

UNIVERSITA' DEGLI STUDI DI VERONA

DEPARTMENT OF

Computer Science

GRADUATE SCHOOL OF

Natural Sciences and Engineering

DOCTORAL PROGRAM IN

Computer Science

XXXIII cycle (2017)

Interpretable task planning and learning
for autonomous robotic surgery with logic programming

S.S.D. INF/01

Coordinator: Prof. Paolo Fiorini




Signature _____

Doctoral Student: Dott. Daniele Meli

Signature _____

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License, Italy. To read a copy of the licence, visit the web page:

<http://creativecommons.org/licenses/by-nc-nd/3.0/>

-  **Attribution** — You must give [appropriate credit](#), provide a link to the license, and [indicate if changes were made](#). You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
-  **NonCommercial** — You may not use the material for [commercial purposes](#).
-  **NoDerivatives** — If you [remix, transform, or build upon the material](#), you may not distribute the modified material.

Interpretable task planning and learning for autonomous robotic surgery with logic programming

Daniele Meli
PhD thesis
Verona, 30 September 2021

UNIVERSITY OF VERONA

**Interpretable task planning and
learning for autonomous robotic
surgery with logic programming**

Author:

Daniele MELI

Supervisor:

Prof. Paolo FIORINI

*A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy*

in the

Department of Computer Science

Declaration of Authorship

I, Daniele MELI, declare that this thesis titled, “Interpretable task planning and learning for autonomous robotic surgery with logic programming” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Signed:

Date:

“Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism.”

Dave Barry

Abstract

This thesis addresses the long-term goal of full (supervised) autonomy in surgery, characterized by dynamic environmental (anatomical) conditions, unpredictable workflow of execution and workspace constraints. The scope is to reach autonomy at the level of sub-tasks of a surgical procedure, i.e. repetitive, yet tedious operations (e.g., dexterous manipulation of small objects in a constrained environment, as needle and wire for suturing). This will help reducing time of execution, hospital costs and fatigue of surgeons during the whole procedure, while further improving the recovery time for the patients. A novel framework for autonomous surgical task execution is presented in the first part of this thesis, based on answer set programming (ASP), a logic programming paradigm, for task planning (i.e., coordination of elementary actions and motions). Logic programming allows to directly encode surgical task knowledge, representing *plan reasoning methodology* rather than a set of pre-defined plans. This solution introduces several key advantages, as reliable human-like interpretable plan generation, real-time monitoring of the environment and the workflow for ready adaptation and failure recovery. Moreover, an extended review of logic programming for robotics is presented, motivating the choice of ASP for surgery and providing an useful guide for robotic designers. In the second part of the thesis, a novel framework based on inductive logic programming (ILP) is presented for surgical task knowledge learning and refinement. ILP guarantees fast learning from very few examples, a common drawback of surgery. Also, a novel action identification algorithm is proposed based on automatic environmental feature extraction from videos, dealing for the first time with small and noisy datasets collecting different workflows of executions under environmental variations. This allows to define a systematic methodology for unsupervised ILP. All the results in this thesis are validated on a non-standard version of the benchmark training ring transfer task for surgeons, which mimics some of the challenges of real surgery, e.g. constrained bimanual motion in small space.

Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor...

Contents

Declaration of Authorship	iii
Abstract	vii
Acknowledgements	ix
1 Introduction	1
1.1 Motivation	1
1.2 Task planning for deliberative robots	4
1.2.1 Domain-dependent vs. domain-independent	5
1.2.2 Probabilistic vs. deterministic	6
1.3 Thesis contribution and outline	8
2 The setup for ring transfer task with da Vinci Research Kit	13
2.1 Introduction	13
2.2 The dVRK	13
2.3 The ring transfer task	15
2.4 The vision system	17
2.4.1 Calibration procedure	17
Arm calibration	20
Camera calibration	21
Hand-eye calibration	21
2.4.2 Object recognition	22
2.5 Conclusion	23
3 Logic task planning for deliberative robots	25
3.1 Introduction	25
3.2 Formalization of the problem	25

3.3	Standard logic planning	28
3.3.1	Prolog-based planners	28
3.3.2	Planners based on the answer set semantics	31
3.3.3	Other non-monotonic planners	34
3.4	Temporal logic planning	37
3.5	Probabilistic logic planning	40
3.6	Discussion	44
3.7	Conclusion	47
4	Answer set programming: formalism and solution methods	49
4.1	Syntax and semantics of ASP programs	50
4.1.1	Standard constructs and axioms	50
4.1.2	Optimal answer sets	52
4.1.3	Structure of an ASP program	53
4.2	Computation of answer sets	55
4.2.1	Grounding	55
4.2.2	Solving	57
5	A logic-based framework for surgical task autonomy	61
5.1	Introduction	61
5.2	The framework	61
	Task reasoning module	63
	Situation Awareness (SA) module	68
	Low-level control module	71
5.3	Experiments	75
5.3.1	Task planner validation	75
5.3.2	Framework validation	79
5.4	Discussion	82
5.5	Conclusion	84
6	Inductive learning of surgical task knowledge	85
6.1	Introduction	85
6.2	Related work	86
6.3	The ILP problem under the AS semantics	88

6.3.1	Task definition	88
6.3.2	Complexity and generality	92
6.4	ILP for the ring transfer task	95
6.4.1	Re-formulation of the original ASP encoding	96
6.4.2	Background knowledge and search space	98
	Pre-conditions and executability conditions	98
	Effects of actions	99
6.4.3	Generation of examples	100
	Action pre-conditions and executability conditions	101
	Action effects	102
6.4.4	Results of ILASP	103
	Pre-conditions and executability constraints	104
	Effects of actions	106
	Validation of learned axioms	108
6.5	Discussion	115
6.6	Conclusion	116
7	Towards unsupervised ILP	117
7.1	Introduction	117
7.2	Unsupervised action identification	118
7.2.1	Action segmentation	120
	Changepoint detection	120
	Changepoint filtering	121
7.2.2	Action classification	122
	Features f_1	123
	Features f_2	124
	Features f_3	124
	Classification algorithm	124
7.3	Experiments	125
7.3.1	Test A: Homogeneous dataset from human executions	128
7.3.2	Test B: Homogeneous dataset with noise	130
7.3.3	Test C: Non-homogeneous dataset	132
7.3.4	Computational performance	133
7.3.5	Discussion	135

7.4	From unsupervised action labels to ILP example definition . . .	136
7.5	Conclusion	141
8	Conclusion	143
8.1	Summary of results	143
8.2	Discussion and future works	145
	Bibliography	149

List of Figures

1.1	The functions of a deliberative robot. This thesis focuses on the task planning function of a deliberative robot.	2
1.2	Standard granularity levels of surgical processes, as described in [178].	3
2.1	The patient side of the dVRK, with the PSMs (lateral) and the ECM (center).	14
2.2	The setup for the ring transfer task. The red dashed line defines reachability regions for the two arms.	15
2.3	The reference frames in the transform tree after the calibration procedure. The orange transformations are known, whereas the black transformations are to be estimated.	18
2.4	The calibration setup with the adapter for ECM (right) and the Aruco marker, with the axes of the common reference frame for all components of the robotic system, as computed after calibration (left).	19
5.1	The proposed framework for surgical task autonomy. Functional modules of the framework are highlighted in red, while arrows show the stream of information between modules, the real system and an external human observer.	62
5.2	Comparison between different ASP encodings for the ring transfer task in the sequential execution for clusters of plans with the same length for Encoding 2 (horizontal axis).	76
5.3	Comparison between different ASP encodings for the ring transfer task in the parallel execution for clusters of plans with the same length for Encoding 2 (horizontal axis).	77

5.4	Comparison between planning times for Encodings 1 and 3 in sequential execution.	78
5.5	a) The set of Cartesian trajectories for the <code>move(A, ring, C)</code> gesture, for both PSMs; b) trajectories after roto-dilatation, to start at the origin, $\mathbf{x}_0 = \mathbf{0}$ and end at the vector of ones, $\mathbf{g} = \mathbf{1}$ for batch learning. The learned DMP is shown in red dashed line.	79
5.6	Real robotic scenario with failed transfer and re-planning (F).	81
5.7	Real robotic scenario with occupied colored pegs.	81
5.8	Real robotic scenario with parallel execution and a ring appearing in the scene, triggering re-planning (D).	82
6.1	Comparison between original and learned Encoding 2 for the ring transfer task in the sequential execution, for clusters of plans with same length for original encoding (horizontal axis).	112
6.2	Comparison between original and learned Encoding 2 for the ring transfer task in the parallel execution, for clusters of plans with same length for original encoding (horizontal axis).	112
6.3	Comparison between original and learned Encoding 1 for the ring transfer task in the sequential execution, for clusters of plans with same length for original encoding (horizontal axis).	113
6.4	Comparison between original and learned Encoding 1 for the ring transfer task in the parallel execution, for clusters of plans with same length for original encoding (horizontal axis).	113
6.5	Comparison between original and learned Encoding 3 for the ring transfer task in the sequential execution, for clusters of plans with same length for original encoding (horizontal axis).	114
6.6	Comparison between original and learned Encoding 3 for the ring transfer task in the parallel execution, for clusters of plans with same length for original encoding (horizontal axis).	114

7.1	Kinematic signature (after filtering) and segmentation results for the execution in Figure 5.7. Blue vertical lines represent real changepoints, red solid lines represent changepoints identified by our algorithm, red dashed lines represent changepoints excluded after fluent detection. Frames on the top correspond to two consecutive changepoints sharing the same set of fluents (<code>on(ring, red, peg, grey)</code> , <code>on(ring, blue, peg, red)</code> and the reachability fluents), so the second one is omitted.	127
7.2	Average F1-score over all actions for Test A, with increasing k value. The minimum value (red bar) is the optimal one.	131

List of Tables

3.1	Summary of the main reviewed frameworks for logic planning. Legend for entries: \mathcal{F} = logic formalism (<i>Pl</i> = Prolog); <i>ROS</i> = ROS support; <i>Impl.</i> = software implementation; <i>Integrated</i> specifies whether the framework is stand-alone or is integrated with other modules; <i>Online</i> = online adaptation / plan revision. Legend for applications: <i>NAV</i> = navigation; <i>S&R</i> = search&rescue; <i>DOM</i> = domestic environment; <i>IND</i> = industry, <i>SRV</i> = service; <i>HRI</i> = human-robot interaction, <i>MRC</i> = multi-robot coordination; <i>MAN</i> = manufacturing; <i>SUR</i> = surveillance; <i>LOC</i> = locomotion, <i>MANIP</i> = manipulation.	48
5.1	Geometric quantities for fluent computation.	70
5.2	Planning time from the ASP module in the real robotic scenarios.	81
6.1	Quantitative results of the ILASP task for pre-conditions and executability constraints. The lengths of original and learned axioms are compared, and the learning time is shown as returned from ILASP.	105
6.2	Quantitative results of the ILASP task for <i>initiated</i> axioms for the effects of actions. The lengths of original and learned axioms are compared, and the learning time is shown as returned from ILASP.	108
6.3	Quantitative results of the ILASP task for <i>terminated</i> axioms for the effects of actions. The lengths of original and learned axioms are compared, and the learning time is shown as returned from ILASP.	109
7.1	Matching scores for Test A. All values are percentages.	129

7.2	Action classification results for Test A. All values are percentages.	130
7.3	Matching scores for Test B. All values are percentages.	131
7.4	Action classification results for Test B. All values are percentages.	132
7.5	Matching scores for Test C. All values are percentages.	133
7.6	Action classification results for Test C. All values are percentages.	134

Chapter 1

Introduction

1.1 Motivation

In the last decades, the use of robots in the operating room has provided help to surgeons in performing minimally invasive surgery, improving the precision of gestures and the recovery time for patients [209, 53, 320]. At present, surgeons tele-operate slave manipulators acting on the patient using a master console. One frontier of research in surgical robotics [231, 41] is the development of a cognitive robotic system able to understand the scene and autonomously execute (a part of) an operation, emulating human reasoning and requiring gradually less monitoring from experts [125]. Levels of autonomy for surgical robotics, and their legal and ethical implications, have been analyzed in [336]. In particular, increasing the level of autonomy could further improve the quality of an intervention, in terms of safety and patient recovery time. Moreover, it could optimize the use of the operating room, solving issues such as surgeon fatigue and reducing hospital costs. So far, most of the research in surgery (especially surgical reasoning) has focused on the interpretation of sensors to guide simple operations, e.g. knot-tying [49] and drilling [54].

This thesis faces the problem of reaching level 2 of autonomy as described in [336]. It consists in the automation of repetitive, yet non-trivial and tedious sub-tasks during surgery (e.g., suturing, tissue removal to expose underlying anatomies). Level 2 of autonomy requires continuous supervision of an expert surgeon, who shall approve the operation of the system and may take control in case of failure or emergency. This guarantees reliability and safety of the autonomous system. The artificial intelligence community has widely investigated

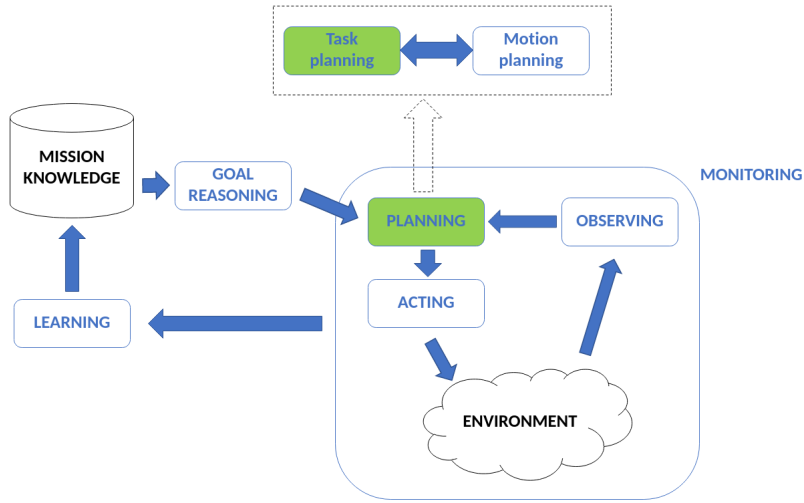


FIGURE 1.1: The functions of a deliberative robot. This thesis focuses on the task planning function of a deliberative robot.

challenges towards autonomous surgical robotics, including real-time situation awareness for monitoring and adaptation of the surgical workflow, interpretable workflow generation for reliability and safety, dexterous and adaptable motion trajectory generation even in limited workspaces. These capabilities are common to the more general class of *deliberative robots*. Deliberation is defined as the *ability to make decisions which are motivated by reasoning on the available resources, i.e. the capabilities of the robot, the actual description of the environment and the given mission* [139]. In the last 20 years, the requirements of a deliberative robotic system have been investigated [109, 139, 269, 213]. Deliberative robots must exhibit six fundamental capabilities, implemented in ad hoc functional modules: planning, acting, monitoring, observing, goal reasoning and learning. Figure 1.1 shows a scheme of the deliberative functions and their integration in a deliberative robot. Planning is defined as the ability to devise a strategy to fulfill a mission. The planning module considers the resources of the robot and information from the sensors, provided by the *observing* module, to generate a feasible plan. Since the planning module combines robot and environmental information and aids in the determination of the robot behaviour in the environment, it is a key requirement for the deliberative robots. Depending on the complexity of the mission, the strategy can be either a motion trajectory

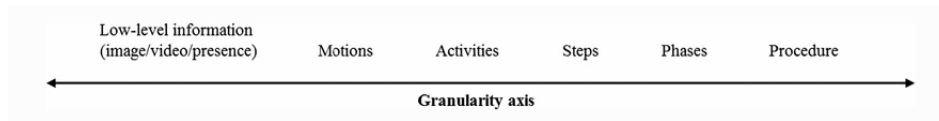


FIGURE 1.2: Standard granularity levels of surgical processes, as described in [178].

to be directly executed by the *acting* module (*motion planning*), or a sequence of elementary operations to be translated into low-level control commands (*task planning*). In some applications, the goal of the mission may be defined as a set of concurrent objectives. This is the case, for instance, of a domestic robot with different objectives in different rooms. In this occurrence, the *goal reasoning* module is in charge of scheduling objectives according to suitable priority or feasibility criteria (e.g., one objective). Moreover, in complex scenarios where interaction with the environment and/or with humans or other robots is involved, a *monitoring* module is needed to supervise the planning-acting-observing loop and guarantee the overall correctness of the execution, triggering re-planning if necessary. Finally, sometimes the specific strategy to complete a mission may not be known at the design stage, hence a (online) *learning* capability is required to discover the best strategy in real time and re-plan. This is particularly useful in surgery, where the environment is unknown or only partially known and the available knowledge about the mission must be obtained/incremented during execution.

This thesis focuses on the problem of task planning for deliberative robots in general, and specifically for surgical robots. The implementation of the planner depends on the description of the task. In surgery, a task can be described at different *granularity levels*, presented in [178] and shown in Figure 1.2. Granularity represents the level of detail in the description. Starting from the very high level, a full surgical *procedure* (e.g., prostatectomy) can be split into a sequence of main *phases* or *steps* (e.g., prostate exposure, prostate removal and final closure). A more fine-grained description could focus on *activities* or *actions*, i.e. single *motion trajectories* (*surgemes*) having semantic information regarding the operating arm of the surgeon, the used instrument and the target anatomy (e.g., cut adipose tissue with left hand using scissors). In this thesis, the surgical task represents a single step of a whole procedure, i.e. a sequence of actions, each

one represented as a motion trajectory (or pair of bimanual trajectories) in the robot's workspace. Starting from a well-defined grammar of actions involved in the step, the main scope of this research is to develop a task planner which provides the (best) sequence of actions reaching the goal of the step. The task planner reasons on a formal description of task knowledge, in terms of causes and consequences of actions, constraints and the goal. In this thesis, first the task knowledge is assumed to be available as a prior. However, this is not realistic for the surgical context, and complex robotic tasks in general, where the workflow of execution depends on multiple environmental variables (e.g., anatomy of the patient), hence cannot be determined in advance. Then, this thesis also shows how to refine and learn task knowledge from few example executions of the surgical step. The performance of the task planner is validated on the da Vinci Research Kit (dVRK), a research-oriented version of the standard surgical da Vinci[®] robot from Intuitive Surgical, equipped with two robotic arms with mounted surgical tools on them. Real execution requires the implementation of motion planning and control. For the scope of this thesis, motion planning is based on learning from human demonstration. The surgeses for each action are learned from few training executions, recording poses (3D positions and orientations) of robotic tool tips. During autonomous execution, the robotic tools then execute trajectories with similar 3D shape as the human ones, and similar orientation along the way, given starting and ending poses for the current task instance. Learning from demonstration allows to replicate human dexterity, which is a fundamental skill of surgeons and guarantees good outcome for the patient.

In the next section, main solutions to the problem of task planning for deliberative robots are briefly presented. This will lead the way to the intended contribution of this thesis, and related implementative choices.

1.2 Task planning for deliberative robots

Inspired from the definitions by [109], a *robotic system* is generally defined as *a set of robots with a common task, which can be achieved through a set of actions, i.e. elementary operations which can be performed by the robots in the environment*. For the scope of this thesis, the robotic systems is composed of the

two arms of the dVRK. The *resources* of the robotic system are the robots with their possible actions, as well as relevant objects in the scenario (e.g., instruments which can be used in the task, or obstacles). The *task planning problem* is then defined by the goal, the resources, the description of the environment and some user-prescribed specifications. The environment is defined through relevant *state variables* related to specific environmental features (e.g., position of objects and spatio-temporal relations between them). Specifications define *pre-conditions* of actions, i.e. the set of conditions which must be verified before executing the actions; *post-conditions or effects* of actions, i.e. conditions which hold after actions are executed; *execution constraints*, i.e. sets of conditions which cannot hold at the same time. Conditions involved in the specifications may be conditions on actions (e.g., an action must occur after another one) or on environmental state variables (e.g., an action can be performed only if the environment is in a specific state). The output of the task planner is a plan, i.e. an *ordered sequence of actions to be executed to accomplish the goal of the task, satisfying the task specifications*.

The task planning problem can be described with a number of formalisms, presented e.g. in [109, 107]. Different taxonomies of task planners have been proposed. In [109], a classification is proposed in domain-dependent and domain-independent task planning. An alternative classification is proposed in [139], distinguishing between deterministic and non-deterministic (or probabilistic) task planners. These two categorizations are presented and briefly discussed next.

1.2.1 Domain-dependent vs. domain-independent

Domain-independent planning [329] is interesting to the robotic community because it is expected to solve the task planning problem in different environments or operational scenarios, starting from a very high-level generic description of the domain. For instance, in a complex robotic task involving transport of objects to pre-defined locations, the domain-independent planner is useful to define a general plan to reach the objective for all objects (e.g., modeling the task as a generic navigation problem). However, grasping and manipulation of single objects before moving them to targets depends on their specific properties and is thus more efficiently managed by a *domain-dependent planner* [74], which

is appropriately designed for the current application. While domain-dependent planning has more practical implementation than domain-independent, it lacks in generality, so it is not suitable on its own for complex heterogeneous robotic tasks. For this reason, in [109] *configurable task planning* is proposed to combine high-level domain-independent task planning with domain-dependent sub-task planning. An explanatory example of this approach can be found in [296], where the full mission is split into a sequence of simpler tasks. While the global strategy is planned through domain-independent hierarchical task planning, a specific domain-dependent planner is in charge of the single tasks. The method is validated successfully on benchmark planning problems, including the famous block-world example.

This categorization of task planners is very coarse, since no detail is provided about the formal characterization of the planner used to describe the features of the environment and the robot.

1.2.2 Probabilistic vs. deterministic

Non-deterministic (or *probabilistic*) task planners are usually based on Markov decision processes and Bayesian models. Stochastic models can capture uncertainty in the environment and sensors, and allow to perform Bayesian inference to output the plan which will most probably satisfy the goal of the task [163, 308, 195]. Probabilistic planners offer natural integration with the acting and observing modules shown in Figure 1.1 because of their data-driven nature, hence they guarantee ready adaptation to changes in the environment and re-planning. However, they do not generally provide enough expressive support to the definition of complex task constraints, e.g. temporal and semantic relations between objects and robots. Moreover, they are typically based on offline/online statistical learning to catch and refine the probabilistic map between environmental variables and robotic actions. This comes at the cost of the *interpretability* of the planning process. Interpretability is defined as *the easy understanding of the robotic actions and their causes, as observed from a human supervisor*. It is an important property of autonomous systems in safety-critical applications involving human-robot interaction as surgery [92], and has been introduced as a fundamental requirement in regulations for high-risk artificial intelligence systems [184].

In fact, the decision-making process of a stochastic planner is strongly driven by what is sensed from the environment and their consequently learned probability mapping to possible actions; however, there is no guarantee that statistical outliers or other sources of error are prevented from affecting the correctness of the final plan.

On the contrary, *deterministic planners* rely on a more formal description of the scenario, in terms of relevant user-defined variables (e.g., environmental and temporal variables, or actions) to reason on. A popular example is temporal planning [95, 96], where variables are used to represent time intervals and generate a plan which satisfies temporal constraints on the sequence of operations towards the goal (e.g., for scheduling robots in industry).

High expressive power to represent the task planning specifications and resources is guaranteed by logic formalism, encoding task knowledge in terms of logic variables and statements. In particular, it is possible to store concepts, e.g. entities in the robotic domain and the environment, with properties and mutual relations, in an *ontology* knowledge base [267].

The expressive capabilities of some logic-based planners are further increased by the *open-world assumption*, i.e. a consequence can be deduced irrespective of whether or not the relative causes are *known to be true*. In other words, assume that a robot shall accomplish the standard manipulation task of moving colored cubes on a shelf. The goal of the task can be represented with a single logic statement as *if a colored cube is found, then it must be placed on the shelf*, assuming that a set of known possible colors is provided (e.g., red, blue and green). Under the closed-world assumption, if only the red and green cubes are found, the robot will only complete the task for them, ignoring the blue one. On the contrary, when the open-world assumption holds, the robot will not consider the task as successfully completed, until the blue cube is either found (e.g., with further exploration) or declared as absent from a human.

Under the open-world assumption, description of the robotic domain and useful resources for planning are typically stored as logic variables and statements in an ontology [267], i.e. a high-level knowledge base. A description logic [13] then provides operators (e.g., first-order-logic operators or standard quantifiers as \exists, \forall [297]), allowing to reason on logic variables and deduce new ones from

known ones. Description logic and ontologies offer high expressivity, allowing to reason on sub-classes, properties and more complex relations between entities of the knowledge base. This is particularly useful for complex scenarios as autonomous surgery [239, 110, 114], human-robot interaction [188, 36] and industrial manipulation [68], involving a large number of variables and hierarchical relations and properties built on them.

However, as evidenced by [321, 241], the improved expressivity and the open-world assumption significantly increase the computational burden, which is a major drawback in robotic applications with real-time demands, as surgery requiring, e.g., prompt re-planning in case an emergency occurs. A more computationally tractable approach, chosen for the scope of this thesis, is *logic programming* [200]. A logic program defines entities of the task domain and logical relations representing specifications. Logic programming does not offer the same expressive power of ontologies and description logics (e.g., support for quantifiers and complex properties of entities). However, this does not represent a significant limitation for the aim of this research, as it will be clarified in the following. Moreover, it will be shown that, while the task description is completely implemented in a single logic program, still several logic programming implementations support the open-world assumption, allowing integration with external ontologies when needed. Finally, logic programming is more suited for robotic task planning, since it can support revision of former deductions and optimal reasoning to choose among different deductions (these aspects will be clarified in the following of this thesis).

1.3 Thesis contribution and outline

This thesis contributes to the ambitious goal of reaching surgical autonomy, and to the more general goal of enhancing deliberation in robotic applications. In particular, the main focus is on reaching level 2 of autonomy. The former Section has specified that logic programming is chosen as the most appropriate solution for this purpose, guaranteeing interpretability (hence more reliability) and enough expressivity.

The first goal of this thesis is to provide an extended review (missing in existing literature) of state-of-the-art logic programming formalisms for task planning in robotic applications, in order to identify the most appropriate solution. In particular, a review focused on robotic applications over the last 20 years is proposed, in order to guarantee that analyzed logic programming formalisms represent the state of the art. Differently from standard categorizations proposed in Section 1.2, a more suitable categorization based on *expressivity* is considered. The expressivity is representative of the complexity of concepts and relations that can be described within each logic formalism, hence it determines the application to surgery. However, complex expressivity usually increases the computational effort, which may represent a limitation for robotic tasks, especially when safety-critical scenarios like surgery are considered, requiring prompt reaction to anomalies and failures. Comparison between different formalisms is presented and discussed on the basis of benefits and drawbacks of differently expressive logics, as well other useful features for the robotic community, including support for online re-planning and open-world reasoning.

Results of the review show that Answer Set Programming (ASP) is the most appropriate candidate for the surgical scenario, since it supports knowledge revision for re-planning, and can encode temporal relations (e.g., arbitrary delays) between entities of the robotic domain, safety and optimization constraints, with real-time computational requirements. Hence, a framework for autonomous surgical task execution is presented in this thesis, based on ASP for task planning and DMPs for motion planning. The framework relies on the continuous monitoring of the surgical robotic scenario through sensors (camera and kinematics). The framework addresses some crucial issues of existing autonomous systems for robotic surgery, including interpretable behavior for better reliability and safety, and online adaptation to anomalies and failures. Interpretability is also guaranteed by a semantic interpretation of raw information from sensors, which allows integration with logic reasoning at task level. The utility and novelty of the framework is remarked by the publications in [114, 115].

One challenge of autonomous robotic surgery is the unpredictability of the anatomical environment and its behavior intra-operatively, depending on the specific patient. The task knowledge encoded in the logic planner only represents

standard description of the task and the environment, but does not cover all possible task instances. Hence, the second part of this thesis focuses on the problem of learning surgical task knowledge through inductive logic programming (ILP) under the ASP syntax, as described in the publications in [218, 219]. ILP allows to learn logic specifications from a very limited number of example executions of a task, solving one fundamental issue of other state-of-the-art data-driven machine learning methods as Markov models, i.e. the lack of extensive surgical datasets for robust model learning. Moreover, task knowledge learned through ILP and added to the ASP program for task planning shows to reduce the complexity of the original specifications, shortening the time for plan computation. The contribution of the proposed ILP approach is relevant also to the generality of the robotic community, since for the first time ILP is applied to the challenging task of learning specifications for a robotic task involving multiple actions and agents (arms of dVRK), and temporal relations expressed in the paradigm of event calculus.

The ILP approach presented in this thesis is supervised, i.e. example executions are manually annotated by experts and labels describing actions and environmental features are provided. This is infeasible for complex surgical procedures, usually lasting very long time. Hence, the final part of this thesis presents an unsupervised approach to ILP. Specifically, a novel unsupervised action identification algorithm is presented as described in [217], which outperforms the performance of state-of-the-art methods even on challenging and small datasets, exploiting semantic environmental features.

The experiments for validating the contributions of this thesis are performed on a benchmark training task for surgeons, the ring transfer from Fundamentals of Laparoscopic Surgery (FLS) [66], consisting in placing rings on same-colored pegs and requiring coordination of multiple actions depending on dynamic environmental conditions. While this task replicates several challenges of real surgery, it excludes others which are covered by other FLS exercises. However, the next chapters will clarify that the generality and the applicability of the proposed methodologies in this thesis are preserved.

The remainder of the thesis is organized as follows. First, Chapter 2 describes the ring transfer task and the robotic setup with dVRK robot for the

autonomous execution. This chapter is useful to clarify the challenges of the considered benchmark task, in order to highlight the differences with other training tasks from FLS and motivate the choice of ring transfer. The benchmark training task is then used to validate the choice of ASP among other logic programming paradigms, in the review presented in Chapter 3. Chapter 4 describes the ASP formalism and solving strategies, in order to introduce the reader to the description of the framework for autonomous surgical task execution in Chapter 5. After presenting and validating the autonomous framework in different experimental conditions, the problem of learning new ASP knowledge from examples is investigated in Chapters 6 (supervised ILP) and 7 (unsupervised ILP). Finally, Chapter 8 summarizes the results of this thesis and proposes future extensions.

Chapter 2

The setup for ring transfer task with da Vinci Research Kit

2.1 Introduction

The benchmark surgical training ring transfer task from Fundamentals of Laparoscopic Surgery (FLS) [66] is considered to validate the contributions of this thesis. The task is executed with the research version of the established surgical da Vinci[®] robot, the da Vinci Research Kit (dVRK)¹.

The scope of this chapter is to describe the ring transfer, detailing the modifications made to the standard version of FLS to highlight the capabilities of the autonomous framework presented in Chapter 5. Also the full setup, including the dVRK robotic platform and the functions of sensors (the vision system), is presented to clarify design choices in the next chapters. A final section discusses a comparison between the ring transfer and other tasks in FLS. This evidences the challenges of the considered task, justifying its choice as a benchmark for this thesis, and is useful to clarify the limitations of the presented framework, and possible directions for future research in Chapter 8.

2.2 The dVRK

As the original da Vinci[®] system, dVRK consists of a patient side and a remote control side. On the remote control side, a console for an expert surgeon is

¹<https://github.com/jhu-dvrk/sawIntuitiveResearchKit>



FIGURE 2.1: The patient side of the dVRK, with the PSMs (lateral) and the ECM (center).

mounted. Through the console, the surgeon can tele-operate the instruments on the patient side.

A picture of the patient side of the dVRK is shown in Figure 2.1. It consists of two cable-driven robotic arms with 6 degrees of freedom (DOFs), denoted as Patient-Side Manipulators (PSMs), and one Endoscopic Camera Manipulator (ECM) with 6 DOFs.

The PSMs may be equipped with different surgical laparoscopic tools, similar to the ones used in standard manual surgery. For the scope of this thesis, two large needle drivers, typically employed for needle manipulation in suturing tasks, are used. Each instrument has a gripper whose opening / closure is commanded separately.

The ECM mounts the surgical stereo camera, and allows for camera motion during the intervention (if requested by the surgeon). In this thesis, the ECM is assumed to be fixed.

The dVRK also offers APIs to control the pose and the joints of the patient-side instruments.

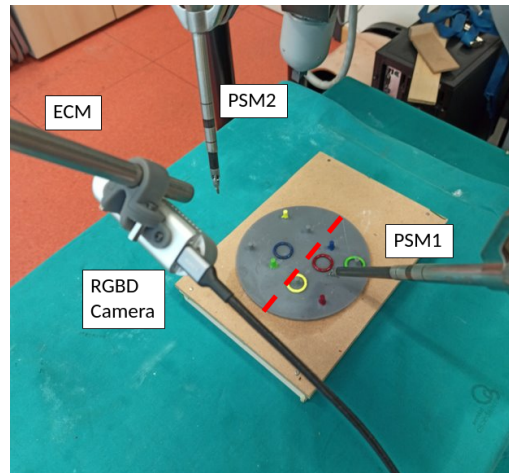


FIGURE 2.2: The setup for the ring transfer task. The red dashed line defines reachability regions for the two arms.

2.3 The ring transfer task

The setup for the bimanual ring transfer task with dVRK is shown in Figure 2.2. The task consists in placing colored rings (4 rings with colors red, green, blue and yellow) on the same-colored pegs. In standard task description from FLS, each ring is initially placed on a grey peg, and it must be transferred between arms of dVRK before placing on the corresponding peg. The standard task definition hence only requires the execution of a pre-defined sequence of actions (moving to a ring, grasping it, transferring to the other arm and placing on the peg) without significant variations in the workflow, except for the failure condition that a ring may fall during the motion of a PSM. In this thesis, a more flexible task description is considered, to increase the variability and highlight the versatility of the framework for autonomous execution.

The first modification consists in defining *reachability* regions for the PSMs, i.e. areas of the peg base which can be reached by the robotic arms. Introducing reachability regions guarantees that the robotic arms do not collide with each other during the execution of the task. In Figure 2.2, they are denoted by a red dashed line. If a ring is in the same reachability region as the corresponding peg (e.g., the red ring in Figure 2.2), a single arm (PSM1) can grasp and place it on the peg; otherwise (e.g., the blue ring) transfer between the arms (PSM2 to PSM1) is needed.

Moreover, colored pegs may be occupied by other rings, so they must be freed before placing another ring on them. This operation may require the temporary placement of a ring on a grey peg, which is an unconventional operation for task completion (validation of the framework in Chapter 5 will consider a similar scenario).

In addition, rings may be placed on a peg or not at the beginning, requiring extraction or not (respectively) before moving them.

Finally, the scenario for the ring transfer task is assumed to be dynamic, i.e. all the environmental conditions may change during the execution. In this way, also the condition that a ring is added to the scene after the beginning of the task can be resolved with re-planning at task level. Furthermore, positions of rings may change over time, triggering re-planning at motion level.

Even if ring transfer is not a proper surgical task, it is widely used as a training exercise for surgeons [240], since it presents several challenges in common with real surgery, and therefore it represents a necessary first step before addressing more realistic surgical tasks. In fact, PSMs must move in a surgical-scale environment, avoiding obstacles (e.g., parts of the anatomy), grasping and positioning small objects with precision and dexterity (like needle grasping in suturing) and adapting to any variation in the scenario. Moreover, object manipulation in a constrained environment is essential for most surgical sub-tasks, from suturing (bimanual coordination in needle passing and knot tying) to debridement (localization of small anatomies, e.g. tumors, to be grasped and removed).

In order to further increase the relevance of the benchmark task for the surgical scenario, criteria of optimality in the plan computation are introduced. In particular, economy of motion (i.e., the distance covered by PSMs during the task) is maximized, specifying that the rings shall be picked and placed on pegs, sorted according to their distance with respect to PSMs. In this way, the ring which is closest to any of the PSMs will be picked up first. Plan optimality is typical of real surgery, where often multiple strategies to accomplish the intervention exist, and the choice of the best solution depends both on the patient's anatomy and the expertise of the surgeon.

It is important and fair to remark that the ring transfer does not obviously

cover all challenges of real surgery, for which novice surgeons are usually evaluated with other exercises from FLS, as suturing, precision cutting and ligation. All other tasks in FLS involve the interaction with soft tissues, hence require adequate low-level control strategies (force control) for proper autonomous execution. Currently, force feedback is not implemented on the dVRK, and the estimation of forces from joint actuations and the dynamical model of the robot is not sufficiently accurate, due to the complexity of the cable-driven mechanics. For this reason, other tasks from FLS have not been considered as benchmark for the methodologies proposed in this thesis. However, next chapters (in particular Chapter 5) will show that the autonomous framework, and mostly the logic-based task-level learning and reasoning approaches, presented in this work can be generalized to more complex and challenging tasks.

2.4 The vision system

In order to execute the task autonomously, the robotic system must be equipped with sensors to deal with the dynamic scenario. Though the endoscopic camera could be directly used, the localization accuracy typically suffers from the small baseline between the stereo cameras. For this reason, in the experiments in this thesis a Realsense D435 RGBD camera² is mounted with a proper adapter on the fixed ECM, as shown in Figure 2.2. The chosen camera has 105 mm minimal depth range of view.

Before the task execution begins, the camera and the PSMs must be calibrated, in order to define a common reference frame (from now on, referred to as *world*) for localization and motion control. Then, an algorithm for real-time object recognition is executed, to identify and localize rings and pegs.

2.4.1 Calibration procedure

The calibration of PSMs and the camera with respect to a common reference frame *world* is realized as described in [276]. This approach guarantees to reach an average localization accuracy of objects in the scene of 1.6 mm, compatible

²<https://www.intelrealsense.com/depth-camera-d435/>

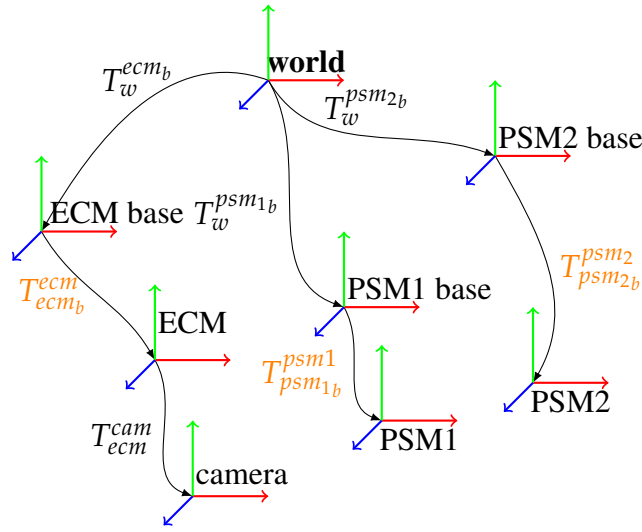


FIGURE 2.3: The reference frames in the transform tree after the calibration procedure. The orange transformations are known, whereas the black transformations are to be estimated.

with measured intrinsic kinematic accuracy of dVRK in standard repeatability tests (multiple positioning of end effectors on known points in the workspace), which is affected by the inaccuracy of the cable-driven actuation [126].

The goal of calibration is to build the transform tree depicted in Figure 2.3. Frames denoted with *base* are part of the kinematic description of the PSMs and the ECM. Base frames represent fixed frames which correspond to the remote center of motion of the robotic arms during the execution. In fact, in surgery, laparoscopic instruments are typically introduced in the body of the patient through pre-defined entry points, constraining the instruments to pivot at them.

The aim of the calibration procedure is twofold. First, the rigid transformations T_{\star}^w between world and the base frame of the arms, assuming $\star \in \{ecm_b, psm1_b, psm2_b\}$, are computed. Second, the transformation T_{ecm}^{cam} between the camera reference frame and the ECM reference frame is obtained.

A custom calibration board is used, shown in Figure 2.4 (left), with an ArUco marker in the center of a circumference of 50 mm radius, with several reference dots. The ECM is equipped with a 3D-printed adapter, shown in Figure 2.4 (right). The adapter has a smaller tip than the ECM to guarantee precise positioning on the dots on the board.

The procedure starts by positioning the calibration board in the workspace of

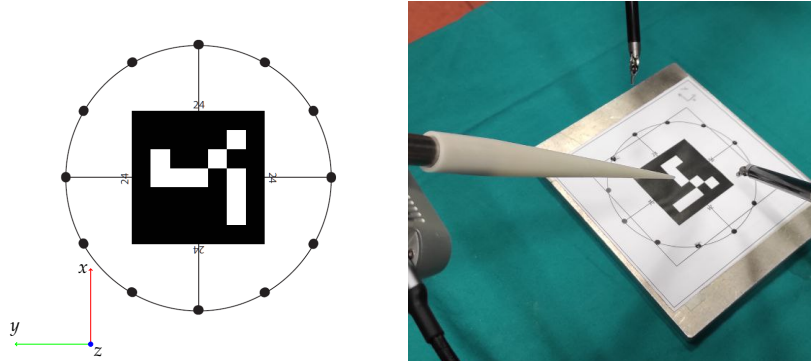


FIGURE 2.4: The calibration setup with the adapter for ECM (right) and the Aruco marker, with the axes of the common reference frame for all components of the robotic system, as computed after calibration (left).

the robot. A set of reference points P is chosen such that each point $p \in P$ is reachable by the three arms and visible from the camera. The points in P must be symmetric with respect to the center of the board to compute the origin of the common reference frame; at least three points are needed to estimate the plane coefficients. The best fitting plane is characterized by the centroid of the point set P , \mathbf{c} , and the normal vector \mathbf{n} . Their optimal estimations are the solution to the optimization problem

$$\{\hat{\mathbf{c}}, \hat{\mathbf{n}}\} = \arg \min_{\mathbf{c}, \|\mathbf{n}\|_2=1} \sum_{i=1}^n ((\mathbf{p}_i - \mathbf{c})^T \mathbf{n})^2 \quad (2.1)$$

As in [100] the centroid is estimated by

$$\hat{\mathbf{c}} = \frac{1}{n} \sum_{i=1}^n \mathbf{p}_i. \quad (2.2)$$

The normal vector \mathbf{n} is obtained by factorizing the distance matrix A with Singular Value Decomposition (SVD)

$$A = USV^T = [\mathbf{p}_1 - \hat{\mathbf{c}}, \dots, \mathbf{p}_n - \hat{\mathbf{c}}] \in \mathbb{R}^{3 \times n} \quad (2.3)$$

and taking the third column of the matrix $U = [\mathbf{u}_1 \ \mathbf{u}_2 \ \mathbf{u}_3]$, $\hat{\mathbf{n}} = \mathbf{u}_3$. To generate a common reference frame for all the tools three algorithmic steps are implemented:

1. Arm calibration
2. Camera calibration
3. Hand-eye calibration

Arm calibration

In order to find the transformation of the arms base frame with respect to the common reference frame, the end effector poses of the arms (PSMs and ECM with adapter) are recorded on each point in the set P . In order to obtain the ECM effective pose, the known rigid transformation between the adapter and the ECM is removed. On this set, the best fitting plane is estimated using 2.1. The set P is then augmented by adding a point above the calibration board acquired by moving the arm's end effector. This last point is used to define the desired plane normal direction

$$\mathbf{n}_d = \frac{\mathbf{p}_{n+1} - \mathbf{c}}{\|\mathbf{p}_{n+1} - \mathbf{c}\|_2}$$

where \mathbf{p}_{n+1} is the last point in the ordered set P , \mathbf{c} is the centroid of P and $\|\cdot\|_2$ is the vector norm. For each arm, the homogeneous transformation T_{\star}^w of the common reference with respect to the arm base frame is defined using the direction versors

$$\mathbf{u} = \text{sign}(\mathbf{n} \cdot \mathbf{n}_d) \mathbf{n}$$

$$\mathbf{l} = \mathbf{u} \times \frac{\mathbf{p}_1 - \mathbf{c}}{\|\mathbf{p}_1 - \mathbf{c}\|_2}$$

$$\mathbf{f} = \mathbf{l} \times \mathbf{u}$$

and the centroid \mathbf{c}

$$T_{\star}^w = \begin{bmatrix} \mathbf{f}_x & \mathbf{l}_x & \mathbf{u}_x & \mathbf{c}_x \\ \mathbf{f}_y & \mathbf{l}_y & \mathbf{u}_y & \mathbf{c}_y \\ \mathbf{f}_z & \mathbf{l}_z & \mathbf{u}_z & \mathbf{c}_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Camera calibration

In order to find the transformation T_{cam}^w for the RGB-D camera, the center of the ArUco marker on the board with respect to the camera frame is first identified. Then, the pose is aligned on the point cloud generated from the depth map acquired by the RGB-D camera. The marker pose and its known radius are used to generate the pose of every dot in the set P in the marker reference frame, as well as the point above the calibration board.

Once the pose set P is obtained, the best fitting plane is computed using 2.1, and the homogeneous transformation T_{cam}^w is found between the common reference frame to the camera base frame by adapting the previous approach used for the arms.

Hand-eye calibration

The hand-eye calibration problem is formulated using the homogeneous transformation matrices:

$$AX = XB$$

where A and B are known homogeneous matrices representing the frames of the base of the robot and the camera, respectively. The unknown transformation X is between the robot coordinate frame and the camera coordinate frame. Given T_{cam}^w , X can be computed as the relative homogeneous transformation between the end effector of the ECM and the RGB-D base frame:

$$T_{ecm}^{cam} = T_w^{cam} (T_w^{ecm})^{-1}.$$

Algorithm 1 Detection Algorithm

```

1: Input: Point cloud  $PC_{in}(t)$  in real time
2: Output: Point clouds of rings,  $PC_r$ , and pegs,  $PC_p$ 
3: Initialize:  $PC_r = []$ ,  $PC_p = []$ 
4: for  $t = 1 : \infty$  do
5:   Subsample  $PC_{in}(t)$  to  $PC_{sub}(t)$ 
6:   for each color do
7:     Color Segmentation of  $PC_{sub}(t)$ 
8:     Euclidean clustering
9:     Update  $PC_r \leftarrow$  RANSAC
10:    if  $t == 1$  then
11:      Update  $PC_p$ 
12:    end if
13:  end for
14: end for

```

2.4.2 Object recognition

Thanks to the calibration procedure, the poses of the robotic arms as read from the built-in encoders of the dVRK can be easily transformed in the world frame, and related to the poses of the point cloud returned by the camera. From the point cloud, relevant objects in the ring transfer scenario must be identified using an appropriate Detection Algorithm 1. The algorithm is based on standard methods from the well-established Point Cloud Library³.

The point cloud is subsampled in order to guarantee real-time performances. The base and the pegs are assumed to be static during the whole execution, and they are identified only at the beginning of the task. The poses of all rings are retrieved at each time step.

The identification of pegs and rings is performed in two steps. First, color segmentation allows to identify same-colored points. Then, Euclidean clustering allows to separate the clouds of ring and peg for each color. Finally, Random Sample Consensus (RANSAC) is used to fit a torus shape on both clusters, and the best fitting cluster is identified as the ring, while the other as the peg.

The output of Algorithm 1 is the point cloud of rings and pegs.

³<https://pointclouds.org/>

2.5 Conclusion

This chapter has described the ring transfer task considered in this thesis, and the setup (dVRK + RGBD camera) used for experiments. The task is a standard training exercise for novice surgeons from FLS, and it represents the most established benchmark for robotic surgery. For this reason, it is chosen to validate the contributions of this thesis. In fact, the ring transfer involves several challenges of real surgery with dVRK, and da Vinci[®] system in general, including dexterous manipulation in small constrained environments, bimanual manipulation and cooperation. These challenges are common, e.g., to needle grasping or knot tying during suturing, or debridement and removal of anatomical parts as tumors. Furthermore, While not all challenges of real surgery are involved in the ring transfer task (e.g., soft tissue interaction, covered by other FLS tasks), the next chapters (especially Chapter 5) will clarify that this does not affect the generality of the methodologies proposed in this thesis.

Chapter 3

Logic task planning for deliberative robots

3.1 Introduction

This chapter reviews the state of the art in logic programming paradigms for robotic task planning. In order to clarify the concepts, examples from the domain of the ring transfer task will be proposed along with the presentation of different paradigms, in order to better motivate the final choice of the most suitable planner.

Before starting the survey, a formalization of (part of) the ring transfer task follows to introduce standard nomenclature and definitions.

3.2 Formalization of the problem

The ring transfer task can be formalized using an action language [83], encoding task entities and specifications as preconditions, effects of actions and constraints, as well as the goal. One of the most popular action languages for robotics is the Planning Domain Definition Language (PDDL) [165].

In PDDL, the ring transfer task can be represented as follows¹:

```
(define (domain ring transfer)
  (:types ring, peg, arm)
```

¹The formal description of the ring transfer task is kept simple and essential for the scope of this chapter. In Chapter 5, a more detailed formalization will be provided.

```

(:functions free ?p - peg;
 at ?a, ?r - arm, ring);
on ?r, ?p - ring, peg)
reachable ?a, ?p - arm, peg)
reachable ?a, ?r - arm, ring)

(:action move
  parameters: (?a - arm; ?r - ring)
  precondition: (reachable ?a, ?r)
  effect: (at ?a, ?r))

(:action move
  parameters: (?a - arm; ?p - peg)
  precondition: (reachable ?a, ?p)
  effect: (on ?r, ?p))

(:action transfer
  precondition: (at ?a, ?r))
(constraints: (and (transfer, reachable ?a ?r,
reachable ?a ?p))
(:goal on ?r, ?p))

```

Domain entities (types) with their properties (functions) and actions (move) are involved in the definition of task specifications. Starting from this high-level representation, a logic programming language encodes the task knowledge using a *sorted signature* \mathcal{D} , with symbols (*terms*) which can be arranged hierarchically to form *atoms*, i.e. predicates of terms representing attributes of the domain. Terms may be variables or constants. A term which is constant is *ground*, and atoms are ground when terms in them are all ground. After grounding, atoms become Boolean variables. Atoms are either *fluents* or *statics*, depending on whether they depend on time or not, respectively. As clarified in the next sections, dependency on time can be made explicit through the use of a temporal variable, depending on the specific logic programming framework. Fluents

typically represent actions or environmental attributes. For instance, in the domain of ring transfer `arm` (with constant value, e.g. `psm1`, `psm2`) `ring` and `peg` (with constant values identifying rings and pegs of specific colors) are statics, while functions and actions shall be defined as fluents. In the following of this chapter, `A`, `R`, `P` will be used as shortcomings for the variable terms `arm`, `ring`, `peg`, respectively. Logic programming languages also introduce specific operators which allow to write the specifications, i.e. preconditions and effects of actions, and executability constraints. For instance, the precondition to `transfer` action can be expressed with the logic implication operator, `at(A, R) → transfer`. Operators define the *expressivity* of a logic language and, hence, of a logic planner, and constrain the specifications which can be defined (e.g., specific languages can support temporal operators, other than logic ones).

The plan is computed using a language-specific solver. The solver grounds atoms according to external information (provided, e.g., by sensors). In this way, the logic program is translated to a set of propositional logic formulas (i.e., all grounded atoms are Booleans). Then, the solver uses logic deduction to return the plan from the specifications, which now represent logic implications between the Booleans of actions and domain entities. For example, in the ring transfer domain the solver could receive the information that the red and green rings are present in the scene, and the red ring is already on the red peg. Hence, the grounding process would return the instances of `ring(red, green)`, `peg(red, green, blue, yellow)`, `on(ring(red), peg(red))`, `reachable(psm1, ring(red, green))`, `reachable(psm1, peg(red, green))`, `reachable(psm2, peg(blue, yellow))` and `free(peg(green))`, with assignments for the generic variables. This will lead to compute a plan as `move(psm1, ring(green))`, `move(psm1, peg(green))`.

In this chapter, a main categorization of logic planners based on their expressivity is proposed, because it allows to highlight their advantages when applied to the description of different robotic domains. Moreover, as the expressivity increases, the computational complexity of solving the logic program rises. Hence, evaluating logic planners according to their expressivity allows to identify a tradeoff for real-time safety-critic applications as surgery. Three main categories of expressivity are considered: planners based on *standard logic*, i.e.

classical Boolean logic with operators as conjunction, disjunction, implication and negation; planners based on *temporal logic*, supporting specific operators to define temporal relations between atoms; and planners based on *probabilistic logic* to account for uncertainty in logic deduction. Other than expressivity, also other important features of logic planners will be analyzed and discussed, including software implementation and support for open-world planning and on-line re-planning.

3.3 Standard logic planning

The terminology *standard logic* refers to logic programming frameworks based on standard Boolean logic, where specifications are expressed using operators as negation, conjunction, disjunction and implication. In order to represent the temporal sequence of actions and fluents, an explicit time variable t must be defined. Hence, with reference to the ring transfer domain, fluents have an additional argument t (e.g., $\text{on}(R, P, t)$). Standard logic planners can be grouped in two main categories, Prolog-based and languages based on answer set programming, which differ in the solving strategy and the computational performance. Also other implementations have been proposed, though they are often chosen only for specific applications. A brief mention to them is provided in the end of this Section.

3.3.1 Prolog-based planners

One of the first examples of standard logic programming languages for planning is *Prolog*, introduced in [318]. Specifically, Prolog is a language for *non-monotonic logic programming*, i.e. the underlying logic reasoning (and knowledge) is non-monotonic [216]. To understand non-monotonicity, consider a generic task knowledge K expressed as a set of entities and specifications, and a Boolean assertion A induced by K , i.e. $K \models A$. The knowledge is said to be monotonic if the addition of a new set S of specifications or entities does not affect the truth of the entailment, i.e. $K \cup S \models A$. Otherwise, the knowledge is said to be non-monotonic. Non-monotonicity of knowledge has allowed to solve the famous qualification problem in artificial intelligence, moving from

the concept of circumscription of human reasoning [215], and it has led to the development of several logic frameworks for decision making of expert systems, e.g. deductive databases [77], abduction theory [5] and stable model semantics [70]. In a robotic task non-monotonicity is needed when the environment is not completely known or dynamic. Considering the ring transfer task, rings may be introduced in the scene (e.g., by a human) during task execution. As a consequence, the solver must continuously update the plan, as new rings are found (hence, new `reachable` fluents are grounded).

The syntax of Prolog extends standard logic formalism with *negation as failure* (NAF). To show the difference between NAF and classical logic negation, consider a generic Boolean variable p . In order to state $\neg p$ in classic monotonic logic, it is necessary to prove that the negation of p holds; on the contrary, to prove $\text{not}(p)$ (NAF syntax in Prolog) it is only required that p does not hold. Hence, NAF introduces non-monotonicity in the task knowledge representation, and it also encodes the closed-world assumption. NAF is used in combination with fluents, since their value can be updated at run time. For instance, in the ring transfer task the goal can be expressed as the condition that the all *and only* reachable rings must be eventually placed on pegs, as $\perp \leftarrow \text{reachable}(A, R, t), \text{reachable}(A, P, t), \text{not}(\text{on}(R, P, t))$. The NAF syntax is used to specify that it is impossible that a reachable ring is not on the corresponding peg (hence, a new plan must be devised until this condition stops holding).

Moreover, starting from Prolog III [52] and up to the latest version Prolog IV [242], Prolog has further extended the standard logic syntax with structures like lists and trees to represent a richer hierarchy between Boolean variables, and is able to manage constraints on real variables which are useful for robotic applications involving continuous environmental and kinematic variables (e.g., positions of objects).

These features have increased the appeal of Prolog in many challenging task planning scenarios for deliberative robots, and prompted the development of several Prolog-based programming languages. Relevant are the many Prolog-based implementations of the Golog action language [190], such as in [141] for multi-robot hierarchical task planning in the context of RoboEarth project [323]; [161], where an useful integration of the Golog dialect Readylog with

the Robot Operating System (ROS) [164] is proposed through C++ bindings (Golog++ [212]), with an application to the Pepper service robot; [111], where INTRGOLOG is presented to implement task interruption and resumption in reaction to anomalous events; [286], where Readylog is used for task planning in domestic scenario with Caesar robot; [91], where TEAMGOLOG is used for multi-robot coordination in a search-and-rescue application under partial observability. Among main implementations of Prolog, also SWI-Prolog [327] shall be mentioned, which implements constructs for optimal query answer and plan generation under the paradigm of preference reasoning [34] and has been used, e.g., in the Tartarus framework for the integration and coordination of multiple robots in cyber-physical systems [290]; [236], where Fuzzy Prolog is used to deal with missing information in real-time robotic soccer; [142], where multi-robot coordination for domestic activities is implemented in Prolog; [335], where an efficient software integration between qualitative Prolog calculus and quantitative C++ processing is implemented for robotic assistance to elderly and disabled people; [263], which proposes SitLog, a Prolog-based planner for service robotic tasks in domestic scenario, in the context of RoboCup@Home international competition; [245, 88] for robotic assembly in industry.

In spite of the success of Prolog in many robotic scenarios, a major drawback comes from the way the task knowledge is represented, and then solved for plan computation. As detailed in [242], Prolog arranges symbols of the sorted signature in a tree structure, following the term-atom hierarchy. For instance, with reference to the ring transfer domain, $\text{free}(P, t)$ is at the root, with P and t variables at the bottom. Operators (Boolean and arithmetic ones for reals) further expand the tree structure. For instance, $\text{free}(P, t)$ is a leaf for $\text{move}(A, P, t)$ since $\text{free}(P, t) \rightarrow \text{move}(A, P, t)$. Given such a tree structure, a solver for Prolog is able to respond to queries from the user implementing a backtracking algorithm. Starting from the grounded leaves of the tree structure (according to the initial grounding of variables provided by the user), the knowledge tree is searched until all atoms corresponding to the input query are grounded. Moreover, last explored branches in the tree are saved in a stack, so as to retrieve the root when a branch returns no solution and to restart the search process. More details on the algorithm can be found in [242] and related

references.

This solving procedure is exponentially complex as the task description becomes more complex, i.e. with more variables and a deeper tree structure, which is the case, e.g., in multi-robot scenarios with heterogeneous environment, or in surgery, where complex anatomies must be described and the procedure is usually not short (more actions, more conditions to be evaluated, hence more demanding tree exploration). Moreover the tree search requires the prior stratification of the program in order to guarantee that solving terminates. Stratification guarantees that a (propositional) logic program can be partitioned in incremental subsets of clauses $P_1 \cup \dots \cup P_n$, such that:

- if a predicate p appears as positive in P_i , then it is defined in $\bigcup_{j \leq i} P_j$;
- if a predicate p appears as negative in P_i , then it is defined in $\bigcup_{j < i} P_j$.

Since Prolog-based programs represent knowledge in a hierarchical tree structure, this subset composition guarantees that the tree search gets never stuck in a loop between predicates defined at different levels of the hierarchy. However, not all logic programs can be stratified [75]. Attempts have been made since the early years of Prolog to reduce the complexity of the solving algorithm, e.g. with intelligent backtracking based on the generator/consumer paradigm [171] or with parallel AND-tree verification in [260]. More recently, in [339] an approach based on linear model reduction has been presented.

However, in the last 20 years research has investigated other logic paradigms relying on standard logic, as the prominent answer set programming described next, which do not need prior stratification for solving.

3.3.2 Planners based on the answer set semantics

A non-monotonic logic programming language alternative to Prolog is *Answer set programming (ASP)*. Introduced in [211], ASP was first proposed in [191] to solve the planning problem for autonomous agents. ASP is based on the *stable-model semantics* [108]. In order to understand what a stable model (also defined as answer set in [191]) for a logic program is, consider a classical (i.e. NAF-free) propositional logic formula F depending on a set X of Boolean variables. The

reduct of F to X , represented as F^X , is the formula resulting from the substitution of all variables appearing with negation with the false \perp . We then define a *stable model* (or *answer set*) of F as a *minimal model satisfying* F^X , namely a set $Y \subseteq X$ with no proper subsets satisfying F^X . For instance, let F be:

$$(q \rightarrow p) \wedge (\neg r \rightarrow q) \wedge (p \rightarrow s)$$

$Y = \{s, p, q\} \subset X = \{s, p, q, r\}$ is an answer set for F , since it satisfies $F^X : (q \rightarrow p) \wedge (\neg \perp \rightarrow q) \wedge (p \rightarrow s)$ and no proper subset of Y does. If F includes NAF, Y is an answer set if and only if it is also a stable model for the original formula when NAF variables are removed.

Generally, stable models are not unique, and they are found using satisfiability checking (SAT). A more detailed description of some strategies for SAT solving will be provided in the next chapter. The basic idea is that specifications are explored and new grounded atoms are inferred, keeping track of violated logic conditions to speed up the exploration. Differently from SLDNF, SAT solving does not require prior stratification of the logic program [193], hence it has been proved to be more efficient [155], while still being exponentially complex. However, SAT solving has applications in a variety of fields of computer science. For this reason, starting from the popular MiniSAT [300], a lot of research has been devoted to improve the efficiency of existing SAT solvers and develop parallel solvers to address the issue of complexity (see [210] for recent advances).

ASP syntax is very similar to Prolog, with differences depending on the specific modeling and solving tool. Starting from the first solvers Smodels [305] and DLV [78], up to the latest extensions as DLV2 [2], ASP syntax allows to define *aggregates* as $l\{\text{Atom} : \text{Condition}\}u$, namely sets of atoms satisfying specific conditions with bounded cardinality between l and u . In order to show the utility of this construct in the task planning problem, consider again the ring transfer domain. Specifying the aggregate $\{\text{move}(A, P, t) : \text{free}(P, t)\}$ asks the solver to generate all possible actions for all possible free pegs in the workspace; similarly, the specification $0\{\text{move}(A, P, t) : \text{free}(P, t)\}1$ imposes the generation of answer sets containing at most one action for each time step. In this way, it is possible to generate multiple plans at once. Differently from Prolog systems, ASP systems are not based on queries; instead,

solvers return the full answer set, i.e. all grounded atoms representing actions and the environment². This increases the interpretability of the robotic system, since a full description of the domain is returned.

Though a multitude of applications of ASP can be found in planning for autonomous agents in artificial intelligence and other fields of scientific research involving reasoning on knowledge (see [82] for a recent review), the applications in robotics are still fewer than Prolog-based systems, which have been well established for a significantly longer time. This is also due to the development of research for efficient SAT solving algorithms, which is significantly advancing only in recent years.

Currently, the most popular implementation for ASP, with applications in the robotic scenario, is Clingo [106], inspired from the DLV system, which from the latest release (Clingo 5) implements constraints on reals and multi-shot ASP solving for more efficient parallel computation. Clingo also supports useful optimization statements for optimal plan generation. An optimization statement acts as a weak constraint on the answer sets generated by the ASP solver. In other words, answer sets *must* guarantee the satisfaction of standard hard constraints defined in the task specifications, but additionally shall *preferably* satisfy the weak constraints from optimization. With reference to the example ring transfer task, assume to minimize the economy of motion, hence to grasp rings which are closer to any of the PSMs first. This can be easily expressed introducing a new fluent in the domain description, $\text{distance}(A, R, X, t)$, defining the distance between an arm and a ring as x ³. Then, it is possible to declare an optimization statement `#minimize{X: move(A, R, t) : distance(A, R, X, t)}`, which minimizes the value of x . It is also possible to define multiple optimization statements with priorities, in order to penalize some optimization scores with respect to others.

Relevant applications of Clingo include [262], where its variant Asprilo [105] for multi-robot logistics and warehouse automation, is exploited; [251], where

²Most ASP solvers allow to specify the grounded atoms of interest, so that for the task planning problem only action atoms, i.e. the plan, are output.

³It is only possible to use integers as numerical constants in ASP syntax. Hence, first all distances between rings and arms are computed; then, an increasing integer from 1 to 8 (2 PSMs, 4 rings) is assigned as a placeholder for real distance in fluents.

Clingo system is integrated with the ALICA language for teamworking in domestic and general-purpose environment [252]; [58] for general-purpose service robotics; [205], which proposes a task planning framework for domestic service robots receiving information and goal description in natural language from humans, at the RoboCup@Home challenge; [115], where an integrated framework for interpretable surgical task-motion planning is proposed; [11], which shows the ROSoClingo package integrating Clingo in ROS. A recent application based on the DLV system is [325], addressing the problem of mechanical assembly sequence in industry.

Several other implementations of ASP have been proposed in [313], where an Answer Set-based Conditional Planner (ASCP) is presented with applications also to robotic examples; [47], where ASP is implemented on the OK-KeJia service robot prototype for planning and decision-making based on multiple sources of information in the context of human-robot interaction; [18], proving the advantages of using ASP for task planning and failure analysis and explanation in human-robot interaction; [143] for the domestic service scenario; [303], which combines ASP with Markov decision processes in a domestic scenario; [26] for dual arm manipulation; [89] proposes a review of recent applications of ASP to industry, while [84] surveys recent applications in various robotic fields.

3.3.3 Other non-monotonic planners

Prolog- and ASP-based systems are the most prominent solutions to the task planning problem in robotics. Their main advantage lies in the non-monotonic representation of task knowledge, which is essential for robotic applications. Moreover, they are general-purpose languages, i.e. they can be adapted to diverse applications in robotics and artificial intelligence.

More application-specific solutions exist, which are optimized to address challenges of different scenarios. An example is Constraint-Handling Rules (CHR) [97], which improves performance of the solving process simplifying task-specific constraints online. An example of this approach is implemented in Fluent Executor (FLUX) [310], which has been used, e.g., for the automation of an office robot, hence in an application involving human-robot interaction [333].

Analogous application-specific approaches rely on a generic description of the task in an action language (e.g., PDDL), and implement an ad hoc formulation for the specific use case. An important example of this class of planners is ROSPlan [42], a framework combining PDDL-based task planning, motion planning, sensing and open-world planning through communication provided by the well-established Robot Operating System (ROS) [164]. Basing on ROS infrastructure offers a considerable implementative advantage for application with most robotic systems. ROSPlan has been applied in several scenarios, such as [220] for multi-robot navigation; [280] which integrates ROSPlan with Petri nets with an application involving human-robot interaction; [86], which tackles the anchoring problem, that is matching abstract symbols from high-level knowledge to perceptions and executed actions, for providing automated code generation of ROS nodes.

Other examples of custom implementations derived from PDDL are based on action graphs [129]; [119] for a PDDL-based planner applied to social robotics; [228] for coordination of unnamed ground and aerial vehicles; [235], where an integrated framework is tested on a robotic manipulator, a surveillance robot and a rover for exploration; [208], where a PDDL-based task planner for mobile robots in the ROS framework is proposed; SHOP [243] for a state-of-the-art implementation of hierarchical task network (HTN) [85], a popular and efficient formalism for task planning with sub-goals.

It is important here to highlight that task-specific approaches derived from generic PDDL descriptions of the task are not the only choice, and other custom implementations based, e.g., on ASP- and Prolog-like syntax are possible and can be adapted to the specific needs of the use case. For instance, in [72] AnsProlog, an implementation of Prolog in the answer set semantics which is the base for the original ASP solver “Smodels”, is used to implement HTN, proving comparable efficiency with respect to SHOP (a more extended comparison is available in [298]). Moreover, recently PDDL- and ASP-based planners have been compared [145]. The authors compare Clingo as the state-of-the-art implementation of ASP against the two most prominent FastDownward PDDL-based planners FDSS-1 [131] and LAMA-2011 [275], in different experimental scenarios including robot navigation. The comparison shows that PDDL-based

systems are generally more efficient when the required plan for the task is composed of a long sequence of actions. However, the computational performance of ASP-based systems are better for shorter plans, and they scale significantly better when the size of the domain description (i.e., the number of entities, actions and specifications) increases.

When very complex robotic domains are considered, planners described so far may suffer from computational inefficiency to keep track of all entities of the domain. A solution is offered by logic programming languages supporting the open-world assumption, integrating external knowledge bases to be queried when needed during the plan execution. The knowledge base encodes a detailed description of the scenario, while the logic program only encodes task specifications and queries the knowledge base for grounding of them. The most popular example of this approach is the CRAM framework [20] for advanced robotic manipulation, and its main backbone Knowrob [309] providing external knowledge base for general service robotic tasks. Knowledge is queried through SWI-Prolog, and this approach has been used in many applications, e.g. [337], where an outdoor search-and-rescue robotic task is fulfilled, performing geometric reasoning and semantic modeling of the environment; [55] for navigation multi-objective tasks; [27] for incomplete assembly task in industry; [118] for task planning in orthopaedic surgery, integrating orthopaedic-specific knowledge; [253] for the safety-critical scenario of mine-countermeasures missions for autonomous underwater vehicles, addressing the challenge of recovery from hardware failure and changes in priority levels. Another implementation, based on the answer set semantics, is a recent extension of DLV, DLVHEX [79], which provides specific constructs to query external knowledge bases. While not providing the same expressive power as description logics, the query-based approach integrating logic programming and knowledge bases is usually more efficient and preserves the non-monotonic assumption, which is essential for most real robotic applications.

3.4 Temporal logic planning

Standard logic planners offer the expressivity to describe many task planning problems in robotics, with operators describing causal relations between atoms. However, in several robotic scenarios additional expressivity is required to describe temporal relations between tasks, actions and environmental conditions. As an example, consider again the ring transfer domain. The user may easily define the final goal using the unary temporal operator *eventually* (\diamond), as $\diamond_{\text{on}}(R, P)$. Notice that the concept of time is encoded in the temporal logic operator, hence the user does not need to define the explicit temporal variable t as for standard logic planners (e.g., in ASP the same goal would be defined as $:- \text{not on}(R, P, t), t > 0$).

This section presents task planners for robotics which rely on the formalism from temporal logic, mainly linear temporal logic (LTL) [266], assuming a linear sequence of events, rather than multiple realizations as in more complex (and computationally inconvenient) non-linear temporal logic (e.g., CTL [81]). Specifically, an interpretation of LTL is considered over *finite traces* (LTL_f), as proposed e.g. in [28, 98, 328]. Formulas from LTL_f are interpreted on a finite time horizon. This is a reasonable assumption for robotic tasks, which typically involve a limited sequence of actions to be performed. The choice to focus on LTL is justified by two reasons. First, LTL_f has been proved to be as expressive as first-order logic [69]. LTL's extension LDL_f [61] including regular expressions from propositional dynamic logic [94] has been proved to be as expressive as monadic second order logic for finite automata [311]. The problem of synthesis for LDL_f formulas (i.e., the assignment of truth values to variables so that the given formulas hold) has been proved to be equivalent to the problem of conditional planning under full observability assumption, and 2EXPTIME-complete [60]. Hence, it is possible to encode a generic task planning problem from high-level action languages (e.g. PDDL) into $\text{LTL}_f/\text{LDL}_f$ formulas, which are then solved through satisfiability checking. For simplicity, the subscript f through the rest of this Section, hence LTL will stand for LTL_f .

Other than eventually, relevant temporal operators are here recalled to show the expressivity of the underlying logic. Unary operators are *next* \circ to define temporal sequences and *always* \square to define conditions which hold during the

whole execution; binary operators include *until* \mathcal{U} and *release* \mathcal{R} . Given two Boolean predicates a, b , $a\mathcal{U}b$ specifies that a must hold at least until b becomes true, while $a\mathcal{R}b$ asks that b holds until a becomes true (in this sense, a *releases* b). Temporal operators do not necessarily act on atoms, but also on specifications. For instance, in the ring transfer task a specification may be to continue moving to a ring until a new ring is in the scene and not placed on its peg. This can be encoded in LTL as $\text{reachable}(A, R) \mathcal{R} \neg_{\text{on}}(R, P) \rightarrow \text{move}(A, R)$.

A relevant tool for LTL in the robotic community is LTLMoP [93], to implement temporal task specifications in the ROS framework, with application e.g. to human-robot interaction. LTLMoP has been used e.g. for multi-target navigation [170]; task planning in a grocery store with an Aldebaran Nao robot [147]; multi-robot cooperative task planning [159, 158]; search-and-rescue mission in partially known environment with a Pioneer 3-DX robot [281]; task planning for modular robots [146]; multi-task planning for a patrol robot [199]; reactive task planning based on game theory [288]. Also relevant is the application of LTL-MoP to the problem of explanation of unsynthesizable plans for robotics [270], which is useful to enhance the interpretability of the robotic system.

LTLMoP implements satisfiability checking to solve the task planning problem from temporal specifications. As evidenced in [168] and in Section 3.3.2, one limitation of this approach is the *state explosion* problem, which is worsened by the introduction of the temporal dimension in the planning problem and hence the higher number of states and conditions to be verified in the solving process. This becomes even more crucial when implementing LTL plans on real robots, typically with hybrid control (connecting discrete-time and continuous-time control) so that the lower-level motion behavior matches the task-level specifications. In this scenario, LTL specifications must also account for the finer granularity of the real environment, thus leading to the infeasibility of analyzing all future scenarios. In order to cope with the computational demand, several solutions have been proposed.

For instance, in [175] the concept of *quantification of satisfiability* has been introduced, in order to guarantee the satisfaction of main LTL constraints while neglecting less important ones. This solution is applied, e.g., in [314], where maximal satisfaction of LTL task specifications for a Nao robot able to grasp,

drop a ball and walk with failure conditions is guaranteed.

Another widely used approach is the *receding time horizon* [331], which relies on the reasonable approximation that in most real robotic applications, the dynamic nature of the environment does not allow to devise a plan which develops long ahead in the future. Hence, a good design choice is to verify LTL formulas with a short time horizon, and to refine the resulting plan on the fly during the execution, according to the current state. In this way, only most probable future traces are analyzed, and the state explosion is significantly mitigated. A tool which implements the receding time horizon approach, differently from LTLMoP, is TuLiP [332]. Though it has been mostly applied to the provably-correct construction of hybrid controllers for integrated task-level (discrete) and motion planning (continuous) problems, worth mentioning are the applications in [197], where a robot surveillance task is planned through TuLiP, while local reactivity as in [30] and refinement is guaranteed through μ -calculus, a modal logic which can be used to express temporal formulas [33]; and [340] for whole-body biped locomotion in unstructured environment.

The two above solutions are not applicable to the scope of this thesis, though. In fact, quantifying relevant constraints to be considered in the solving process to neglect others is a design choice. The surgical scenario is often very complex, with a high number of safety constraints for the patient. Hence, the choice of relevant specifications is not easy, and may be dangerous in such a safety-critical scenario. As for limiting the maximum time horizon, it is possible for simple training tasks as, e.g., the ring transfer, consisting of a simplified domain. However, real surgeries often last for long time, and it would not be possible to gain enough efficiency because the time horizon would be exceedingly large.

Other LTL-based paradigms for robotic task planning exist, which have the same issues as the above mentioned most popular ones, though. They find mostly application in multi-robot systems coordination [162, 330], e.g. [173] coordinating single-robot tasks represented as Petri nets; [123], with dynamic leader selection under communication constraints; [122] with mutual distance constraints; [167] for search-and-rescue and coverage scenarios; [130] for manipulation tasks; [287] for time optimality under the ROS framework; [120], where

Signal Temporal Logic (STL) [73], a temporal logic devised for motion planning, is combined with standard LTL to deal with event-based task planning in a multi-robot domain. It is also important to mention other applications, including [334] where dynamic planning in a robotic fire-fighting scenario is investigated; [51] where the construction of behavioral trees from LTL specifications is used for simulated task planning of a mobile robot in adversarial environment; [16] for autonomous flexible manufacturing with experimental evaluation on a Gantry robot in Siemens NX Mechatronics Concept Designer simulator; [127] for warehouse robotics; [121] for robot navigation, integrated with lower-level model-checking-based motion planning through hybrid control; [40], where LTL over finite and infinite traces specifications derived from PDDL encoding are solved through translation to non-deterministic automata; [48], where LTL is used to design a medical robot tasked with roles of patient reception and triage.

As a final remark, there exist extensions of the syntax of standard logic programming to support temporal operators, which are then translated into standard logic formalism with the explicitation of the time variable by the solver. Examples of Prolog extensions can be found in [1], while recently temporal extensions to the ASP syntax *telingo* [38] and *tasplan* [37] have been proposed, based on Clingo solver. These extensions are based on temporal equilibrium logic [3], which integrates LTL operators into equilibrium logic [259], the logical characterization of stable models. Although no relevant applications of these paradigms are known to the author, temporal extensions of ASP and Prolog may well find a wide range of applications in the future, also due to the recent success of ASP solvers as Clingo. Moreover they introduce non-monotonicity in the task knowledge representation, while standard LTL is generally monotonic (though non-monotonic temporal logic has been introduced, e.g. in [19]).

3.5 Probabilistic logic planning

The logic planners presented in the previous sections allow to describe task planning problems with complex constraints, involving temporal and spatial constraints. Specifications can be checked at runtime, allowing for plan revision thanks to non-monotonic logic programming as in Prolog and ASP. However, an

important challenge for deliberative robots interacting with unknown environment is the quality of the perception. This requires a real-time planning system which not only accounts for the real-time information from sensors to update the environmental model, but also considers uncertainty of the sensed model and generates a sequence of most probable actions to be refined as new evidence is acquired. In this section, logic programming languages which enrich their semantics with concepts from probability theory are analyzed, in order to express specifications with a degree of uncertainty. Considered planners merge the standard and temporal logic syntax with the one of probabilistic logic presented in [246] such as the distribution semantics (DS) [282] and extensions (e.g. [206, 207]).

The syntax of probabilistic logic programs is enriched with p-annotated atoms and rules. Given $\mu \subseteq [0, 1]$, a *p-annotated Boolean* $a : \mu$ is a Boolean variable which is true with probability in μ . Similarly, in the formalism of logic programming it is possible to define p-atoms which can be grounded within some probability threshold. For instance, the location of an object `ob` may be known as `X` with accuracy between 80 – 85%, thus the p-atom `location(ob, X, t) : [0.8, 0.85]` can be defined. A *p-rule* is a rule containing p-annotated atoms. In order to understand how this syntax can be useful in a robotic scenario, consider the ring transfer task, and assume that the pose of a ring is only known with a probability between 60% and 70% due to noise in the point cloud from the RGBD camera. This affects the probability of the grounded fluents, e.g., `reachable(A, R, t)`. It must be then defined a p-rule `move(A, R, t) ← reachable(A, R, t) : [0.6, 0.7]`. `move` atom is not annotated, but it inherits the same probability range as the p-atom pre-condition `reachable`. If multiple pre-conditions must hold for an atom to be grounded, the annotations of all of them concur at defining its probability range.

This further complicates the solving process for probabilistic logic programs, which now consists not only in grounding atoms according to the specifications, but also checking the overlap of annotations and computing probability ranges for grounded atoms accordingly. Moreover, differently from the deterministic case, in probabilistic logic programming atoms cannot be grounded as false and ignored, unless the corresponding annotation vanishes or reduces to $\{0\}$. Hence,

an explosion problem similar to that of temporal logic can occur.

Several solutions have been proposed to face the computational issues related to probabilistic logic programming. For instance, it is possible to restrict the solution of the logic program to the deduction of tight logic consequences. Considering a set of (propositional logic) clauses (resulting from grounding of p-rules) \mathcal{P} with annotation μ (i.e., all clauses in \mathcal{P} have annotation μ), a clause $[H|B] : \mu_1$ is said to be a *tight logic consequence* of \mathcal{P} if $\sup \mu = \max \mu_1, \inf \mu = \min \mu_1$. Intuitively, a clause is a tight logic consequence of a set of clauses if it can be deduced from all their possible realizations. [206] shows that the deduction of tight logic consequences can be reduced to solving two linear programs. However, the problem can still be NP, depending on the number of possible realizations. Hence, the author proposes an algorithm to further mitigate the complexity of the problem when probabilistic clauses are separable from deterministic ones.

Another solution to the computational complexity of solving probabilistic logic programs is the use of sampling methods from stochastic systems, e.g. Monte Carlo. This is used e.g. in Problog [160], a popular probabilistic extension to Prolog which allows to define a weight assigning discrete probabilities to atoms and specifications. Problog and its variants have been applied in several robotic task planning scenarios, e.g. for opening doors [224] and object manipulation [226, 306, 12]. An alternative approach to Problog is the framework of distributional clauses (DC) [124], which extends Problog allowing to define continuous probability distributions over rules and atoms. DC has been used for robotic manipulation and grasping [225, 248, 227].

Both proposed solutions are inspired from the quantification of relevant constraints already proposed for LTL, since they induce a selection of relevant and discarded specifications. Hence, as already explained in the former section, they would not represent an appropriate solution for the surgical domain.

In spite of the computational issues which still make it inapplicable to most real-time robotic domains, probabilistic logic programming is promising to fill the gap between deterministic formal logic and the intrinsic uncertainty deriving from the planning-acting-sensing loop of deliberative robots.

Recent research is in fact focusing on integrating query-based planners with probabilistic knowledge in open-world domains, which can be updated as new

evidenced is acquired (similarly, e.g., to what is done with the CRAM framework in standard logic). A relevant example is proposed in [140], where the ProbCog system allows to query and reason on a Bayesian logic network (BLN) derived from the probabilistic logic of multi-entity Bayesian networks (MEBN) [179] in a domestic scenario with a B21 service robot. Basically, a MEBN is a knowledge representation formalism which connects fragments of Bayesian networks through statements in first-order logic. This framework has been used, e.g., in [21] for complex task decomposition and planning with a robot for home chore; manipulation [22, 249]; cognitive underwater vehicles [214].

A similar approach is presented in [250] for domestic tasks, using a Markov logic network (MLN) [273] instead of a BLN. MLNs are based on probabilistic weighted logic statements. In [222] task planning for manipulation and navigation scenarios is implemented querying a MLN with the STRIPS-based Metric-FF planner [133]; in [198] a robot for chemical experiment automation is proposed querying probabilistic action cores (PRACs) [249] based on MLNs. A mention is also deserved by LP^{MLN} [187, 186] to reason on MLNs using the stable model semantics under the action language $\mathcal{BC}+$ [14] for transition systems, though the actual application in robotic scenarios is still part of ongoing and under review research.

It is also important to highlight that research effort has been recently made towards the extension of the answer set semantics with probability theory. The main example is P-log [17] which is based on weighted specifications as Problog, used e.g. in the LCORPP framework for sequential robot planning [9]. In this way, in the next future it may be possible to exploit recent advances in SAT solving strategies to overcome the computational limitations of probabilistic logic.

At the current state of the art, an efficient solution of combining formal logics with probability theory is merging standard logic programming with Markov models in hierarchical task and motion planning, as e.g. in [303] for domestic robotic systems based on ASP and Markov decision processes.

3.6 Discussion

Based on the state of the art in logic task planning for robotics presented in the former sections, a number of considerations should be made before choosing the right planner for the specific use case.

First, the *expressivity* of the logic planner is important directly related to the field of application. Several applications of logic planning in robotics have been identified in this chapter, including domestic/service scenarios, navigation, industry, search & rescue, human-robot interaction, multi-robot coordination, surveillance and manipulation tasks. Most of these applications are covered by standard logic planners, which allow to specify conditional laws, constraints and optimization statements for optimal plan generation. The state-of-the-art Prolog-based planner, SWI-Prolog [327], can cope with different data types, including real variables, supporting the designer in the definition of proper specifications for real robots dealing with continuous data from sensors. ASP-based planners are still limited in this sense, though Clingo [106] allows to define arithmetic constraints.

Temporal planners shall be preferred for applications requiring provably-correct plans and safety guarantees, e.g. obstacle avoidance in navigation, multi-robot coordination and human-robot interaction, since they allow to express conditions from system theory as invariance and reachability of sets (e.g., for definition of system stability). For this reason, temporal planners usually combine task planning with control specification at the lower level of planning and help define hybrid controllers for the robotic application (e.g., TuLiP toolbox [331]).

Probabilistic planners represent a valid choice when integration with sensors is crucial to operate in an uncertain environment, e.g. in manipulation tasks and navigation. Distributional Clauses (DC) [124] also allows to represent continuous probability distributions, better representing the flow of information from sensors.

However, another important requirement for robotic scenarios is efficiency. The *computational complexity* of plan generation depends on the complexity of the task planning problem and the solving algorithm of the chosen planner. Standard logic planners are able to solve problems involving complex environment description and non-trivial autonomous capability (multiple actions on multiple

resources). Prolog-based planners suffer from the definition of loops in the logic program, hence they require prior stratification which represents an added complexity for the robotic designer and is not always possible.

On the contrary, more recent ASP-based planners overcome this limitation relying on SAT solving of logic programs. Moreover, (stochastic) SAT solving has been proved to be more efficient (while still NP-complete) and scalable than classical approaches as SLDNF of Prolog and first-order deduction on a number of benchmark problems [155]. The recent push in research on efficient and parallel SAT solvers is rapidly prompting the popularity of ASP-based planners in complex robotic scenarios.

Temporal planners as LTLMoP [93], even if based on linear temporal logic, suffer from the state explosion problem, which rises especially when long time horizons are needed for task completion. Techniques have been developed to overcome this limitation, e.g. assuming a receding time horizon in the state-of-the-art tool TuLiP.

Current probabilistic planners are not able to cope with very complex scenarios because of a similar state explosion problem. In fact, the distribution semantics is defined over sets of possible models, i.e. models from standard logic with an associated probability distribution. Hence, most applications involve manipulation tasks and simple (multi-robot) navigation scenarios.

Software implementation is an important factor in the choice of the task planner too, since it enhances the integration with external modules in charge of the other deliberative functions (e.g., sensing and motion planning). Standard logic planners are well established in the robotic community, hence they offer integration with the standard ROS framework for robotic research. ROSPlan [42] is a particularly versatile frameworks, since it does not require any specific logic formalism for task specifications, but generic PDDL interpretations.

The only temporal planner providing integration with ROS is LTLMoP [93], while ProbCog [140] was born as a probabilistic extension to the CRAM project [20], hence it offers built-in ROS interface. However, relevant frameworks as TuLiP and Problog [160] have Python and Java libraries, hence they can be easily integrated in the ROS framework and similars.

Support for open-world planning is an interesting feature for complex robotic

applications, where external knowledge bases may provide useful information about resources of the robotic system. SWI-Prolog is part of the well established CRAM project for open-world planning, but also novel ASP-based DLVHEX [79] offers easy integration with external knowledge bases, though applications to robotics are not known at the writing of this survey.

Among probabilistic planners, ProbCog is the natural statistical extension to the CRAM project, though the scalability to very complex scenarios remains an issue.

Finally, *plan revision* is a key ability for online robotic task planners, since it allows to cope with dynamic conditions which occur in many robotic scenarios. Prolog- and ASP-based planners rely on the formalism of non-monotonic logic, hence they inherently support online plan adaptation, as multi-shot solving with Clingo.

Also several probabilistic planners allow plan revision, since they are usually stochastic extensions of Prolog-based planners. Most temporal planners do not offer this feature, because they are based on monotonic versions of LTL. This is partly due to the different scope of temporal planners, which are intended to generate correct-by-constructions plans.

A summary of the results of the analysis of the state of the art in logic task planning for robotics is presented in Table 3.1. Examples of different logic programming paradigms referred to the ring transfer task lead to the conclusion that standard logic programming offers adequate expressivity to represent the considered task knowledge. Moreover, it provides several desired guarantees for safety-critical surgical scenario, as good real-time computational performance which is not usually reachable with temporal and probabilistic logic, and non-monotonicity of knowledge representation and reasoning, which is essential for dynamic scenarios as surgery.

Specifically, ASP paradigm is chosen for the purpose of this thesis, since it is rapidly becoming more and more popular in the research community, thanks to the fast development of efficient SAT solvers. Furthermore, ASP supports optimal plan generation and returns a full description of the scenario in terms of grounded terms, hence guaranteeing more interpretability and thus safety.

It is also important to recall that recent extensions of ASP supporting temporal logics and integration with stochastic motion-level models have been proposed, which can be exploited when considering more complex surgical tasks in the future.

3.7 Conclusion

This chapter has reviewed and discussed state of the art over the last 20 years in logic task planning for deliberative robots. Different features of planners have been compared, including expressivity, computational efficiency, re-planning capabilities and software support. Analysis has been performed considering the ring transfer task which represents the case study of this thesis. In this way, a proper evaluation of the best choice of planner for the surgical context has been carried out. ASP has been chosen as the logic programming paradigm for the scope of this thesis, and for surgery in general, because of its main advantages with respect to, e.g., temporal and probabilistic planners, or even its well established standard logic-based competitor Prolog. Among all benefits, high computational efficiency, strong expressive power and active research on the ASP paradigm have mainly determined the final choice discussed in the former section.

TABLE 3.1: Summary of the main reviewed frameworks for logic planning. Legend for entries: \mathcal{F} = logic formalism (*Pl* = Prolog); *RO S* = ROS support; *Impl.* = software implementation; *Integrated* specifies whether the framework is stand-alone or is integrated with other modules; *Online* = online adaptation / plan revision. Legend for applications: NAV = navigation; S&R = search&rescue; DOM = domestic environment; IND = industry, end for applications; NAV = navigation; S&R = search&rescue; DOM = domestic environment; IND = industry, SRV = service; HRI = human-robot interaction, MRC = multi-robot coordination; MAN = manufacturing; SUR = surveillance; LOC = locomotion, MANIP = manipulation.

	\mathcal{F}	Applications	Impl.	ROS	Integrated	On-line
SWI-Prolog [327]	Pl	DOM, SRV, NAV, IND, S&R	C++, Java, Python	✓	In CRAM [20] (open-world), Tartarus [290] (MRC)	✓
Readylog [161]	Pl	SRV, DOM	C++	✓		✓
Clingo [106]	ASP	DOM, SRV, IND	C, Python	✓		✓
ROSPlan [42]	PDDL	NAV, HRI, MRC		✓		
LTLMoP [93]	LTL	NAV, S&R, HRI, MRC	Python	✓		
TuLiP [332]	LTL	SUR, LOC	Python		Hybrid controller design	
Problog [160]	DS	MANIP	Python, Java			✓
DC [124]	DS	MANIP				✓
ProbCog [140]	BLN	DOM, MANIP, NAV	Python, Java	✓	Statistical learning&reasoning	✓
PCTL [277]	CTL	MAN, S&R, MRC				

Chapter 4

Answer set programming: formalism and solution methods

This chapter defines the syntax and semantics of answer set programming (ASP), introducing standard nomenclature which will be useful through the rest of this thesis. The reference for the contents of this chapter is the formalism of the grounder/solver Clingo, which provides useful syntax to define the robotic task planning problem and integrate task planning with the sensing module of the conceptual architecture of a deliberative robotic system. Specifically, Clingo v. 5.3¹ is used, since it offers the possibility to structure the task encoding conveniently for iterative temporal solving. Clingo also introduces additional constructs with respect to the standard ASP formalism described in [39], e.g. optimization statements (needed for the ring transfer scenario, and the surgical scenario in general to reason about preferences and real-time convenience depending on the environment / anatomy) and support for external atoms, needed for integration with sensors. Finally, the ASP solving process is described, to clarify how the task planning problem will be solved using answer sets in the remainder of this thesis, and highlight some advantages of Clingo in terms of efficiency.

¹<https://github.com/potassco/guide/releases/tag/v2.1.0>

4.1 Syntax and semantics of ASP programs

4.1.1 Standard constructs and axioms

As defined in Chapter 3 for a generic logic programming paradigm, ASP syntax is based on a sorted signature \mathcal{D} , defining terms which can appear in a program. A term in ASP syntax is either:

- a constant, either an integer or a string starting with a lower-case letter and containing letters, digits or the underscore symbol `_`;
- a variable, represented by a string starting with an upper-case letter. Values can be explicitly assigned to variables with the notation `Var(value)`, being `Var` the name of the variable and `value` a constant. Alternatively, it is possible to compactly specify a set of possible values for a variable. For instance, consider `Var` as a string variable which can have three possible strings as values, say `s1`, `s2`, `s3`. `Var(s1)`, `Var(s2)`, `Var(s3)` is equivalent to `Var(s1;s2;s3)`. In case `Var` is an integer variable with possible values in an interval, say from 1 to 4, `Var(1..4)` is shorthand for the explicit definition of possible values of `Var`;
- an arithmetic term, either with the unary operator `-` or as obtained from two terms combined with a binary arithmetic operator in $\{+, -, \times, /\}$;
- a functional term, or predicate, `f(t1, . . . tn)`, where each `ti` is a term. The arity of a predicate is the number of terms in it. Notice that predicates introduce a hierarchy in the sorted signature (function terms depending on other terms).

An *atom* in ASP is defined as a predicate with any arity (also 0, which reduces to a simple term). A *literal* is either:

- an atom `a` or its classical logical negation $\neg a$, denoted as `-a`²;
- a NAF-literal, i.e. an atom with negation as failure (NAF) `not a`;

²Typically, classical negation is not used in ASP programs, since it can be easily expressed with integrity constraints, which will be introduced in few lines.

- a comparison literal, i.e. two terms combined with a comparison operator in $\{<, >, \leq, \geq, ==, !=\}$;
- an aggregate literal $l \#agg\{e_1=w_1; \dots; e_n=w_n\} u$, where each e_i is an aggregate element in the form $t_1, \dots, t_j : l_1, \dots, l_k$, with t_i terms subject to conditions represented by l_i literals (possibly with NAF). w_i are weights associated to each aggregate element, while l, u are the bounds of the aggregate. agg is the name of the aggregate. In general, an aggregate defines an operation over a multiset of weighted literals. Clingo supports different aggregates; in this thesis, `sum` and `count` are used. `sum` returns all terms t_i subject to their conditions and such that the sum of their weights is between l and u . `count` is equivalent to `sum` with unitary weights for all aggregate elements, hence it simply imposes a constraint on the arity of the returned set of terms. Notice that `#count` can be omitted in the syntax of Clingo.

It is possible to define *axioms* (or *rules*) over the signature, with the following syntax:

$$h_1 \vee \dots \vee h_n : - b_1, \dots, b_m.$$

where $: -$ denotes logical implication \leftarrow , h_i are literals forming the *head* of the axiom, and b_i are literals forming the *body* of the axiom. The intuitive meaning of an axiom is that all literals in the body must hold (" $,$ " is shorthand for logical conjunction) for the literals in the head to be valid. In this thesis, an axiom is either:

- a fact, i.e. an axiom with empty body, meaning that the head always holds.
- a normal axiom with only one literal in the head³. Notice that also aggregate literals can appear in the axiom;

³A generic axiom with literals in logical disjunction can be easily replaced by a set of normal axioms, one for each head literal and with the same body of the original axiom.

- an integrity constraint with no literals in the head. This is equivalent to the statement

$$\perp : - b_1, \dots, b_m.$$

meaning that body literals cannot hold contemporarily, otherwise leading to logical inconsistency⁴.

Axioms in Clingo must be *safe*, i.e. all variables in the head of rules (or conditions) must also appear in the corresponding body, in a non-NAF literal. The reason for this will be clarified in Section 4.2.1.

4.1.2 Optimal answer sets

ASP allows to define *weak constraints* which express ordering between possible answer sets, according to the well established framework of preference reasoning and planning with non-monotonic logic [34]. A weak constraint has the form:

$$:\sim b_1, \dots, b_m.[w@l, t_1, \dots, t_n]$$

where w is denoted as weight, l as priority level, b_i are literals and t_i are terms. $[w@l, t_1, \dots, t_n]$ is denoted as the tail of the weak constraint. The weight defines the penalty assigned to answer sets containing b_i , while the priority level ranks different weak constraints. For better clarity, consider the following very simple ASP program:

```
0{p(1); p(2); p(3)}.
:\sim p(X).[1@1]
```

The first line specifies that any of $p(1..3)$ can be returned as an answer set of the program, even the empty answer set (notice that the lower bound of the aggregate is 0, and no upper bound nor conditions on the literals are specified). However, the second line assigns a weight of 1 to each answer set containing any

⁴In this thesis, integrity constraints will be denoted as *executability constraints*, since they are used to specify constraints on the execution of task actions.

of $p(1..3)$. Hence, the optimal answer set is the empty set, while any other answer set is sub-optimal with weight 1. Consider now a similar ASP program, but replace the weak constraint with:

```
:~ p(X).[1@1, X]
```

In this case, a term in the tail of the weak constraint is specified. This new statement assigns weight 1 *for all* atoms $p(1..3)$ in the answer set. Hence, though the optimal answer set is still empty, other answer sets are not equally sub-optimal (for instance, $\{p(1)\}$ has penalty 1, while $\{p(1), p(3)\}$ has penalty 2).

Clingo offers useful specific optimization statements which can be used in place of weak constraints, for clearer and more explicit syntax. In particular, it is possible to define optimization statements in the form:

```
#opt {e1=w1@p1; ...; en=wn@pn}.
```

where e_i are elements in the form $t_1, \dots, t_j : l_1, \dots, l_k$, i.e. terms subject to conditions similarly to aggregate elements; w_i are the weights for penalties; and l_i determine priority levels. `opt` is the name of the optimization statement, which can be either `maximize` or `minimize`. The meaning of the optimization statement is easy to understand: the answer set shall maximize (or minimize) the sum of the weights assigned to each element, favoring elements with higher priority.

4.1.3 Structure of an ASP program

As of version 5.3, Clingo allows to structure an ASP program into functional parts:

```
#base.
%Statements
#step(t).
%Statements
```

```
#check( $\tau$ ).
%Statements
```

This structure is intended for iterative computation of answer sets, as originally introduced with the iterative version of Clingo, iClingo [104]. Hence, it is particularly useful for the task planning problem in robotics, where the plan is composed of actions at consecutive discrete timesteps, and the pre-conditions and effects of each action shall be evaluated at each timestep to account for the dynamic nature of the scenario. τ is an in-built integer variable of Clingo, which is iteratively increased during solving. The `base` part contains ASP statements which are evaluated by Clingo only at the beginning of the solving process ($\tau=0$). Typical assertions in this part of the program are definitions of variables, constants and facts. An useful syntax of Clingo is the definition of `#external` variables, i.e. variables which can be set from other programs (e.g., sensory modules). Typically, `external` variables are specified with the syntax `#external term` : L_1, \dots, L_n , i.e. terms subject to conditions represented by literals.

On the contrary, `step` and `check` parts are evaluated iteratively, with increasing value of τ . Specifically, `check` contains a time-dependent definition of the terminating condition for the ASP program, which is usually identified as the goal in case the encoding represents the description of a robotic task. The `check` part is evaluated first, at each timestep (after `base` for $\tau=0$). If the terminating condition is not satisfied, then the `step` part is evaluated. It contains transition axioms, i.e. rules which define how variables in the ASP programs interact and change in time. This is where the specifications of the task planning problem lie in the ASP program. All atoms appearing in `check` and `step` parts of the program are fluents of the domain of the logic program, as of the generic description provided in the introduction to Chapter 3. For this reason, in the following of this thesis atoms depending from τ may be referred to as fluents for simplicity.

In Section 3.3, an answer set has been defined as the minimal set of logic variables satisfying the reduct of a logical program. However, usually not all variables are of interest for the purpose of the user. Hence, Clingo allows to define meta-statements as `#show` and `#hide` to specify which terms shall be included or hidden in the answer set returned to the user (e.g., `#show τ /ar`),

where ar denotes the arity of t .

4.2 Computation of answer sets

This section describes the two main steps involved in the computation of answer sets, given an ASP encoding in the form presented above: grounding and solving. Grounding transforms an ASP encoding in a propositional logic program. The stable model of the propositional ground program is then computed by the ASP solver. Clingo combines the grounding and solving capabilities. This section is not meant to be exhaustive, but to explain how plan generation is performed for a robotic task in the following of this thesis, and to evidence some advantages of Clingo with respect to other solvers. A deeper insight in the topic is available in [154].

Before describing the grounding/solving process, it is necessary to introduce the preliminary notions of Herbrand universe for an ASP program and substitution function for a set of variables.

Definition 1 (Herbrand universe and base). *Consider an ASP program P . The Herbrand universe $\mathcal{H}_{P,u}$ is defined as the set of all possible terms which can be constructed from constants and values in P . The Herbrand base $\mathcal{H}_{P,b}$ is the set of all atoms which can be constructed from predicates in P , having terms in $\mathcal{H}_{P,u}$ as arguments.*

Definition 2 (Global substitution). *Consider an ASP program P , containing a set of variables V . A substitution $\theta : V \mapsto \mathcal{H}_{P,u}$ is a function assigning variables with constant terms and values. A global substitution for an axiom A in P , denoted as $\theta(A) : V_A \mapsto \mathcal{H}_{P,b}$, is the application of a substitution function to all variables V_A appearing in A . In case aggregates are present in the axiom, only variables which appear also outside of the aggregate are substituted.*

4.2.1 Grounding

It is now possible to define the first step of ASP solving, which is *grounding*.

Definition 3 (Grounding of axioms). *Consider an axiom A in an ASP program P . If A is free of aggregates, the ground instance of A is the axiom $A' = \theta(A)$,*

such that no unassigned variables are in A . If A contains aggregates, grounding is performed in two steps:

- computing the set $G = \{A' = \theta(A)\}$;
- substituting each aggregate literal $\#agg\{e_1; \dots; e_n\}$ in G with the ground instance defined as $\#agg\{e_1; \dots; e_n\} = \{\theta(e_i) \mid i \in [1, \dots, n]\}$.

For better clarity, consider a simple ASP program with $\mathcal{H}_{p,u} = \{1, 2\}$ and containing the axiom A :

$$p(X) :- q(X, Y), \#sum\{Z, X: q(X, Z)\} < Y.$$

There exist four possible global substitutions for A , with all possible combinations of values in $\mathcal{H}_{p,u}$ assigned to the non-aggregate variables X, Y . For each global substitution, Z can be grounded with any of the two values in $\mathcal{H}_{p,u}$. For instance, assuming the substitutions $X \mapsto 1, Y \mapsto 2$, the grounded instance of the aggregate rule is as follows:

$$p(1) :- q(1, 2), \#sum\{1, 1: q(1, 1); 2, 1: q(1, 2)\} < Y.$$

It is straightforward to define a *ground program* P as the program obtained from grounding of all axioms in P . Notice that, in general, the grounding of an ASP program is EXPTIME-hard [154]. This is why many grounders (as the one of Clingo) require that axioms are safe, so that each variable in the head has a range bounded by the specifications in the body, and the grounding is faster. The computational burden is typically reduced by modern ASP grounders using optimization algorithms which aim at restricting the number of axioms to be grounded, identifying the relevant ones [7, 261], and optimizing the order of evaluation of atoms in axioms according to a dependency graph between them (for instance, if a rule R_1 has a predicate p in the head, and a second rule R_2 has p in the body, then R_1 should be evaluated first to reduce the size of the final ground program, while preserving the final answer sets of the ASP program).

4.2.2 Solving

With grounding, the ASP program is translated to a propositional logic program, i.e. all variables have been replaced by constant values and atoms represent Boolean variables, while axioms standard logical implications. As explained in Section 3.3, the goal of an ASP solver is then to compute the stable models, i.e. the minimal set of Booleans which satisfies the ground program. This is done using efficient strategies of Conflict Driven Constraint Learning (CDCL), pioneered in the area of satisfiability testing (SAT) [29]. CDCL is based on the identification of *nogoods*, which are intuitively invalid partial assignments of Boolean variables⁵. Nogoods can be already present implicitly in the ground program, e.g. completion and loop nogoods [102], or they can be generated during the solving process. The basic strategy of CDCL is sketched in Algorithm 2. Completion and loop nogoods of the ground program P_G ($\Delta_C(P_G)$ and $\Delta_L(P_G)$, respectively) are stored and used to constrain the logic program originated from grounding. Then theory propagation, i.e. partial assignments of logic variables, is performed, storing any found nogood δ in a set Δ_T of theory nogoods. The assignment of variables is iterated until all variables are assigned, or a (solvable) conflict with any of the stored nogoods in $\Delta_C(P_G) \cup \Delta_L(P_G) \cup \Delta_T$ is found. In the latter case, backjumping is performed: assigned variables are unassigned until the conflict is unit (only variables in the nogood are assigned), then the procedure continues. If all variables can be assigned without any conflict, the set of assigned variables is the stable model for the propositional logic program, hence the corresponding answer set is returned.

The following example clarifies how loop and completion nogoods, implicit in a ground program, can be found. Consider the ground ASP program (containing only normal axioms for simplicity):

```
a :- not b.
b :- not a.
x :- a, not c; x :- y.
```

⁵Notice that from now on, only Boolean variables are considered in this chapter, with reference to ASP solving, hence assignments are not related to the substitution of variables in the grounding process.

Algorithm 2 Basic CDCL

```

1: Input: Ground program  $P_G$  (in propositional logic form)
2: Output: Final set of assigned Booleans  $AS(P_G)$  (answer set)
3: Initialize: Compute  $\Delta_C(P_G)$  and  $\Delta_L(P_G)$ ;  $AS(P_G) = \emptyset$ 
4: while True do
5:   if no conflict  $c$  then
6:     if all variables assigned in  $AS(P_G)$  then
7:       return  $AS(P_G)$ 
8:     else
9:       Theory Propagation and variables assigned in  $AS(P_G)$ 
10:      Update  $\Delta_T$  with new  $\delta$ 
11:    end if
12:  else
13:    if  $c$  cannot be resolved then
14:      return UNSAT %Not satisfiable
15:    else
16:      while  $c$  is not unit do
17:        Backjumping and variables removed from  $AS(P_G)$ 
18:      end while
19:    end if
20:  end if
21: end while

```

$$y :- x, b.$$

This is translated to the following propositional logic program P_G :

$$\begin{aligned}
 a &\leftarrow \neg b & (4.1) \\
 b &\leftarrow \neg a \\
 x &\leftarrow a \wedge \neg c \vee y; y \leftarrow x \wedge b
 \end{aligned}$$

where the two rules with x in the head have been translated to a single logical implication. Notice that NAF has been translated to classical logic negation, according to the underlying principle of non-monotonic reasoning and ASP that *only provable theories are admitted*, i.e. unknown logic variable cannot be assumed to hold true [192]. Not all models of the logic program are valid for

ASP: for instance, variable c is not supported (implied) by any rule, thus it cannot appear in a stable model. Hence, c ⁶ is defined as a completion nogood for the program. Similarly, b can be unsupported if $\neg a$ holds. Completion nogoods are eliminated from final models turning simple implications into logical equivalences [50]:

$$\begin{aligned} a &\leftrightarrow \neg b \\ b &\leftrightarrow \neg a \\ x &\leftrightarrow a \wedge \neg c \vee y \\ y &\leftrightarrow x \wedge b \\ c &\leftrightarrow \perp \end{aligned}$$

Notice the last equivalence to directly add c into $\Delta_C(P_G)$. This new logic program has obviously fewer models than the original one in (4.1). Specifically, the three possible models are $\{b\}$, $\{b, x, y\}$ and $\{a, x\}$. However, the program still contains loop nogoods, i.e. nogoods generated from loops of implications. In the example program, a loop between x and y exists. As shown in [196], it is possible to exclude loop nogoods adding a support rule which contains the loop variables in the head, and does not contain any positive variable in the body which corresponds to a loop variable. For the proposed example, such a support rule is the following:

$$(x \vee y) \rightarrow a \wedge \neg c \tag{4.2}$$

which excludes the model $\{b, x, y\}$ from the possible models. (4.1) and (4.2) characterize stable models of P_G . While the size of rules generated to exclude completion nogoods is linear in the size of P_G , the size of rules generated to exclude loop nogoods may be exponential [194]. For this reason, the solving system of Clingo does not generate the rules at the beginning of the solving process, but it detects loop violations while executing Algorithm 2 exploiting efficient techniques of unfounded set detection [189, 103]. Moreover, Clingo

⁶This is shorthand for $c = \text{True}$, as well as $\neg c$ is shorthand for $c = \text{False}$.

further speeds up the solving process exploiting parallel theory propagation: different propagators are designed to propagate different rules and assign different variables, and only interested propagators are invoked when variables are unassigned during backjumping. Finally, as of Clingo 5 propagators are lazy, i.e. they do not test all rules for checking violation of nogoods in the assignments; when all variables are assigned (Line 6 of Algorithm 2), propagators are invoked for a final check of validity [149].

Chapter 5

A logic-based framework for surgical task autonomy

5.1 Introduction

This chapter presents a novel modular framework for autonomous execution of surgical tasks. As explained in Chapter 1, a proper framework must address challenging issues typical of advanced autonomous robotic systems, including real-time situation awareness for monitoring and adaptation of the workflow, interpretable plan generation for safety, dexterous trajectory generation and adaptation even in a small workspace. Previous chapters have evidenced the suitability of ASP as a logic programming paradigm to meet the aforementioned requirements. Hence, the framework proposed in Section 5.2 combines an ASP-based task planner with adaptive motion planning and control. The high- and low-level modules are tightly connected with sensors, which provide a semantic interpretation of the robotic scenario and guarantee real-time situation awareness to guide the workflow of execution. The performance of ASP-based task planning (in simulation) and the full framework (with dVRK) are validated in Section 5.3 on the ring transfer.

5.2 The framework

A scheme of the proposed framework for autonomous surgical task execution is shown in Figure 5.1. The exchange of information between its modules and the real system (robot + sensors) is shown. The flow of information towards an

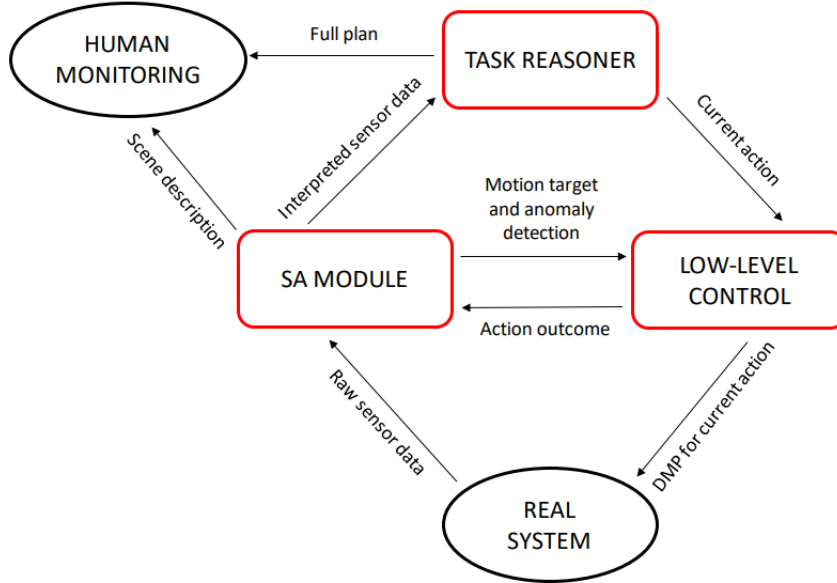


FIGURE 5.1: The proposed framework for surgical task autonomy. Functional modules of the framework are highlighted in red, while arrows show the stream of information between modules, the real system and an external human observer.

external human observer is also represented. The human can read the semantic conditions identified by sensors and the plan scheduled for execution at runtime, monitoring the correctness of the overall system. The framework consists of three main functional modules.

A *task reasoner* takes action-level decisions based on an ASP description of the task. In particular, it analyzes environmental conditions (expressed in terms of logic atoms) and generates a plan, i.e. a sequence of elementary actions to be executed to reach the final goal.

Actions in the plan are commanded sequentially to the *low-level (motion) control* module. As mentioned in Chapter 1, the main objective of this thesis is to preserve the safety and efficiency guaranteed by expert surgeons. For this reason, the motion control module stores learned motion trajectories for each elementary action in the task, and is able to generalize (based on imitation learning) learned trajectories from humans to the contingent situation of the task (i.e., the initial and target pose). In other words, the goal of the low-level control module is to replicate the *shape* of expert gestures, and only adapt shape to specific targets

identified by sensors. As an example, consider the case of moving with a PSM to the red ring first, and to the yellow ring then. The motion trajectory, including the shape in Cartesian space and the orientation adaptation while moving, should be similar for both actions, though the targets are different given the different locations of the two rings on the peg base. In this way, the dexterity of gestures of surgeons is preserved also in the autonomous framework, guaranteeing proper manipulation and, in general, interaction with the anatomical environment.

The connection between motion control and task reasoning is guaranteed by a *situation awareness (SA) module*. It combines information coming from sensors (kinematics of dVRK and RGBD camera) in real time, and computes logic atoms describing the environment and the task situation. This information is communicated to the task reasoner for planning and re-planning, and to the motion control module in case anomalies or failure conditions occur, requiring stop of the execution.

The following sections describe the single modules in detail.

Task reasoning module

The ASP encoding of the ring transfer task is structured following the architecture shown in Section 4.1.3. The base part of the program defines the statics relevant to the task: `object`, with values in `{ring, peg}`; `color`, with values in `{red, blue, green, yellow, grey}`; and `arm`, with values in `{psm1, psm2}`. From now on, the shortcodes `O`, `C`, `A` will be used to refer to the aforementioned statics, respectively.

This part of the program also contains definitions of `external` atoms, which describe the environmental features computed from the SA module that concur in the decision of the workflow of execution in real time. These atoms are `reachable(A, O, C)` to encode the concept of reachability regions for the two PSMs; `at(A, O, C)` and `at(A, center)`, describing the location of a robotic arm with respect to rings and pegs, and with respect to the center of the base (meeting point to transfer the ring between the PSMs); `in_hand(A, ring, C)`, to describe that a ring is held by an arm; `closed_gripper(A)` representing the

gripper status of an arm; and `on(ring, C1, peg, C2)`¹ indicating that a ring is placed on a peg.

The `step(t)` part of the program contains the specifications of the task. First, the possible elementary actions for the PSMs are defined, based on the task description presented in Chapter 2:

- `move(A, O, C)` to move an arm to either a ring or a peg with color `C`;
- `move(A, center, C)` to move an arm to the center of peg base while holding a ring with color `C`;
- `extract(A, ring, C)` to extract a ring with color `C` from a peg;
- `grasp(A, ring, C)` to grasp a ring with color `C`;
- `release(A)` to open the gripper.

The definition of actions follows, when possible, the standard of SPMs [178]. In fact, each action represents an elementary robotic operation which can be directly executed as a single motion trajectory (or, for gripper actions, joint actuation). Moreover, the syntax of action includes the identifier of the agent performing the action (one of the PSMs), the semantic target of the action (for instance, for `move` actions, either the ring or the peg) and a relevant property for the specific action (for this task, the color of the ring or the peg, recalling the instrument used by the surgeon in action definitions of real surgery). The only exception to this syntax is `release` action, which is not necessarily associated to a specific semantic target (for instance, at the beginning of the task the gripper may be closed, hence it must be open to start the execution and grasp a ring).

Each action is subject to *pre-conditions* and *executability constraints*. Pre-conditions for actions of the ring transfer task are defined as follows:

$$\begin{aligned}
 0 \{ & \text{move}(A, O, C, t) : \text{reachable}(A, O, C, t); & (5.1) \\
 & \text{move}(A, \text{center}, C, t) : \text{in_hand}(A, \text{ring}, C, t); \\
 & \text{extract}(A, \text{ring}, C, t) : \text{in_hand}(A, \text{ring}, C, t);
 \end{aligned}$$

¹`C1, C2` represent two possibly different `color` variables. From now on, this will be the syntax to represent different variables of the same type.

```

grasp(A, ring, C, t) : at(A, ring, C, t);
release(A, t) : in_hand(A, ring, _, t) } 1.

```

The temporal variable is the same both for pre-conditions and actions, meaning that an action can start as soon as the corresponding necessary environmental fluents hold. The semantics of pre-conditions in 5.1 is clear; the aggregate structure (a set of atoms with lower l and upper u bounds on the cardinality, $l \{ \} u$, see Chapter 4) specifies that at most $u=1$ action (among the ones which are possible if the corresponding pre-condition holds) per time is allowed, hence the execution of the task is *sequential*. The sequential execution is one possible paradigm for the execution of the ring transfer task, forcing that only one PSM moves per time, except for transfer with $\text{move}(A, \text{center}, C)$. In fact, this action will be translated, at the motion level, into two trajectories, one per arm (arms must meet at the center eventually).

Since dVRK has two PSMs, *parallel* execution is also considered in this thesis, i.e. arms can move and operate on different rings contemporarily, hence finding a satisfiable plan within a possibly shorter time horizon. This can be easily implemented in ASP modifying the aggregate rule in 5.1 as follows:

```

0 {move(A, O, C, t) : reachable(A, O, C, t);      (5.2)
move(A, center, C, t) : in_hand(A, ring, C, t);
extract(A, ring, C, t) : in_hand(A, ring, C, t);
grasp(A, ring, C, t) : at(A, ring, C, t);
release(A, t) : in_hand(A, ring, _, t) } 1 :- arm(A).

```

The meaning is that one action *per arm* can be returned at each timestep. As a consequence, $\text{move}(A, \text{center}, C)$ action now translates to the single motion of the arm which is holding ring C towards the center (another action $\text{move}(A1, \text{ring}, C)$ will be commanded at the same time step for the other PSM).

Executability constraints of the ring transfer task specify forbidden actions under specific environmental conditions. They are:

```

:- move(A1, peg, _, t), in_hand(A1, ring, C, t),

```

$$\text{in_hand}(A2, \text{ring}, C, t), A1 \neq A2. \quad (5.3a)$$

$$\begin{aligned} &:- \text{move}(A, \text{center}, C, t), \text{in_hand}(A, \text{ring}, C, t), \\ &\quad \text{on}(\text{ring}, C, \text{peg}, _, t). \end{aligned} \quad (5.3b)$$

$$\begin{aligned} &:- \text{move}(A, \text{peg}, _, t), \text{in_hand}(A, \text{ring}, C, t), \\ &\quad \text{on}(\text{ring}, C, \text{peg}, _, t). \end{aligned} \quad (5.3c)$$

$$:- \text{move}(A, \text{ring}, _, t), \text{closed_gripper}(A, t). \quad (5.3d)$$

$$:- \text{move}(A, \text{peg}, C, t), \text{on}(\text{ring}, _, \text{peg}, C, t). \quad (5.3e)$$

Axiom 5.3a guarantees that no arm moves if they are both holding the same ring (during transfer), to avoid rupture of the ring; axioms 5.3b-c prevent the motion of a ring which is placed on a peg (extraction is needed first); axiom 5.3d forbids movement of an arm to a ring if the gripper is closed (grasping would be impossible); and axiom 5.3e specifies that it is not possible to move to an occupied peg (only one ring may be placed on a same peg).

The *step* part of the ASP program finally includes *effects* of actions on the environmental fluents:

$$\text{in_hand}(A, \text{ring}, C, t) :- \text{grasp}(A, \text{ring}, C, t-1). \quad (5.4)$$

$$\begin{aligned} \text{in_hand}(A, \text{ring}, C, t) &:- \text{in_hand}(A, \text{ring}, C, t-1), \\ &\quad \text{not release}(A, t-1). \end{aligned}$$

$$\text{closed_gripper}(A, t) :- \text{grasp}(A, \text{ring}, _, t-1).$$

$$\begin{aligned} \text{closed_gripper}(A, t) &:- \text{closed_gripper}(A, t-1), \\ &\quad \text{not release}(A, t-1). \end{aligned}$$

$$\begin{aligned} \text{on}(\text{ring}, C1, \text{peg}, C2, t) &:- \text{in_hand}(A, \text{ring}, C1, t-1), \\ &\quad \text{at}(A, \text{peg}, C2, t-1), \text{release}(A, t-1). \end{aligned}$$

$$\begin{aligned} \text{on}(\text{ring}, C1, \text{peg}, C2, t) &:- \text{on}(\text{ring}, C1, \text{peg}, C2, t-1), \\ &\quad \text{not extract}(_, \text{ring}, C1, t-1). \end{aligned}$$

$$\text{at}(A, \text{ring}, C, t) :- \text{move}(A, \text{ring}, C, t-1).$$

$$\text{at}(A, \text{peg}, C, t) :- \text{move}(A, \text{peg}, C, t-1).$$

$$\text{at}(A, \text{center}, t) :- \text{move}(A, \text{center}, _, t-1).$$

Notice that the head of axioms (effects) have a different time step than atoms in the bodies (causes). In fact, e.g., the gripper is closed *right after* the grasping actions is concluded. The above axioms hold for parallel execution. For sequential execution, the following axiom must be included:

$$\text{at}(A, \text{ring}, C, t) \text{ :- move}(A1, \text{center}, C, t-1), \text{arm}(A), A \neq A1. \quad (5.5)$$

In this way, the program considers that $\text{move}(A, \text{center}, C)$ actually involves the motion of both arms to the center for transfer, hence the PSM which does not carry the ring reaches it at the end of the action.

Axioms 5.4 express that fluents $\text{in_hand}(A, \text{ring}, C)$, $\text{closed_gripper}(A)$, $\text{on}(\text{ring}, C1, \text{peg}, C2)$ keep holding until specific conditions or actions occur (e.g., $\text{closed_gripper}(A)$ stops holding when $\text{release}(A)$ is executed). Defining the persistence of these fluents is strictly needed to guarantee that a plan for the ring transfer can be computed. In fact, e.g., $\text{in_hand}(A, \text{ring}, C)$ must hold until the ring is placed on a peg, otherwise it is not possible to execute, e.g., $\text{move}(A, \text{center}, C)$ nor $\text{extract}(A, \text{ring}, C)$ from axioms 5.1. A more complete ASP encoding should include also the persistence conditions for $\text{at}(A, O, C)$ and $\text{at}(A, \text{center})$ atoms. Intuitively, these atoms represent locations of PSMs, hence they stop holding only when arms move. Hence, it is sufficient to include the following axioms in the program:

$$\begin{aligned} \text{at}(A, O, C, t) & \text{ :- at}(A, O, C, t-1), & (5.6) \\ & \text{not move}(A, O, C, t-1), \text{not move}(A, \text{center}, t-1). \\ \text{at}(A, \text{center}, t) & \text{ :- at}(A, \text{center}, t-1), \\ & \text{not move}(A, O, C, t-1), \text{not move}(A, \text{center}, C, t-1). \end{aligned}$$

Finally, the $\text{check}(t)$ part of the program contains the definition of the goal of the ring transfer task:

$$\text{:- reachable}(_, \text{ring}, C, t), \text{reachable}(_, \text{peg}, C, t),$$

```
not on(ring, C, peg, C, t).
```

which specifies that the plan computation must continue until there is at least one visible ring (i.e., reachable by any arm) not placed on the corresponding peg.

The ASP program ends with the specification of relevant atoms in the answer set. These atoms are the ones that are needed for the proper working of the framework. Relevant atoms are the ones representing actions, since they must be communicated to the low-level control module for execution. Hence the following declarations are specified, with syntax described in Chapter 4:

```
#show move/4.
#show extract/4.
#show release/2.
#show grasp/4.
```

Situation Awareness (SA) module

This module is in charge of the semantic interpretation of data from sensors, to ground external atoms in the ASP planner, provide target poses to the low-level control module (for `move` actions) and recognize failure conditions during the execution. Moreover, the high-level description of the environment in real-time can be easily read from human supervisors and enhances explainability and safety of the framework.

The input to the SA module are the kinematic information from dVRK, namely position \mathbf{p}_A , orientation \mathbf{o}_A (quaternion) and gripper angle j_A of PSM A (for both PSMs); and the point clouds of rings and pegs returned by Detection Algorithm 1.

Point clouds of rings and pegs are used to compute poses (position + orientation) of rings, pegs, and the center of the peg base (in the common world frame, see Figure 2.4 left for axis convention). In particular, for peg C with point cloud $PC = PC_{pc}$, the position is chosen as $\mathbf{p}_{pc} = \{\bar{x}_{PC}, \bar{y}_{PC}, \max(z_{PC})\}$, being $\bar{\cdot}$ the average of coordinates. The orientation of pegs is not relevant for the task, since they are assumed to be fixed and normal to the peg base. As a consequence,

when $\text{move}(A, \text{peg}, C)$ is commanded, the goal pose has the position of the peg, and the orientation of the arm² is only constrained to be normal to the peg base.

Given the point cloud PC_{rc} of ring C , the position of the ring is considered as the position of its center, and it is computed as the average of 3D coordinates in PC_{rc} . This position is used to compute environmental fluents (e.g., relative position of rings and pegs). It is also important to compute the grasping pose for each ring, which will be the target for $\text{move}(A, \text{ring}, C)$. The grasping position is computed in order to maximize the distance from pegs (hence reducing the probability of collision), as $\arg \max_{r \in PC_{rc}} \sqrt{\sum_{p \in X} \|r - p\|_2^2}$, being X the set of positions of pegs. As for the orientation of grasping, it is chosen such that the arm arrives at the grasping position normal with respect to the ring. The normal to the ring is the same of a plane fit on the points in PC_{rc} . Moreover, the rotation of the gripper around the z -axis of the world is prescribed such that the gripper opens in a plane orthogonal to $x - y$ plane of world frame, and containing both the center and grasping point of the ring. In this way, the PSM grasps the ring with an orientation which is radial with respect to the ring itself.

Finally, the pose of the center of the base is needed to define reachability regions and the meeting point for transfer. The position of the center is identified as the average position of pegs, and it is chosen as the target position for $\text{move}(A, \text{center}, C)$ (in case of sequential execution, this action involves two motion trajectories, hence also the grasping target on ring C is specified). The orientation of A is just constrained to be normal to the peg base.

The SA algorithm uses the aforementioned quantities (together with kinematic information from dVRK) to compute (external) environmental fluents for the ring transfer task. Table 5.1 collects the useful geometric variables for convenience. The following relations are used to compute external fluents from geometric quantities.

$$\begin{aligned} \text{at}(A, \text{ring}, C) &\leftarrow \|\mathbf{p}_A - \mathbf{p}_{rc}\|_2 < rr \wedge j_A > \frac{\pi}{8}^3 & (5.7) \\ \text{at}(A, \text{peg}, C) &\leftarrow \|\mathbf{p}_A - \mathbf{p}_{pc}\|_2 < rr \wedge z_{pc} < z_{pos,A} \\ \text{in_hand}(A, \text{ring}, C) &\leftarrow \|\mathbf{p}_A - \mathbf{p}_{rc}\|_2 < rr \wedge j_A < \frac{\pi}{8} \end{aligned}$$

²From now on, when referring to position and orientation of an arm, the end effector of the arm will be implicitly taken into account.

TABLE 5.1: Geometric quantities for fluent computation.

Name	Description
$\mathbf{p}_{rc} = \{x_{rc}, y_{rc}, z_{rc}\}$	position of the center of ring with color C
$\mathbf{p}_{pc} = \{x_{pc}, y_{pc}, z_{pc}\}$	position of the tip of peg with color C
$\mathbf{p}_b = \{x_b, y_b, z_b\}$	position of the center of the peg base
$\mathbf{p}_A = \{x_{pos,A}, y_{pos,A}, z_{pos,A}\}$	position of the tip of arm A
j_A	opening angle of the gripper of arm A
rr	ring radius

$$\begin{aligned}
\text{on}(\text{ring}, C1, \text{peg}, C2) &\leftarrow \|\mathbf{p}_{rc1} - \mathbf{p}_{pc2}\|_2 < rr \wedge z_{rc1} < z_{pc2} \\
\text{reachable}(A_x, O, C) &\leftarrow \text{argmin}_A |y_{oc} - y_{pos,A}| = A_x \\
\text{closed_gripper}(A) &\leftarrow j_A < \frac{\pi}{8} \\
\text{at}(A, \text{center}) &\leftarrow \|\{x_{pos,A}, y_{pos,A}\} - \{x_b, y_b\}\|_2 < rr
\end{aligned}$$

Another important function of the SA module is the real-time check of failure and anomalies during the task execution. Different failure conditions are identified for (move) actions:

- for action $\text{move}(A, \text{ring}, C)$, failure is detected when $\text{reachable}(A, \text{ring}, C)$ does not hold anymore, meaning that the ring has been removed from the scene or it has been moved to a non-reachable region for arm A ;
- for action $\text{move}(A, \text{peg}, C)$, failure is detected either when $\text{on}(\text{ring}, C1, \text{peg}, C)$ holds (a ring has been placed on the target peg) or $\|\mathbf{p}_A - \mathbf{p}_{rc}\|_2 > 2 \cdot rr$ (the ring has fallen);
- for action $\text{move}(A, \text{center}, C)$, failure is detected when $\|\mathbf{p}_A - \mathbf{p}_{rc}\|_2 > 2 \cdot rr$

In case failure is identified, the low-level control module is notified, current DMP stops and an updated grounding of external atoms is sent to the task reasoner to compute a new plan.

The functions of SA module are summarized in Algorithm 3.

³The limit value to consider the gripper as closed is chosen empirically.

Algorithm 3 SA Algorithm

```

1: Input: Robot kinematics  $K$ , geometry from vision  $V$ , current action
2: Output: DMP target  $g$ , set of external atoms  $EA$ 
3: Initialize:  $EA = \emptyset$ , fail = False
4: while not fail do
5:   if action = null %No plan available then
6:      $EA = \text{compute\_ext}(K, V)$ 
7:   else
8:      $g = \text{compute\_goal}(\text{action}, V)$ 
9:     fail = fail\_check(action, K, V)
10:    if fail then
11:      Stop motion
12:       $EA = \text{compute\_ext}(K, V)$ 
13:    end if
14:  end if
15: end while

```

Low-level control module

The low-level control module receives actions by the task reasoner, and commands the motion of PSMs accordingly. Actions of the gripper and `extract(A, ring, C)` are simple actions, which require opening/closing of the gripper joint of PSMs and upward motion normal to the peg base, respectively. `move` actions vary depending on the initial / target pose, and the presence of obstacles (pegs). For this reason, the low-level control module must guarantee adaptability to the current scenario, while preserving the dexterity of usual motions by surgeons or expert operators of dVRK. To this aim, Dynamic Movement Primitives (DMPs) for `move` actions are implemented. DMPs are a framework for trajectory learning. They are based upon an Ordinary Differential Equation (ODE) of spring-mass-damper type with a forcing term. This framework has numerous advantages that make it well suited for robotic applications. First, any trajectory can be learned and subsequently executed while changing starting and goal poses. Second, the executed trajectory will always converge to the goal, maintaining a similar shape to the learned trajectory. Third, the learned trajectory can be executed at different speed simply by changing a single parameter. Finally, DMPs have been proven to be flexible enough to being extended in multiple ways: for instance, the formulation can be modified to deal with periodic

movements [136, 316], to learn sensory experience [258, 257]. An exhaustive description of the framework is out of the scope of this thesis, hence here only key concepts are presented for completeness. A more detailed description of the topic can be found in [254, 132, 256, 113] and the original formulation proposed by [137, 136, 284, 138].

For each `move` action, a DMP for Cartesian motion and a DMP for orientation space are implemented (for `move(A, center, C)`, two DMPs for each metric space are needed in case of sequential execution, one per arm). In the following, DMPs for Cartesian space are described for simplicity. A very similar formulation is used for quaternion space, as presented in [315, 283].

DMPs for Cartesian trajectories consist of the following system of ordinary differential equations:

$$\begin{cases} \tau \dot{\mathbf{v}} = \mathbf{K}(\mathbf{g} - \mathbf{x}) - \mathbf{D}\mathbf{v} - \mathbf{K}(\mathbf{g} - \mathbf{x}_0)s + \mathbf{K}\mathbf{f}(s) + \mathbf{f}_{obs} & (5.8a) \\ \tau \dot{\mathbf{x}} = \mathbf{v} & (5.8b) \end{cases}$$

Vectors $\mathbf{x}, \mathbf{v} \in \mathbb{R}^d$ are, respectively, the *position* and *velocity* of the system; and $\mathbf{x}_0, \mathbf{g} \in \mathbb{R}^d$ are, respectively, the *starting* and *goal positions*. Matrices $\mathbf{K}, \mathbf{D} \in \mathbb{R}_+^{d \times d}$ are, respectively, the *elastic* and *damping terms* of the system. Both are diagonal matrices, $\mathbf{K} = \text{diag}(K_1, K_2, \dots, K_d)$, $\mathbf{D} = \text{diag}(D_1, D_2, \dots, D_d)$, and satisfy the critical damping relation $D_i = 2\sqrt{K_i}$, so that the un-perturbed system, i.e. when $\mathbf{f} \equiv \mathbf{0}$, converges as fast as possible to the unique equilibrium $(\mathbf{x}, \mathbf{v}) = (\mathbf{g}, \mathbf{0})$. Scalar $\tau \in \mathbb{R}_+$ is a *temporal scaling factor* which can be used to make the execution of the trajectory faster or slower. Function $\mathbf{f} : \mathbb{R} \rightarrow \mathbb{R}^d$ is the *forcing* (also called *perturbation*) *term*. Scalar $s \in (0, 1]$ is a re-parametrization of time $t \in [0, T]$ governed by the so called *canonical system*

$$\tau \dot{s} = -\alpha s, \quad (5.9)$$

where $\alpha \in \mathbb{R}_+$ and the initial state is $s(0) = 1$.

The forcing term $\mathbf{f}(s) = [f_1(s), f_2(s), \dots, f_d(s)]^\top$ is written in term of basis functions. Each component $f_p(s)$, $p = 1, 2, \dots, d$ has then the form

$$\mathbf{f}(s) = \frac{\sum_{i=0}^N \boldsymbol{\omega}_i \psi_i(s)}{\sum_{i=0}^N \psi_i(s)} s, \quad (5.10)$$

where ${}^p\omega_i \in \mathbb{R}$ is called *weight*, and $\psi_i(s)$ is a *Gaussian Radial Basis (GRB) function* defined as

$$\psi_i(s) = \exp\left(-h_i(s - c_i)^2\right), \quad (5.11)$$

with *centers* c_i defined as

$$c_i = \exp\left(-\alpha i \frac{T}{N}\right), \quad i = 0, 1, \dots, N, \quad (5.12)$$

and *widths* defined as

$$h_i = \frac{1}{(c_{i+1} - c_i)^2}, \quad i = 0, 1, \dots, N-1, \quad (5.13)$$

$$h_N = h_{N-1}.$$

During the *learning phase*, a desired trajectory $\tilde{\mathbf{x}}(t)$ and its velocity $\tilde{\mathbf{v}}(t)$ are recorded. Then, from (5.8a), the desired forcing term $\tilde{\mathbf{f}}(s(t))$ is computed (after fixing matrices \mathbf{K} and \mathbf{D}). Finally, the weights ${}^p\omega_i$, $i = 0, 1, \dots, N$, $p = 1, 2, \dots, d$ that best approximate the desired forcing term $\tilde{\mathbf{f}}$ using formulation (5.10) are computed. Notice that weights can be learned in closed-form solution, since the fitting problem is linear in the parameters (assuming basis functions are pre-computed). Hence, it is theoretically possible to learn weights only from one example trajectory. In the ring transfer task, weights for each `move` action are saved as prior (both for position and orientation trajectories), to be used in the real execution phase.

During the *execution phase*, starting and goal positions \mathbf{x}_0, \mathbf{g} are set (as computed from the situation awareness module of the framework), and the forcing term \mathbf{f} is computed using (5.10) with learned weights. Solving the dynamical system (5.8) will give a trajectory of similar shape to the learned one, that starts from \mathbf{x}_0 and converges to \mathbf{g} .

For effective motion planning of the PSMs of the dVRK robot, also obstacle avoidance with DMPs must be implemented. In the ring transfer scenario, obstacles are represented by pegs. Obstacle avoidance is implemented in Cartesian space with the framework proposed in [116, 117]. First, a potential function for

each obstacle is defined:

$$U_S(\mathbf{x}) = \frac{A \exp(-\eta C(\mathbf{x}))}{C(\mathbf{x})}, \quad (5.14)$$

where $A, \eta \in \mathbb{R}_+$ are gain parameters. Functional $C : \mathbb{R}^d \rightarrow \mathbb{R}$ is an *isopotential* function that vanishes on the surface of the obstacle. In \mathbb{R}^3 it is defined as:

$$C(\mathbf{x}) = \left(\left(\frac{x_1}{f_1(\mathbf{x})} \right)^{2n} + \left(\frac{x_2}{f_2(\mathbf{x})} \right)^{2n} \right)^{\frac{2m}{2n}} + \left(\frac{x_3}{f_3(\mathbf{x})} \right)^{2m} - 1. \quad (5.15)$$

that vanishes on the surface of a *generalized ellipsoid*. Notice that any function C can be used as an isopotential function, as long as the following is satisfied:

- the boundary of the obstacle is the zero-level set of the isopotential;
- the value of C increases when the distance from the obstacle increases;

The choice of the specific formulation in (5.15) originates from the work on superquadric potential functions presented in [322], and it guarantees proofs of convergence and maximization of the available workspace for the robotic agent. In fact, by tuning parameters m, n and functions f_1, f_2, f_3 it is possible to model obstacles of any shape (their boundary will be the zero-level set of (5.15)). For the ring transfer task, pegs can be modeled as simple cylinders, hence parameters in (5.15) are set as $n = m = 1, p = 2$ and f_1, f_2, f_3 are constant functions with value representing the dimensions of pegs. Starting from the potential formulation in (5.14), a repulsive forcing term which guarantees obstacle avoidance is added to the dynamic system (5.8) as:

$$\boldsymbol{\phi}(\mathbf{x}, \mathbf{v}) = \boldsymbol{\phi}(\mathbf{x}) = -\nabla_{\mathbf{x}} U_S(\mathbf{x})$$

5.3 Experiments

5.3.1 Task planner validation

Validation of the ASP-based task planner is first performed, evaluating the plan computation time and the length of the generated plan in a number of representative scenarios for the ring transfer. These measures quantify the real-time performances of the task planner. Experiments are executed on a PC using 2.6 GHz Intel Core i7-6700HQ processor, mounting 4 cores and 8 threads, and 16 GB RAM.

1000 scenarios are generated (in simulation for convenience, to avoid burdening the robotic system and sensors) considering all possible combinations of four rings on the peg base, with the constraint that rings are placed on a peg at the beginning. In this way, all actions are included in the dataset of executions for proper validation of all axioms, and multiple workflows of execution are required to solve the task planning problem (e.g., moving to grey pegs, transfer between arms, single-arm execution). A maximum allowed time for plan computation by Clingo is set to 200 s. Three different ASP encodings of the ring transfer task are compared based on these metrics:

- **Encoding 1:** standard task description, obtained combining axioms 5.1-5.3-5.4;
- **Encoding 2:** task description with *durative environmental atoms* (i.e., all environmental fluents keep holding until specific events occur, e.g. a particular action is executed), obtained combining axioms 5.1-5.3-5.4-5.6;
- **Encoding 3:** task description with *optimization* based on distances between rings and PSMs. Specifically, the goal of this encoding is to compute a plan which places rings on pegs ranging them from the closest to the farthest (to any of the dVRK arms). This can be implemented in ASP and Clingo exploiting preference reasoning and optimization axioms as explained in Section 4.1.2. First, an `external atom distance(A, ring, C, Value)` is added to the base part of the program, with `Value` as a variable representing the distance between an arm and a colored ring. Then, in

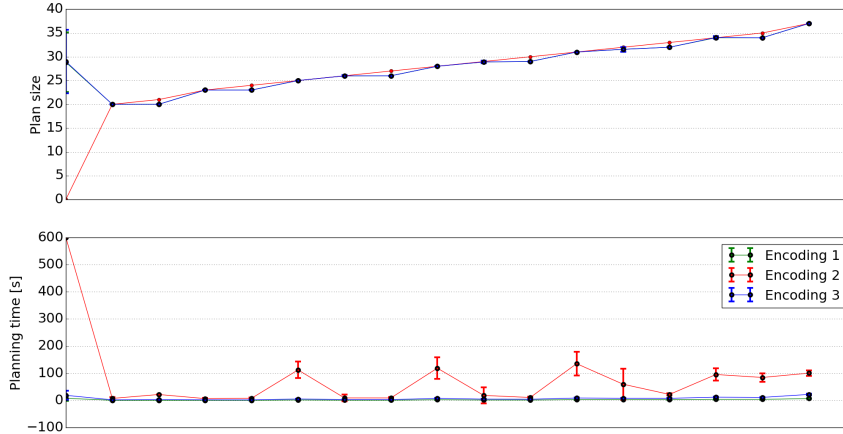


FIGURE 5.2: Comparison between different ASP encodings for the ring transfer task in the sequential execution for clusters of plans with the same length for Encoding 2 (horizontal axis).

the $\text{step}(t)$ an optimization axiom is introduced as follows:

$$\# \text{minimize}\{X: \text{move}(A, \text{ring}, C, t), \text{distance}(A, \text{ring}, C, X)\}. \quad (5.16)$$

The optimization axiom is added to the encoding represented by axioms [5.1-5.3-5.4](#).

The three encodings capture different challenges of the ring transfer task, as presented in [Chapter 2](#). In particular, Encoding 1 just describes task knowledge of the novel ring transfer definition considered in this thesis. Encoding 2 introduces the complexity of more temporal relations between atoms, which is relevant especially for the surgical domain with long lasting execution workflow and complex anatomies subject to continuous modification through interaction (effects of surgical actions). Finally, Encoding 3 captures the complexity of optimal / preference reasoning, linked to the ability of surgeons to take the best choice depending on the patient and the anatomical contingencies. All of the three encodings are evaluated for both sequential and parallel executions. [Figures 5.2 and 5.3](#) show the results for sequential and parallel execution, respectively. Plots

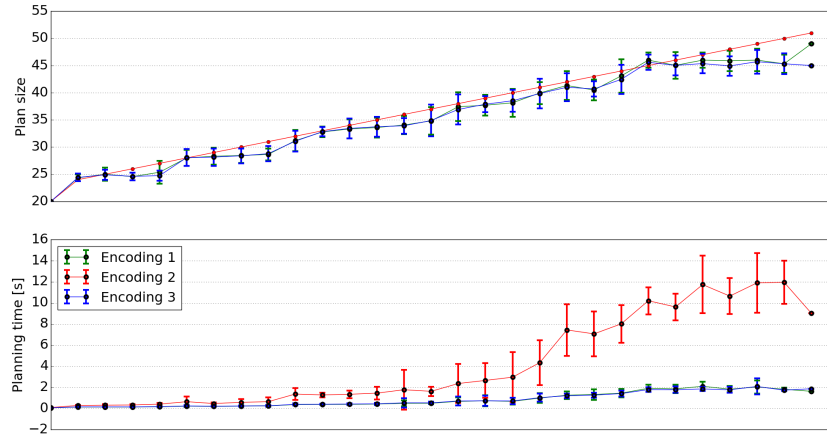


FIGURE 5.3: Comparison between different ASP encodings for the ring transfer task in the parallel execution for clusters of plans with the same length for Encoding 2 (horizontal axis).

are generated in the following way. First, collected executions from Encoding 2 (generating longer plans) are sorted according to increasing plan size, measured as the number of actions in the plan. In this way several clusters of plans of the same size from Encoding 2 are generated. For each cluster, plans from Encodings 1 and 3 are evaluated, matching plans with the same environmental fluents as Encoding 2; then, mean and standard deviation of the plan length (top of Figures 5.2-5.3) and the planning time (bottom of Figures 5.2-5.3) are computed for each cluster and encoding. Horizontal axes of Figures 5.2 and 5.3 report the clusters of plans as explained above.

On average, all encodings generate plans of similar length for sequential execution, while in the parallel execution plans generated by Encoding 2 are slightly longer, especially for final clusters corresponding to more complex environments (e.g., colored pegs occupied or more transfers needed), but still comparable.

As for the plan computation time, longer plans require higher times, due to the increased complexity of the environmental fluent representation. Notice that the first cluster (6 executions) for Encoding 2 in the sequential execution generates plans with null size, meaning that Clingo is not able to find a feasible plan within the maximum allowed time. The corresponding environmental fluents represent challenging scenarios. This cluster is represented with a planning time

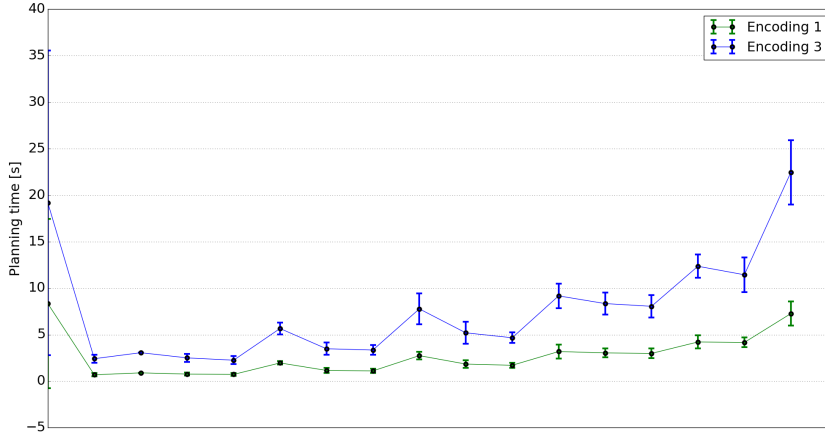


FIGURE 5.4: Comparison between planning times for Encodings 1 and 3 in sequential execution.

of 600 s^4 in the lower plot, which justifies the apparent initial decrease in the computational time.

Encoding 2 for the sequential execution takes significantly longer time for plan computation in most scenarios, with peaks above 100 s on average which limit the application in real time, especially for a critical scenario as surgery. This is probably due to the increased complexity in grounding (more atoms and axioms to be grounded), hence solving. Figure 5.4 highlights the time results with Encodings 1 and 3 for sequential execution. Encoding 1 generates plans in very fast times even for challenging scenarios (approximately 5 s for longest plans).

Encoding 3 with optimization requires longer computational times, but still acceptable for real tasks. Also in this case, the apparent initial decrease in the planning time is due to the first cluster, which corresponds to challenging environmental conditions where Encoding 2 fails.

The comparison between planning times of different encodings holds also for the parallel execution, though in this case the computation is significantly faster, with Encodings 1 and 3 able to generate plans in at most 2 s, and Encoding 1 in

⁴This value is chosen because it is higher than the maximum planning time for Encoding 2, so as not to impact on the meaning of the plot while guaranteeing better clarity.

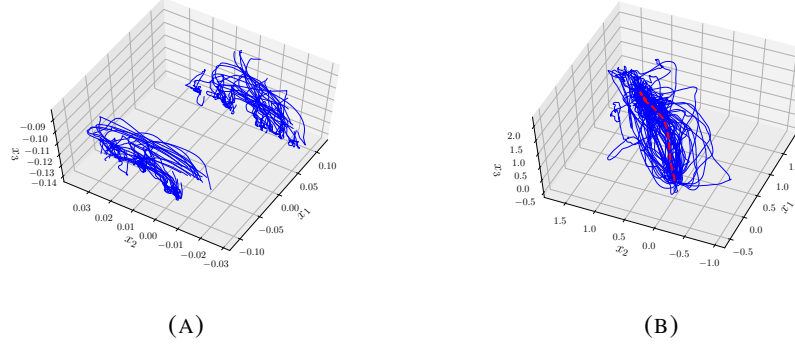


FIGURE 5.5: a) The set of Cartesian trajectories for the move (A, ring, C) gesture, for both PSMs; b) trajectories after roto-dilatation, to start at the origin, $\mathbf{x}_0 = \mathbf{0}$ and end at the vector of ones, $\mathbf{g} = \mathbf{1}$ for batch learning. The learned DMP is shown in red dashed line.

at most 10 seconds on average.

5.3.2 Framework validation

The validation of the overall framework presented in Section 5.2 is now performed with the real robotic setup. The dVRK is used to control the surgical robot. The DMPs for move actions are learned from multiple human executions using the approach presented in [113]. Three users with different dexterity perform five trials each of the task in tele-operation. The initial positions of rings and pegs and the order of the rings (red, green, blue, yellow) are the same for all executions. Rings are always transferred between the arms. This generates 120 executions of the move (A, ring, C) gesture (at the beginning and during transfer for each ring), 60 executions of the move (A, peg, C) and move (A, center, C) gestures. The learning process averages over all human trajectories. Figure 5.5 shows the learned Cartesian DMP for the move (A, ring, C) gesture as an example. The framework implements Encodings 1 and 3 in the task planning module, because of the higher computational efficiency for real-time purposes shown in former section. Three different scenarios are evaluated, shown in Figures 5.6-5.7-5.8. The scenarios are designed such that they represent unconventional situations in the ring transfer task, in order to prove the

versatility of ASP planning.

In scenario 5.6 the red ring is placed on a grey peg and must be extracted, while the blue ring requires transfer. In spite of the calibration accuracy, the small size of the setup and light conditions still originate vision errors. In this scenario, the ASP reasoner is also able to re-plan when the transfer of the blue ring fails. Moreover, Encoding 3 is exploited to perform optimization and take the closest ring (red) first.

In scenario 5.7, colored pegs are occupied, hence a ring must be temporarily placed on a grey peg. This scenario highlights the difference between a standard sequential planner or finite state machine and ASP: the logic program does not specify all possible workflows of execution, instead it describes the constraints of the task and the environment. Hence, though placing a ring on a grey peg is not a standard operation, it is still recognized by the ASP program (Encoding 1) as the best solution to reach the final goal. Moreover, in this scenario the SA module identifies the green ring as already placed, hence the reasoner ignores it.

Finally, in scenario 5.8 the parallel execution of the two arms with Encoding 1 is shown.

Table 5.2 shows the task planning times for the tested scenarios and re-planning. The results confirm the real-time capabilities of the ASP module proved in the former Section. The planning time reduces as the number of rings in the scene is lower (smaller grounding and shorter plan). Optimization in Encoding 3 increases the planning time, though it is still acceptable for real-time computation.

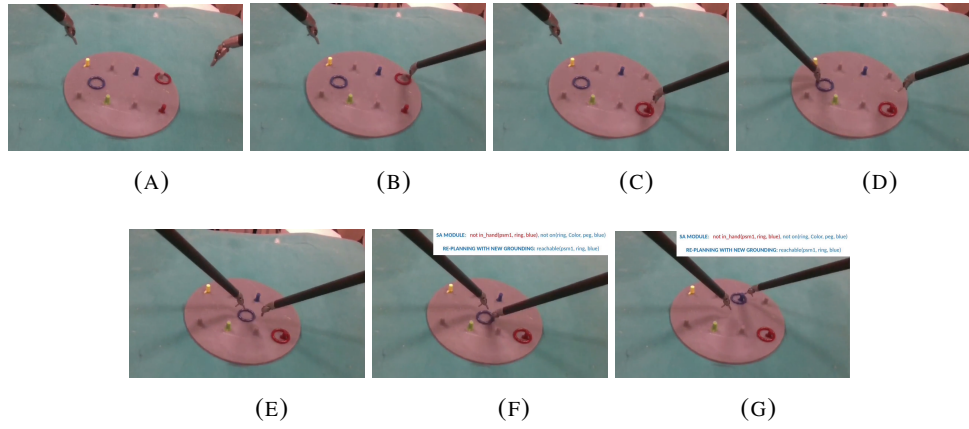


FIGURE 5.6: Real robotic scenario with failed transfer and re-planning (F).

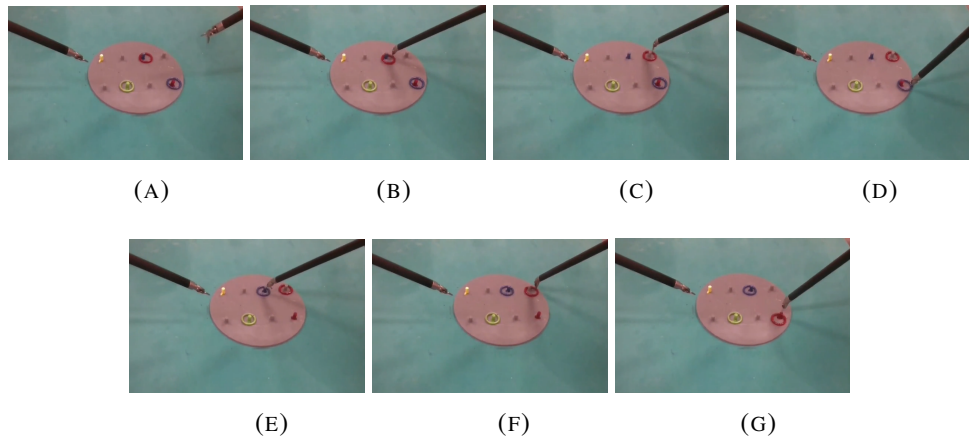


FIGURE 5.7: Real robotic scenario with occupied colored pegs.

TABLE 5.2: Planning time from the ASP module in the real robotic scenarios.

Scenario	Encoding	Planning time [s]
Figure 5.6 (re-planning)	3	0.42 (0.05)
Figure 5.7	1	0.13
Figure 5.8 (re-planning)	1	0.11 (0.05)

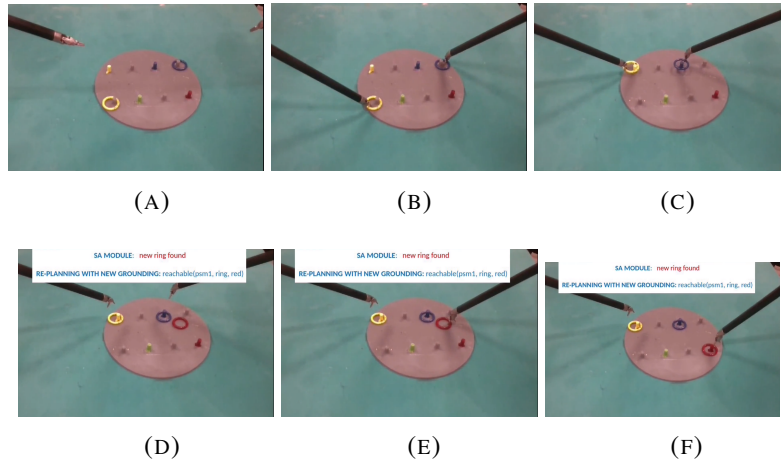


FIGURE 5.8: Real robotic scenario with parallel execution and a ring appearing in the scene, triggering re-planning (D).

5.4 Discussion

The proposed framework for autonomous surgical task execution addresses a number of issues in the surgical scenario.

First, adaptation to different environmental conditions (including failure conditions) is guaranteed both at task level with ASP, and at motion level with DMPs. This is important for most surgical operations, from standard suturing performed on wounds with different shape and with different anatomical surrounding and constraints, to complex operations as tumor removal from different organs.

Moreover, the framework exhibits interpretable, hence more reliable behavior, thanks to the high expressivity of task logic at ASP level, and the continuous monitoring of relevant environmental variables from sensors, with situation awareness module.

The ring transfer task used for validation allows to demonstrate that the proposed framework can efficiently control the operation of a surgical robot as dVRK in constrained dexterous manipulation tasks in small environment, thanks to the use of DMPs to replicate the manual capabilities of expert surgeons, one key skill towards efficient and safe surgery.

An important feature for an autonomous surgical system is also the real-time performance. The use of logics may slow down the reasoning process in complex scenarios, where task description involves a significant computational burden for grounding. This holds also for efficient ASP paradigm, as explained in Chapter 4.

In order to investigate this aspect, 1000 scenarios of the ring transfer task have been tested in simulation, evaluating the solving performance of state-of-the-art ASP solver Clingo when considering ASP encodings of increasing complexity. Performance is evaluated using a computer with standard commercial hardware specifications.

When considering standard ASP description and optimization (Encodings 1 and 3), the solving time is below 10 s even for very complex scenarios, including occupied pegs (hence requiring multiple extractions) and transfers. However, when temporal persistence of environmental fluents is assumed in Encoding 2 (i.e., all fluents hold until specific events or actions occur), the grounding process requires more time (more than 100 s). Considering temporal persistence of environmental fluents is realistic in robotic scenarios in general, and is very important even for the specific ring transfer task, since several fluents must hold in order to guarantee proper workflow of execution (e.g., status of gripper). This becomes even more crucial for more complex and realistic surgical tasks, where anatomical conditions must be stored and considered during the whole (possibly long) procedure.

It is important to compare the real-time (task) planning capabilities of the framework with the performance of a surgeon / user of dVRK, in order to further assess the benefits of autonomy. Some preliminary tests in this direction have been performed.

In particular, the ring transfer task in standard environmental conditions and in the same situation as Figures 5.6-5.8 (including failure conditions) is executed by a human operator with experience in using the dVRK robot and its remote control console, though not a certified surgeon. The setup is prepared while the human operator is not observing, and recording of the kinematic readings from dVRK and ROS topics is started as soon as the user is ready sitting at the console. Then, the human planning time is considered as the initial delay

between the beginning of the recording and the actual significant variation of any of the kinematic readings. The ASP solver of the autonomous framework appears able to generate plans faster than human in unconventional conditions (0.13 s vs. 1.56 s for Figure 5.7, 0.11 s vs. 0.83 s for Figure 5.8) and when failures occur (0.05 s vs. 0.37 s for failed grasp in Figure 5.6, 0.05 s vs. 0.33 s for new ring introduced in Figure 5.8), while the human performance is better in standard environmental conditions (1.36 s vs. 0.90 s). This is reasonable, since a human operator usually trains on the standard version of the ring transfer task.

Though more tests need to be performed to validate the real-time capabilities of the reasoning system, including a more rigorous evaluation of the planning time of a human, preliminary results are encouraging.

5.5 Conclusion

Combining task-level ASP and motion-level DMPs, integrated with situation awareness for sensor interpretation, has allowed to develop a framework which addresses important issues towards surgical task autonomy, and successfully solves the benchmark task of ring transfer.

Two main issues remain to be addressed. First, task knowledge is usually not priorly known, hence it must be acquired from experience and example executions from experts. In fact, the same surgical task is executed differently depending on the anatomy of the patient and his needs.

Secondly, computational efficiency of ASP planning must be improved to cope with temporal specifications, e.g. on persistence of fluents.

In the next chapter, these two problems will be addressed with inductive logic programming.

Chapter 6

Inductive learning of surgical task knowledge

6.1 Introduction

An important limitation of the framework presented in Chapter 5 is that it assumes comprehensive knowledge of the domain and the task in terms of domain attributes (e.g., object properties) and axioms governing domain dynamics (e.g., constraints, and action pre-conditions and effects). In practical robotics domains, especially in surgical scenarios, this is not feasible. In fact, the peculiarity of each patient's anatomy and the high number of anomalies which can occur during an intervention introduce high variability in the surgical workflow. Hence, it is not reliable to develop an autonomous surgical system which operates only on the basis of prior knowledge encoded by a human designer, or automatically extracted from textbooks of standard surgery (following the popular approach of text mining in artificial intelligence [229, 144, 24]).

This chapter is focused on the problem of learning previously unknown task-level knowledge from observations of real surgical task executions. The surgical context presents two major challenges to this purpose. First, usually very few records of executions are available for real surgeries, because of the task complexity, duration and privacy issues for the patients and the hospitals. Secondly, iterative learning is needed to refine existing knowledge as new observations are acquired, in order to develop a surgical system which is able to improve after every instance of a task. The proposed solution to the learning problem is based on inductive logic programming (ILP), a framework founded on logic induction

from examples expressed in a specific logic programming paradigm. The ILP approach presented in this chapter is applied to the three ASP encodings for the ring transfer task presented in Chapter 5. This chapter is organized as follows: Section 6.2 reviews the state of the art in surgical process learning, as well as general methods for knowledge acquisition and refinement from the artificial intelligence community; Section 6.3 briefly describes the general formalism of ILP, and specifies the ILP task under the AS semantics; Section 6.4 presents an alternative yet equivalent formulation for the ASP encoding proposed in Chapter 5, based on the temporal logic paradigm of event calculus, which is needed to solve the ILP problem more efficiently. Experiments which validate the learning framework on the ring transfer task are then presented, proving the advantages of the ILP framework and its feasibility even for more realistic surgical scenarios.

6.2 Related work

Learning a surgical process model (SPM) requires to choose the level of granularity at which the task will be analyzed, according to the standard definition proposed in [178] and shown in Figure 1.2. Learning an SPM for motions is challenging since surgical gestures present high variability [244]. Hence, statistical methods such as Markov models and Gaussian mixture models (GMMs) [204, 307, 178] are typically used to infer a motion-level SPM. With the increasing use of deep neural networks, GMMs have been combined with deep reinforcement learning to transfer learned motions to different surgical tools [46]. Although deep learning provides a small improvement in accuracy over traditional learning methods, it is computationally expensive to tune the network parameters [71, 87]. A further limitation of deep networks and related methods is that they need many (labeled) examples to build a good SPM. This chapter focuses on learning SPMs at a coarser granularity, i.e. at the level of relations between activities (or actions) that constitute a surgical step or phase. An action is an elementary motion associated with semantics specifying, e.g., the arm and the surgical tool at use. The sequence of actions is affected by the variations in the anatomical conditions. Bayesian networks (BNs) represent the state of the art for learning SPMs at this granularity [31, 44]. Recurrent (deep) neural networks have

also been explored, exhibiting improvement in the accuracy at the expense of increased computational effort for training [65]. A hidden Markov model is used to model a ring transfer exercise involving cooperation between a human and a robot [25]. However, though the authors consider a simplified version of the task, 80 labeled human executions are required as a training set for learning, making scalability to more complex tasks a challenging problem. Another major limitation of statistical methods is that they generate black-box models that do not provide any guarantees in terms of correctness and soundness, affecting the reliability of the surgical system. On the contrary, logic-based formalisms for representing and reasoning with domain knowledge inherently provide correctness guarantees [244], and they make the underlying reasoning easier to understand. However, such logic-based formalisms for the ring transfer task have required comprehensive domain knowledge to be encoded a priori [115, 134], which is rather difficult to do in more complex surgical scenarios.

There is an established literature in AI on methods for learning domain knowledge. Examples include the incremental revision of a first-order logic representation of action operators [112], the expansion of the theory of actions to revise or inductively learn ASP system descriptions [15], and the combination of non-monotonic logical reasoning, inductive learning, and relational reinforcement learning to incrementally acquire previously unknown actions and their pre-conditions and effects [302]. Previously unknown state constraints have also been learned using decision tree induction in a framework that combines ASP-based non-monotonic logical reasoning with deep learning for scene understanding [230]. All these approaches may be viewed as instances of interactive task learning, a general framework for acquiring domain knowledge using labeled examples or reinforcement signals obtained from domain observations, demonstrations, or human instructions [176]. Learning unknown domain axioms representing action-level knowledge is performed in this thesis using ILP, first developed in [233]. The choice rises from the requirements expressed at the beginning of this chapter. First, ILP supports learning from labeled examples and incremental learning from a given background knowledge. For this reason it has been used by an international research community in different domains, e.g., to identify cognitive stress and distraction in a driver [221]; for event recognition in city

transport [151]; and to learn logic programs in robotics [56]. ILP has also been successfully applied to the learning of programs based on the paradigm of event calculus, firstly in [232]. Defining event calculus specifications is a non-trivial task, which has limited the application of this temporal logic formalism [232]. Hence, research has been devoted to the automatic learning of event calculus definitions, e.g. with incremental and scalable approaches [6, 153]. Moreover, ILP has been used to learn in non-monotonic logic programs [183] and probabilistic logic programs [63]. In complex application domains such as surgical robotics, learning with probabilistic logics is computationally challenging [246], but non-monotonic logical reasoning is still necessary. The approach presented in this chapter builds on ILASP, an implementation of ILP under the AS semantics [180]. Unlike another popular implementation called Inspire [289], ILASP has fewer hyper-parameters and supports learning from a batch of examples, which significantly speeds up the learning process. A drawback of ILASP is that it does not natively support automatic predicate invention, which helps grouping logic variables and generating shorter rules, while Inspire does. However, here presented experiments show that this limitation can be partially overcome by using an iterative version of the algorithm of ILASP. In [180], ILASP has been shown to be more general than XHAIL [271], a state-of-the-art tool for inductive learning of event calculus-based axioms, and its competitor ILED [152]. Moreover, it guarantees the optimality of the set of axioms in terms of minimal length (see Section 6.3 for details).

6.3 The ILP problem under the AS semantics

6.3.1 Task definition

Several formulations of ILP tasks have been proposed and investigated by the research community, including learning from satisfiability [62], learning from entailment [234, 272] and learning from interpretations [64]. All ILP tasks define some *background knowledge* B expressed in a logic formalism \mathcal{F} , a *search space* S_M of possible axioms to be learned in the syntax of \mathcal{F} , and a set of *examples* E expressed in the syntax of \mathcal{F} . The goal of any ILP task is to find a subset of

the search space, $H \subseteq S_M$, such that $H \cup B$ supports the evidence contained in the examples. The ILP problem under the AS semantics is a specific instance of learning from entailment. The learning from entailment task is defined as follows:

Definition 4 (Learning from entailment). *Consider a set of examples $E = \langle E^+, E^- \rangle$, where E^+ is a subset of positive examples and E^- is a subset of negative examples. The task of learning from entailment is defined as the tuple $T = \langle B, S_M, E \rangle$. The goal of T is to find $H \subseteq S_M$ such that:*

$$\begin{aligned} B \cup H &\models E^+ \\ B \cup H \cup E^- &\models \perp \end{aligned}$$

The above definition introduces the definitions of positive and negative examples. Intuitively, a positive example must be *covered* by $B \cup H$, i.e. positive examples must be entailed from the knowledge encoded in the logic program. On the contrary, negative examples typically represent undesired observations and task situations which must not be covered by the final knowledge. The above generic definition is specialized to the AS semantics expressing B and S_M in the ASP language as presented in Chapter 4, and considering examples which are *partial interpretations*.

Definition 5 (Partial interpretation). *Let P be an ASP program. Any set of ground atoms that can be generated from axioms in P is an interpretation of P . Given an interpretation I of P , a pair of subsets of ground atoms $e = \langle e^{inc}, e^{exc} \rangle$ is said to be a partial interpretation extended by interpretation I if $e^{inc} \subseteq I$ and $e^{exc} \cap I = \emptyset$.*

In other words, a positive example is a partial interpretation of an ASP program specifying atoms which may appear in an answer set of the program (e^{inc}), and atoms which cannot appear (e^{exc}). Conversely, a negative example is a partial interpretation of an ASP program specifying atoms which do not appear in an answer set of the program (e^{inc}), and atoms which shall appear (e^{exc}).

Given these definitions of examples, two different variants of the ILP task are defined under the AS semantics: *brave induction* and *cautious induction* tasks, first formalized in [278].

Definition 6 (Brave induction task). Let $\mathcal{T} = \langle B, S_M, E \rangle$, H , and $E = \langle E^+, E^- \rangle$ be as defined for learning from entailment under the AS semantics. Let A be an answer set of the ASP program defined by $H \cup B$. \mathcal{T} is said to define a brave induction task if the goal set H of hypotheses must satisfy:

$$\exists A \text{ s.t. } B \cup H \models A : A \cap E^- = \emptyset \wedge E^+ \subseteq A$$

Definition 7 (Cautious induction task). Let $\mathcal{T} = \langle B, S_M, E \rangle$, H , and $E = \langle E^+, E^- \rangle$ be as defined for learning from entailment under the AS semantics. Let A be an answer set of ASP program defined by $H \cup B$. \mathcal{T} is said to define a cautious induction task if the goal set H of hypotheses must satisfy:

$$\forall A \text{ s.t. } B \cup H \models A : A \cap E^- = \emptyset \wedge E^+ \subseteq A$$

In other words, the hypotheses H selected by a brave induction task must entail *at least* one answer set that satisfies the examples, and the H returned by a cautious induction task must entail *only* answer sets that satisfy the examples. For any set of examples, hypotheses returned by a cautious induction task are stricter than those returned by a brave induction task since it must satisfy additional constraints.

As stated at the beginning of this chapter, example executions (which generate observed answer sets in E^+) of surgical tasks present only few possible occurrences, hence the ILP task must be able to generate an hypothesis which covers positive examples, but does not exclude different (unavailable at the moment) workflows. At the same time, a surgical task involves a number of constraints which must be safely guaranteed for the overall reliability of the surgical system. The objective is to provide these constraints in terms of negative examples, hence negative examples must be excluded from any answer set of the learned ASP program. For this reason, the framework for ILP under the AS semantics adopted in this chapter is founded on the state-of-the-art tool ILASP [180], based on Clingo syntax, which combines brave induction for positive examples with cautious induction for negative examples as follows:

Definition 8 (ILASP task). Let $\mathcal{T} = \langle B, S_M, E \rangle$, H , and $E = \langle E^+, E^- \rangle$ be defined as for learning from entailment under the AS semantics. Let A be an

answer set from the ASP program defined by $H \cup B$. The goal of the ILASP task is to find H such that:

$$\begin{aligned} \forall e \in E^+ \exists A \text{ s.t. } B \cup H \models A : e \subseteq A \\ \forall e \in E^- \nexists A \text{ s.t. } B \cup H \models A : e \subseteq A \end{aligned}$$

ILASP implements a number of algorithms for solving the ILP task. It is out of the scope of this chapter to describe the full details of ILASP algorithms, which can be found in [180]. However, a short intuitive description of the main algorithm, ILASP2, is here presented to illustrate the main advantages of the tool and justify its choice for the purposes of this thesis. ILASP2 represents the ILP task as a meta-level ASP program to be solved with standard ASP solvers as Clingo. The meta-level program contains axioms which represent the background knowledge B , and its answer sets are the axioms in the final hypothesis H . The search space S_M is represented as a set of constraints on the final answer sets. Examples are added to the meta-level ASP program using *reification*, i.e. they are converted to constraints which limit the set of feasible hypotheses in the search space. ILASP2 iteratively solves the ASP program collecting all violating reasons, i.e. hypotheses which do not satisfy examples. The solving process stops as soon as an answer set (H for the ILP task) is found including none of the violating reasons. Notice ILASP offers the advantage to find not only a supporting hypothesis for the examples, but the *shortest* one. In fact, the search in S_M is performed starting from the axioms with minimal length. The length of an axiom is defined as follows:

Definition 9 (Length of an axiom). *Let \mathcal{A} be an axiom in an ASP program. The length of \mathcal{A} , $|\mathcal{A}|$, is defined as the number of atoms that appear in its body and head. For an aggregate rule, i.e. an axiom with an aggregate $l \{a_1; a_2; \dots; a_n\} u$ in the head, the length of the head is defined as $\sum_{i=1}^n i \cdot n$.*

The length of an hypothesis is directly defined as the sum of the length of the axioms composing it. In this chapter, a more efficient variant of ILASP2 algorithm is used, ILASP2i [181]. ILASP2i is an incremental version of ILASP2, meaning that it does not consider all examples as constraints in the meta-level ASP program. On the contrary, the algorithm first finds H to satisfy the first

example. Then, at each iteration only examples which are not supported by H are reified. In this way, the meta-level ASP program is not burdened by a high number of constraints corresponding to all examples, hence ILASP2i is computationally more efficient when many examples are provided. Another important advantage of ILASP is that it allows to define the search space S_M with compact syntax, using *mode bias* that specifies the atoms that can occur in the body and head of axioms. For the ring transfer task, only atoms identifying actions and fluents can be part of the heads of candidate rules; the body of axioms can have both actions and domain attributes.

In the rest of this chapter, a more specific ILP task than the one defined in Definition 8 is considered, which is *learning from context-dependent examples* [182]. The difference is that examples are not only observed or unwanted partial interpretations, but they also include the *context* C , i.e. the specific constraints / atoms which hold for the example. In this way, examples become *context-dependent partial interpretations* (CDPI), defined as tuples $\langle e, C \rangle$, and the task in Definition 8 is re-formulated as follows:

Definition 10 (ILASP task with CDPIs). *Let $\mathcal{T} = \langle B, S_M, E \rangle$, H , and $E = \langle E^+, E^- \rangle$ be defined as for learning from entailment under the AS semantics, with E as CDPIs. Let $AS(K)$ define the answer sets of an ASP program K . The goal of the ILASP task with CDPIs is to find H such that:*

$$\begin{aligned} \forall e \in \langle E^+, C \rangle \exists A \in AS(B \cup H \cup C) \text{ s.t. } B \cup H \cup C \models A : e \subseteq A \\ \forall e \in \langle E^-, C \rangle \nexists A \in AS(B \cup H \cup C) \text{ s.t. } B \cup H \cup C \models A : e \subseteq A \end{aligned}$$

Using CDPIs allows to relate different plans of the task to different contextual environmental conditions, hence capturing the variability of the workflow without defining a shared background knowledge which is logically consistent with all examples.

6.3.2 Complexity and generality

[183] has extensively studied the computational complexity and the generality of ILP frameworks under the AS semantics. In this section, results which are relevant to the specific task of Definition 10 of interest in this thesis are reported.

The computational complexity is evaluated for two problems: *verification* and *satisfiability*.

Definition 11 (Verification and satisfiability problems). *Let $\mathcal{T} = \langle B, S_M, E \rangle$ be a generic ILP task, e.g. as the one defined in Definition 4. Verification is the problem of deciding whether hypothesis H is a solution of T . Satisfiability is the problem of deciding whether at least such hypothesis exists.*

The following result holds (demonstration can be found in [183]):

Theorem 1. *The complexity of the verification problem for the ILP tasks in Definitions 8-10 is DP-complete. The complexity of the satisfiability problems for the ILP tasks in Definitions 8-10 is Σ_2^p -complete.*

Notice that these results are originated from the results for the cautious induction task described above, which intuitively is more complex than brave induction task, since ILP tasks in Definitions 8-10 use cautious induction on negative examples. In fact, [183] proves that the cautious induction task can be polynomially reduced to the ILP tasks in Definitions 8-10.

The generality of an ILP framework is a measure of the possible axioms which can be learned, given a sufficient number and variety of examples. The generality of an ILP framework is then affected by the definition of the search space S_M (e.g., through mode bias). For this reason, the generality of an ILP task shall be evaluated assuming no restriction on S_M , i.e. $S_M = A^{\mathcal{F}}$, where $A^{\mathcal{F}}$ is the set of all possible axioms in the syntax of the specific logic formalism \mathcal{F} underlying the ILP task (in this case, ASP). Hence, from now on the generic ILP task \mathcal{T} of previous definitions is denoted as $\mathcal{T} = \langle B, E \rangle$, meaning that $S_M = A^{\mathcal{F}}$. An important definition for studying generality is *strong reduction* of one ILP framework to another:

Definition 12 (Strong reduction). *Consider two ILP tasks $\mathcal{T}_1 = \langle B, E_1 \rangle$, $\mathcal{T}_2 = \langle B, E_2 \rangle$. \mathcal{T}_1 is said to strongly reduce to \mathcal{T}_2 (denoted as $\mathcal{T}_1 \rightarrow_{sr} \mathcal{T}_2$) if, for every instance of \mathcal{T}_1 there exists an instance of \mathcal{T}_2 such that $H_1 = H_2$, given that H_1 is a valid hypothesis for \mathcal{T}_1 and H_2 is a valid hypothesis for \mathcal{T}_2 .*

In other words, $\mathcal{T}_1 \rightarrow_{sr} \mathcal{T}_2$ means that any instance of \mathcal{T}_1 maps to an instance of \mathcal{T}_2 , given that the *same background knowledge is provided for both tasks*.

Hence, it is possible to define a notion of generality as derived from the strong reduction relation, which is *sr-generality*. \mathcal{T}_1 is said to be as sr-general as \mathcal{T}_2 if $\mathcal{T}_2 \rightarrow_{sr} \mathcal{T}_1$; similarly, \mathcal{T}_1 is said to be more sr-general than \mathcal{T}_2 if $\mathcal{T}_2 \rightarrow_{sr} \mathcal{T}_1$, but $\mathcal{T}_1 \not\rightarrow_{sr} \mathcal{T}_2$. The following important theorem holds:

Theorem 2. *The ILASP task with CDPIs in Definition 10 is more sr-general than the ILASP task in Definition 8, which is more sr-general than the brave induction task. At the same time, the ILASP task in Definition 8 is more sr-general than the cautious induction task.*

A good measure of generality of an ILP task is the *one-to-many distinguishability class*, which evaluates the capability of one task \mathcal{T} to distinguish between a target hypothesis H_T and a set of unwanted hypotheses S . In fact, simply stating that H_T can be found by \mathcal{T} does not necessarily mean that \mathcal{T} is general enough to find H_T . A clarifying example is $\mathcal{T} = \langle B, E \rangle$ with $E = \emptyset$: the empty hypothesis is obviously satisfying for \mathcal{T} , whatever ILP task is considered. The formal definition of one-to-many distinguishability is as follows:

Definition 13 (One-to-many distinguishability). *The one-to-many distinguishability class of an ILP task $\mathcal{T} = \langle B, E \rangle$ (denoted as $\mathcal{D}_m^1(\mathcal{T})$) is the set of all tuples $\langle B, H, S \rangle$ such that there exists an instance of \mathcal{T} which can distinguish between H and any hypothesis contained in a set of hypotheses S , given the same background knowledge B . Given two ILP tasks $\mathcal{T}_1, \mathcal{T}_2$, \mathcal{T}_1 is said to be at least as \mathcal{D}_m^1 -general as \mathcal{T}_2 if $\mathcal{D}_m^1(\mathcal{T}_2) \subseteq \mathcal{D}_m^1(\mathcal{T}_1)$.*

It is possible to establish the following relation between strong reduction and one-to-many distinguishability:

Theorem 3. *Given two ILP tasks \mathcal{T}_1 and \mathcal{T}_2 , $\mathcal{T}_1 \rightarrow_{sr} \mathcal{T}_2 \implies \mathcal{D}_m^1(\mathcal{T}_1) \subseteq \mathcal{D}_m^1(\mathcal{T}_2)$.*

This result, together with Theorem 2, leads to the conclusion that the one-to-many distinguishability class for the ILASP task in Definition 10 is larger than

the other ILASP tasks, hence it is the most general framework for the purposes of learning task knowledge in ASP syntax¹.

6.4 ILP for the ring transfer task

In this section, the experimental setup and the results of evaluating the capabilities of ILASP in the context of the ring transfer task are presented. The main objective was to learn previously unknown axioms describing actions' preconditions (e.g., Statement 5.1) and effects (e.g., Statement 5.4), and executability conditions (Statement 5.3). In order to restrict the search space and improve the computational efficiency, separate ILASP tasks for each action were defined to learn the different types of axioms. Also, separate ILASP tasks were defined for each domain fluent, one each for the `initiated` and the `terminated` conditions respectively.

The background knowledge and the search space for ILASP tasks are then defined (Section 6.4.2), and the generation of training examples is explained (Section 6.4.3). Finally, the results of comparing the learned axioms with the ground truth information provided by the designer are discussed (Section ??). In the first experiment, the length of axioms and the computational time required by ILASP are used as the evaluation measures, under the hypothesis that the learned axioms would closely match the ground truth information. In the next experiment, considered 1000 simulated scenarios are considered that mimic challenging conditions for the ring transfer task, including both sequential and parallel execution of actions as described in Chapter 5. In each scenario, paired trials are conducted with the learned and ground truth axioms respectively. In these trials, planning time and plan length were the evaluation measures (with plans computed using the Clingo ASP solver).

¹In [183], results of complexity and generality are explicitly reported for the task in Definition 10 with ordered examples, \mathcal{T}_{LOAS}^c , i.e. when the target is to learn weak ASP constraints. However, it is straightforward to apply the proofs of theorems in [183] (in particular the equivalence of complexity class between \mathcal{T}_{LOAS}^c and the task in Definition 8) and show that the same results for complexity hold for the task in Definition 10. For generality, Definition 10 is obviously less general without ordered examples; however, learning weak constraints is out of the scope of this thesis, hence Definition 10 ensures the highest needed generality.

Before presenting the ILASP approach, a slight modification in the syntax of task knowledge for ring transfer is required for computational efficiency, as described in the following section.

6.4.1 Re-formulation of the original ASP encoding

The ASP formulation of effects of actions for the ring transfer task presented in Section 5.2 is challenging for the ILP task from CDPIs of Definition 10. In fact, as explained in Section 6.3, ILASP2 (and ILASP2i) algorithm solves a meta-level ASP program, with constraints obtained from reification of examples and the definition of the search space S_M . The ASP formulation of durative effects includes a high number of NAFs, since the concept of duration of atoms is implicit in the axioms, meaning that they specify that *an atom holds until some other atom does not hold*. Axioms with NAFs are more challenging for ILASP, since a NAF represents unobserved entities of the domain, hence the S_M is usually larger when NAFs must be included in it. For this reason, here a more convenient, yet equivalent formulation for durative effects of actions is presented, based on the temporal logic paradigm of *event calculus*. which specifies the starting (*initiating*) and *terminating* conditions for each fluent, based on event calculus [166, 148]. The event calculus is a general approach to representing and reasoning about events and their effects in a logic programming framework, born as an attempt to solve the frame problem [166]. An event calculus program relates the properties of a domain to triggering events, following the common sense *law of inertia* [291]. This law formalises the notion that properties of a domain remain unchanged throughout time unless specific events occur. Hence, in ASP syntax, events (specific sets of atoms) which trigger the starting and the ending of a specific atom (*initiating* and *terminating* conditions, respectively) are explicitly and separately defined, thus reducing the use of NAFs in the axioms. Given a generic `fluent` subject to the law of inertia, the core ASP axioms of the event calculus are as follows:

$$\begin{aligned}
 \text{occurs}(\text{fluent}, t) & :- \text{initiated}(\text{fluent}, t). & (6.1) \\
 \text{occurs}(\text{fluent}, t) & :- \text{occurs}(\text{fluent}, t-1), \\
 & \text{not terminated}(\text{fluent}, t).
 \end{aligned}$$

These are equivalent to the following axioms in temporal logic:

$$\begin{aligned} & \text{not occurs}(\text{fluent}) \mathcal{U} \text{initiated}(\text{fluent}) \\ & \text{terminated}(\text{fluent}) \mathcal{R} \text{occurs}(\text{fluent}) \end{aligned}$$

where \mathcal{U}, \mathcal{R} are the fundamental temporal operators *until*, *release* respectively [80]. In the ASP program of the ring transfer task, the inertial property 6.1 and initiating / terminating conditions for effects are defined as follows:

$$\text{initiated}(\text{at}(A, \text{peg}, C), t) :- \text{move}(A, \text{peg}, C, t-1). \quad (6.2)$$

$$\begin{aligned} \text{initiated}(\text{at}(A, \text{center}, C), t) :- \text{move}(A, \text{center}, C, \\ t-1). \end{aligned}$$

$$\begin{aligned} \text{initiated}(\text{in_hand}(A, \text{ring}, C), t) :- \text{grasp}(A, \text{ring}, C, \\ t-1). \end{aligned}$$

$$\begin{aligned} \text{initiated}(\text{closed_gripper}(A), t) :- \text{grasp}(A, \text{ring}, _, \\ t-1). \end{aligned}$$

$$\begin{aligned} \text{initiated}(\text{on}(\text{ring}, C1, \text{peg}, C2), t) :- \text{in_hand}(A, \text{ring}, \\ C1, t-1), \text{at}(A, \text{peg}, C2, t-1), \text{release}(A, t-1). \end{aligned}$$

$$\text{initiated}(\text{at}(A, \text{ring}, C), t) :- \text{move}(A, \text{ring}, C, t-1).$$

$$\begin{aligned} \text{terminated}(\text{in_hand}(A, \text{ring}, C), t) :- \text{release}(A, t-1), \\ \text{in_hand}(A, \text{ring}, C, t-1). \end{aligned} \quad (6.3a)$$

$$\begin{aligned} \text{terminated}(\text{closed_gripper}(A), t) :- \text{release}(A, t-1). \\ \end{aligned} \quad (6.3b)$$

$$\begin{aligned} \text{terminated}(\text{on}(\text{ring}, C1, \text{peg}, C2), t) :- \text{extract}(_, \text{ring}, \\ C1, t-1), C1 \neq C2. \end{aligned} \quad (6.3c)$$

$$\begin{aligned} \text{terminated}(\text{at}(A, \text{ring}, C), t) :- \text{move}(A, \text{ring}, C1, t-1), \\ \text{color}(C), C1 \neq C. \end{aligned} \quad (6.3d)$$

$$\begin{aligned} \text{terminated}(\text{at}(A, \text{ring}, C), t) :- \text{move}(A, \text{peg}, _, t-1), \\ \text{color}(C). \end{aligned} \quad (6.3e)$$

$$\text{terminated}(\text{at}(A, \text{ring}, C), t) :- \text{move}(A, \text{center}, _, t-1),$$

$$\text{color}(C). \quad (6.3f)$$

$$\begin{aligned} \text{terminated}(\text{at}(A, \text{peg}, C), t) &:- \text{move}(A, \text{peg}, C1, t-1), \\ &\text{color}(C), C \neq C1. \end{aligned} \quad (6.3g)$$

$$\begin{aligned} \text{terminated}(\text{at}(A, \text{peg}, C), t) &:- \text{move}(A, \text{ring}, _, t-1), \\ &\text{color}(C). \end{aligned} \quad (6.3h)$$

$$\begin{aligned} \text{terminated}(\text{at}(A, \text{peg}, C), t) &:- \text{move}(A, \text{center}, _, t-1), \\ &\text{color}(C). \end{aligned} \quad (6.3i)$$

$$\text{terminated}(\text{at}(A, \text{center}), t) :- \text{move}(A, O, _, t-1). \quad (6.3j)$$

Notice that these axioms are complete when parallel execution is considered. In the case of sequential execution, the $\text{move}(A, \text{center}, C, t)$ action implicitly moves both arms. Hence, two additional axioms need to be encoded:

$$\begin{aligned} \text{initiated}(\text{at}(A1, \text{ring}, C), t) &:- A2 \neq A1, \text{arm}(A1), \\ &\text{move}(A2, \text{center}, C, t-1). \\ \text{terminated}(\text{at}(A1, \text{center}), t) &:- A2 \neq A1, \text{arm}(A1), \\ &\text{move}(A2, \text{center}, _, t-1). \end{aligned}$$

6.4.2 Background knowledge and search space

In this thesis, action pre-conditions and executability conditions are learned in one set, and the action effects in another set. Below, the initial ILASP setup for these two sets of axioms is described.

Pre-conditions and executability conditions

For the experiment that focused on learning action pre-conditions and executability conditions, the background knowledge of each ILASP learning task (one per action) included the definitions of sorts and helper axioms describing the difference between two different arms or colors:

$$\text{different}(A1, A2) :- \text{arm}(A1), \text{arm}(A2), A1 \neq A2.$$

```
different(C1, C2) :- color(C1), color(C2), C1 != C2.
```

The search space for each ILASP task was defined using mode bias for compactness. Specifically, for the task of learning pre-conditions and executability conditions for any given `action`, the search space was defined such that the action can only occur in the head of an aggregate rule (to capture pre-conditions) or in the body of axioms (for executability conditions). In ILASP syntax, this corresponded to the statements `#modeha(action)` and `#modeb(1, action)`, respectively; `#modeb(1, action)` specifies that `action` can appear in the body of an axiom only once. Also, each environmental (i.e., domain) fluent presented in Chapter 5 may appear in the body of axioms, by adding the mode bias statement `#modeb(1, fluent)`. Similarly, the statement `#modeb(1, different)` was introduced. When defining the search space, arguments of atoms which are variables or constants must be clearly stated in ILASP. Axioms with more variables generally require more computational effort. For the task of learning pre-conditions and executability constraints, only `arm` and `color` were defined as variables in atoms. Finally, the length of the body of axioms is limited to three atoms using a specific ILASP flag from command line, to reduce the dimension of the problem.

Effects of actions

In order to learn the effects of action, we set up two ILASP learning tasks per environmental fluent, one each for the axioms associated with the `initiated` and `terminated` relations. The background knowledge for these learning tasks contained the same ASP statements presented in the previous section, and the laws of inertia (Statement 6.1). Moreover, since effects are delayed with respect to actions, the concept of temporal sequence was included:

```
delay(1..N).
prev(T1, T2, D) :- time(T1), time(T2), delay(D),
T2 = T1+D.
```

where `delay` is a variable constrained to the set $1..N$ and N is an estimate of the maximum delay between actions and effects in the domain; N can be increased until ILASP is able to find a suitable hypothesis with the minimum temporal delay. For the ring transfer task, ILASP found the minimum value of $N=1$. Then, the search space was defined using the mode bias `#modeh(Initiated(Fluent, t))` or `#modeh(Terminated(Fluent, t))`, which specified the head of candidate normal axioms. Moreover, for each environmental fluent f and each action $action$ of the task, `#modeb(1, ft)`, `#mode(1, actiont)` was stated to allow them in the body of candidate rules. The notation `atomt` is shorthand to specify that the time step appears as an argument of `atom`, from now on. Also `#modeb(1, prev)` was included in the mode bias. Note that the inertia laws (Statement 6.1) imply that `fluentt :- initiated(fluent, t)`, which would lead ILASP to learn the trivial axiom:

```
initiated(fluent, t) :- fluentt
```

As a result, in the ILASP task to learn `initiated` conditions for a specific fluent, the mode bias `#modeb(1, fluentt)` was omitted. ILASP variables included `color`, `arm`, and `time`, while `delay` was defined as a constant `#constant(delay, 1..N)` to reduce the size of the search space. The maximum body length of axioms was limited to three.

6.4.3 Generation of examples

The training and testing examples were extracted from videos of a human or the robot performing the target task. Specifically, a human execution of the ring transfer task in standard environmental conditions was considered from the dataset used for learning DMPs in Chapter 5. In addition, the executions of Figures 5.6-5.8 were added, to capture anomalous situations and learn full task knowledge.

From each video, fluents were extracted as explained in Chapter 5. Each frame in the videos was labeled with the recognized fluents and action being

executed. This process was repeated in all the videos to generate the set of labeled examples that serves as the input to our approach to learn previously unknown axioms corresponding to executability conditions (Statement 5.3), action pre-conditions (Statement 5.1), and action effects (Statement 5.4). The target axioms define logical relations between atoms describing actions and domain fluents; the corresponding examples will only contain these atoms. The structure of examples for these types of axioms is described next.

Action pre-conditions and executability conditions

Since all atoms in the axioms corresponding to the pre-conditions and executability conditions of actions refer to the same timestep t (see Chapter 5), the timestep in the literals were omitted to reduce the number of variables in the search space and speed up learning. For each timestep, the positive examples were defined as CDPIs of the form $\langle e^{inc}, e^{exc}, C \rangle$, where e^{inc} was the executed action, C contained the atoms of the fluents describing the environmental state, and $e^{exc} = \emptyset$. Also actions that could not occur at each timestep were specified, simulating knowledge from an expert designer analyzing the video under consideration. Then negative examples were defined with forbidden actions in e^{inc} and $e^{exc} = \emptyset$. Although it is possible to add forbidden actions in the set e^{exc} in the positive examples, the fact that ILASP learns through brave induction from positive examples implies there is no guarantee that actions in e^{exc} will always be excluded by the solution hypothesis. On the contrary, negative examples are cautiously entailed by adding executability conditions to the hypothesis set to ensure that the learned axioms are reliable. As an illustrative example, consider the scene in Figure 5.6. The first action moves PSM1 towards the red ring, providing a positive example:

```
#pos{ex1, {move(psm1, ring, red)}, {},
{reachable(psm1, ring, red), reachable(psm2, ring, blue),
reachable(psm1, peg, red), reachable(psm1, peg, blue),
reachable(psm2, peg, green), reachable(psm2, peg, yellow),
reachable(psm1, peg, grey), reachable(psm2, peg, grey),
```

```
on (R, red, peg, grey) } }
```

At the same time, it is not possible to move PSM1 to the blue ring, providing the negative example:

```
#neg{ex2, {move(psm1, ring, blue)}, {} ,
{reachable(psm1, ring, red), reachable(psm2, ring, blue),
reachable(psm1, peg, red), reachable(psm1, peg, blue),
reachable(psm2, peg, green), reachable(psm2, peg, yellow),
reachable(psm1, peg, grey), reachable(psm2, peg, grey),
on (R, red, peg, grey) } }
```

To reduce the complexity of the learning task, *redundant examples* are omitted, i.e., examples that only differ in the grounding of variables in the atoms. For example, in the scenario in Figure 5.8, both arms moved to a ring (PSM1 moved to blue ring, PSM2 moved to yellow one) at $t=1$. This generated two examples that differ not in context but only in the grounding of $\text{move}(A, R, C)$; only one example was added. Overall, 8 positive examples and 8 negative examples were generated for $\text{move}(A, P, C)$; 9 positive examples and 20 negative examples for $\text{move}(A, R, C)$; 2 positive examples and 1 negative example for $\text{move}(A, \text{center}, C)$; 11 positive examples for $\text{grasp}(A, R, C)$; 10 positive examples and 4 negative examples for $\text{release}(A)$; and 1 positive example for $\text{extract}(A, R, C)$.

Action effects

Since atoms in axioms corresponding to action effects do not share the same timestep, examples for these axioms must account for the temporal aspects. Since the ASP task knowledge formulation includes predicates inspired by event calculus, two examples were generated for each fluent and for each task execution, one each for the *initiated* and the *terminated* axioms of this fluent. Only positive examples were considered since they would not be used to learn executability conditions (see above). Examples were CDPIs of the form

$\langle e^{inc}, e^{exc}, C \rangle$, where e^{inc} was the set of initiated (or terminated) conditions at all timesteps, while e^{exc} was the set of initiated (or terminated) conditions that did not hold at all timesteps. The context C was the task history, i.e., the set of atoms corresponding to actions and fluents that were true at all timesteps. The set e^{exc} was needed to guarantee that only relevant causal laws were learned, given that positive examples are subject to brave induction—see Definition 8. Consider the scene in Figure 5.7 as an illustrative example. For the fluent $at(A, R, C, t)$, the positive example (considering only the initiated condition for simplicity) is shown below, with the atoms corresponding to the set e^{exc} underlined:

```
#pos{ex3,
{init(at(psm1,ring,red,2)),init(at(psm1,ring,blue,7)),...},
{init(at(psm1,ring,blue,2)),init(at(psm1,ring,red,7)),...},
{reachable(psm1,ring,red,_),reachable(psm2,ring,blue,_),
reachable(psm1,peg,red,_),reachable(psm1,peg,blue,_),
reachable(psm2,peg,green,_),reachable(psm2,peg,yellow,_),
reachable(psm1,peg,grey,_),reachable(psm2,peg,grey,_),
on(ring,red,peg,blue,1),on(ring,blue,peg,red,1),...}}
```

As before, redundant examples were omitted for action effects. Overall, 1 example was generated for the initiated (equivalently, terminated) condition for $closed_gripper(A)$; 2 examples for $in_hand(A, ring, C)$; 1 example for $at(A, center)$; 2 examples for $on(ring, C1, peg, C2)$; and 3 examples for $at(A, O, C)$.

6.4.4 Results of ILASP

The results of ILP task are now presented, distinguishing between executability constraints and pre-conditions of actions on one hand, and effects on the other hand. Validation of the learned ASP encoding is performed finally, both for sequential and parallel task executions. Learning is performed only for the most complex and general Encoding 2, involving persistence of environmental fluents,

hence more complex and realistic temporal specifications. In fact, Encoding 1 (standard task description) is represented with the same axioms of Encoding 2, simply removing persistence in (5.6); Encoding 3 is the same as Encoding 1, simply adding (5.16) for optimization. ILASP is run on a PC with standard commercial hardware specifications, using 2.6 GHz Intel Core i7 processor and 16 GB RAM (the same hardware used for experiments in Chapter 5).

Pre-conditions and executability constraints

The learned action pre-conditions are as follows (a single aggregate is added manually after learning, since ILASP tasks learn pre-conditions for single actions):

```
0 {move(A, O, C, t) : reachable(A, O, C, t);      (6.4)
move(A, center, C, t) : in_hand(A, ring, C, t);
extract(A, ring, C, t) : in_hand(A, ring, C, t);
grasp(A, ring, C, t) : at(A, ring, C, t);
release(A, t) : in_hand(A, ring, C, t) } 1 :- arm(A).
```

Executability constraints are:

```
:- move(A, peg, C2, t), in_hand(A, ring, C, t),
   on(ring, C, peg, C1, t).      (6.5a)
```

```
:- move(_, peg, _, t), in_hand(A, ring, C, t),
   not reachable(A, peg, C).    (6.5b)
```

```
:- move(A, center, C, t), on(ring, C, peg, C1, t).
                                          (6.5c)
```

```
:- move(A, ring, C, t), closed_gripper(A, t).    (6.5d)
```

```
:- on(ring, C1, peg, C, t), move(A, peg, C, t).  (6.5e)
```

Notice that the timestep variable is added to each atom after learning. Axiom 6.4 matches the action pre-conditions in axiom 5.1. Axioms 6.5(a-c) represent the same conditions as in Axioms 5.3(e), 5.3(a) (forbidding motion when both

TABLE 6.1: Quantitative results of the ILASP task for pre-conditions and executability constraints. The lengths of original and learned axioms are compared, and the learning time is shown as returned from ILASP.

Actions	Original length	Learned length	Time [s]
<code>move(A, ring, C)</code>	4	4	0.49
<code>move(A, peg, C)</code>	11	10	49.17
<code>move(A, center, C)</code>	5	4	0.09
<code>extract(A, ring, C)</code>	2	2	0.06
<code>grasp(A, ring, C)</code>	2	2	0.09
<code>release(A)</code>	2	2	0.79
total	26	24	50.69

arms hold the same ring is equivalent to forbidding motion when the arm which cannot reach the peg is holding the ring, after transfer) and 5.3(b) respectively. Axioms 6.5(e-f) match the constraints in axioms 5.3(c-d). Notice that axiom 6.5b contains placeholders (`_`) in the action atom. ILASP is not able to generate placeholders, hence they are added after learning. In fact, executability constraints are learned by a separate ILASP instance for each action, with only examples relevant to that action. This results in the following executability constraint without the action fluent:

```
:- in_hand(A, ring, C, t), not reachable(A, peg, C, t).
```

This constraint is obviously infeasible when combined with the set of axioms for the other actions. Hence, the action atom with placeholders in (6.5)b is added to relate this constraint only to the action of moving to a peg. Notice that placeholders make the substituted variables arbitrary in the axiom, hence the generality of the learned constraint is preserved. Table 6.1 shows the computational time of the learning process for each action specification, as well as the length of the learned axioms with respect to the original ASP encoding of the domain. The action `move(Arm, peg, Color)` has the most influence on the learning time. In fact, it has the highest length of axioms, hence more time is needed to search the set of hypotheses and find the correct ones for it. An overall reduction from 26 to 24 of the length of the axioms is gained with ILASP, which finds more

optimal logical connections between fluents.

Effects of actions

The learned axioms for starting conditions for effects (replacing $\text{prev}(T1, T2, 1)$) are as follows:

$$\text{initiated}(\text{in_hand}(A, \text{ring}, C), t) \text{ :- grasp}(A, \text{ring}, C, t-1). \quad (6.6a)$$

$$\text{initiated}(\text{closed_gripper}(A), t) \text{ :- grasp}(A, \text{ring}, _, t-1). \quad (6.6b)$$

$$\text{initiated}(\text{on}(\text{ring}, C1, \text{peg}, C2), t) \text{ :- in_hand}(A, \text{ring}, C1, t-1), \text{at}(A, \text{peg}, C2, t-1), \text{release}(A, t-1). \quad (6.6c)$$

$$\text{initiated}(\text{at}(A, \text{ring}, C), t) \text{ :- move}(A, \text{ring}, C, t-1). \quad (6.6d)$$

$$\text{initiated}(\text{at}(A1, \text{ring}, C), t) \text{ :- move}(A2, \text{center}, C, t-1), \text{different}(A1, A2). \quad (6.6e)$$

$$\text{initiated}(\text{at}(A, \text{peg}, C), t) \text{ :- move}(A, \text{peg}, C, t-1). \quad (6.6f)$$

$$\text{initiated}(\text{at}(A, \text{center}), t) \text{ :- move}(A, \text{center}, _, t-1). \quad (6.6g)$$

while the terminated conditions are as follows:

$$\text{terminated}(\text{in_hand}(A, \text{ring}, C), t) \text{ :- release}(A, t-1), \text{in_hand}(A, \text{ring}, C, t-1). \quad (6.7a)$$

$$\text{terminated}(\text{closed_gripper}(A), t) \text{ :- release}(A, t-1). \quad (6.7b)$$

$$\text{terminated}(\text{on}(\text{ring}, C1, \text{peg}, C2), t) \text{ :- in_hand}(A, \text{ring}, C1, t-1), \text{on}(\text{ring}, C1, \text{peg}, C2, t-1). \quad (6.7c)$$

$$\text{terminated}(\text{at}(A, \text{ring}, C), t) \text{ :- extract}(A, \text{ring}, C, t-1). \quad (6.7d)$$

$$\begin{aligned} \text{terminated}(\text{at}(A1, \text{ring}, C), t) \text{ :- at}(A1, \text{peg}, C, t), \\ \text{at}(A2, \text{center}, t), \text{release}(A1, t). \end{aligned} \quad (6.7e)$$

$$\begin{aligned} \text{terminated}(\text{at}(A, \text{peg}, C), t) \text{ :- at}(A, \text{peg}, C, t-1), \\ \text{grasp}(A, \text{ring}, C1, t). \end{aligned} \quad (6.7f)$$

$$\begin{aligned} \text{terminated}(\text{at}(A1, \text{center}), t) \text{ :- at}(A1, \text{center}, _), \\ \text{move}(A2, \text{ring}, C, t), \text{grasp}(A1, \text{ring}, C, t). \end{aligned} \quad (6.7g)$$

Notice that ILASP actually finds the following causal law for the initiating condition of `closed_gripper(A)`:

$$\text{initiated}(\text{closed_gripper}(A), t) \text{ :- in_hand}(A, \text{ring}, C, t). \quad (6.8)$$

In 6.6b, the body of axiom 6.6a is replaced just for clarification (the goal is to highlight the effects of actions on environmental fluents).

Axiom 6.6c is learnt using intermediate predicate invention. In fact, algorithm ILASP2i returns the partial hypothesis after evaluation of each example. The algorithm is not able to find a valid hypothesis for all examples at the first try, but it generates the hypothesis:

$$\begin{aligned} \text{initiated}(\text{on}(\text{ring}, C, \text{peg}, C), t) \text{ :- release}(A, t-1), \\ \text{at}(A, \text{peg}, C, t-1). \end{aligned}$$

This hypothesis only covers examples in which a ring is placed on the same-colored peg, but it does not cover anomalous scenarios in which a ring has to be placed on a grey peg (Figure 5.7). This partial hypothesis is then added as a new axiom to the background knowledge:

$$\begin{aligned} \text{flag}(A, C, T1, T2) \text{ :- release}(A, T1), \text{at}(A, \text{peg}, C, \\ T1), \text{prev}(T1, T2). \end{aligned} \quad (6.9)$$

TABLE 6.2: Quantitative results of the ILASP task for initiated axioms for the effects of actions. The lengths of original and learned axioms are compared, and the learning time is shown as returned from ILASP.

Actions	Original length	Learned length	Time [s]
<code>at(A, ring, C)</code>	6	5	17.77
<code>at(A, peg, C)</code>	2	2	24.89
<code>at(A, center)</code>	2	2	13.50
<code>in_hand(A, ring, C)</code>	2	2	10.15
<code>on(ring, C1, peg, C2)</code>	4	4	67.06 ²
<code>closed_gripper(A)</code>	2	2	10.15
total	18	17	143.52

and the mode bias is modified to include `flag` in the search space. ILASP is now able to find the correct axiom. Notice that increasing the maximum axiom length in the hyperparameters would lead to the same result without intermediate predicate invention, though increasing the search space.

In Section 6.4.3, the importance of specifying not holding fluents in the examples for effects of actions has been asserted. This axiom is now clarified with an example. Consider the `initiated` condition for the fluent `at(A, ring, C)`. Excluding non-occurring fluents from examples generates the following initiating axiom:

```
initiated(at(A, ring, C), t) :- in_hand(A, ring, C, t+1).
```

which does not always hold actually, and does not properly describe the causal relation since it inverts the causality condition between body and head of rule.

Validation of learned axioms

Learned axioms are validated in the same set of 1000 simulated scenarios used in Section 5.3.1, to mimic challenging environmental conditions for the ring transfer task, both for parallel and sequential executions.

²Adding new predicate in axiom 6.9 to the background knowledge.

TABLE 6.3: Quantitative results of the ILASP task for terminated axioms for the effects of actions. The lengths of original and learned axioms are compared, and the learning time is shown as returned from ILASP.

Actions	Original length	Learned length	Time [s]
<code>at(A, ring, C)</code>	10	8	18.23
<code>at(A, peg, C)</code>	10	3	24.79
<code>at(A, center)</code>	6	4	20.86
<code>in_hand(A, ring, C)</code>	3	3	10.76
<code>on(ring, C1, peg, C2)</code>	3	3	92.36
<code>closed_gripper(A)</code>	2	2	10.76
total	34	23	177.76

The set of learned axioms presented in previous sections does not allow to compute a plan for all scenarios. Then a systematic approach is used to refine learned axioms, showing that the proposed ILP approach allows knowledge refinement.

First, axioms which mostly affect the plan computational time are identified, replacing learned axioms with the original ones presented in Chapter 5. Axioms 6.7c-e-f are recognized as the bottleneck for plan computation, by iteratively removing learned axioms for each single environmental fluent. Then, a new ILASP task is run for each of the corresponding fluents, i.e. `at(A, ring, C)`, `at(A, peg, C)`, removing axioms 6.7c-e-f from the search space. This results in the following final set of terminated axioms (new axioms are underlined), which allows to compute plans for all simulated scenarios:

$$\begin{aligned} \text{terminated}(\text{in_hand}(A, \text{ring}, C), t) &:- \text{release}(A, t-1), \\ &\quad \text{in_hand}(A, \text{ring}, C, t-1) \end{aligned} \quad (6.10a)$$

$$\begin{aligned} \text{terminated}(\text{closed_gripper}(A), t) &:- \text{release}(A, t-1) \end{aligned} \quad (6.10b)$$

$$\begin{aligned} \text{terminated}(\text{on}(\text{ring}, C1, \text{peg}, C2), t) &:- \text{on}(\text{ring}, C1, \text{peg}, \\ &\quad \underline{C2, t-1}, \text{extract}(A, \text{ring}, C1, t-1)) \end{aligned} \quad (6.10c)$$

$$\begin{aligned} \text{terminated}(\text{at}(A, \text{ring}, C), t) &:- \text{extract}(A, \text{ring}, C, t-1) \end{aligned} \quad (6.10d)$$

$$\frac{\text{terminated}(\text{at}(A, \text{ring}, C), t) \text{ :- } \text{at}(A, \text{ring}, C, t-1),}{\text{release}(A, t-1)} \quad (6.10e)$$

$$\frac{\text{terminated}(\text{at}(A, \text{ring}, C), t) \text{ :- } \text{at}(A, \text{ring}, C, t-1),}{\text{move}(A, \text{center}, C, t-1)} \quad (6.10f)$$

$$\frac{\text{terminated}(\text{at}(A, \text{peg}, C), t) \text{ :- } \text{at}(A, \text{peg}, C, t-1),}{\text{at}(A, \text{ring}, C1, t)} \quad (6.10g)$$

$$\frac{\text{terminated}(\text{at}(A1, \text{center}), t) \text{ :- } \text{at}(A1, \text{center}, _),}{\text{move}(A2, \text{ring}, C, t), \text{grasp}(A1, \text{ring}, C, t)} \quad (6.10h)$$

Tables 6.2-6.3 show the learning performances for the initiated axioms 6.6 and the new terminated axioms. `closed_gripper` requires the same learning time as `in_hand` because of the detected semantic equivalency (6.8). Initiating and terminating axioms for `on(ring, C1, peg, C2)`, `at(A, O, C)` and `at(A, center)` require most of the overall learning time because the search space is wider than other axioms. The original ASP encoding for pre-conditions, executability constraints and effects contains more axioms than learned ones. Notice that the terminating condition for `at(A, peg, C)` is significantly shorter. In fact, a comparison of axioms 6.10g and 6.3g-h evidences that ILASP finds a single axiom describing the terminating condition, connecting fluents instead of actions, differently from standard human design logic.

Figures 6.1-6.2 show the comparison between learned and original versions for ASP Encoding 2 for the sequential and parallel task execution, respectively. The size of the plans returned by the two ASP programs and the plan computational times are compared in all simulated scenarios. Plots are obtained similarly to the ones in Section 5.3.1, sorting and clustering plans from the original ASP program according to their length and matching environmental fluents for a fair comparison.

The results show that the learned ASP program for the sequential execution generates plans of similar length with respect to the original program, while slightly longer plans are generated with parallel execution.

The average planning time and its variability are significantly lower for the

learned ASP program in the sequential execution, due to the shorter and more optimal axioms found by ILASP. In particular, the average planning time for sixth, ninth and twelfth clusters is reduced of approximately 100 s, solving critical issues for the real-time application of logic planning to real surgery. The result is not evidenced in the parallel execution. This is probably because the planning time is significantly lower with respect to the sequential execution. However, the computational time is comparable to the original encoding, hence guaranteeing similar performances in a real planning scenario.

Moreover, in Section 5.3.1 one cluster of six scenarios has been identified where Clingo with the original Encoding is not able to find a feasible plan within allowed maximum time of 200 s (as in Chapter 5, the limit of vertical axis is set to 600 s for visibility reasons). Experiments show that the learned program fails to find a plan in time only in one scenario (the average planning time in Figure 6.1-bottom is higher than the maximum allowed time). The dataset of simulated scenario is used to compare learned and original axioms also for Encoding 1 (Figure 6.3 for sequential execution, Figure 6.4 for parallel) and Encoding 3 (Figure 6.5 for sequential execution, Figure 6.6 for parallel). The length of the plans is overall comparable for all Encodings. Some slight improvement is introduced by the learned axioms for sequential execution with Encoding 1, but plans are slightly longer using the learned axioms for parallel execution with Encoding 3. As for the planning time, it does not vary for Encodings 1 and 3 in the sequential execution overall. Some initial clusters require longer planning time with learned Encoding 3, but the original encoding performs worse for last clusters (longer plans). In the parallel execution, the learned encodings require slightly higher computational times, but the maximum average time is 5 s, suitable for a real robotic scenario. In general, learned axioms do not significantly affect the applicability of the ASP encoding to a real robotic scenario.

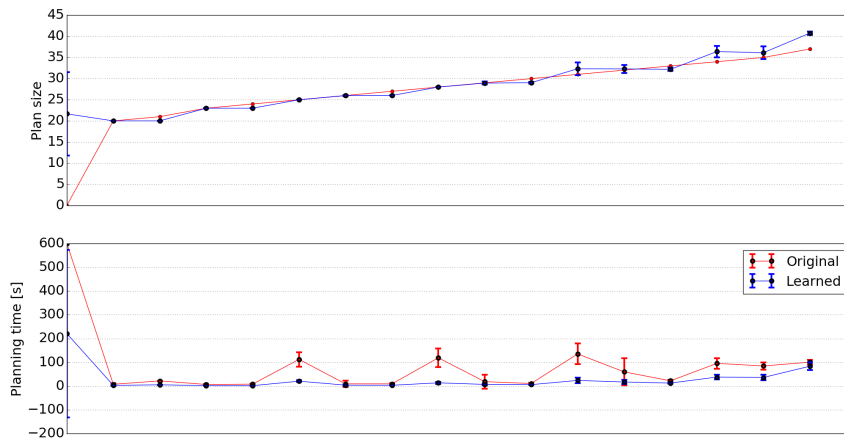


FIGURE 6.1: Comparison between original and learned Encoding 2 for the ring transfer task in the sequential execution, for clusters of plans with same length for original encoding (horizontal axis).

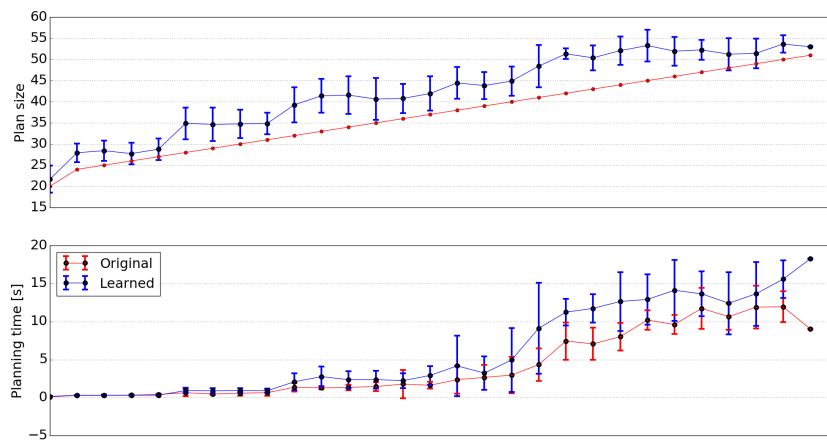


FIGURE 6.2: Comparison between original and learned Encoding 2 for the ring transfer task in the parallel execution, for clusters of plans with same length for original encoding (horizontal axis).

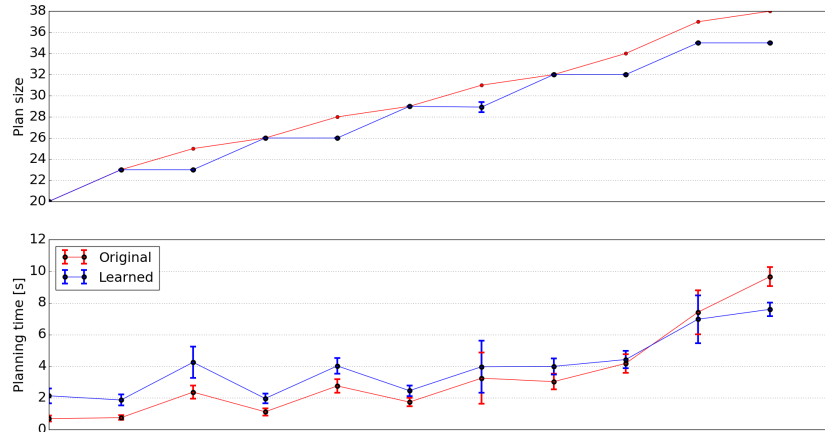


FIGURE 6.3: Comparison between original and learned Encoding 1 for the ring transfer task in the sequential execution, for clusters of plans with same length for original encoding (horizontal axis).

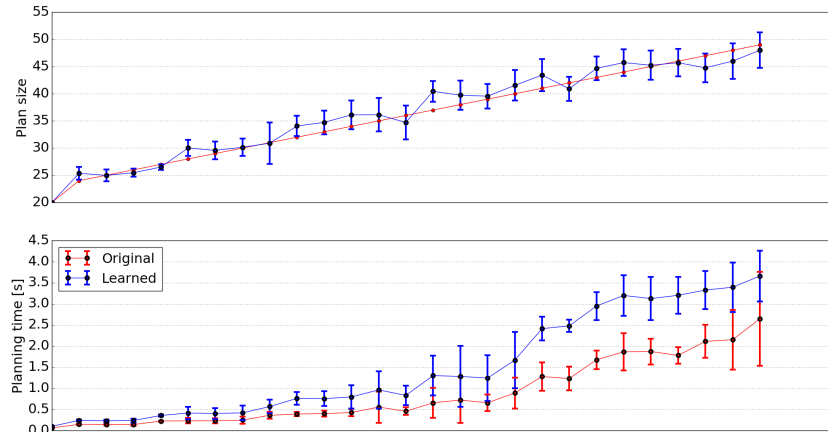


FIGURE 6.4: Comparison between original and learned Encoding 1 for the ring transfer task in the parallel execution, for clusters of plans with same length for original encoding (horizontal axis).

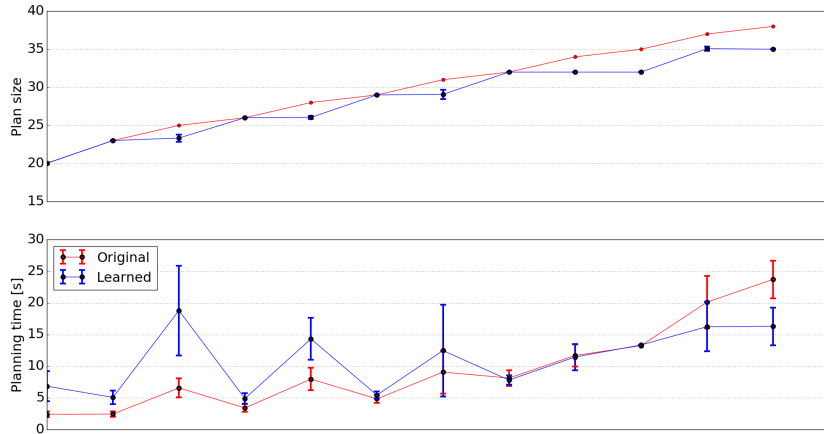


FIGURE 6.5: Comparison between original and learned Encoding 3 for the ring transfer task in the sequential execution, for clusters of plans with same length for original encoding (horizontal axis).

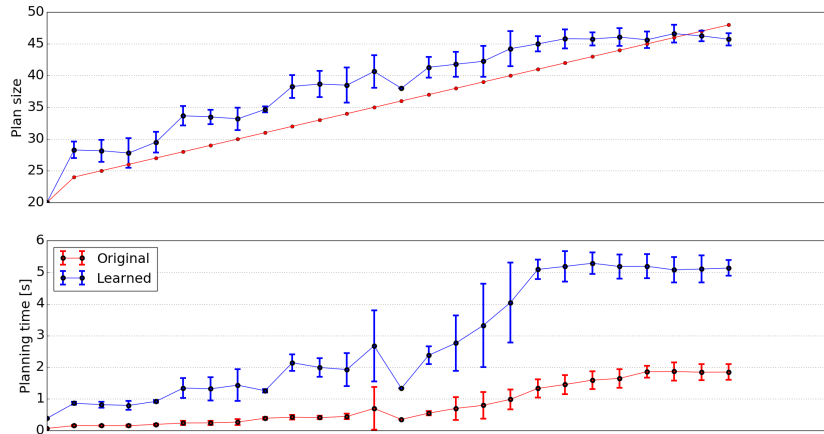


FIGURE 6.6: Comparison between original and learned Encoding 3 for the ring transfer task in the parallel execution, for clusters of plans with same length for original encoding (horizontal axis).

6.5 Discussion

The learned ASP axioms have been compared to the hand-written ones in the three encodings presented in Section 5.3.1, in terms of plan length and computation time. The same set of 1000 simulated scenarios of Section 5.3.1 is used as a benchmark.

Learned encodings exhibit comparable or slightly worse results with respect to hand-written one in terms of plan size, for sequential and parallel execution respectively. However, the most relevant metric is the plan computation time, which affects the real-time performances of a task planner for robotics in general, and is particularly relevant for a critical scenario like surgery.

For the sequential execution of the task, the average planning time with the learned encoding is reduced of approximately 100 s in some scenarios for Encoding 2, with reduced variance. This allows to guarantee real time performances which are not reachable with the original ASP encoding. The result is due to the optimality of the learned rules by ILASP, which are shorter than the original ones and identify more efficient semantic relations between atoms in the domain.

In the parallel execution with Encoding 2, the planning time with the learned encoding is slightly higher. However, the planning time is significantly lower with respect to sequential execution (< 20 s), hence acceptable computational performances are still guaranteed.

Learned axioms for Encodings 1 and 3 require slightly higher computational time, but still comparable to the original ones and acceptable for application in real robotic scenarios (5 s maximum average planning time with optimization and parallel execution in Encoding 3).

Moreover, learned axioms fail to generate a plan (within maximum allowed time) with Encoding 2 in only one environmental condition, against six failures for original axioms. No failure is recorded with other encodings.

Finally, learning time is low (178 sat most for terminating conditions of effects) even on a computer with standard commercial hardware specifications.

6.6 Conclusion

In this chapter, an ILP-based approach to surgical task knowledge learning has been presented, based on learning from context-dependent examples under the AS semantics with the state-of-the-art tool ILASP. This method can cope with several issues of the surgical scenario, as the unavailability of large and complete training datasets and the need for interpretable surgical task description for reliability of an autonomous surgical system.

In order to apply ILASP to the ring transfer task (and in general to complex surgical robotic tasks) involving time-dependent relations between atoms (encoding, e.g., durative environmental fluents and continuous mutual influence between the robot and environment), first a re-formulation of axioms based on the paradigm of event calculus has been proposed.

Given a set of only four incomplete executions of the ring transfer task, from human and autonomous robotic agents, all task-relevant axioms in ASP formalism have been learned. Moreover, ILASP is able to identify the minimum arbitrary delay between atoms. This is crucial for application to general complex robotic tasks.

The ILASP approach solves a number of issues in the context of step-level SPM learning, including learning of interpretable and reliable action models and learning from a limited number of (possibly different) examples.

Moreover, ILASP guarantees fast learning, and an improvement in the quality of learned task knowledge with respect to manually-encoded one, reflecting in ASP solving with higher computational efficiency.

Finally, distinguishing between different parts of a generic robotic task knowledge (pre-conditions, executability constraints and effects of actions, with or without dependency on time) allows to easily generalize the framework proposed in this Chapter to other use cases.

A drawback of the presented approach is the need for fully labeled executions of the task. The annotation of real, long surgical task is tedious and prone to errors even for expert users, hence it is often difficult. This problem will be partially solved in the next chapter.

Chapter 7

Towards unsupervised ILP

7.1 Introduction

The ILP approach presented in former Chapter 6 relies on the assumption that a complete semantic description of executions in the training dataset is made available from expert user annotations (labeling) of the execution traces, both in terms of environmental features (fluents) and actions executed by the robot. This requirement strongly limits the scalability of the ILP approach to more complex surgical tasks, since traces of executions are usually long and hence the labeling process is tedious and prone to errors, even from an expert operator. This is demonstrated, for instance, by the recently developed supervised algorithms for surgical action identification, as [293, 338] based on statistical model learning or [247, 319] based on movement primitives. For this reason, the goal of this chapter is to investigate an unsupervised approach to the labeling problem, which can lead to an unsupervised (or at least partially supervised) ILP methodology for scalable surgical task knowledge learning.

In order to construct examples for ILP, both environmental features and robotic actions must be labeled. The problem of automatic recognition of environmental features from surgical videos has been widely investigated by researchers [202], mainly for surgical tool detection, e.g. in laparoscopic surgery [268], eye surgery [326] and neurosurgery [32]; event recognition as detection of smoke from electrosurgery [203]; and anatomical feature identification as cystic artery damage [174], tumor [150], vessels [223] and blood flow for autonomous robotic suction [274]. The above mentioned applications make use of standard and well established algorithms from machine learning, especially deep neural networks,

to match raw environmental data to higher-level semantic features. For the ring transfer task considered in this thesis, environmental fluents represent relations between geometric quantities retrieved from the robot kinematics and the video (e.g., positions of objects). Hence, they can be computed automatically as already explained for the SA module of the autonomous framework proposed in Chapter 5, based on relations in (5.7).

The focus of this chapter is on the automatic unsupervised identification of actions from execution traces. This is a more complex problem than the extraction of semantic features from the environment, since it does not involve a mere evaluation and matching of raw information from sensors, but more structured analysis is needed to discern, e.g., between actions which may occur under very similar environmental conditions. A novel algorithm for action identification is proposed in Section 7.2 which exploits semantic features and kinematic analysis to solve several issues of state-of-the-art methods, including the need for large datasets, repetitive task executions and downgrade of performances for short actions. Section 7.3 shows the results of the algorithm as applied to the ring transfer task. Finally, a methodology for partially unsupervised ILP is proposed in the scenario of the ring transfer task.

7.2 Unsupervised action identification

Recent research has focused on the unsupervised identification of actions from surgical datasets. Inspired from advances in human activity recognition [156, 238], researchers have investigated approaches to unsupervised recognition of surgical actions. [237, 292] use transition-state classification (TSC) with deep neural networks on videos and kinematics of JIGSAWS dataset of surgical actions [101], with an identification accuracy below 67%. Approaches based on statistical models, e.g. TSC with Gaussian mixture models [169], and soft-boundary algorithms with fuzzy classification scores [90] have shown improve the accuracy. Recently, weakly supervised methods as [10] have been proposed, initializing the transition parameters of a Gaussian mixture model from a set of only three example executions to improve the accuracy of subsequent unsupervised segmentation based on expectation maximization. While this significantly

mitigates the issue of the size of the training dataset for supervised approaches, the method is only based on kinematic analysis, without including visual features. Moreover, the mentioned works are validated on (often large) datasets containing *homogeneous* repetitions of the same task, i.e. executions which do not differ in the sequence of actions, but present only some kinematic variations due to execution from multiple users (with possibly different expertise). Unsupervised identification algorithms usually perform better on homogeneous datasets, because of the higher similarity between different instances of the same action. Homogeneity is possible for repetitive tasks as suturing or knot tying, or standard versions of training tasks as the peg transfer. However, this requirement is infeasible when tasks which involve multiple possible workflows depending on dynamic environmental (anatomical) contingencies (as the ring transfer) are considered. This problem is even more crucial for long complex tasks which are more relevant to the surgical scenario, e.g. more realistic procedures than training exercises. A further limitation of state-of-the-art approaches (especially when only kinematic features are analyzed, as [67]) is the poorer performance for actions with short durations. The algorithm for unsupervised action identification proposed in this section tries to solve all the aforementioned issues, exploiting semantic environmental features extracted from each frame of videos in the execution traces. The algorithm is presented and validated with reference to the case study of the ring transfer task, hence environmental features are extracted from videos with relations in (5.7). Extension to other surgical (and, more generally, robotic) tasks implies considering specific semantics for the interpretation of geometric features from videos. Environmental features are combined with a *kinematic signature* consisting of the Cartesian position \mathbf{p}_A and the gripper angle j_A for each PSM, as reported in Table 5.1, plus the orientation of PSMs $\mathbf{q}_A = \{x_{or,A}, y_{or,A}, z_{or,A}, w_{or,A}\}$ (16 kinematic features are considered as a total, 8 per PSM). Notice that other works include additional features in the kinematic signature signature, e.g. the speed of the end effectors [10]) and invariant geometric measures of Cartesian trajectories as curvature and torsion [67]. However, these measures have not shown to improve the identification performances in the experiments presented in this chapter (specifically, [10] evidences that the speed of end-effectors even downgrades the accuracy), hence they are omitted.

Action identification is a sequence of two main steps:

- *action segmentation* to identify changepoints delimiting segments in the execution trace;
- *action classification* to group together segments corresponding to the same action class.

7.2.1 Action segmentation

Given an execution trace, the segmentation algorithm 4 identifies changepoints corresponding to starting / ending timesteps of actions of the task. The algorithm involves two main steps: changepoint detection from kinematics, and changepoint filtering from the semantic video stream.

Changepoint detection

Several algorithms for changepoint detection in timeseries have been proposed in literature, as extensively revised in [312, 35, 45]. Some algorithms assume that the number of segments in a timeseries is known as a prior. However, this is not usually the case when a completely unsupervised analysis is required. Changepoint detection algorithms typically seek to optimize the adherency of the input timeseries to an approximate sequence of segments, also minimizing the number of segments when it is unknown. Both parametric and non-parametric cost functions exist to evaluate the goodness of the changepoint approximation [312]. Parametric methods are based on a finite set of parameters to be tuned, while non-parametric methods (e.g., non-parametric maximum likelihood estimation [341]) are usually more robust and generic. Moreover, some algorithms solve the optimization problem exactly [23, 157], under certain assumptions on the cost function (e.g., linearity in the parameters for [157]). This obviously results in expensive computation, especially for long timeseries. For this reason, approximate algorithms, e.g. based on tunable sliding window approximation of the full timeseries [128], have gained increased popularity.

The algorithm proposed in this section does not derive from any assumption on the number of segments. The features in the kinematic signature are first normalized by their maximum values to allow comparison between different configurations of the setup for the task (different locations of objects in the scene and distances to the PSMs). Then, filtering at 1.5 Hz is applied to each feature, to consider only relevant motions within the fundamental frequency of human gestures [201]. Changepoints in the kinematic signature are identified evaluating peaks in the 2nd derivative of the features. Specifically, peaks above a user-defined threshold α are selected as candidate changepoints. In order to remove implicit noise in the 2nd derivative, it is computed using Savitzky-Golay filter (SGF) [285]. SFG is a digital filter which focuses on a pre-defined time window of the signal, and performs polynomial fitting for smoothing. Hence,SGF is able to capture higher-order moments in the data (depending on the size of the time window), while removing noise in the signal. SGF has a long history of successful applications in time series analysis, e.g. for kinematic analysis of the human arm [324] and neural signals [265]. One advantage of this approach is the very limited number of parameters to be tuned for segmentation (time window, set to 20 in the following experiments as empirically determined from the speed of variation of the kinematic features; and $\alpha = 20\%$ relative to the maximum absolute value of the 2nd order derivative for each kinematic feature). This makes the algorithm more robust and general.

Changepoint filtering

The analysis of only kinematic features can lead to the detection of spurious changepoints, corresponding e.g. to abrupt motions during the execution of the task. Hence, first consecutive changepoints which are distant less than 1s is performed (implicitly assuming an empirical minimum duration for actions, corresponding to the duration of `grasp(A, ring, C)`, `release(A)` actions involving the grippers of PSMs). Then, fluents from the video stream are computed in correspondence of each changepoint. If two or more consecutive changepoints correspond to the same set of fluents, only the first is considered valid, and others are excluded. This choice is made under the assumption that *different actions*

Algorithm 4 Action segmentation algorithm

```

1: Input: Execution trace with temporal kinematic signature  $K(t)$  and video
   stream  $V(t)$ , threshold for peak detection  $\alpha$ 
2: Output: Set of changepoints  $C$ 
3: Initialize: Normalize and filter  $K(t)$ ,  $C = []$ 
4: % Changepoint detection
5: for  $k(t)$  kinematic feature  $\in K(t)$  do
6:    $\ddot{k}(t) = \text{SGF}(k(t))$ 
7:   peaks = PeakDetect( $\ddot{k}(t)$ )
8:   for  $p \in \text{peaks}$  do
9:     if  $|\ddot{k}(p)| > \alpha \cdot \max |\ddot{k}(p)|$  then
10:       $C.append(p)$ 
11:    end if
12:  end for
13: end for
14: % Changepoint filtering
15: fluents  $F = []$ 
16:  $c_{old} = 0$ 
17: for  $c \in C$  do
18:   old_fluents  $F_{old} = F$ 
19:    $F = \text{FluentCompute}(V(c))$ 
20:   if  $F == F_{old} \vee c - c_{old} < 1s$  then
21:      $C.remove(c)$ 
22:   end if
23:    $c_{old} = c$ 
24: end for
25: return  $C$ 

```

derive from different environmental conditions: hence, two consecutive changepoints defining the starting timesteps of two different actions cannot share the same set of fluents.

7.2.2 Action classification

Given the correct changepoints of the execution trace(s), Hence, the segments corresponding to actions of the task, k -NN classification is used to group segments corresponding to same action classes. k -NN is preferred over other established classification methods as support vector machines (SVMs) [67] and self-organizing maps based on neural networks [238] because it usually exhibits

better performance on small datasets. For instance, in [67] k -NN results in better identification than SVM. Each segment in the execution trace is associated with a feature vector $\mathbf{f} = [\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3]$, containing both kinematic information as in [67] and semantic environmental information from the video stream:

Features \mathbf{f}_1

It is an array of reals representing the kinematic features of each segment. Each feature $k_i(t)$ in the kinematic signature of the segment is approximated by a polynomial $p_i(t) = \sum_{j=1}^n a_{j,i} t^j$, with $n \in \mathbb{N}$ arbitrary degree as in [67]. Then, \mathbf{f}_1 is built concatenating coefficients $a_{j,i}$, resulting in an array of dimension $(n + 1) \cdot 16$ (2 arms \times 8 kinematic features, i.e. 4 for orientation, 3 for position and 1 for gripper joint). After empirical evaluation, $n = 5$ is chosen as the optimal polynomial order for action identification. In order to properly compare kinematic features corresponding to different instances of the same action, they are first shifted in time to start from $t = 0$. Polynomial approximation is not the only option for kinematic representation. For instance, in [238] coefficients of Fourier expansion are shown to be more robust against noise in the kinematic signature for identification of actions in the benchmark CAVIAR dataset for human action recognition [57]. However, the results of the experiments proposed in this chapter do not show any variation when Fourier approximation is used in place of polynomial one. Notice that the classification algorithm must be able to generalize over the colors of objects in the setup and the actual PSM executing a specific action: for instance, two segments corresponding to `move(psm1, ring, red)`, `move(psm2, ring, yellow)` must be classified in the same cluster, corresponding to the abstract action `move(A, ring, C)`. Hence, evaluation of which PSM moves in each segment is needed, evaluating the difference between the kinematic signatures at the starting and ending changepoints. In case only one arm moves, the coefficients of the polynomial approximations of the kinematic features for that specific arm are added to \mathbf{f}_1 , and null coefficients for the other arm are appended to complete the vector. In case both PSMs move along the segment, coefficients of PSM1 and PSM2 are concatenated to build \mathbf{f}_1 . In this way, there is no distinction whether an action is executed by one arm or the other.

Features f_2

It is a Boolean array representing fluents holding at the beginning changepoint of each segment (i.e., grounded by Algorithm 3). Each entry in the array corresponds to an environmental fluent defined in Section 5.2, excluding `reachable(A, ring, C)` which is identified at all timesteps in the video stream (all rings and pegs are always reachable at least by one arm), Hence, it is neglected. Each entry in the array has true value if the corresponding fluent holds at the beginning of the segment, otherwise it is set to false. In order to guarantee the invariance of the classification algorithm with respect to the arm and color instances, f_2 has two entries for each fluent, one per PSM, and the color attribute is ignored. Similarly to the generation of f_1 , if a fluent holds only for one arm, then the first corresponding entry is set to true, regardless of the specific arm. The final dimension of f_2 is 12.

Features f_3

It is a 16D Boolean vector with entries corresponding to each feature in the kinematic signature of the segment. If one signature varies from the beginning to the ending changepoint, the value of the corresponding entry is set to true (as for f_1 , the order of entries for the two arms does not depend on the specific PSM to guarantee the generality of the classification algorithm). f_3 is needed because instances of the same action may significantly differ from a kinematic perspective, due to the different relative position of objects and arms in the scene. However, it is reasonable to expect that the same kinematic features vary along the segment.

Classification algorithm

k -NN classification must then compare feature vectors for different segments in the execution trace(s) containing both Boolean and real values. Hence, the standard Euclidean distance metric over the set of polynomial coefficients $a_{j,i}$ is not suitable. Instead, a mixed distance metric $d_{i,l}$ between segments i, l is defined

as:

$$d_{i,l} = \sqrt{\frac{d_e^2}{d_{emax}^2} + \frac{d_h^2}{d_{hmax}^2}} \quad (7.1)$$

where d_e is the standard Euclidean distance between coefficients in \mathbf{f}_1 for the two segments, $d_e = \sqrt{\sum_{j=1}^n (a_{j,i} - a_{j,l})^2}$; while d_h is the Hamming distance between $[\mathbf{f}_2, \mathbf{f}_3]$ for the two segments

$$d_h = \frac{h_{i,l}}{\dim([\mathbf{f}_2, \mathbf{f}_3]_i)}$$

being $h_{i,l}$ the number of different values in $[\mathbf{f}_2, \mathbf{f}_3]$ for segment i with respect to segment l , and $\dim(\cdot)$ the dimension of an array. d_{emax}, d_{hmax} are normalizing factors which are needed to properly combine Euclidean and Hamming distances, and they are chosen as the maximum cost resulting from k -NN classification with only Euclidean distance over \mathbf{f}_1 and only Hamming distance over $[\mathbf{f}_2, \mathbf{f}_3]$, respectively.

k for k -NN classification is chosen as the number of occurrences of the most frequent action in the dataset of executions. In this way, the algorithm is able to recognize all instances of actions in the dataset. A higher k value would also be valid; however, the experiments will show that the best classification performance is achieved with the minimum k .

7.3 Experiments

The unsupervised action identification algorithm is evaluated with three different experiments.

In Test A, 9 human executions of the ring transfer task are considered from the dataset used in Chapter 5 to learn DMPs. The executions are in *standard environmental conditions*, i.e. with all rings placed on grey pegs and requiring transfer between arms. Human users are not surgeons, and they have different expertise in using dVRK. Hence, this test emulates learning from senior and novice surgeons or trainees, or in general surgeons with a different know-how and style of operating.

In Test B, a single autonomous task execution (with the framework proposed in Chapter 5) under standard environmental conditions is considered. Synthetic task replications are then generated adding low-frequency noise to the original execution, to emulate variability between human users with different expertise and test the robustness of the proposed algorithm with respect to noise.

Test C considers a dataset consisting of only the noiseless standard execution from Test B and three autonomous executions of the task under different environmental conditions, shown in Figures 5.6-5.8. This test shows the performance in the presence of non-homogeneity in the dataset.

Notice that the use of DMPs in the autonomous executions make the kinematic signature very similar to an average one from a human operator. All execution traces (kinematic sensor readings from dVRK robot and point clouds from a Reasense D435 RGB-D camera) are collected through ROS topics to ensure synchronization between videos and kinematics.

It is useful to show how it works with reference to the autonomous execution in 5.7 (occupied pegs). Figure 7.1 shows that the proposed changepoint detection algorithm finds more changepoints (red solid lines) than real ones from manual segmentation (blue lines). However, changepoint filtering exploits fluent detection from video stream to discard wrong changepoints (red dashed lines) when two (or more) consecutive changepoints share the same set of fluents.

The goodness of the identification algorithm is evaluated using standard metrics. Specifically, a rate of the segmentation accuracy is the *matching score*:

$$M_i = \frac{|\cap(t_i, g_i)|}{|g_i|}$$

where t_i is the segment identified by the algorithm, g_i is the real segment corresponding to the i -th action, and $|\cdot|$ denotes the temporal length of a segment. The matching score measures the overlapping between the identified and actual segment, normalized by the length of the actual segment. The results of action classification are quantified using three standard metrics: *precision*, *recall* and *F1-score*. Precision for a specific action class A is defined as:

$$Pr = \frac{TP}{FP + TP}$$

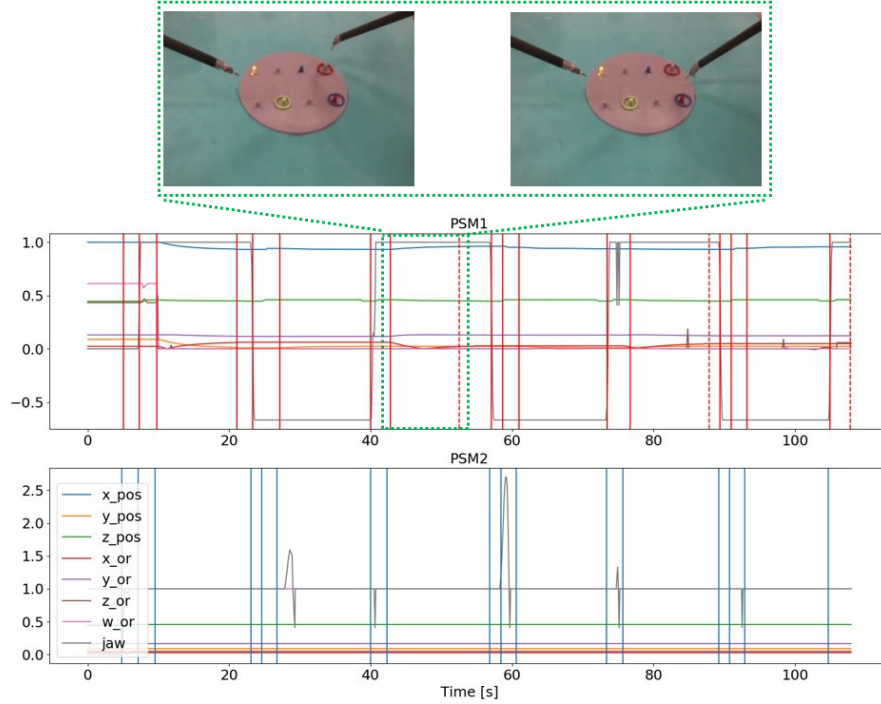


FIGURE 7.1: Kinematic signature (after filtering) and segmentation results for the execution in Figure 5.7. Blue vertical lines represent real changepoints, red solid lines represent changepoints identified by our algorithm, red dashed lines represent changepoints excluded after fluent detection. Frames on the top correspond to two consecutive changepoints sharing the same set of fluents ($\text{on}(\text{ring}, \text{red}, \text{peg}, \text{grey})$, $\text{on}(\text{ring}, \text{blue}, \text{peg}, \text{red})$ and the reachability fluents), so the second one is omitted.

being TP the number of true positives, i.e. segments correctly categorized in A , and FP the number of false positives, i.e. segments wrongly classified as instances of A . Since k for k -NN classification is chosen as the maximum number of occurrences of the most frequent action in the dataset, $FP + TP$ is chosen for the i -th action as $\dim(P)$, where P is the maximum set of segments such that:

- $\dim(P) \geq n_{occ,i}$, where $n_{occ,i}$ is the number of occurrences of the i -th action in the dataset;
- the score of the last segment in P is $s_P = s_{max}$, being s_{max} the score of the $n_{occ,i}$ -th segment in P .

In this way, the scores for less frequent actions are not strongly affected by the results of most frequent ones.

Recall for an action class A is defined as:

$$Rec = \frac{TP}{FN + TP}$$

with FN the number of false negatives, i.e. segments wrongly excluded from class A .

F1-score combines precision and recall as follows:

$$F1 = 2 \frac{Pr \cdot Rec}{Pr + Rec}$$

An evaluation of the computational performance of the proposed identification algorithm with respect to the state of the art is presented in the end of this section.

In order to make a fair comparison, all experimental results are reported with respect to works in the state of the art which address similar issues to the ones presented at the beginning of this chapter and relevant to surgery. In particular, comparison is made to [67], which considers a similar task to ring transfer, i.e. peg transfer, and addresses the problem of identification from small datasets. Moreover, results in [67] allow a comparison on a per-action base, especially for classification quality. An average comparison is also made, finally, with [295], a very recent research addressing the problem of identification in non-homogeneous datasets of ring transfer.

7.3.1 Test A: Homogeneous dataset from human executions

In this test, 9 executions of the ring transfer task in standard environmental conditions, executed by 3 users with different expertise in using the dVRK. Gripper actions and `extract` last 1.05 ± 1.09 s, while `move` actions last 3.69 ± 2.90 s (comparably with peg transfer in [67] on average). The high variance is due to the different expertise of users. Each execution trace contains 36 actions (actions appear multiple times in each execution). Table 7.1 reports the matching scores

TABLE 7.1: Matching scores for Test A. All values are percentages.

Action		Matching score
<code>move(A, ring, C)</code>		85.65
<code>move(A, peg, C)</code>		90.69
<code>move(A, center, C)</code>		82.43
<code>grasp(A, ring, C)</code>		77.05
<code>extract(A, ring, C)</code>		82.73
<code>release(A)</code>		90.18
Average	Proposed method	84.79
	Method in [67]	81.90

(for each action and on average over all actions). The segmentation results are slightly better than [67], though comparable.

In Table 7.2, results of our classification algorithm are reported. $k = 36$ is chosen for k -NN classification, i.e. the number of occurrences of the most frequent action (`release`) in the dataset. As explained in Section 7.2.2, this is the minimum possible value for k to recognize at least all action instances. However, there is no prior guarantee that this is the optimal value for the classification performance. Hence, in Figure 7.2 a preliminary test is made with increasing values of $k \in [36, 56]$ with a step of 2, and the average F1-scores over all actions in the dataset are compared to identify the best choice of k . Results show that $k = 36$ is the optimal value. Hence, also in the following experiments k is chosen as the minimum possible value. Given k , first the full feature array $[\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3]$ presented in Section 7.2.2 is given as input to the classification algorithm. However, this results in poor performance, especially for short actions. Hence, only Boolean features $[\mathbf{f}_2, \mathbf{f}_3]$ are considered for short actions to obtain the results in Table 7.2. Results are significantly improved with respect to [67], whose method suffers from the executions by non-expert users, on the contrary.

TABLE 7.2: Action classification results for Test A. All values are percentages.

Action	Feature array	<i>Pr</i>	<i>Rec</i>	<i>F1</i>
move (A, ring, C)	$[f_1, f_2, f_3]$	100.00	83.33	90.91
	f_1 as in [67]	22.22	22.22	22.22
move (A, peg, C)	$[f_1, f_2, f_3]$	55.56	55.56	55.56
	f_1 as in [67]	19.44	19.44	19.44
move (A, center, C)	$[f_1, f_2, f_3]$	58.33	58.33	58.33
	f_1 as in [67]	25.00	25.00	25.00
grasp (A, ring, C)	$[f_2, f_3]$	41.23	40.00	40.61
	f_1 as in [67]	17.78	22.22	19.66
extract (A, ring, C)	$[f_2, f_3]$	66.67	66.67	66.67
	f_1 as in [67]	30.56	30.56	30.56
release (A)	$[f_2, f_3]$	40.00	40.00	40.00
	f_1 as in [67]	26.67	26.67	26.67
Average	Proposed method	60.30	57.32	58.68
	f_1 as in [67]	23.61	24.35	23.93

7.3.2 Test B: Homogeneous dataset with noise

The dataset for this test consists of an autonomous task execution in standard conditions learned from humans with DMPs, and nine more executions generated adding low-frequency noise to the kinematic signature. This generates an homogeneous dataset with noise, which increases the kinematic variability between different users and requires additional robustness. Noise is generated with power frequency spectrum:

$$S(f) = \beta \frac{1}{f^\lambda}$$

with $\lambda = 7.5$ so that frequencies above the fundamental frequency 1.5 Hz of human hand [201] have power below 0.05β , Hence, they can be neglected. β is related to the noise variance, and varies in the range $[0.01, 0.09]$ with a step of 0.01 to generate the synthetic executions. Table 7.3 reports the matching score considering the full synthetic dataset. The average matching score over

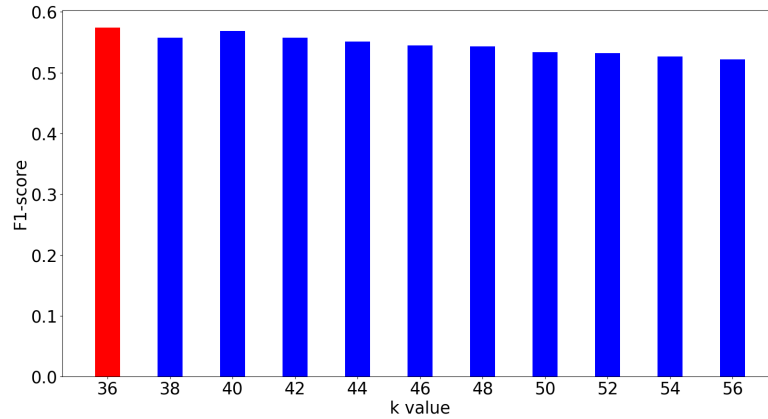


FIGURE 7.2: Average F1-score over all actions for Test A, with increasing k value. The minimum value (red bar) is the optimal one.

all actions with the proposed Algorithm 4 improves the results of [67].

The performance of action classification algorithm is reported in Table 7.4. $k = 36$ (number of occurrences of `release` action) is chosen for k -NN classification. Results are comparable with Test A, though the improvement with respect to [67] is significant only for short actions.

TABLE 7.3: Matching scores for Test B. All values are percentages.

Action	Matching score	
<code>move(A, ring, C)</code>	94.78	
<code>move(A, peg, C)</code>	90.47	
<code>move(A, center, C)</code>	97.53	
<code>grasp(A, ring, C)</code>	76.02	
<code>extract(A, ring, C)</code>	71.80	
<code>release(A)</code>	93.18	
Average		
	Proposed method	87.30
	Method in [67]	81.90

TABLE 7.4: Action classification results for Test B. All values are percentages.

Action	Feature array	<i>Pr</i>	<i>Rec</i>	<i>F1</i>
move (A, ring, C)	$[f_1, f_2, f_3]$	75.00	75.00	75.00
	f_1 as in [67]	75.00	75.00	75.00
move (A, peg, C)	$[f_1, f_2, f_3]$	100.00	100.00	100.00
	f_1 as in [67]	100.00	100.00	100.00
move (A, center, C)	$[f_1, f_2, f_3]$	40.82	55.56	47.06
	f_1 as in [67]	45.00	50.00	47.37
grasp (A, ring, C)	$[f_2, f_3]$	40.00	50.00	44.44
	f_1 as in [67]	17.78	22.22	19.66
extract (A, ring, C)	$[f_2, f_3]$	51.02	69.44	58.82
	f_1 as in [67]	47.37	50.00	48.65
release (A)	$[f_2, f_3]$	40.00	40.00	40.00
	f_1 as in [67]	26.67	26.67	26.67
Average	Proposed method	57.81	65.00	60.89
	f_1 as in [67]	51.97	53.98	52.89

7.3.3 Test C: Non-homogeneous dataset

The dataset for Test C consists of only 4 autonomous executions under non-homogeneous environmental conditions, i.e. the initial conditions of the setup and the action sequence vary between executions. One execution is in standard environmental conditions (36 actions), while 3 other ones are depicted in Figures 5.6-5.8.

Specifically, execution in Figure 5.6 (the ring falls and re-planning is needed) includes 18 actions; execution in Figure 5.7 (occupied colored pegs) includes 17 actions; execution in Figure 5.8 (simultaneous motion of PSMs) includes 12 actions. Actions in each execution repeat multiple times. This is a challenging dataset for unsupervised action identification, since not all actions appear in all executions, and both kinematic and semantic features do not repeat in the same way over the whole dataset. Table 7.5 shows that the matching score is comparable with Test B both for each action and on average, and higher than [67] on average. Results of k -NN classification with $k = 21$ (number of occurrences of

TABLE 7.5: Matching scores for Test C. All values are percentages.

Action		Matching score
<code>move(A, ring, C)</code>		95.36
<code>move(A, peg, C)</code>		92.83
<code>move(A, center, C)</code>		96.35
<code>grasp(A, ring, C)</code>		78.40
<code>extract(A, ring, C)</code>		83.08
<code>release(A)</code>		85.45
Average	Proposed method	88.58
	Method in [67]	81.90

`release` action) are shown in Table 7.6. The proposed method outperforms the scores in [67] (average F1-score rises from 54% to almost 67%). The results for `extract` action are significantly improved, reaching precision of 100% and F1-score of 77% (both scores reach only 12% on the dataset of [67]). Also F1-score and precision (almost double) for `move(A, center, C)` are better. This is even more significant considering that this action is the least frequent in the dataset, appearing only in two scenarios (once in Figure 5.6 and 4 times under standard environmental conditions). A slight decrease in the performance is only recorded for `move(A, ring, C)`, though the F1-score is the highest among all actions.

7.3.4 Computational performance

The computational performance of the proposed action identification algorithm is tested on a PC using 2.6 GHz Intel Core i7-6700HQ CPU (4 cores / 8 threads). The full action identification procedure (segmentation + classification) for one execution trace requires 0.45 s on average (maximum 0.58 s for the standard execution with the highest number of actions). For changepoint filtering, the time required for fluent identification from video frames must be also accounted for. Fluent identification from a single frame requires 0.09 s on average; thus, fluent identification in a full execution trace requires at most 2.88 s for the execution with most actions (standard environmental conditions).

TABLE 7.6: Action classification results for Test C. All values are percentages.

Action	Feature array	<i>Pr</i>	<i>Rec</i>	<i>F1</i>
move (A, ring, C)	$[f_1, f_2, f_3]$	81.82	90.00	85.72
	f_1 as in [67]	90.00	90.00	90.00
move (A, peg, C)	$[f_1, f_2, f_3]$	80.00	80.00	80.00
	f_1 as in [67]	80.00	80.00	80.00
move (A, center, C)	$[f_1, f_2, f_3]$	40.00	40.00	40.00
	f_1 as in [67]	22.22	40.00	28.57
grasp (A, ring, C)	$[f_2, f_3]$	60.00	75.00	66.66
	f_1 as in [67]	56.25	75.00	64.29
extract (A, ring, C)	$[f_2, f_3]$	100.00	62.50	76.92
	f_1 as in [67]	12.50	12.50	12.50
release (A)	$[f_2, f_3]$	52.38	52.38	52.38
	f_1 as in [67]	52.38	52.38	52.38
Average	Proposed method	69.03	66.65	66.95
	f_1 as in [67]	52.23	58.31	54.62

This results in better performance than [67], where 5 s are needed for action identification using a better CPU (3.4 GHz Intel Core i7-3770 with 4 cores / 8 threads). In fact, one main limitation of [67] lies in the use of persistence analysis [76] and dynamic time warping (DTW) [279] to identify changepoints. Persistence analysis removes noise and identifies relevant local minima and maxima in the kinematic features, but still results in spurious changepoints which must be removed with further post-processing through DTW based on Euclidean distance. DTW evaluates the alignment of *all possible* consecutive segments in the kinematic signature, including spurious ones. As also claimed by the authors of [67], this increases computational complexity. Notice that the computational performance also depends on the number of segments to be identified in execution traces.

In [67], 12 actions are involved in a single execution trace (with possibly some repetitions), while the highest number of segments in an execution trace under standard environmental conditions for the experiments in this thesis is 32;

hence, the comparison of computational performance is fair.

The computational performance of the proposed algorithm is also comparable with TSC [krishnan2018transition] ($\approx 1 - 10$ s, though the authors do not provide hardware information and do not consider automatic extraction of visual features, presented as a future work).

7.3.5 Discussion

The proposed algorithm for unsupervised action identification addresses some limitations introduced by the surgical scenario, as identification from datasets including short actions, few executions of the task and possibly non-homogeneous (anomalous) flows of actions. The algorithm has been validated in three different experimental conditions. First, the performance on a limited dataset of 9 executions from humans with different expertise in using dVRK has been evaluated. The algorithm has proved able to significantly improve results in the state of the art under different experimental conditions (but considering only expert executions), doubling F1-score also for short actions.

Then, the robustness of the algorithm has been evaluated on a similar dataset, where a single human execution was augmented with low-frequency kinematic noise (10 executions total). Results were comparable with the previous test, though the improvement with respect to the state of the art was not similarly relevant.

Finally, the algorithm has been able to improve the state of the art (especially for short actions) on a non-homogeneous dataset including only 4 executions and actions appearing rarely (only 2 times).

The average F1-score over all actions is 58 – 66%, comparable or better than state-of-the-art methods based on deep learning [295] (51%) for a similar task to ring transfer. Improvements mainly arise from the inclusion of semantic visual features for classification, which compensate kinematic variability.

In addition, the algorithm exhibits comparable or better computational performance than state of the art, allowing feature extraction and action identification within few seconds.

7.4 From unsupervised action labels to ILP example definition

The action identification algorithm presented in Section 7.2 returns unsupervised labels action_i (with $i \in \{1, \dots, n\}$, being n the number of actions, 6 for the ring transfer task), i.e. without any specific meaning with respect to the semantics of the task. Labels for actions used in Chapter 6, on the contrary, contain information about the specific instance of action inside the example. This allows to infer ASP axioms which relate relevant variables appearing in environmental and action fluents, and it is necessary for proper definition of the ILP problem. For better clarity, consider the scenario in Figure 5.7. The first action of the autonomous execution is `move(psm1, ring, red)`, associated to the following ILP example:

```
#pos{ex1, {move(psm1, ring, red)}, {}} (7.2)
{reachable(psm1, ring, red), reachable(psm1, ring, blue),
reachable(psm1, peg, red), reachable(psm1, peg, blue),
reachable(psm2, peg, green), reachable(psm2, peg, yellow),
reachable(psm1, peg, grey), reachable(psm2, peg, grey),
on(ring, red, peg, blue), on(ring, blue, peg, red),
on(ring, green, peg, green)}}
```

While the fifth action is `move(psm1, ring, blue)`, associated to the following ILP example:

```
#pos{ex2, {move(psm1, ring, blue)}, {}} (7.3)
{reachable(psm1, ring, red), reachable(psm1, ring, blue),
reachable(psm1, peg, red), reachable(psm1, peg, blue),
reachable(psm2, peg, green), reachable(psm2, peg, yellow),
reachable(psm1, peg, grey), reachable(psm2, peg, grey),
on(ring, red, peg, grey), on(ring, blue, peg, red),
on(ring, green, peg, green)}}
```

Unsupervised action segmentation clusters both actions together with the same label, say action_1 . This makes the ILP problem ill-defined, since examples cannot be undistinguished, hence pre-condition $\text{action}_1 :- \text{reachable}(A, \text{ring}, C)$ as in (5.1) cannot be inferred. This section is to propose a methodology to overcome this limitation, towards a feasible approach for unsupervised ILP applied to the ring transfer case study. The goal is to enrich the semantics of unsupervised labels action_i with argument (logic) variables characterizing the single actions. The first observation is that actions in ASP command specific gestures to either PSM1 or PSM2, thus all action labels must have the variable Arm as an argument, resulting in $\text{action}_i(A)$. Notice that $\text{move}(A, \text{center}, C)$ action actually corresponds to the simultaneous motion of the PSMs to the center of the peg base for transfer. As a consequence, the unsupervised action label shall be $\text{action}_i(A1, A2)$. However, this is still not enough to solve the issue for examples (7.2)-(7.3). In fact, instances of actions in both examples are executed by PSM1, hence they still remain undistinguished. As a second observation, it is reasonable to assume that relevant argument variables should be the ones which are influenced by the occurrence of actions. Hence, it is possible to retrieve them from environmental fluents which are modified after the action has been executed. Specifically, fluents which have the variable Arm as an argument are of interest, since they are explicitly related to the action of one PSM. Consider again the examples (7.2)-(7.3). After the $\text{move}(A, \text{ring}, C)$ actions are executed, the $\text{grasp}(A, \text{ring}, C)$ actions occur, generating the following ILP examples:

```
#pos{ex3, {grasp(psm1, ring, red)}, {}                                     (7.4)
{reachable(psm1, ring, red), reachable(psm1, ring, blue),
reachable(psm1, peg, red), reachable(psm1, peg, blue),
reachable(psm2, peg, green), reachable(psm2, peg, yellow),
reachable(psm1, peg, grey), reachable(psm2, peg, grey),
on(ring, red, peg, blue), on(ring, blue, peg, red),
on(ring, green, peg, green), at(psm1, ring, red)}
```

```

#pos{ex4, {grasp(psm1,ring,blue)}, {} (7.5)
{reachable(psm1,ring,red), reachable(psm1,ring,blue),
reachable(psm1,peg,red), reachable(psm1,peg,blue),
reachable(psm2,peg,green), reachable(psm2,peg,yellow),
reachable(psm1,peg,greyscale), reachable(psm2,peg,greyscale),
on(ring,red,peg,greyscale), on(ring,blue,peg,red),
on(ring,green,peg,green), at(psm1,ring,blue)}}

```

Analyzing the contexts of examples (7.4)-(7.5), the only changing environmental fluent with `Arm` variable as its argument is `at(A, ring, C)` (`at(psm1, ring, red)` in (7.4), `at(psm1, ring, blue)` in (7.5)). Hence, the unsupervised action labels shall be enriched as `action1(psm1, red)`, `action1(psm1, blue)` for examples (7.2)-(7.3), respectively. In a similar way, it is possible to identify the relevant environmental fluents which change after the execution of other actions, as resulting from the execution traces for experiments in Chapter 6:

- `move(A, peg, C): at(A, peg, C), at(A, ring, C)` (only after transfer), resulting in unsupervised label `action2(A, C)`;
- `move(A, center, C): at(A, center), at(A, ring, C), at(A1, ring, C)`, resulting in unsupervised label `action3(A, A1, C)`;
- `extract(A, ring, C): on(ring, C, peg, C1)` (neglected since it does not contain the variable `Arm`), `at(A, ring, C)`, resulting in unsupervised label `action4(A, C)`;
- `grasp(A, ring, C): in_hand(A, ring, C), closed_gripper(A)`, resulting in unsupervised label `action5(A, C)`;
- `release(A): in_hand(A, ring, C), on(ring, C, peg, C1)` (ignored), `closed_gripper(A)`, resulting in unsupervised label `action6(A, C)`.

Notice that unsupervised labels depend on the same argument variables as original labels from supervised ILP. Hence, the ILP approach presented in Chapter 6 returns identical results even with unsupervised labels. The only exceptions are

actions `release(A)` and `move(A, center, C)`, with unsupervised labels having an additional argument variable, `action6(A, C)` and `action3(A, A1, C)`, respectively. However, axioms involving `release(A)` are the pre-condition `release(A) :- in_hand(A, ring, C)` in (6.4), and the effect axioms 6.6c-6.10a-b-e. They all already depend on the variable `Color`, hence the ILP results are identical even with unsupervised labeling. As for `move(A, center, C)`, it appears in the pre-condition `move(A, center, C) :- in_hand(A, ring, C)` of (6.4), in (6.5)c, and in effect axioms 6.6e-f and 6.10f. All of these axioms are not affected when the unsupervised label `action3(A, A1, C)` is introduced, since the original label appears as a safe atom. Only the pre-condition needs a slight modification, since axiom `action3(A, A1, C) :- in_hand(A, ring, C)` is not safe because of `A1`. Running ILASP for this specific action results in the following new pre-condition:

```
action3(A, A1, C, t) :- reachable(A1, peg, C, t),
closed_gripper(A, t)
```

This is equivalent to the former pre-condition: in fact, it prescribes that `move(A, center, C)` is possible when `A` is holding some ring and the peg with color `C` is reachable only with the other PSM (hence transfer is needed).

As a final remark, the proposed approach to unsupervised ILP does not take into account negative examples, which usually lead to learning executability constraints. As explained in Section 6.4.3, negative examples represent forbidden actions which are manually annotated by expert users. Then, they cannot be directly identified with unsupervised methods in execution traces. The problem of automatically generating negative examples for ILP applications has been faced by several researchers. One simple approach [294] relies on the closed-world assumption, and considers as negative examples all occurrences which are not covered by positive examples. However, this approach is computationally inefficient in non-trivial scenarios, since many examples have to be generated. Moreover, it is non-feasible in most realistic applications, since it assumes only one action is possible in any environmental conditions. For instance, consider the ring transfer scenario in Figure 5.6. Given that the first action to be executed in `move(psm1, ring, red)`, the following negative examples should be generated

under the closed-world assumption:

```

#neg{ex5, {move(psm1, ring, blue)}, {}                                (7.6)
{reachable(psm1, ring, red), reachable(psm2, ring, blue),
reachable(psm1, peg, red), reachable(psm1, peg, blue),
reachable(psm2, peg, green), reachable(psm2, peg, yellow),
reachable(psm1, peg, grey), reachable(psm2, peg, grey),
on(ring, red, peg, grey)}}

#neg{ex6, {move(psm1, ring, green)}, {}
{reachable(psm1, ring, red), reachable(psm2, ring, blue),
reachable(psm1, peg, red), reachable(psm1, peg, blue),
reachable(psm2, peg, green), reachable(psm2, peg, yellow),
reachable(psm1, peg, grey), reachable(psm2, peg, grey),
on(ring, red, peg, grey)}}

#neg{ex7, {move(psm1, ring, yellow)}, {}
{reachable(psm1, ring, red), reachable(psm2, ring, blue),
reachable(psm1, peg, red), reachable(psm1, peg, blue),
reachable(psm2, peg, green), reachable(psm2, peg, yellow),
reachable(psm1, peg, grey), reachable(psm2, peg, grey),
on(ring, red, peg, grey)}}

#neg{ex8, {move(psm2, ring, _)}, {}
{reachable(psm1, ring, red), reachable(psm2, ring, blue),
reachable(psm1, peg, red), reachable(psm1, peg, blue),
reachable(psm2, peg, green), reachable(psm2, peg, yellow),
reachable(psm1, peg, grey), reachable(psm2, peg, grey),
on(ring, red, peg, grey)}}

```

These examples exclude the axiom $0\{\text{move}(A, \text{ring}, C) : \text{reachable}(A, \text{ring}, C)\}1$ in (6.4) from the feasible hypotheses. In fact, it is in contradiction with the negative example $\text{ex}:8$, forbidding the motion of PSM2 to the blue

ring, though it is reachable. A more computationally efficient solution is proposed in [185], generating negative examples by randomly modifying variables in positive examples. Similarly, in [4] negative examples are first generated randomly, then omitted if they do not lead to learning more specific hypotheses. This approach still does not solve the issue evidenced with examples in (7.6), since negative examples may be generated which prevent the learning of correct axioms, especially when aggregate axioms are in the target hypothesis as in the ring transfer case study. Very recent research in [59] has investigated the implementation of generative policy models for autonomous agency in unexperienced environment. Policies expressed in ASP syntax are generated and evaluated online by an oracle, marking them as feasible or not. Infeasible policies are used as negative examples in an ILP framework based on ILASP to refine available task knowledge and generate an acceptable strategy. This methodology appears the most correct approach to the problem of automatic generation of negative examples in an unsupervised ILP framework. Moreover, experiments in Section 6.4.4 have shown the possibility of hypothesis refinement with ILASP as new examples are acquired. In a surgical context, the autonomous robotic system may query a supervisory expert surgeon when unexperienced situations occur, and learn online from his choice for future executions.

7.5 Conclusion

This chapter has presented an approach to unsupervised ILP for the ring transfer task, and the surgical scenario in general, exploiting automatic recognition of environmental features and action identification. This solves issues of supervised ILP, related to the tedious and error-prone labeling of long execution traces.

In particular, a novel algorithm has been proposed for action identification, which combines kinematic and semantic visual features to identify clusters of actions in execution traces. Presented experiments highlight a number of advantages of the algorithm with respect to state-of-the-art approaches.

The algorithm can be generalized to different robotic (surgical) tasks, defining appropriate environmental fluents to be extracted from the video stream of the executions.

Chapter 8

Conclusion

8.1 Summary of results

This thesis has faced the problem of reaching level 2 of autonomy in robotic surgery, as of the classification of autonomy levels described in [336]. The goal is the automation of repetitive and often tedious surgical steps, in order to reduce surgeon's fatigue, improve recovery time for the patient with more precise motion, and optimize hospital resource usage. Specifically, in the first part of the thesis the task planning problem has been analyzed, which is a fundamental functionality of general-purpose deliberative robots. The paradigmatic scenario of the ring transfer training task for novice surgeons with dVRK robot has been considered for validation. The framework for autonomous surgical step execution presented in this thesis implements for the first time a task planning module based on logic programming, in detail answer set programming (ASP). This allows to address some major issues of state-of-the-art applications:

- *explainability* of the surgical workflow, helpful for monitoring purposes and to guarantee reliability of the overall surgical system;
- *adaptability* of the surgical workflow in presence of dynamic and partially unknown environmental conditions, thanks to the non-monotonicity of knowledge representation in ASP;
- the possibility to encode surgical knowledge as directly retrieved from experts, hence representing the *reasoning process* of the surgeon, rather than an explicit enumeration of possible surgical workflows.

Moreover, an extended literature review of logic programming formalisms, together with experiments on the ring transfer task, has shown the feasibility of ASP for most practically relevant robotic scenarios, even when temporal relations between domain entities are to be expressed (e.g., arbitrary delays between actions and effects).

In the second part of the thesis, the problem of surgical knowledge learning has been studied. An approach based on inductive logic programming (ILP) under the answer set semantics with benchmark tool ILASP has been proposed for the first time in the surgical context. The advantages of ILP with respect to state-of-the-art learning approaches for surgery are:

- *limited learning time* (≈ 180 s maximum) of full task knowledge for the ring transfer;
- the need for a *very limited training dataset*, specifically only four incomplete executions of the ring transfer task. This solves the problem of the lack of training data for surgery;
- learning of *interpretable* human-readable knowledge for reliability.

Moreover, ILP has shown to improve the efficiency of hand-written ASP specifications in terms of length of axioms, thus significantly reducing the plan computation time even for a complex task description involving delayed effects of actions. This guarantees the applicability of learned ASP encoding to real robotic setups, with a maximum planning time of approximately 10 s even for long sequences of actions, when both PSMs of dVRK can move simultaneously.

Finally, for the first time (to the best of the knowledge of the author) ILP has been successfully applied to complex learning tasks in robotics, involving multiple agents and actions, and temporal constraints under the paradigm of event calculus. In order to overcome the limitation of annotated executions of the task needed for ILP, an unsupervised learning approach has also been proposed, based on a novel unsupervised action identification algorithm relying on semantic environmental features extracted from videos of example executions. The proposed algorithm is able to improve the performance of state-of-the-art approaches to unsupervised surgical action identification, addressing for the first time issues including:

- *limited datasets of actions*, including executions which present increasing hand-frequency noise to emulate different expertise of surgeons;
- *non-homogeneous datasets* with non-uniform action sequences under various environmental conditions, hence lowering the performance of identification algorithms based only on kinematics;
- identification of actions with *short duration* (e.g., actions of the grippers).

The proposed algorithm for action identification is also able to outperform state of the art in terms of computational efficiency.

8.2 Discussion and future works

The proposed framework for autonomous surgical task execution has been validated only on one standard training task for novice surgeons from the Fundamentals of Laparoscopic Surgery (FLS) [299]. The considered ring transfer task involves several challenges of real surgery, as dual-arm coordination, motion in constrained (possibly with obstacles) space, optimization of the solving strategy depending on user-defined (or patient-specific) criteria, reasoning on dynamic environmental conditions due to the continuous interaction with the environment.

However, the ring transfer domain involves a rigid environment, hence the proposed strategy for motion control in the framework only accounts for pose control. Other standard tasks from FLS, as suturing, require interaction control based on force feedback, in order to cope with the deformability of real anatomical environment.

Though DMPs can be easily extended to account for force feedback from the environment, using the framework of Associative Skill Memories [255], the main limitation lies in the lack of accurate force feedback from the dVRK. In fact, positioning force sensors directly at the tip of surgical instruments would be infeasible both from a mechanical point of view (instruments are usually small and offer very little space for sensor integration), and considering the safety of the patient (sensors cannot be close to anatomical parts).

A possible solution is using current / torque readings from built-in sensors of dVRK at joint level, and translating them to force information at the tip of

instruments through a dynamical model of the robotic system. Recently, the dynamical model of dVRK has been estimated in [264], achieving an error of at most 15% in the estimation of force / torque at the end effectors, when compared with actual sensors on the tips of the instruments. Implementing force feedback would lead the way to the validation of the proposed framework to other tasks from FLS, which is a major development of the work in this thesis.

It is important to remark, however, that the limitation of the applicability of the framework depends mainly from low-level control aspects. The task-level reasoning system and context interpretation, which are the main proposed contributions in this thesis, remain valid and can be extended to other tasks, considering only a different semantic description of the new domain.

Other research problems remain to be investigated after the results of this thesis. A first limitation regards the kinematic precision of the autonomous system. In fact, though a state-of-the-art localization precision (≈ 1.6 mm) is reached with accurate hand-eye calibration of PSMs and the camera, an error is registered in arm positioning, which may become infeasible in precision surgery. This is due to the very small scale of the setup for surgical applications, and to the intrinsic kinematic accuracy of the dVRK, which has been recently proved to have an average accuracy in localization of fiducial markers of 1 mm on average (2.7 mm maximum error) [126, 172]. Moreover, the use of a standard surgical endoscope with smaller baseline and depth range will further deteriorate the performance. In order to solve this issue, the implementation of visual servoing techniques is needed to guarantee accurate positioning even in the presence of localization errors, exploiting continuous visual information from the camera. Most efficient visual servoing techniques in robotics currently generate velocity commands to match visual features during the execution, hence without being affected from model inaccuracy [304, 135]. The main limitation of this approach is that dVRK does not currently implement velocity control for the instruments of PSMs.

Subsequently, a comparison with human performance will be needed to assess the performance of the autonomous surgical system towards its application in a real surgical scenario. Currently used metrics in the surgical context are speed of execution, smoothness of motion trajectories and deviation from human

demonstration [317], number of errors and workspace range [8].

Another important metric is the planning time, i.e. the delay between the beginning of the task execution session and the actual beginning of the motion of either one of the robotic arms. While it is easy to evaluate the planning time for the autonomous framework from the output of Clingo, the planning time of a human user cannot be derived in a straightforward way. When testing the autonomous framework in Chapter 5, some very preliminary experiments in this direction have been conducted, with promising results in terms of real-time capabilities. However, further and more robust validation is needed, especially making the evaluation of the human planning time more rigorous.

This thesis has approached the problem of surgical knowledge learning using ILP. This method has shown a number of advantages with respect to other standard machine learning tools. However, several other frameworks exist in the context of inductive learning. In particular, though ILP has been shown to successfully refine knowledge as new examples are available or unconventional situations occur, a comparison with hybrid methods as semantic reinforcement learning (SRL) [302, 177, 43] will be interesting to investigate. SRL is designed for real-time knowledge refinement, generalizing over new labeled examples, e.g. from human instructions. SRL has been recently shown to guarantee proofs of correctness and reliability in human-robot interaction, with accuracy performance gracefully degrading as noise in the observations increases [302]. Notice that also ILASP tool for ILP used in this thesis allows to deal with noise in the examples [180]. However, the level of noise of one example is simply defined as a penalty which increases the cost of learning axioms satisfying that example. On the contrary, SRL provides a more statistically relevant mechanism to account for noisy observations, in the context of classical statistical and reinforcement learning. Hence, it is possible to integrate ILP- and SRL-based learning to incrementally refine surgical task knowledge.

Finally, the unsupervised learning approach presented in Chapter 7 still relies on existing knowledge of the surgical scenario, encoded in the environmental (anatomical) description. Though the research community agrees that ontologies encoding such knowledge are needed for the surgical operating room of the future [110], it is unrealistic to assume that all relevant features for surgery may

be perfectly formally described and standardized. Hence, one interesting advance of the here presented work is the comparison and integration of reliable ILP approach with unsupervised event classification and recognition techniques, recently proposed in the context of human activity recognition [99, 301].

Bibliography

- [1] Martin Abadi and Zohar Manna. “Temporal logic programming”. In: *Journal of symbolic computation* 8.3 (1989), pp. 277–295.
- [2] Weronika T Adrian et al. “The ASP system DLV: advancements and applications”. In: *KI-Künstliche Intelligenz* 32.2 (2018), pp. 177–179.
- [3] Felicidad Aguado et al. “Temporal equilibrium logic: a survey”. In: *Journal of Applied Non-Classical Logics* 23.1-2 (2013), pp. 2–24.
- [4] Naveed Akhtar et al. “Towards iterative learning of autonomous robots using ILP”. In: *2011 15th International Conference on Advanced Robotics (ICAR)*. IEEE. 2011, pp. 409–414.
- [5] Jose Julio Alferes et al. “Planning as abductive updating”. In: *Proc. of the Symposium on AI Planning and Intelligent Agents (AISB’00)*. 2000, pp. 1–8.
- [6] Dalal Alrajeh et al. “Extracting requirements from scenarios with ILP”. In: *International Conference on Inductive Logic Programming*. Springer. 2006, pp. 64–78.
- [7] Mario Alviano et al. “Magic sets for disjunctive datalog programs”. In: *Artificial Intelligence* 187 (2012), pp. 156–192.
- [8] Tarek Alzahrani et al. “Validation of the da Vinci Surgical Skill Simulator across three surgical disciplines: a pilot study”. In: *Canadian Urological Association Journal* 7.7-8 (2013), E520.
- [9] Saeid Amiri, Mohammad Shokrolah Shirazi, and Shiqi Zhang. “Learning and Reasoning for Robot Sequential Decision Making under Uncertainty”. In: *arXiv preprint arXiv:1901.05322* (2019).

-
- [10] Beatrice van Amsterdam et al. “Weakly supervised recognition of surgical gestures”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 9565–9571.
- [11] Benjamin Andres et al. “Integrating ASP into ROS for reasoning in robots”. In: *International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer. 2015, pp. 69–82.
- [12] Laura Antanas et al. “Semantic and geometric reasoning for robotic grasping: a probabilistic logic approach”. In: *Autonomous Robots* 43.6 (2019), pp. 1393–1418.
- [13] Franz Baader et al. *The description logic handbook: Theory, implementation and applications*. Cambridge university press, 2003.
- [14] Joseph Babb and Joohyung Lee. “Action language BC+”. In: *Journal of Logic and Computation* (2015).
- [15] Marcello Balduccini. “Learning Action Descriptions with A-Prolog: Action Language C”. In: *AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning*. Mar. 2007.
- [16] H. S. Bank, S. D’Souza, and A. Rasam. “Temporal Logic (TL)-Based Autonomy for Smart Manufacturing Systems”. In: *Procedia Manufacturing*. Vol. 26. 2018, pp. 1221–1229.
- [17] Chitta Baral, Michael Gelfond, and Nelson Rushton. “Probabilistic reasoning with answer sets”. In: *International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer. 2004, pp. 21–33.
- [18] Chitta Baral and Tran Cao Son. ““Add Another Blue Stack of the Same Height!”: ASP Based Planning and Plan Failure Analysis”. In: *International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer. 2015, pp. 127–133.
- [19] Chitta Baral and Jicheng Zhao. “Non-monotonic Temporal Logics for Goal Specification.” In: *IJCAI*. Vol. 7. 2007, pp. 236–242.

-
- [20] Michael Beetz, Lorenz Mösenlechner, and Moritz Tenorth. “CRAM—A Cognitive Robot Abstract Machine for everyday manipulation in human environments”. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2010, pp. 1012–1017.
- [21] Michael Beetz et al. “Cognition-enabled autonomous robot control for the realization of home chore task intelligence”. In: *Proceedings of the IEEE* 100.8 (2012), pp. 2454–2471.
- [22] Michael Beetz et al. “Towards performing everyday manipulation activities”. In: *Robotics and Autonomous Systems* 58.9 (2010), pp. 1085–1095.
- [23] Richard Bellman. “On a routing problem”. In: *Quarterly of applied mathematics* 16.1 (1958), pp. 87–90.
- [24] Michael W Berry and Malu Castellanos. “Survey of text mining”. In: *Computing Reviews* 45.9 (2004), p. 548.
- [25] Pierre Berthet-Rayne et al. “Hubot: A three state Human-Robot collaborative framework for bimanual surgical tasks based on learned models”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 715–722.
- [26] Riccardo Bertolucci et al. “An ASP-based framework for the manipulation of articulated objects using dual-arm robots”. In: *International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer. 2019, pp. 32–44.
- [27] D. Beßler, M. Pomarlan, and M. Beetz. “Owl-enabled assembly planning for robotic agents”. In: *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*. Vol. 3. 2018, pp. 1684–1692.
- [28] Meghyn Bienvenu, Christian Fritz, and Sheila A McIlraith. “Planning with Qualitative Temporal Preferences.” In: *KR* 6 (2006), pp. 134–144.
- [29] Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of satisfiability*. Vol. 185. IOS press, 2009.
- [30] Roderick Bloem et al. “Synthesis of reactive (1) designs”. In: *Journal of Computer and System Sciences* 78.3 (2012), pp. 911–938.

- [31] Tobias Blum et al. “Modeling and online recognition of surgical phases using hidden markov models”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2008, pp. 627–635.
- [32] David Bouget et al. “Detecting surgical tools by modelling local appearance and global shape”. In: *IEEE transactions on medical imaging* 34.12 (2015), pp. 2603–2617.
- [33] Julian Bradfield and Colin Stirling. “Modal mu-calculi”. In: *Handbook of modal logic* 3 (2007), pp. 721–756.
- [34] Gerhard Brewka, Ilkka Niemela, and Miroslaw Truszczyński. “Preferences and nonmonotonic reasoning”. In: *AI magazine* 29.4 (2008), pp. 69–69.
- [35] E Brodsky and Boris S Darkhovsky. *Nonparametric methods in change point problems*. Vol. 243. Springer Science & Business Media, 2013.
- [36] Barbara Bruno et al. “The CARESSES EU-Japan project: making assistive robots culturally competent”. In: *Italian Forum of Ambient Assisted Living*. Springer. 2017, pp. 151–169.
- [37] Pedro Cabalar, Manuel Rey, and Concepcion Vidal. “A Complete Planner for Temporal Answer Set Programming”. In: *EPIA Conference on Artificial Intelligence*. Springer. 2019, pp. 520–525.
- [38] Pedro Cabalar et al. “telingo= ASP+ Time”. In: *International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer. 2019, pp. 256–269.
- [39] Francesco Calimeri et al. “ASP-Core-2 input language format”. In: *Theory and Practice of Logic Programming* 20.2 (2020), pp. 294–309.
- [40] A. Camacho et al. “Non-deterministic planning with temporally extended goals: LTL over finite and infinite traces”. In: *31st AAAI Conference on Artificial Intelligence, AAAI 2017*. 2017, pp. 3716–3724.
- [41] David B Camarillo, Thomas M Krummel, and J Kenneth Salisbury Jr. “Robotic technology in surgery: past, present, and future”. In: *The American Journal of Surgery* 188.4 (2004), pp. 2–15.

- [42] Michael Cashmore et al. “Rosplan: Planning in the robot operating system”. In: *Twenty-Fifth International Conference on Automated Planning and Scheduling*. 2015.
- [43] Joyce Y Chai et al. “Language to Action: Towards Interactive Task Learning with Physical Agents.” In: *IJCAI*. 2018, pp. 2–9.
- [44] Katia Charrière et al. “Real-time analysis of cataract surgery videos using statistical models”. In: *Multimedia Tools and Applications* 76.21 (2017), pp. 22473–22491.
- [45] Jie Chen and Arjun K Gupta. *Parametric statistical change point analysis: with applications to genetics, medicine, and finance*. Springer Science & Business Media, 2011.
- [46] Jie Chen et al. “Towards transferring skills to flexible surgical robots with programming by demonstration and reinforcement learning”. In: *2016 Eighth International Conference on Advanced Computational Intelligence (ICACI)*. IEEE. 2016, pp. 378–384.
- [47] Xiaoping Chen et al. “Toward open knowledge enabling for human-robot interaction”. In: *Journal of Human-Robot Interaction* 1.2 (2013), pp. 100–117.
- [48] B.J. Choi, J. Park, and C.H. Park. “Formal verification for human-robot interaction in medical environments”. In: *ACM/IEEE International Conference on Human-Robot Interaction*. 2021, pp. 181–185.
- [49] Der-Lin Chow et al. “A novel vision guided knot-tying method for autonomous robotic surgery”. In: *2014 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE. 2014, pp. 504–508.
- [50] Keith L Clark. “Negation as failure”. In: *Logic and data bases*. Springer, 1978, pp. 293–322.
- [51] M. Colledanchise, R. M. Murray, and P. Ogren. “Synthesis of correct-by-construction behavior trees”. In: *IEEE International Conference on Intelligent Robots and Systems*. Vol. 2017-September. 2017, pp. 6039–6046.

- [52] Alain Colmerauer. “An introduction to Prolog III”. In: *Computational Logic*. Springer. 1990, pp. 37–79.
- [53] F Corcione et al. “Advantages and limits of robot-assisted laparoscopic surgery: preliminary experience”. In: *Surgical Endoscopy and Other Interventional Techniques* 19.1 (2005), pp. 117–119.
- [54] CJ Coulson et al. “An autonomous surgical robot for drilling a cochleostomy: preliminary porcine trial”. In: *Clinical Otolaryngology* 33 (2008), pp. 343–347.
- [55] J. Crespo et al. “Reasoning Systems for Semantic Navigation in Mobile Robots”. In: *IEEE International Conference on Intelligent Robots and Systems*. 2018, pp. 5654–5659.
- [56] Andrew Cropper and Stephen H Muggleton. “Learning efficient logic programs”. In: *Machine Learning* 108.7 (2019), pp. 1063–1083.
- [57] James L Crowley, Patrick Reignier, and Sebastien Pesnel. “Context Aware Vision using Image-based Active Recognition”. In: (2004).
- [58] G. Cui, W. Shuai, and X. Chen. “Semantic task planning for service robots in open worlds”. English. In: *Future Internet* 13.2 (2021), pp. 1–19. URL: www.scopus.com.
- [59] Daniel Cunningham et al. “A generative policy model for connected and autonomous vehicles”. In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE. 2019, pp. 1558–1565.
- [60] Giuseppe De Giacomo and Moshe Vardi. “Synthesis for LTL and LDL on finite traces”. In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*. 2015.
- [61] Giuseppe De Giacomo and Moshe Y Vardi. “Linear temporal logic and linear dynamic logic on finite traces”. In: *Twenty-Third International Joint Conference on Artificial Intelligence*. 2013.
- [62] Luc De Raedt and Luc Dehaspe. “Learning from satisfiability”. In: *Ninth Dutch Conference on Artificial Intelligence (NAIC'97)*. 1997, pp. 303–312.

- [63] Luc De Raedt and Kristian Kersting. “Probabilistic inductive logic programming”. In: *Probabilistic Inductive Logic Programming*. Springer, 2008, pp. 1–27.
- [64] Luc De Raedt and Wim Van Laer. “Inductive constraint logic”. In: *International Workshop on Algorithmic Learning Theory*. Springer. 1995, pp. 80–94.
- [65] Olga Dergachyova, Xavier Morandi, and Pierre Jannin. “Knowledge transfer for surgical activity prediction”. In: *International journal of computer assisted radiology and surgery* 13.9 (2018), pp. 1409–1417.
- [66] Anna M Derossis et al. “Development of a model for training and evaluation of laparoscopic skills”. In: *The American journal of surgery* 175.6 (1998), pp. 482–487.
- [67] Fabien Despinoy et al. “Unsupervised trajectory segmentation for surgical gesture recognition in robotic training”. In: *IEEE Transactions on Biomedical Engineering* 63.6 (2015), pp. 1280–1291.
- [68] Mohammed Diab et al. “PMK—A Knowledge Processing Framework for Autonomous Robotics Perception and Manipulation”. In: *Sensors* 19.5 (2019), p. 1166.
- [69] V Diekert and P Gastin. “First-order definable languages”. In: *Logic and Automata: History and Perspectives* (), pp. 261–306.
- [70] Yannis Dimopoulos, Bernhard Nebel, and Jana Koehler. “Encoding planning problems in nonmonotonic logic programs”. In: *European Conference on Planning*. Springer. 1997, pp. 169–181.
- [71] Robert DiPietro et al. “Recognizing surgical activities with recurrent neural networks”. In: *International conference on medical image computing and computer-assisted intervention*. Springer. 2016, pp. 551–558.
- [72] Jürgen Dix, Ugur Kuter, and Dana Nau. “Planning in answer set programming using ordered task decomposition”. In: *Annual Conference on Artificial Intelligence*. Springer. 2003, pp. 490–504.

- [73] Alexandre Donzé and Oded Maler. “Robust satisfaction of temporal logic over real-valued signals”. In: *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer. 2010, pp. 92–106.
- [74] Christian Dornhege et al. “Semantic attachments for domain-independent planning systems”. In: *Nineteenth International Conference on Automated Planning and Scheduling*. 2009.
- [75] Wklodzimierz Drabent. “Completeness of SLDNF-resolution for non-floundering queries”. In: *The Journal of logic programming* 27.2 (1996), pp. 89–106.
- [76] Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. “Topological persistence and simplification”. In: *Proceedings 41st annual symposium on foundations of computer science*. IEEE. 2000, pp. 454–463.
- [77] Thomas Eiter et al. “A deductive system for non-monotonic reasoning”. In: *International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer. 1997, pp. 363–374.
- [78] Thomas Eiter et al. “Declarative problem-solving using the DLV system”. In: *Logic-based artificial intelligence*. Springer, 2000, pp. 79–103.
- [79] Thomas Eiter et al. “The DLVHEX system”. In: *KI-Künstliche Intelligenz* 32.2-3 (2018), pp. 187–189.
- [80] E Allen Emerson. “Temporal and modal logic”. In: *Formal Models and Semantics*. Elsevier, 1990, pp. 995–1072.
- [81] E Allen Emerson and Edmund M Clarke. “Using branching time temporal logic to synthesize synchronization skeletons”. In: *Science of Computer programming* 2.3 (1982), pp. 241–266.
- [82] Esra Erdem, Michael Gelfond, and Nicola Leone. “Applications of answer set programming”. In: *AI Magazine* 37.3 (2016), pp. 53–68.
- [83] Esra Erdem and Volkan Patoglu. “Applications of action languages in cognitive robotics”. In: *Correct Reasoning*. Springer, 2012, pp. 229–246.
- [84] Esra Erdem and Volkan Patoglu. “Applications of ASP in Robotics”. In: *KI-Künstliche Intelligenz* 32.2-3 (2018), pp. 143–149.

- [85] Kutluhan Erol. “Hierarchical task network planning: formalization, analysis, and implementation”. PhD thesis. 1996.
- [86] D. Escudero-Rodrigo and R. Alquezar. *Study of the anchoring problem in generalist robots based on ROSPlan*. Vol. 288. Frontiers in Artificial Intelligence and Applications. 2016, pp. 45–50.
- [87] B Estebanez et al. “Maneuvers recognition in laparoscopic surgery: Artificial Neural Network and hidden Markov model approaches”. In: *2012 4th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*. IEEE. 2012, pp. 1164–1169.
- [88] H. FakhruLdeen et al. “Human robot cooperation planner using plans embedded in objects”. English. In: *IFAC-PapersOnLine* 49.21 (2016), pp. 668–674.
- [89] Andreas Falkner et al. “Industrial applications of answer set programming”. In: *KI-Künstliche Intelligenz* 32.2-3 (2018), pp. 165–176.
- [90] Mahtab Jahanbani Fard et al. “Soft boundary approach for unsupervised gesture segmentation in robotic-assisted surgery”. In: *IEEE Robotics and Automation Letters* 2.1 (2016), pp. 171–178.
- [91] A. Farinelli, A. Finzi, and T. Lukasiewicz. “Team programming in golog under partial observability”. In: *IJCAI International Joint Conference on Artificial Intelligence*. 2007, pp. 2097–2102.
- [92] Maria-Camilla Fiazza and Paolo Fiorini. “Design for Interpretability: Meeting the Certification Challenge for Surgical Robots”. In: *2021 IEEE International Conference on Intelligence and Safety for Robotics (ISR)*. IEEE. 2021, pp. 264–267.
- [93] Cameron Finucane, Gangyuan Jing, and Hadas Kress-Gazit. “LTLMoP: Experimenting with language, temporal logic and robot control”. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2010, pp. 1988–1993.
- [94] Michael J Fischer and Richard E Ladner. “Propositional dynamic logic of regular programs”. In: *Journal of computer and system sciences* 18.2 (1979), pp. 194–211.

- [95] Jeremy Frank and Ari Jonsson. “Constraint-based attribute and interval planning”. In: *Constraints* 8.4 (2003), pp. 339–364.
- [96] Simone Fratini et al. “Apsi-based deliberation in goal oriented autonomous controllers”. In: *ASTRA* 11 (2011).
- [97] Thom Frühwirth. “Theory and practice of constraint handling rules”. In: *The Journal of Logic Programming* 37.1-3 (1998), pp. 95–138.
- [98] Alfredo Gabaldon. “Precondition Control and the Progression Algorithm.” In: *ICAPS*. 2004, pp. 23–32.
- [99] Chuang Gan et al. “Concepts not alone: Exploring pairwise relationships for zero-shot video activity recognition”. In: *Thirtieth AAAI conference on artificial intelligence*. 2016.
- [100] Walter Gander and Jiri Hrebicek. *Solving problems in scientific computing using Maple and Matlab®*. Springer Science & Business Media, 2011.
- [101] Yixin Gao et al. “Jhu-isi gesture and skill assessment working set (jigsaws): A surgical activity dataset for human motion modeling”. In: *MIC-CAI Workshop: M2CAI*. Vol. 3. 2014, p. 3.
- [102] Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. “Conflict-driven answer set solving: From theory to practice”. In: *Artificial Intelligence* 187 (2012), pp. 52–89.
- [103] Martin Gebser et al. “Answer set solving in practice”. In: *Synthesis lectures on artificial intelligence and machine learning* 6.3 (2012), pp. 1–238.
- [104] Martin Gebser et al. “Engineering an incremental ASP solver”. In: *International Conference on Logic Programming*. Springer. 2008, pp. 190–205.
- [105] Martin Gebser et al. “Experimenting with robotic intra-logistics domains”. In: *Theory and Practice of Logic Programming* 18.3-4 (2018), pp. 502–519.
- [106] Martin Gebser et al. “Multi-shot ASP solving with clingo”. In: *Theory and Practice of Logic Programming* 19.1 (2019), pp. 27–82.

- [107] Hector Geffner and Blai Bonet. “A concise introduction to models and methods for automated planning”. In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 8.1 (2013), pp. 1–141.
- [108] Michael Gelfond and Vladimir Lifschitz. “The stable model semantics for logic programming.” In: *ICLP/SLP*. Vol. 88. 1988, pp. 1070–1080.
- [109] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated planning and acting*. Cambridge University Press, 2016.
- [110] Bernard Gibaud et al. “Toward a standard ontology of surgical process models”. In: *International journal of computer assisted radiology and surgery* 13.9 (2018), pp. 1397–1408.
- [111] G. Gierse et al. *Interruptible task execution with resumption in GOLOG*. Vol. 285. Frontiers in Artificial Intelligence and Appl. 2016, pp. 1265–1273.
- [112] Yolanda Gil. “Learning by Experimentation: Incremental Refinement of Incomplete Planning Domains”. In: *International Conference on Machine Learning*. New Brunswick, USA, July 1994, pp. 87–95.
- [113] Michele Ginesi, Nicola Sansonetto, and Paolo Fiorini. “Overcoming some drawbacks of dynamic movement primitives”. In: *Robotics and Autonomous Systems* (2021), p. 103844.
- [114] Michele Ginesi et al. “A knowledge-based framework for task automation in surgery”. In: *2019 19th International Conference on Advanced Robotics (ICAR)*. Dec. 2019, pp. 37–42. DOI: [10.1109/ICAR46387.2019.8981619](https://doi.org/10.1109/ICAR46387.2019.8981619).
- [115] Michele Ginesi et al. *Autonomous task planning and situation awareness in robotic surgery*. 2020. arXiv: [2004.08911](https://arxiv.org/abs/2004.08911) [cs.RO].
- [116] Michele Ginesi et al. “Dynamic Movement Primitives: Volumetric Obstacle Avoidance”. In: *2019 19th International Conference on Advanced Robotics (ICAR)*. Dec. 2019, pp. 234–239. DOI: [10.1109/ICAR46387.2019.8981552](https://doi.org/10.1109/ICAR46387.2019.8981552).

- [117] M. Ginesi et al. “Dynamic Movement Primitives: Volumetric Obstacle Avoidance Using Dynamic Potential Functions”. In: *Journal of Intelligent and Robotic Systems: Theory and Applications* 101.4 (2021).
- [118] P. J. S. Gonçalves and P. M. B. Torres. “Knowledge representation applied to robotic orthopedic surgery”. In: *Robotics and Computer- Integrated Manufacturing* 33 (2015), pp. 90–99.
- [119] Alba Gragera, Alba Maria Garcia, and Fernando Fernandez. “A Modelling and Formalisation Tool for Use Case Design in Social Autonomous Robotics”. In: *Iberian Robotics conference*. Springer. 2019, pp. 656–667.
- [120] David Gundana and Hadas Kress-Gazit. “Event-Based Signal Temporal Logic Synthesis for Single and Multi-Robot Tasks”. In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 3687–3694. DOI: [10.1109/LRA.2021.3064220](https://doi.org/10.1109/LRA.2021.3064220).
- [121] M. Guo, K. H. Johansson, and D. V. Dimarogonas. “Motion and action planning under LTL specifications using navigation functions and action description language”. In: *IEEE International Conference on Intelligent Robots and Systems*. 2013, pp. 240–245.
- [122] M. Guo, J. Tumova, and D. V. Dimarogonas. “Communication-Free Multi-Agent Control under Local Temporal Tasks and Relative-Distance Constraints”. In: *IEEE Trans. on Automatic Control* 61 (2016), pp. 3948–3962.
- [123] M. Guo, J. Tumova, and D. V. Dimarogonas. “Cooperative decentralized multi-agent control under local LTL tasks and connectivity constraints”. In: *Proceedings of the IEEE Conference on Decision and Control*. Vol. 2015-February. 2014. Chap. February, pp. 75–80.
- [124] Bernd Gutmann et al. “The magic of logical inference in probabilistic programming”. In: *Theory and Practice of Logic Programming* 11.4-5 (2011), pp. 663–680.
- [125] T. Haidegger. “Autonomy for surgical robots: Concepts and paradigms”. In: *IEEE Trans. on Medical Robotics and Bionics* 1 (2019), pp. 65–76.

- [126] Tamas Haidegger et al. “The importance of accuracy measurement standards for computer-integrated interventional systems”. In: *EURON GEM Sig Workshop on The Role of Experiments in Robotics Research at IEEE ICRA*. 2010, pp. 1–6.
- [127] S. Hao et al. “An optimal task decision method for a warehouse robot with multiple tasks based on linear temporal logic”. In: *2017 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2017*. Vol. 2017-January. 2017, pp. 1453–1458.
- [128] Zaid Harchaoui et al. “A regularized kernel-based approach to unsupervised audio segmentation”. In: *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2009, pp. 1665–1668.
- [129] H. Harman and P. Simoens. “Action graphs for proactive robot assistance in smart environments”. In: *Journal of Ambient Intelligence and Smart Environments* 12.2 (2020), pp. 79–99.
- [130] K. He et al. “Towards manipulation planning with temporal logic specifications”. In: *Proceedings - IEEE International Conference on Robotics and Automation*. Vol. 2015-June. 2015. Chap. June, pp. 346–352.
- [131] Malte Helmert, Gabriele Röger, and Erez Karpas. “Fast downward stone soup: A baseline for building planner portfolios”. In: *ICAPS 2011 Workshop on Planning and Learning*. Citeseer. 2011, pp. 28–35.
- [132] Heiko Hoffmann et al. “Biologically-inspired dynamical systems for movement generation: automatic real-time goal adaptation and obstacle avoidance”. In: *Robotics and Automation, IEEE International Conference on*. IEEE. 2009, pp. 2587–2592.
- [133] Jörg Hoffmann and Bernhard Nebel. “The FF planning system: Fast plan generation through heuristic search”. In: *Journal of Artificial Intelligence Research* 14 (2001), pp. 253–302.
- [134] Minsik Hong and Jerzy W Rozenblit. “Modeling of a transfer task in computer assisted surgical training”. In: *Proceedings of the Modeling and Simulation in Medicine Symposium*. 2016, pp. 1–6.

- [135] MT Hussein. “A review on vision-based control of flexible manipulators”. In: *Advanced Robotics* 29.24 (2015), pp. 1575–1585.
- [136] Auke J Ijspeert, Jun Nakanishi, and Stefan Schaal. “Learning attractor landscapes for learning motor primitives”. In: *Advances in neural information processing systems*. 2003, pp. 1547–1554.
- [137] Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. “Movement imitation with nonlinear dynamical systems in humanoid robots”. In: *Robotics and Automation, 2002. Proceedings. ICRA’02. IEEE International Conference on*. Vol. 2. IEEE. 2002, pp. 1398–1403.
- [138] Auke Jan Ijspeert et al. “Dynamical movement primitives: learning attractor models for motor behaviors”. In: *Neural computation* 25.2 (2013), pp. 328–373.
- [139] Felix Ingrand and Malik Ghallab. “Deliberation for autonomous robots: A survey”. In: *Artificial Intelligence* 247 (2017), pp. 10–44.
- [140] Dominik Jain, Lorenz Mosenlechner, and Michael Beetz. “Equipping robot control programs with first-order probabilistic reasoning capabilities”. In: *2009 IEEE International Conference on Robotics and Automation*. IEEE. 2009, pp. 3626–3631.
- [141] Rob Janssen et al. “Cloud based centralized task control for human domain multi-robot operations”. In: *Intelligent Service Robotics* 9.1 (2016), pp. 63–77.
- [142] B. Javia and P. Cimiano. *A Logic Programming Approach to Collaborative Autonomous Robotics*. Vol. 10505 LNAI. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2017, pp. 309–315.
- [143] Jianmin Ji. “A cognitive architecture for a service robot: an answer set programming approach”. In: *Proceedings of the 25th International Conference on Logic Programming, LNCS*. Vol. 5649. 2009, pp. 532–533.
- [144] J. Jiang. “Information extraction from text”. In: *Mining text data*. Springer, 2012, pp. 11–41.

- [145] Y. -. Jiang et al. “Task planning in robotics: an empirical comparison of PDDL- and ASP-based systems”. In: *Frontiers of Information Technology and Electronic Engineering* 20.3 (2019), pp. 363–373.
- [146] Gangyuan Jing et al. “An End-To-End System for Accomplishing Tasks with Modular Robots.” In: *Robotics: Science and systems*. 2016, pp. 1–5.
- [147] Gangyuan Jing et al. “Correct high-level robot control from structured english”. In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 3543–3544.
- [148] Antonis C Kakas and Antonia Michael. “Integrating Abductive and Constraint Logic Programming.” In: *ICLP*. 1995, pp. 399–413.
- [149] Roland Kaminski, Torsten Schaub, and Philipp Wanko. “A tutorial on hybrid answer set solving with clingo”. In: *Reasoning Web International Summer School*. Springer. 2017, pp. 167–203.
- [150] Stavros A Karkanis et al. “Tumor recognition in endoscopic video images using artificial neural network architectures”. In: *Proceedings of the 26th Euromicro Conference. EUROMICRO 2000. Informatics: Inventing the Future*. Vol. 2. IEEE. 2000, pp. 423–429.
- [151] Nikos Katzouris, Alexander Artikis, and Georgios Paliouras. “Incremental learning of event definitions with inductive logic programming”. In: *Machine Learning* 100.2-3 (2015), pp. 555–585.
- [152] Nikos Katzouris, Alexander Artikis, and Georgios Paliouras. “Incremental learning of event definitions with inductive logic programming”. In: *Machine Learning* 100.2-3 (2015), pp. 555–585.
- [153] Nikos Katzouris, Alexander Artikis, and Georgios Paliouras. “Parallel online event calculus learning for complex event recognition”. In: *Future Generation Computer Systems* 94 (2019), pp. 468–478.
- [154] Benjamin Kaufmann et al. “Grounding and solving in answer set programming”. In: *AI Magazine* 37.3 (2016), pp. 25–32.
- [155] Henry Kautz and Bart Selman. “Pushing the envelope: Planning, propositional logic, and stochastic search”. In: *Proceedings of the National Conference on Artificial Intelligence*. 1996, pp. 1194–1201.

- [156] Shian-Ru Ke et al. “A review on video-based human activity recognition”. In: *Computers* 2.2 (2013), pp. 88–131.
- [157] Rebecca Killick, Paul Fearnhead, and Idris A Eckley. “Optimal detection of changepoints with a linear computational cost”. In: *Journal of the American Statistical Association* 107.500 (2012), pp. 1590–1598.
- [158] S. Kim and G. Kwon. “Multi-robot collaboration simulation using LTL synthesis”. In: *Information (Japan)* 17.5 (2014), pp. 1763–1769.
- [159] Sunghae Kim and Gihwon Kwon. “Simulation of Collaborative Multi-robots”. In: *International Conference on Hybrid Information Technology*. Springer. 2012, pp. 588–593.
- [160] Angelika Kimmig et al. “On the implementation of the probabilistic logic programming language ProbLog”. In: *Theory and Practice of Logic Programming* 11.2-3 (2011), pp. 235–262.
- [161] M. Kirsch et al. “Integrating golog++ and ROS for practical and portable high-level control”. In: *ICAART 2020 - Proceedings of the 12th International Conference on Agents and Artificial Intelligence*. Vol. 2. 2020, pp. 692–699.
- [162] M. Kloetzer and C. Belta. “LTL planning for groups of robots”. In: *Proceedings of the 2006 IEEE International Conference on Networking, Sensing and Control, ICNSC’06*. 2006, pp. 578–583.
- [163] Andrey Kolobov. “Planning with Markov decision processes: An AI perspective”. In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 6.1 (2012), pp. 1–210.
- [164] Anis Koubâa et al. *Robot Operating System (ROS)*. Vol. 1. Springer, 2017.
- [165] Daniel L Kovacs. “BNF definition of PDDL 3.1”. In: *International Planning Competition IPC-2011 website* (2011).
- [166] Robert Kowalski and Marek Sergot. “A logic-based calculus of events”. In: *Foundations of knowledge base management*. Springer, 1989, pp. 23–55.

- [167] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. “Temporal-logic-based reactive mission and motion planning”. In: *IEEE Transactions on Robotics* 25.6 (2009), pp. 1370–1381.
- [168] Hadas Kress-Gazit, Tichakorn Wongpiromsarn, and Ufuk Topcu. “Correct, reactive, high-level robot control”. In: *IEEE Robotics & Automation Magazine* 18.3 (2011), pp. 65–74.
- [169] Sanjay Krishnan et al. “Transition state clustering: Unsupervised surgical trajectory segmentation for robot learning”. In: *The International Journal of Robotics Research* 36.13-14 (2017), pp. 1595–1618.
- [170] Anil Kumar and Rahul Kala. “Linear Temporal Logic-based Mission Planning”. In: *IJIMAI* 3.7 (2016), pp. 32–41.
- [171] Vipin Kumar and Yow-Jian Lin. “A data-dependency-based intelligent backtracking scheme for Prolog”. In: *The Journal of Logic Programming* 5.2 (1988), pp. 165–181.
- [172] David M Kwartowitz, S Duke Herrell, and Robert L Galloway. “Toward image-guided robotic surgery: determining intrinsic accuracy of the da Vinci robot”. In: *International Journal of Computer Assisted Radiology and Surgery* 1.3 (2006), pp. 157–165.
- [173] B. Lacerda and P. U. Lima. “Designing Petri net supervisors from LTL specifications”. In: *Robotics: Science and Systems*. Vol. 7. 2012, pp. 169–176.
- [174] Ashwini Lahane et al. “Detection of unsafe action from laparoscopic cholecystectomy video”. In: *Proceedings of the 2nd ACM SIGHIT International Health Informatics Symposium*. 2012, pp. 315–322.
- [175] Morteza Lahijanian et al. “This time the robot settles for a cost: A quantitative approach to temporal logic planning with partial satisfaction”. In: *Twenty-Ninth AAAI Conference on Artificial Intelligence*. 2015.
- [176] John E. Laird et al. “Interactive Task Learning”. In: *IEEE Intelligent Systems* 32.4 (2017), pp. 6–21.
- [177] John E Laird et al. “Interactive task learning”. In: *IEEE Intelligent Systems* 32.4 (2017), pp. 6–21.

- [178] Florent Lalys and Pierre Jannin. “Surgical process modelling: a review”. In: *International journal of computer assisted radiology and surgery* 9.3 (2014), pp. 495–511.
- [179] Kathryn Blackmond Laskey. “MEBN: A language for first-order Bayesian knowledge bases”. In: *Artificial intelligence* 172.2-3 (2008), pp. 140–178.
- [180] Mark Law. “Inductive learning of answer set programs”. PhD thesis. University of London, 2018.
- [181] Mark Law, Alessandra Russo, and Krysia Broda. “Iterative Learning of Answer Set Programs from Context Dependent Examples”. In: *Theory and Practice of Logic Programming* 16.5-6 (2016), p. 834.
- [182] Mark Law, Alessandra Russo, and Krysia Broda. “Iterative learning of answer set programs from context dependent examples”. In: *Theory and Practice of Logic Programming* 16.5-6 (2016), pp. 834–848.
- [183] Mark Law, Alessandra Russo, and Krysia Broda. “The complexity and generality of learning answer set programs”. In: *Artificial Intelligence* 259 (2018), pp. 110–146.
- [184] *Laying down harmonized rules on artificial intelligence and amending certain Union legislative acts*. URL: <https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52021PC0206>.
- [185] Gregor Leban, Jure Žabkar, and Ivan Bratko. “An experiment in robot discovery with ilp”. In: *International Conference on Inductive Logic Programming*. Springer. 2008, pp. 77–90.
- [186] Joohyung Lee and Yi Wang. “A Probabilistic Extension of Action Language $\mathcal{BC}+$ ”. In: *Theory and Practice of Logic Programming* 18.3-4 (2018), pp. 607–622.
- [187] Joohyung Lee and Yi Wang. “Weighted rules under the stable model semantics”. In: *Fifteenth International Conference on the Principles of Knowledge Representation and Reasoning*. 2016.

- [188] Séverin Lemaignan et al. “ORO, a knowledge management platform for cognitive architectures in robotics”. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2010, pp. 3548–3553.
- [189] Nicola Leone, Pasquale Rullo, and Francesco Scarcello. “Disjunctive stable models: Unfounded sets, fixpoint semantics, and computation”. In: *Information and computation* 135.2 (1997), pp. 69–112.
- [190] Hector J Levesque et al. “GOLOG: A logic programming language for dynamic domains”. In: *The Journal of Logic Programming* 31.1-3 (1997), pp. 59–83.
- [191] Vladimir Lifschitz. “Answer set planning”. In: *International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer. 1999, pp. 373–374.
- [192] Vladimir Lifschitz. “Answer sets and the language of answer set programming”. In: *AI Magazine* 37.3 (2016), pp. 7–12.
- [193] Vladimir Lifschitz. “What Is Answer Set Programming?.” In: *AAAI*. Vol. 8. 2008. 2008, pp. 1594–1597.
- [194] Vladimir Lifschitz and Alexander Razborov. “Why Are There so Many Loop Formulas?” In: *ACM Trans. Comput. Logic* 7.2 (2006), pp. 261–268.
- [195] Maxim Likhachev, Sebastian Thrun, and Geoffrey J Gordon. “Planning for markov decision processes with sparse stochasticity”. In: *Advances in neural information processing systems*. 2005, pp. 785–792.
- [196] Fangzhen Lin and Yuting Zhao. “ASSAT: Computing answer sets of a logic program by SAT solvers”. In: *Artificial Intelligence* 157.1-2 (2004), pp. 115–137.
- [197] Hai Lin. “Mission accomplished: An introduction to formal methods in mobile robot motion planning and control”. In: *Unmanned Systems* 2.02 (2014), pp. 201–216.

- [198] Gheorghe Lisca et al. “Towards robots conducting chemical experiments”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, pp. 5202–5208.
- [199] L. Liu et al. “Temporal logic task and motion planning of a smart robot-towards a smart substation environment”. In: *2017 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2017*. Vol. 2017-January. 2017, pp. 1110–1115.
- [200] John W Lloyd. *Foundations of logic programming*. Springer Science & Business Media, 2012.
- [201] Ian D Loram, Peter J Gawthrop, and Martin Lakie. “The frequency of human, manual adjustments in balancing an inverted pendulum is constrained by intrinsic physiological factors”. In: *The Journal of physiology* 577.1 (2006), pp. 417–432.
- [202] Constantinos Loukas. “Video content analysis of surgical procedures”. In: *Surgical endoscopy* 32.2 (2018), pp. 553–568.
- [203] Constantinos Loukas and Evangelos Georgiou. “Smoke detection in endoscopic surgery videos: a first step towards retrieval of semantic events”. In: *The International Journal of Medical Robotics and Computer Assisted Surgery* 11.1 (2015), pp. 80–94.
- [204] Constantinos Loukas and Evangelos Georgiou. “Surgical workflow analysis with Gaussian mixture multivariate autoregressive (GMMAR) models: a simulation study”. In: *Computer Aided Surgery* 18.3-4 (2013), pp. 47–62.
- [205] Dongcai Lu et al. “Integrating Answer Set Programming with Semantic Dictionaries for Robot Task Planning.” In: *IJCAI*. 2017, pp. 4361–4367.
- [206] T. Lukasiewicz. “Probabilistic Logic Programming.” In: *ECAI*. 1998, pp. 388–392.
- [207] Thomas Lukasiewicz. “Probabilistic logic programming with conditional constraints”. In: *ACM Transactions on Computational Logic (TOCL)* 2.3 (2001), pp. 289–339.

- [208] C. Ma, F. Fang, and X. Ma. “Task planning of mobile robots in distributed service framework”. In: *IECON Proceedings (Industrial Electronics Conference)*. 2016, pp. 342–347.
- [209] Michael J Mack. “Minimally Invasive and Robotic Surgery”. In: *Journal of American Medical Association* 285.5 (2001), pp. 568–572.
- [210] Norbert Manthey. “Towards next generation sequential and parallel SAT solvers”. In: *KI-Künstliche Intelligenz* 30.3-4 (2016), pp. 339–342.
- [211] Victor W Marek and Mirosław Truszczyński. “Stable models and an alternative logic programming paradigm”. In: *The Logic Programming Paradigm*. Springer, 1999, pp. 375–398.
- [212] V. Mataré, S. Schiffer, and A. Ferrein. “Golog++: An integrative system design”. In: *CEUR Workshop Proceedings*. Vol. 2325. 2018, pp. 29–35.
- [213] Maja J Mataric. “Situated robotics”. In: *Encyclopedia of cognitive science* (2006).
- [214] Francesco Maurelli, Zeyn Saigol, and David Lane. “Cognitive knowledge representation under uncertainty for autonomous underwater vehicles”. In: *Proceedings of IEEE ICRA’14 IEEE Hong Kong, Workshop on Persistent Autonomy for Underwater Robotics*. Citeseer. 2014.
- [215] John McCarthy. “Circumscription—a form of non-monotonic reasoning”. In: *Artificial intelligence* 13.1-2 (1980), pp. 27–39.
- [216] Drew McDermott and Jon Doyle. “Non-monotonic logic I”. In: *Artificial intelligence* 13.1-2 (1980), pp. 41–72.
- [217] Daniele Meli and Paolo Fiorini. “Unsupervised identification of surgical robotic actions from small non homogeneous datasets”. In: *arXiv preprint arXiv:2105.08488* (2021).
- [218] Daniele Meli, Paolo Fiorini, and Mohan Sridharan. “Towards inductive learning of surgical task knowledge: a preliminary case study of the peg transfer task”. In: *Procedia Computer Science* 176 (2020), pp. 440–449.
- [219] Daniele Meli, Mohan Sridharan, and Paolo Fiorini. “Inductive learning of answer set programs for autonomous surgical task planning”. In: *Machine Learning* (2021), pp. 1–25.

- [220] Dannilo Samuel Silva Miranda, Luiz Edival de Souza, and Guilherme Sousa Bastos. “A ROSPlan-Based Multi-Robot Navigation System”. In: *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)*. IEEE. 2018, pp. 248–253.
- [221] Fumio Mizoguchi et al. “Identifying Driver’s Cognitive Distraction Using Inductive Logic Programming”. In: *Proceedings of the 25th International Conference on Inductive Logic Programming (ILP ‘15)*. 2015.
- [222] Ahmed Abdulhadi Al-Moadhen et al. “Robot task planning in deterministic and probabilistic conditions using semantic knowledge base”. In: *International Journal of Knowledge and Systems Science (IJKSS) 7.1* (2016), pp. 56–77.
- [223] Sara Moccia et al. “Blood vessel segmentation algorithms—review of methods, datasets and evaluation metrics”. In: *Computer methods and programs in biomedicine* 158 (2018), pp. 71–91.
- [224] B. Moldovan, L. Antanas, and M. Hoffmann. *Opening doors: An initial SRL approach*. Vol. 7842 LNAI. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2013, pp. 178–192.
- [225] Bogdan Moldovan and Luc De Raedt. “Learning relational affordance models for two-arm robots”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 2916–2922.
- [226] Bogdan Moldovan et al. “Learning relational affordance models for robots in multi-object manipulation tasks”. In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 4373–4378.
- [227] Bogdan Moldovan et al. “Relational affordances for multiple-object manipulation”. In: *Autonomous Robots* 42.1 (2018), pp. 19–44.
- [228] J. Moon and B. -. Lee. “PDDL planning with natural language-based scene understanding for UAV-UGV cooperation”. In: *Applied Sciences (Switzerland)* 9.18 (2019).

- [229] Raymond J Mooney and Razvan Bunescu. “Mining knowledge from text using information extraction”. In: *ACM SIGKDD explorations newsletter* 7.1 (2005), pp. 3–10.
- [230] Tiago Mota and Mohan Sridharan. “Commonsense Reasoning and Knowledge Acquisition to Guide Deep Learning on Robots”. In: *Robotics Science and Systems*. Freiburg, Germany, June 2019.
- [231] George P Moustiris et al. “Evolution of autonomous and semi-autonomous robotic surgical systems: a review of the literature”. In: *The international journal of medical robotics and computer assisted surgery* 7.4 (2011), pp. 375–392.
- [232] Stephen Moyle and Stephen Muggleton. “Learning programs in the event calculus”. In: *International Conference on Inductive Logic Programming*. Springer. 1997, pp. 205–212.
- [233] Stephen Muggleton. “Inductive logic programming”. In: *New generation computing* 8.4 (1991), pp. 295–318.
- [234] Stephen Muggleton. “Inverse entailment and Progol”. In: *New generation computing* 13.3-4 (1995), pp. 245–286.
- [235] P. Muñoz et al. “MoBAR: a Hierarchical Action-Oriented Autonomous Control Architecture”. In: *Journal of Intelligent and Robotic Systems: Theory and Applications* 94.3-4 (2019), pp. 745–760.
- [236] S. Muñoz-Hernandez and W. S. Wiguna. *Fuzzy cognitive layer in RoboCup Soccer*. Vol. 4529 LNAI. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2007, pp. 635–645.
- [237] Adithyavairavan Murali et al. “Tsc-dl: Unsupervised trajectory segmentation of multi-modal surgical demonstrations with deep learning”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 4150–4157.

- [238] Andrew Naftel and Shehzad Khalid. “Classification and prediction of motion trajectories using spatiotemporal approximations”. In: *Proc. Int. Workshop on Human Activity Recognition and Modelling*. 2005, pp. 17–26.
- [239] Dénes Ákos Nagy et al. “Ontology-based surgical subtask automation, automating blunt dissection”. In: *Journal of Medical Robotics Research* 3.03n04 (2018), p. 1841005.
- [240] Tamás D Nagy and Tamás P Haidegger. “Towards Standard Approaches for the Evaluation of Autonomous Surgical Subtask Execution”. In: *2021 IEEE 25th International Conference on Intelligent Engineering Systems (INES)*. IEEE. 2021, pp. 000067–000074.
- [241] Hirenkumar Nakawala et al. ““Deep-Onto” network for surgical workflow and context recognition”. In: *International journal of computer assisted radiology and surgery* 14.4 (2019), pp. 685–696.
- [242] Guy A Narboni. “From prolog iii to prolog iv: The logic of constraint programming revisited”. In: *Constraints* 4.4 (1999), pp. 313–335.
- [243] Dana Nau et al. “SHOP: Simple hierarchical ordered planner”. In: *Proceedings of the 16th international joint conference on Artificial int.- Volume 2*. 1999, pp. 968–973.
- [244] Thomas Neumuth et al. “Acquisition of process descriptions from surgical interventions”. In: *International Conference on Database and Expert Systems Applications*. Springer. 2006, pp. 602–611.
- [245] I. Nevlyudov, O. Tsymbal, and S. Milyutina. “The logical model of product assembly technological process design”. In: *TCSET 2008 - Modern Problems of Radio Engineering, Telecommunications and Computer Science - Proceedings of the International Conference*. 2008, pp. 500–501.
- [246] Raymond Ng and V.S. Subrahmanian. “Probabilistic logic programming”. In: *Information and computation* 101.2 (1992), pp. 150–201.
- [247] Scott Niekum et al. “Learning grounded finite-state representations from unstructured demonstrations”. In: *The International Journal of Robotics Research* 34.2 (2015), pp. 131–157.

- [248] Davide Nitti, Vaishak Belle, and Luc De Raedt. “Planning in discrete and continuous markov decision processes by probabilistic programming”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2015, pp. 327–342.
- [249] Daniel Nyga and Michael Beetz. “Everything robots always wanted to know about housework (but were afraid to ask)”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 243–250.
- [250] Daniel Nyga et al. “Grounding robot plans from natural language instructions with incomplete world knowledge”. In: *Conference on Robot Learning*. 2018, pp. 714–723.
- [251] S. Opfer, S. Jakob, and K. Geihs. *Reasoning for autonomous agents in dynamic domains: Towards automatic satisfaction of the module property*. Vol. 10839 LNAI. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2018, pp. 22–47.
- [252] S. Opfer et al. *ALICA 2.0 - Domain-Independent Teamwork*. Vol. 11793 LNAI. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2019, pp. 264–272.
- [253] G. Papadimitriou, Z. Saigol, and D. M. Lane. “Enabling fault recovery and adaptation in mine-countermeasures missions using ontologies”. In: *MTS/IEEE OCEANS 2015 - Genova: Discovering Sustainable Ocean Energy for a New World*. 2015.
- [254] Dae-Hyung Park et al. “Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields”. In: *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on*. IEEE. 2008, pp. 91–98.
- [255] Peter Pastor et al. “From dynamic movement primitives to associative skill memories”. In: *Robotics and Autonomous Systems* 61.4 (2013), pp. 351–361.

- [256] Peter Pastor et al. “Learning and generalization of motor skills by learning from demonstration”. In: *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*. IEEE. 2009, pp. 763–768.
- [257] Peter Pastor et al. “Towards associative skill memories”. In: *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*. IEEE. 2012, pp. 309–315.
- [258] P. Pastor et al. “Online movement adaptation based on previous sensor experiences”. In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sept. 2011, pp. 365–371.
- [259] David Pearce. “A New Logical Characterisation of Stable Models and Answer Sets”. In: *Non-Monotonic Extensions of Logic Programming: Second International Workshop NMELP'96, Germany, 1996, Selected Papers*. 1216. Springer Science & Business Media. 1997, p. 57.
- [260] Luis Moniz Pereira et al. “Delta Prolog: a distributed backtracking extension with events”. In: *International Conference on Logic Programming*. Springer. 1986, pp. 69–83.
- [261] Simona Perri et al. “Enhancing DLV instantiator by backjumping techniques”. In: *Annals of Mathematics and Artificial Intelligence* 51.2-4 (2007), p. 195.
- [262] P. Pianpak et al. “A distributed solver for multi-agent path finding problems”. In: *ACM International Conference Proceeding Series*. 2019.
- [263] L. A. Pineda et al. “Sit log: A programming language for service robot task”. In: *International Journal of Advanced Robotic Systems* 10 (2013), pp. 1–12.
- [264] Francesco Piqué et al. “Dynamic modeling of the da Vinci Research Kit arm for the estimation of interaction wrench”. In: *2019 International Symposium on Medical Robotics (ISMR)*. IEEE. 2019, pp. 1–7.
- [265] Tobias Pistohl et al. “Prediction of arm movement trajectories from ECoG-recordings in humans”. In: *Journal of neuroscience methods* 167.1 (2008), pp. 105–114.

- [266] Amir Pnueli. “The temporal logic of programs”. In: *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. IEEE, 1977, pp. 46–57.
- [267] Roberto Poli, Michael Healy, and Achilles Kameas. *Theory and applications of ontology: Computer applications*. Springer, 2010.
- [268] M.J. Primus. “Segmentation and indexing of endoscopic videos”. In: *Proceedings of the 22nd ACM international conference on Multimedia*. 2014, pp. 659–662.
- [269] Kanna Rajan, Frederic Py, and Javier Barreiro. “Towards deliberative control in marine robotics”. In: *Marine Robot Autonomy*. Springer, 2013, pp. 91–175.
- [270] Vasumathi Raman and Hadas Kress-Gazit. “Explaining impossible high-level robot behaviors”. In: *IEEE Transactions on Robotics* 29.1 (2012), pp. 94–104.
- [271] Oliver Ray. “Nonmonotonic abductive inductive learning”. In: *Journal of Applied Logic* 7.3 (2009), pp. 329–340.
- [272] Oliver Ray and Katsumi Inoue. “Mode-directed inverse entailment for full clausal theories”. In: *International Conference on Inductive Logic Programming*. Springer, 2007, pp. 225–238.
- [273] Matthew Richardson and Pedro Domingos. “Markov logic networks”. In: *Machine learning* 62.1-2 (2006), pp. 107–136.
- [274] Florian Richter et al. “Autonomous Robotic Suction to Clear the Surgical Field for Hemostasis using Image-based Blood Flow Detection”. In: *arXiv preprint arXiv:2010.08441* (2020).
- [275] Silvia Richter, Matthias Westphal, and Malte Helmert. “LAMA 2008 and 2011”. In: *International Planning Competition*. 2011, pp. 117–124.
- [276] A. Roberti et al. “Improving Rigid 3-D Calibration for Robotic Surgery”. In: *IEEE Transactions on Medical Robotics and Bionics* 2.4 (2020), pp. 569–573. DOI: [10.1109/TMRB.2020.3033670](https://doi.org/10.1109/TMRB.2020.3033670).
- [277] Jan JMM Rutten et al. *Mathematical techniques for analyzing concurrent and probabilistic systems*. 23. American Mathematical Soc., 2004.

- [278] Chiaki Sakama and Katsumi Inoue. “Brave induction: a logical framework for learning from incomplete information”. In: *Machine Learning* 76.1 (2009), pp. 3–35.
- [279] Hiroaki Sakoe and Seibi Chiba. “Dynamic programming algorithm optimization for spoken word recognition”. In: *IEEE transactions on acoustics, speech, and signal processing* 26.1 (1978), pp. 43–49.
- [280] Valerio Sanelli et al. “Short-term human-robot interaction through conditional planning and execution”. In: *Twenty-Seventh International Conference on Automated Planning and Scheduling*. 2017.
- [281] Shahar Sarid, Bingxin Xu, and Hadas Kress-Gazit. “Guaranteeing high-level behaviors while exploring partially known maps”. In: *Robotics*. MIT Press, 2013, pp. 377–384.
- [282] Taisuke Sato. “A statistical learning method for logic programs with distribution semantics”. In: *IN PROCEEDINGS OF THE 12TH INTERNATIONAL CONFERENCE ON LOGIC PROGRAMMING (ICLP’95*. Cite-seer. 1995.
- [283] Matteo Saveriano, Felix Franzel, and Dongheui Lee. “Merging Position and Orientation Motion Primitives”. In: *International Conference on Robotics and Automation (ICRA), 2019*. 2019.
- [284] Stefan Schaal. “Dynamic movement primitives—a framework for motor control in humans and humanoid robotics”. In: *Adaptive motion of animals and machines*. Springer, 2006, pp. 261–280.
- [285] R. W. Schafer. “What Is a Savitzky-Golay Filter? [Lecture Notes]”. In: *IEEE Signal Processing Magazine* 28.4 (2011), pp. 111–117.
- [286] S. Schiffer, A. Ferrein, and G. Lakemeyer. “Caesar: An intelligent domestic service robot”. In: *Intelligent Service Robotics* 5.4 (2012), pp. 259–273.
- [287] P. Schillinger, M. Burger, and D.V. Dimarogonas. “Multi-objective search for optimal multi-robot planning with finite LTL specifications and resource constraints”. In: *Proceedings - IEEE International Conference on Robotics and Automation*. 2017, pp. 768–774.

- [288] A. Schmuck, R. Majumdar, and A. Leva. “Dynamic hierarchical reactive controller synthesis”. In: *Discrete Event Dynamic Systems: Theory and Applications* 27.2 (2017), pp. 261–299.
- [289] Peter Schüller and Mishal Benz. “Best-effort inductive logic programming via fine-grained cost-based hypothesis generation”. In: *Machine Learning* 107.7 (2018), pp. 1141–1169.
- [290] T. Semwal et al. “Tartarus: A multi-agent platform for integrating cyber-physical systems and robots”. In: *ACM International Conference Proceeding Series*. Vol. 02-04-July-2015. 2015.
- [291] Murray Shanahan et al. *Solving the frame problem: a mathematical investigation of the common sense law of inertia*. MIT press, 1997.
- [292] Zhenzhou Shao et al. “Unsupervised trajectory segmentation and promoting of multi-modal surgical demonstrations”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 777–782.
- [293] Aidean Sharghi et al. “Automatic Operating Room Surgical Activity Recognition for Robot-Assisted Surgery”. In: *International Conf. on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2020, pp. 385–395.
- [294] Wei-Min Shen. “Discovery as autonomous learning from the environment”. In: *Machine Learning* 12.1-3 (1993), pp. 143–165.
- [295] Xueying Shi et al. “Domain Adaptive Robotic Gesture Recognition with Unsupervised Kinematic-Visual Data Alignment”. In: *arXiv preprint arXiv:2103.04075* (2021).
- [296] Vikas Shivashankar et al. “The GoDeL planning system: a more perfect union of domain-independent and hierarchical planning”. In: *Twenty-Third International Joint Conference on Artificial Intelligence*. 2013.
- [297] Raymond M Smullyan. *First-order logic*. Courier Corporation, 1995.
- [298] Tran Cao Son et al. “Domain-dependent knowledge in answer set planning”. In: *ACM Transactions on Computational Logic (TOCL)* 7.4 (2006), pp. 613–657.

- [299] Nathaniel J. Soper and Gerald M. Fried. “The fundamentals of laparoscopic surgery: its time has come.” In: *Bull Am Coll Surg* 93.9 (2008), pp. 30–32.
- [300] Niklas Sorensson and Niklas Een. “Minisat v1.13 - a sat solver with conflict-clause minimization”. In: *SAT 2005.53* (2005), pp. 1–2.
- [301] Muralikrishna Sridhar, Anthony G Cohn, and David C Hogg. “Unsupervised learning of event classes from video”. In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*. AAAI Press. 2010, pp. 1631–1638.
- [302] Mohan Sridharan and Ben Meadows. “Knowledge Representation and Interactive Learning of Domain Knowledge for Human-Robot Collaboration”. In: *Advances in Cognitive Systems* 7 (Dec. 2018), pp. 77–96.
- [303] M. Sridharan et al. “ReBA: A refinement-based architecture for knowledge representation and reasoning in robotics”. In: *Journal of Artificial Intelligence Research* 65 (2019), pp. 87–180.
- [304] Xiaoying Sun et al. “A review of robot control with visual servoing”. In: *2018 IEEE 8th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*. IEEE. 2018, pp. 116–121.
- [305] Tommi Syrjänen and Ilkka Niemelä. “The smodels system”. In: *International Conference on Logic Programming and NonMonotonic Reasoning*. Springer. 2001, pp. 434–438.
- [306] Zhi-Xuan Tan, Jake Brawer, and Brian Scassellati. “That’s mine! Learning ownership relations and norms for robots”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 8058–8065.
- [307] Lingling Tao et al. “Sparse hidden markov models for surgical gesture classification and skill evaluation”. In: *International conference on information processing in computer-assisted interventions*. Springer. 2012, pp. 167–177.

- [308] Florent Teichteil-Königsbuch, Ugur Kuter, and Guillaume Infantes. “Incremental plan aggregation for generating policies in MDPs”. In: *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*. International Foundation for Autonomous Agents and Multiagent Systems. 2010, pp. 1231–1238.
- [309] Moritz Tenorth and Michael Beetz. “Representations for robot knowledge in the KnowRob framework”. In: *Artificial Intelligence 247* (2017), pp. 151–169.
- [310] M. Thielscher. *Handling implication and universal quantification constraints in FLUX*. Vol. 3709 LNCS. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2005, pp. 667–681.
- [311] Boris A Trakhtenbrot. “Finite automata and monadic second order logic”. In: *Siberian Math. J* 3 (1962), pp. 101–131.
- [312] Charles Truong, Laurent Oudre, and Nicolas Vayatis. “Selective review of offline change point detection methods”. In: *Signal Processing* 167 (2020), p. 107299.
- [313] Phan Huy Tu, Tran Cao Son, and Chitta Baral. “Reasoning and planning with sensing actions, incomplete information, and static causal laws using answer set programming”. In: *Theory and Practice of Logic Programming* 7.4 (2007), pp. 377–450.
- [314] J. Tumova et al. “Maximally satisfying LTL action planning”. In: *IEEE International Conf. on Intelligent Robots and Systems*. 2014, pp. 1503–1510.
- [315] Aleš Ude et al. “Orientation in cartesian space dynamic movement primitives”. In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE. 2014, pp. 2997–3004.
- [316] A. Ude et al. “Task-Specific Generalization of Discrete and Periodic Dynamic Movement Primitives”. In: *IEEE Transactions on Robotics* 26.5 (Oct. 2010), pp. 800–815.

- [317] Jur Van Den Berg et al. “Superhuman performance of surgical tasks by robots using iterative learning from human-guided demonstrations”. In: *2010 IEEE International Conference on Robotics and Automation*. IEEE. 2010, pp. 2074–2081.
- [318] Maarten H Van Emden and Robert A Kowalski. “The semantics of predicate logic as a programming language”. In: *Journal of the ACM (JACM)* 23.4 (1976), pp. 733–742.
- [319] Balakrishnan Varadarajan et al. “Data-derived models for segmentation with application to surgical assessment and training”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2009, pp. 426–434.
- [320] Tamas J Vidovszky et al. “Robotic cholecystectomy: learning curve, advantages, and limitations”. In: *Journal of Surgical Research* 136.2 (2006), pp. 172–178.
- [321] N. Vitucci and G. Gini. “Reasoning on objects and grasping using description logics”. In: *Advanced Robotics* 33.13 (2019), pp. 616–635.
- [322] Richard Volpe. “Real and artificial forces in the control of manipulators: theory and experiments”. PhD thesis. PhD thesis, Carnegie Mellon University, Department of Physics, 1990.
- [323] Markus Waibel et al. “Roboearth”. In: *IEEE Robotics & Automation Magazine* 18.2 (2011), pp. 69–82.
- [324] Xuguang Wang. “Three-dimensional kinematic analysis of influence of hand orientation and joint limits on the control of arm postures and movements”. In: *Biological Cybernetics* 80.6 (1999), pp. 449–463.
- [325] X. Wang et al. “An ASP based solution to mechanical assembly sequence planning”. In: *3rd International Conference on Genetic and Evolutionary Computing, WGECC 2009*. 2009, pp. 205–208.
- [326] JD Westwood et al. “Surgical tools recognition and pupil segmentation for cataract surgical process modeling”. In: *Medicine Meets Virtual Reality 19: NextMed* 173 (2012), p. 78.

-
- [327] Jan Wielemaker et al. “Swi-prolog”. In: *Theory and Practice of Logic Programming* 12.1-2 (2012), pp. 67–96.
- [328] Thomas Wilke. “Classifying discrete temporal properties”. In: *Annual symposium on theoretical aspects of computer science*. Springer. 1999, pp. 32–46.
- [329] David E Wilkins. “Domain-independent planning representation and plan generation”. In: *Artificial Intelligence* 22.3 (1984), pp. 269–301.
- [330] K. W. Wong and H. Kress-Gazit. “Need-based coordination for decentralized high-level robot control”. In: *IEEE International Conference on Intelligent Robots and Systems*. Vol. 2016-November. 2016, pp. 2209–2216.
- [331] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M Murray. “Receding horizon control for temporal logic specifications”. In: *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*. 2010, pp. 101–110.
- [332] Tichakorn Wongpiromsarn et al. “TuLiP: a software toolbox for receding horizon temporal logic planning”. In: *Proceedings of the 14th international conference on Hybrid systems: computation and control*. ACM. 2011, pp. 313–314.
- [333] Wei-hua WU and Yi-song LIU. “Office robot controlling based on fluent calculus and FLUX [J]”. In: *Computer Engineering and Design* 11 (2008).
- [334] N. Xu et al. “An LTL-Based Motion and Action Dynamic Planning Method for Autonomous Robot”. In: *IFAC-PapersOnLine* 49.5 (2016), pp. 91–96.
- [335] S. Xu et al. “Human-machine-environment cyber-physical system and hierarchical task planning to support independent living”. In: *Proceedings - 2014 IEEE International Conf. on Bioinformatics and Biomedicine, IEEE BIBM 2014*. 2014, pp. 568–573.

-
- [336] Guang-Zhong Yang et al. “Medical robotics—Regulatory, ethical, and legal considerations for increasing levels of autonomy”. In: *Science Robotics* 2.4 (2017), p. 8638.
- [337] F. Yazdani et al. “Query-based integration of heterogeneous knowledge bases for search and rescue tasks”. In: *Robotics and Autonomous Systems* 117 (2019).
- [338] Luca Zappella et al. “Surgical gesture classification from video and kinematic data”. In: *Medical image analysis* 17.7 (2013), pp. 732–745.
- [339] Wei Zhang and Shenggui Hong. “A Linear Reduction Model for Parallel Prolog System”. In: *2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*. IEEE. 2019, pp. 650–654.
- [340] Ye Zhao, Ufuk Topcu, and Luis Sentis. “High-level planner synthesis for whole-body locomotion in unstructured environments”. In: *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE. 2016, pp. 6557–6564.
- [341] Changliang Zou et al. “Nonparametric maximum likelihood approach to multiple change-point problems”. In: *The Annals of Statistics* 42.3 (2014), pp. 970–1002.