.K., Verl, A.: Cooperation of human and machines in assembly lines. CIRP Annals 58(2), 628 – 646 (2009)
21. Lee, J., Lifschitz, V., Yang, F.: Action language BC: preliminary report. In: Rossi, F. (ed.) Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013). pp. 983–989. IJCAI/AAAI (2013)
22. Lifschitz, V.: Answer set programming and plan generation. Artificial Intelligence Journal 138(1-2), 39–54 (2002)
23. Nair, A., Chen, D., Agrawal, P., Isola, P., Abbeel, P., Malik, J., Levine, S.: Combining self-supervised learning and imitation for vision-based rope manipulation. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA). pp. 2146–2153. IEEE (2017)
24. Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm. Annals of Mathematics and Artificial Intelligence 25(3-4), 241–273 (1999)
25. Schäpers, B., Niemueller, T., Lakemeyer, G., Gebser, M., Schaub, T.: ASP-Based Time-Bounded Planning for Logistics Robots. In: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS). pp. 509–517. AAAI Press (2018)
26. Schulman, J., Ho, J., Lee, C., Abbeel, P.: Learning from demonstrations through the use of non-rigid registration. In: Proceedings of the International Symposium on Robotics Research (ISRR). pp. 339–354. Springer (2013)
27. Wakamatsu, H., Arai, E., Hirai, S.: Knotting/unknotting manipulation of deformable linear objects. International Journal of Robotic Research 25(4), 371–395 (2006)
28. Yamakawa, Y., Namiki, A., Ishikawa, M.: Dynamic high-speed knotting of a rope by a manipulator. International Journal of Advanced Robotic Systems 10, 1–12 (2013)