SAPIENZA
UNIVERSITÀ DI ROMA

Knowledgeable Robots Through Multimodal HRI

Department of Computer, Control, and Management Engineering
"Antonio Ruberti", "Sapienza" University of Rome

Dottorato di Ricerca in Ingegneria Informatica – XXVIII Ciclo

Candidate

Guglielmo Gemignani
ID number 1401792

Thesis Advisor

Prof. Daniele Nardi

A thesis submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Engineering
in Computer Science

September 2015

Thesis defended on December 10th 2015
in front of a Board of Examiners composed by:

Prof. Daniele Nardi (chairman)
Prof. Giuseppe De Giacomo
Prof. Domenico Lembo
Prof. Barbara Caputo
Prof. Tiziana Catarci

Knowledgeable Robots Through Multimodal HRI
Ph.D. thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Version: December 29, 2015

Author's email: gemignani@dis.uniroma1.it

Acknowledgments

I like to think of this work as the result of a long-term project stemmed from the unique relationship established with my PhD advisor, Prof. Daniele Nardi. In these last three years, Prof. Nardi has been not only the best advisor I could ever hope for, but also a friend and a life mentor. A big part of who I am now is the result of his teachings and suggestions. So a big thanks goes to him, to one of the persons that have shaped my life the most.

During my PhD, I was often helped by Prof. Luca Iocchi, Dr. Giorgio Grisetti, and Prof. Luigia Carlucci Aiello to whom I would also like to express my deep gratitude. I owe a big part of what I know now about robotics and artificial intelligence to them.

This PhD has given me the opportunity to work with great researchers from all over the world. In particular, I would like to thank Prof. Manuela Veloso for hosting me in her lab and for her invaluable research guidance during my visit at Carnegie Mellon University. I would also like to thank Dr. Nick Hawes for the great collaboration we had and Dr. Subramanian Ramamoorthy for all the ideas shared during his visit.

The work undertaken would have been so much harder if I didn't meet several great friends during these three intense years. In particular I would like to express my gratefulness to Fabio, Roberto, Francesco, Jacopo, Claudio, Andrea, Steve, Dominik and all the other Lab Ro.Co.Co. members. They shared with me great moments in Rome, inside and outside the university. Also, I would like to acknowledge my friends at Carnegie Mellon University. In particular, I would like to thank Steven, Tiago, Rui, Juan Pablo, Vittorio, Harri, Andres, Manuel and all the members of the CORAL Group for making my six months in Pittsburgh a memorable experience.

I would like to thank my physicist friends, specifically Livia, Robertone, Francesco, Andrea, Peter, Guglielmo, Bruno, Guido, Ivan, and Silvio. Thank you for being present in my life despite the distance that divides us. Equally, my lifelong friends have an incessant presence in my life despite my recurrent habit of being absent from home; so thank you Andrea, Emanuele, Francesca, Francesco, Giulia, and Lara for your invaluable friendship.

A big thanks goes to Caterina, for her incessant love and support given to me in spite of life's attempts to keep us separated. I thank every day for her presence in my life. I wouldn't be the person that I am now without her.

I would not be here if it wasn't for my relatives. Thanks for your constant teaching and valuable support. You have been some of the most important life teaching figures in my entire life. Finally, thank you Mom for everything you have ever done for me. I miss you.

Contents

Chapter 1

Introduction

Robots are expected to be the next technological frontier, impacting our society in many sectors, including security, industry, agriculture, transportation, and services. In particular, robots are expected to become consumer products, that massively enter our homes to be part of our everyday life. This naturally brings back the view of the robot as an intelligent agent, capable of smoothly interacting with humans and operating in real life environments, whose features are undoubtedly challenging. In addition to the recent developments in robotics, other technological advancements make it more feasible to address the design of robots as intelligent agents. Specifically, new sensors allow for more effective approaches to perception, new devices support multi modal interaction with the user, and cloud-based technologies greatly enhance robots' computational capabilities.

This notwithstanding, it is currently unrealistic to deploy robots in domestic environments, enabling them to autonomously deal with complex scenarios and different requests from the users. A key limiting factor is the lack of awareness and specific knowledge that robots currently have. Also, in order to support the implementation of simple commands, state of the art systems typically require a significant engineering effort to be deployed in a specific scenario. Additionally, even with such an effort, robots often are unable to accomplish their tasks due to unexpected situations unforeseen by their developers at programming time.

Because of these issues, a growing number of researchers has started to accept the fact that our robots will have to cope with their own limitations for a long period of time. Consequently, in recent years such deficiencies have started to be encoded in the robots' internal representation, in order to recognize situations that can not be managed autonomously. For example, robots are currently unable to autonomously recognize all the common objects found in a given environment. Also, they are currently unable to reliably manipulate the significant variety of ordinary items found in domestic or office environments. In such situations, multiple authors have proposed to rely on the interaction with the users to enable robot operation. By proactively asking for help through a vocal interface, multiple

robots have been successfully deployed in real scenarios [1, 2].

Nevertheless, even if we accept robots' limitations and allow them to exploit human help to successfully achieve their goals, there are still various issues that need to be addressed. Reliably grounding the given commands into the operational environment, understanding the specific needs of every unique user or resolving the ambiguities arising from human-robot interactions are among the problems that still require a complete solution. As we will show more in detail in the next chapter, recent works have proposed different solutions to such problems. However, such solutions often rely on general representations of the operational environment and the users in it. For instance, common models of objects and rooms found on the web or handcrafted are used for recognition algorithms. Models of common language expressions or user behaviors are assumed while designing human-robot interactions. General user needs are considered while devising the task that the robots will have to execute. We believe that such generalizations of peculiar aspects of the operational environment are not always valid. Instead, we imagine robots able of adapt to the need of every user and to every setting they are deployed in to effectively carry out their tasks.

Following this approach, in this work we aim at enabling robots to better adapt to such scenarios by exploiting human-robot interactions. Specifically, through the interaction with the user, we aim at acquiring specific knowledge about the operational environment, the tasks that need to be accomplished, and the individual humans living in it. We believe that these three types of knowledge will support adaptable robotic behaviors.

Environmental Knowledge

When first deployed in a new environment a robot needs to first obtain a specific spatial and semantic representation of the environment to support its task execution. In literature, the process of acquiring environmental knowledge for robots is called *semantic mapping* [3]. The maps created during this process are an abstract representations of the objects and rooms contained in the environment, associated to physical and spatial properties that are used by robots to reason during task execution. Indeed, when robots are assigned a command, they need to "contextualize" it in the operational environment through a grounding process. Semantic maps are thus needed to ground commands in the physical world.

In recent years, multiple approaches have been proposed to build semantic maps. Many of the techniques proposed try to exploit knowledge found on the web or specifically handcrafted, for enabling robots to autonomously learn about the environment they are deployed in. However, current approaches for autonomous semantic mapping still have problems allowing a robotic system to enter an unknown environment and create a rich and accurate semantic map. One issue that withholds the realization of such a task is the difficulty of reliably detecting and classifying the objects found in everyday environments. Dedicated algorithms are being developed to robustly recognize common objects and rooms

given a huge variety of prior examples [4]. While these approaches will in the future improve their performance on common objects, the problem is even more difficult in the case of uncommon or less conventional ones.

Accepting the current limitations of our robots, we propose to exploit human help to build semantic maps. Specifically, we contribute an approach to perform semantic mapping in indoor environments, leveraging multimodal interactions with non-expert users. The user guides the robot through vocal commands, teaching it about objects and rooms in the environment with the aid of a laser pointer. Through this mechanism, our robots are able to enter an unknown environment and create a specific and rich semantic map. This map is then used to execute tasks involving previously unknown objects and rooms, also supporting various form of reasoning. Finally, this semantic map can be updated through human-robot interaction to cope with the dynamic nature of the environment, and allow for long term deployment.

Procedural Knowledge

Once a robot is able to acquire environmental knowledge, it also needs to be able to learn tasks specified by the user. In fact, robots often need to cope with multiple situations that cannot be foreseen at design time, due to the complexity of the environment they are deployed in and the specific features and habits of every human they interact with. In literature, different mechanisms have been proposed to enable robots to perform tasks. For instance, many robots require their task steps to be directly programmed. Others require actions, states and goals to generate tasks or policies. Others are able to learn through the observation or interaction with a user.

Nevertheless, only recently new approaches have been proposed to allow non-expert users to teach robots new tasks. In particular, some researchers have proposed to exploit natural language to teach robots. In these approaches, the robot is provided with a set of sensing and action capabilities that it already knows how to perform. The user then describes the new tasks through natural language as a combination of these primitives. As an example, let's consider a manipulator robot. This robot might be able to open and close its grippers, detect objects and move its arms toward them. In this case, the user could describe how to pick up and store an item in terms of these primitives.

As we will see in the next chapter, most of the teaching approaches based on natural language only enable users to teach specific tasks (e.g., delivering the blue book to room 123), without allowing them to teach more general ones (e.g., delivering an object to a location). Additionally, these approaches do not leverage previous task knowledge to speed up the teaching process, while considering only single non interacting robots. Our contribution to the state of the art of interactive task teaching addresses three problems. We first discuss an approach for allowing non-expert users to teach to a robot more general and parametric

tasks. Next, we discuss a method for extracting general tasks from a library of known ones to support the user in the teaching process by suggesting possible autocompletions. Finally, we show how specific coordination techniques can be exploited to teach multiple robots how to coordinate through a natural language interaction with the user. With these contributions we show how to enable multiple-coordinating robots to proactively learn new parametric tasks through the natural language interaction with the user.

User Knowledge

All the contributions previously outlined heavily rely on the interaction with the user. Due to the peculiarity of every human, a robot should also be able to learn and adapt to specific user characteristics. To be able to achieve this goal, robots should first be able to acquire specific models of users. In literature, the process of acquiring knowledge about users is called *user profiling*. A user profile can be defined as a model that summarizes the essential information about a specific human. These models are used to represent the different preferences, interests, background and goals of a human using a specific technology. Discovering and modeling these differences is vital to provide users with personalized services.

While a considerable amount of user profiling approaches has been developed for software applications [5], robots able to adapt to user preferences have been rarely considered due to the limited amount of systems operating in human-populated environments. Nonetheless, in the future we imagine robots capable of adapting to various aspects of every user they interact with. In particular, while observing users interacting with our robots, we noted two specific aspects that varied between users: different language expressions used to interact with a robot and different understandings of qualitative spatial relations.

Specifically, we consider two problems related to user profiling. First, we consider the scenario in which a human needs to instruct an unknown autonomous robot through a natural language interface. In this case, the user may be able to imagine its capabilities, while not knowing how to instruct it. To this end, we present an approach that allows a robot to learn new expressions used by the user to refer to objects in the operational environment. This mechanism also enables the user to recognize the robot understanding capabilities and learn how to interact with it. Additionally, we consider the scenario in which a user instructs a robot using a qualitative spatial relation (e.g., "go in front of the desk near the entrance"). In this setting, we notice that different users understand different positions given the same spatial relation. To this end, we present a preliminary approach that allows the robot to learn specific models of these spatial relations through natural language interaction. This learning process is carried out by asking the user for a feedback after the completion of each task. While multiple other issues regarding user profiling still need to be tackled, these two contributions provide valuable examples of successful adaptations to specific aspects of human individuals.

Structure of the Thesis

The rest of the dissertation is organized in the following way: Chapter 2 discusses the work most closely related to this thesis, pointing out the aspects in which our work contributes to the state of the art. Chapter 3 describes our approach to environmental knowledge acquisition based on multimodal human-robot interaction. Next, Chapter 4 discusses our contributions to task teaching, considering three issues related to this research topic. Finally, Chapter 5 addresses the problem of user profiling for improving robotic task execution, while Chapter 6 concludes the thesis.

Related Publications

We would like to note that parts of this thesis have been published in the following articles, conference and workshop papers:

- ◇ *Living with Robots: Interactive Environmental Knowledge Acquisition*. G. Gemignani, R. Capobianco, E. Bastianelli, D. Bloisi, L.Iocchi and D. Nardi. Robotics and Autonomous Systems, RAS 2015.

- ◇ *Multi-Robot Task Acquisition through Sparse Coordination*. G. Gemignani, S. Klee, M. Veloso, and D. Nardi. International Conference on Intelligent Robots and Systems, IROS 2015.

- ◇ *Language-Based Sensing Descriptors for Robot Object Grounding*. G. Gemignani, M. Veloso, and D. Nardi. 19th Annual RoboCup International Symposium, 2015. Best Paper Award.

- ◇ *Approaching Qualitative Spatial Reasoning About Distances and Directions in Robotics*. G. Gemignani, R. Capobianco, and D. Nardi. 14th Conference of the Italian Association for Artificial Intelligence, AI*IA 2015.

- ◇ *Graph-Based Task Libraries for Robots: Generalization and Autocompletion*. S. D. Klee, G. Gemignani, D. Nardi, and M. Veloso. 14th Conference of the Italian Association for Artificial Intelligence, AI*IA 2015.

- ◇ *Teaching Robots Parametrized Executable Plans Through Spoken Interaction*. G. Gemignani, E. Bastinaelli, and D. Nardi. 14th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015.

- ◇ *On Task Recognition and Generalization in Long-Term Robot Teaching (Extended Abstract)*. G. Gemignani, S. Klee, M. Veloso, and D. Nardi. 14th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015.

- ◇ *Task Recognition and Generalization in Long-Term Robot Teaching*. G. Gemignani, S. Klee, M. Veloso, and D. Nardi. Autonomous Robots and Multirobot Systems, ARMS 2015.

- ◇ *Knowledge-Based Reasoning on Semantic Maps*. R. Capobianco, G. Gemignani, D. Nardi, D. Bloisi, and L. Iocchi. Knowledge Representation and Reasoning in Robotics Symposium at AAAI Spring Symposium, 2014.

- ◇ *Symbiotic Semantic Mapping*. G. Gemignani , D. Nardi, D. D. Bloisi, R. Capobianco, and L. Iocchi. International Symposium on Experimental Robotics, ISER 2014.

- ◇ *Automatic Extraction of Structural Representations of Environments*. R. Capobianco, G. Gemignani, D. Bloisi, D. Nardi, and L. Iocchi. 13th International Conference on Intelligent Autonomous Systems, IAS 2014.

- ◇ *On-line Semantic Mapping*. E. Bastianelli, D. Bloisi, R. Capobianco, F. Cossu, G. Gemignani, L. Iocchi, and D. Nardi. 16th International Conference on Advanced Robotics, ICAR 2013.

- ◇ *Knowledge Representation for Robots through Human-Robot Interaction*. E. Bastianelli, D. Bloisi, R. Capobianco, G. Gemignani, L. Iocchi, and D. Nardi. Knowledge Representation and Reasoning in Robotics Workshop at 29th International Conference on Logic Programming, 2013.

Chapter 2

Related Work

There is a growing consensus that some form of knowledge representation needs to be embedded in robotic systems, in order to enable them to operate in various types of environments and interact with humans through natural language. Such ideas are traceable in different forms within multiple areas of robotics. Historically, common sense knowledge has tried to be embedded in many computer programs, starting as early as 1963 [6]. Subsequently, common sense knowledge has started to be applied to robots. In particular, Shakey [7] has been the first example of general-purpose mobile robot able to reason about its action and interact with users. Such a robot could perform tasks that required planning, route-finding, and rearranging simple objects.

After the early-day work in AI, knowledge representation for robots has mainly focused on addressing the problems of representing the environment [8] and the actions [9] to be performed. In the former work, Kuipers and Byun present one of the first methods for robot exploration, mapping, and navigation in large-scale spatial environments. In the latter article, Levesque and Reiter present one of the first languages for representing and executing actions on an autonomous robot deployed in a dynamic and incompletely known world. Such works have inspired a notable amount of projects, affecting, up to these days, several aspects of the research fields regarding robotics and artificial intelligence.

One of the ultimate goals of the Robotics and Artificial Intelligent research community consists of designing an autonomous robot able to intelligently operate in human-populated environments. However, there still exist numerous technical limitations that stop the achievement of such a goal. To this end, many researchers have started to represent perceptual, physical, and reasoning limitations in their robot internal representation and began leveraging humans by proactively asking for their help. In literature, the concept of allowing a robot to recognize its own limitations and proactively ask for help is known as Symbiotic Autonomy [1] or Symbiotic Robotics [10].

This thesis embraces such an approach of relying on humans to overcome robotics limitations and to allow them to adapt to the different scenarios encountered. Due to this

fact, in the rest of this chapter we will first address the related work on the central topic of Human Robot Interaction and Symbiotic Autonomy. Next, we will discuss the related work on acquiring and representing environmental knowledge (also known in literature as semantic mapping), later discussing the literature on action representation and task teaching. We then conclude the chapter with a brief overview of the main works on Qualitative Spatial Reasoning in robotics. Such a research topic is in fact mainly related to the problems addressed in Chapter 5 of this thesis.

2.1 Related Work on Symbiotic Autonomy and HRI

As previously stated, Symbiotic Autonomy [1] or Symbiotic Robotics [10] is a general philosophy adopted for robot design. When embracing such a principle, robots are not seen any more as fully autonomous, solitary machines working in an unmodified and unknown environment. Instead, they are seen as pervasive robotic systems working in symbiosis with people and their environments. By embracing this developing philosophy, researchers have started to explicitly represent inside robots their own limitations, in order to decide when to exploit human help to overcome their inabilities. Due to the nature of such an approach, Symbiotic Autonomy heavily relies on human-robot interaction for task execution.

Human-Robot Interaction (HRI) is the field that studies and designs robotic systems that need to interact with humans. Interaction, by definition, requires communication. Communication between a human and a robot may take several forms, but these forms are largely influenced by whether the human and the robot are close to each other or not. Generally, remote interaction with mobile robots is often referred to as teleoperation or supervisory control, while remote interaction with a physical manipulator is often referred to as telemanipulation. Instead, proximate interaction with mobile robots may take the form of a robot assistant, and it may include a physical and social interaction. In this thesis, we will mainly focus on proximate interaction through natural language briefly surveyed in the following.

2.1.1 Natural Language HRI

Initial studies on natural language understanding can be traced back to SHRDLU [11], a system able to process natural language instructions to perform actions in a virtual environment. Inspired by this system, multiple researchers extended SHRDLU's capabilities into real-world scenarios, soon starting to tackle related problems, including natural language on robotics systems.

Research has applied speech-based approaches to deploy robotic systems in a wide variety of environments. For example, these approaches have been used in manipulators [12, 13], aerial vehicles [14], and wheeled platforms [15, 16]. Moreover, some robots present

on the market support vocal interactions with users, such as the NAO Humanoid [17], and robuBOX-Kompa [18]. Moreover, several prototypes have been developed for social robots carrying out specialized tasks, such as attending as a waiter [19], as a receptionist [20] or as a bartender [21]. Some of these specialized tasks target industrial goals, such as assembly [22], or moving objects [23]. Dialog has also been used to teach robots how to accomplish a given task, such as giving a tour [24], delivering objects [25], or manipulating them [26]. Finally, other related works have combined speech-based approaches with other types of interactions [27, 16]. Specifically, in the former work the authors have developed a theory of mind for the interacting user, built upon perspective taking, multi-modal communication, and a symbol grounding capability. Instead, in the latter case, the authors present a multi-modal approach for building on-line a semantic map of the environment.

More recently, several domain-specific systems that allow users to instruct robots through natural language have been presented in literature. For example, Kollar *et al.* [28] and MacMahon *et al.* [29] present different methods for following natural language route instructions by decoupling the semantic parsing problem from the grounding problem. In these works, the input sentences are first translated to intermediate representations, which are then grounded into the available knowledge base. Instead, Chernova *et al.* [30] show how to enable natural language human-robot interaction in a scenario of collaborative human-robot tasks, by data-mining past interactions between humans. Dzifcak *et al.* [31] address the problem of translating natural language instructions into goal descriptions and actions by exploiting $\lambda-$calculus. However, these approaches are not able to incrementally enhance their natural language understanding from the continuous interaction with the user. Such a problem has been faced by Kollar *et al.* [32]. By exploiting the dialog with the user, in this work the authors present a probabilistic approach able to learn referring expressions for robot primitives and physical locations in a map.

In the works presented in this thesis we heavily adopt natural language interactions. Specifically, during the interactions we first capture the meaning of the commands uttered by a user through the concept of *frames* [33], later grounding them as discussed in Section 3.5.2. Additionally, in Chapter 5 we present an approach inspired to the work presented by Kollar *et al.* [32]. However, we make an additional step forward, assuming the user to be unaware of the capabilities and the internal representation of the robot. With this assumption, we propose an approach for allowing a robot to recognize unknown object properties contained in the received commands and warn the user about them. With this approach, on one hand the user is able to understand over time what a robot can and cannot ground. On the other hand, the robot can leverage past interactions to learn new references used by the user to refer to specific objects in the environment.

2.2 Related Work on Semantic Mapping

While developing cognitive robots, there has been a growing tendency to introduce high-level semantic knowledge into the developed systems. The question of what should count as semantic knowledge has been asked repeatedly in the history of philosophy. More recently, semantic knowledge has been the subject of much debate in the Artificial Intelligence community.

Hertzberg and Saffiotti [3] characterized *semantic knowledge* in robotics by two properties: the need for an explicit representation of knowledge inside the robot; and the need for grounding the symbols used in this representation in real physical objects, parameters, and events. In the editorial, the two authors stress the fact that, independently from the specific formalism adopted to represent semantic knowledge, semantic mapping goes far beyond attaching labels to a set of sensor data. These labels have in fact to be embedded in a domain theory in order to allow the robot to manipulate and perform reasoning on them. In other words, to become meaningful concepts, the labels need to co-exist with other categories in some form of ontology in order to avoid inducing meaning just to a human observer and not to the robot. Additionally, the acquired semantic knowledge should be effectively grounded in the robot's sensor and motor signals. Knowledgeable robots should in fact be able to recognize objects and assess the truth or falsity of propositions from the sensor data stream of the given sensors.

Nowadays, many robotic systems embody some sort of semantic knowledge. In general, the methods proposed for semantic mapping can be grouped into two main categories, by distinguishing fully automatic methods from approaches involving a human user to help the robot in the semantic mapping process.

2.2.1 Fully Automated Approaches

The category of *fully automatic* methods rely only on the sensors data available to the robot, not considering the possibility of interacting with the human. This category can be divided into three different approaches.

A first set of techniques aim at acquiring features of the environment from laser based metric maps to support labeling and extract high-level information. For example, in Nüchter *et al.* [34], the author present a system able to extract semantic information from 3D models built from a laser scanner. This work is inspired by previous work on 3D scene analysis in vision [35], however it is limited to the classification of surface elements, such as ceilings, floors, and doors. Such a work has been later enhanced by Galindo *et al.* [36]. In this work, environmental knowledge is represented by augmenting a topological map with semantic knowledge provided by the anchoring concept [37]. In particular, the topological map is extracted using fuzzy morphological operators from an occupancy grid map of the environment. This map is later enriched with semantic knowledge through a previously

developed computational framework for anchoring [38].

An alternative second set of techniques uses classification and clustering for automatic segmentation and labeling of metric maps. For example, the generation of 2D topological maps from metric maps is described in Goerke and Braun [39]. In this work, the authors present a framework to build semantic annotated maps by using a classifier based on AdaBoost on the laser range measurements acquired by a mobile robot. Alternatively, Brunskill *et al.* [40] present an online method for generating topological maps from raw sensor information. Specifically, the authors first describe an algorithm to automatically decompose a map into submap segments using a graph partitioning technique known as spectral clustering. Then they show how to train a classifier to recognize graph submaps from laser signatures. Finally, Friedman *et al.* [41] propose a technique for mapping the topological structure of indoor environments based on Voronoi Random Fields.

A third set of techniques for object recognition and place categorization is based on visual features. For example, Wu *et al.* [42] describe an approach to Visual Place Categorization based on sequential processing of images acquired with a conventional video camera. In particular, the authors present a solution based upon a visual feature known as Census Transform Histogram. Alternatively, Mozos *et al.* [43] present a system that transforms depth and grey scale images taken at each place into histograms of local binary patterns whose dimensionality is further reduced following a uniform criterion. The histograms are then combined into a single feature vector which is categorized using a supervised method. Finally, Hermans *et al.* [44] propose a 2D-3D label transfer based on Bayesian updates and dense pairwise 3D Conditional Random Fields. This approach allows to use 2D semantic segmentations to create a consistent 3D semantic reconstruction of indoor scenes.

Significant progress has been made in fully automated semantic mapping [45], however even the most recent approaches still lack of robustness and generality. For such a reason, multiple members of the Artificial Intelligence and Robotics community have proposed to rely on human interaction to obtain a better semantic representation of the environment.

2.2.2 Human Augmented Mapping

In the *human augmented mapping* approaches, the user actively supports the robot in acquiring the required knowledge about the environment. Additionally, the user role is exploited in grounding symbols to objects that are still autonomously recognized by the robotic platform. In this case, the human-robot interaction is often unimodal, and typically achieved through speech.

For example, Zender *et al.* [46] describe a system able to create conceptual representations of indoor environments. In this work, a robotic platform owns an *a priori* knowledge about spatial concepts and, through them, builds up an internal representation of the environment acquired through low-level sensors. The user, throughout the acquisition process,

supports the robot in place labeling. Alternatively, Pronobis and Jensfelt [47] present a multi-layered semantic mapping algorithm that combines information about the existence of objects and semantic properties about the space, such as room size, shape, and appearance. These properties decouple low-level information from high-level room classification. The user input, whenever provided, is integrated in the system as additional properties about existing objects. Moreover, Peltason *et al.* [48] present a mixed initiative strategy for robotic learning by interacting with a user in a joint map acquisition process. This work however considers only rooms, not enabling the system to embed objects in the semantic map. Finally, Nieto-Granda *et al.* [49] adopt human augmented mapping based on a multivariate probabilistic model to associate a spatial region to a semantic label. A user guide supports a robot in this process, by instructing the robot in selecting the labels.

Few approaches aim at a more advanced form of human-robot collaboration, where the user actively cooperates with the robot to build a semantic map, not only for place categorization and labeling, but also for object recognition and positioning. Such an interaction is more complex and requires natural paradigms to avoid a tedious effort for non-expert users. For this reason, multi-modal interaction is preferred to naturally deal with different types of information. For example, Kruijff *et al.* [50] introduce a system to improve the mapping process by clarification dialogs between human and robot, using natural language. A similar construction of the representation is also addressed in Hemachandra *et al.* [51], where the robot learns the features of the environment through the use of narrated guided tours, and it builds both the metrical and topological representations of the environment during the tour. Spatial and semantic information are then associated to the evolving situations through "events labeling", that occur during a tour, and are later attached to the nodes of a Topological Graph. Finally, Randelli *et al.* [52] propose a rich multi-modal interaction, including speech, vision, and the use of a pointing device (similarly to Kemp *et al.* [53]), enabling for a semantic labeling of environment landmarks that makes the knowledge about the environment actually usable. However, the authors do not attach any additional semantic information to the landmarks other than their position.

In Chapter 3 of this thesis, we approach the problem of semantic labeling the environment through the combined use of a laser pointer and natural language, allowing the robot to learn incrementally over time, thus realizing a form of on-line semantic mapping. Specifically, in Section 3 we present a novel and automatic system able to extract from a generic metric map a Topological Graph of the environment, on top of a symbolic grid-like representation. We also describe how such an abstract representation can be enriched through the interaction of the system with a human, later used to solve spatial referring expressions. Compared with the related work, we not only represent objects as points in the metric map, but we associate a semantic meaning to them, creating a semantic map that holds multiple information besides the position of the object (e.g., dimensions, colors, 3D models), which is needed by the robot for task execution and reasoning.

2.3 Related Work on Task Representation and Teaching

A second major issue addressed while developing cognitive robots consists of understanding how to represent the tasks to be executed by the robots. In this section we will briefly describe the most common methods used for representing and executing robotic tasks. We then conclude this section giving an overview of the most important related work on task teaching found in literature.

2.3.1 Task Representation and Execution

Most of the approaches that have been proposed in the past few years for representing and executing robotic behaviors can be grouped into three broad classes: Finite state automata (FSA) based approaches, Belief Desire Intention (BDI) approaches, and Petri Net (PN) based approaches. It is worth noticing that there are a number of approaches that do not fall into these three categories. Such approaches are mostly programming tools for robotics, with limited or no underlying formal model. These approaches usually have ad-hoc semantics and do not support formal analysis, making it difficult to develop robust and effective behaviors. The resulting tools often take the form of frameworks for ordinary programming languages. For example, Execution Support Language (ESL) [54] is a language for encoding execution knowledge in embedded autonomous agents. In particular, this language extends LISP by defining several constructs commonly used in robotics. In a similar way, the Task Description Language (TDL) [55] extends C++. TDL programs have hierarchical representations, called Task Trees, which are composed by asynchronous processes with explicitly represented constraints, called Tasks. The Reactive Action Packages (RAPs) [56] are instead expressed in LISP-like syntax and describe concurrent tasks along with execution constrains. RAPs are an ad-hoc tool for execution of concurrent tasks in robotic applications, which have some similarities with PNs. In these frameworks no analysis of the resulting behavior is possible and coding coherent behaviors requires a considerable modeling effort. For these reasons many approaches have relied on formal models for behavior representation and execution.

FSA-based approaches

Many robot programming languages are based on Finite State Automata. FSA are either used explicitly, possibly supported by a graphical language, or they are used to provide the underlying semantic model for the chosen programming language. For example, Colbert [57] is a robot programming language that was developed as a component of the Saphira architecture [58]. Colbert has a syntax that is a subset of ANSI C, while its semantic is based on FSA. In particular, in this language states correspond to actions while edges are events associated to conditions. Colbert allows some simple form of concurrency even though, in this case, the semantics are considerably different from standard FSA semantics and it

is very hard to ensure coherence in the behaviors. Xabsl [59] is another and more recent example of an approach based on hierarchical finite state automata. Xabsl is bundled with a set of language specific tools, which allow for an efficient development of behaviors. The language has been successfully applied to specify the behaviors of many robotic platforms, mainly in the domain of RoboCup robot soccer. Also one of the representations for robotic tasks used in this thesis is also based on Finite State Automata. In particular, Instruction graphs have been devised to represent tasks as acyclic graphs, which are composed of nodes representing finite state machines [24]. Finally, it is worth noticing that some methods for an automated FSA-based plan generation have also been developed (e.g. [60]).

FSA-based approaches stem from the need to implement effective behaviors in real-time systems [61]. Several other frameworks have been implemented using this approach, proving the effectiveness of FSA in real world applications. Although modeling behaviors based on FSAs is a very intuitive task, these approaches have been mostly limited to single-robot systems due to the lack of expressiveness in modeling concurrency.

BDI-based approaches

The Belief, Desire, Intention framework (BDI) [62] has been proposed as an alternative to FSA-based robot programming. In a BDI architecture, an agent selects behaviors to be executed (intentions), based on its goals (desires), and the current representation of the environment's state (beliefs). The procedural knowledge is typically encoded in a predefined library of plans (e.g. [63]). In order to obtain the desired balance between reactivity and goal-directed behaviors, an agent can commit to the execution of plans and periodically reconsider them. One key advantage of BDI over FSA-based robot programming is that the designer does not needs to specify a predefined ordering of basic behaviors, allowing the robot to draw the executed plans from a potentially very large search space. Several architectures inspired by the BDI framework have been proposed for modeling Multi-Agent Systems (MAS). Two notable examples of such architectures that also model collaboration among multiple agents are STEAM [64] and, more recently, BITE [65]. STEAM is implemented with a focus on collaborative behaviors, by relying on Cohen and Levesque's Joint Intentions Theory [66]. In STEAM, agents distributedly monitor the execution of collaborative behaviors (which are organized in a partial hierarchy of joint intentions), possibly reorganizing the team. In STEAM, cooperation is implemented through a set of complex domain-independent rules, incorporated in the architecture to form sophisticated hierarchical team structures. Instead, the BITE architecture was specifically designed for robotic applications that involve collaboration and coordination. To manage teamwork, BITE maintains an organization hierarchy, a task/sub-task behavior graph, and a library of hierarchically linked social interaction behaviors. Although no explicit methodological guidance to teamwork design is provided, one of BITE's strengths consists of the possibility of specifying different types

of interaction templates (e.g., for synchronization and task allocation). Such templates can be reused to automate the task selection of individual robots in different scenarios. BITE is focused on the automation of teamwork, while the design of individual behaviors is not extensively addressed. As with other systems based on BDI, BITE does not provide formal validation tools to verify the consistency of the designed behaviors.

PN-based approaches

A solution to robot programming that recently gained interest in the scientific community is the modeling of robotic behaviors through Petri Nets. In particular, there has been a considerable effort in modeling MAS through PNs [67], given their capability of representing concurrent systems and shared resources. PN-based systems offer two main advantages with respect to FSA [68]: PN languages are a super-set of FSA-based languages, due to memory and concurrency characteristics. Therefore, the set of modeled roles and behaviors is potentially richer when using PNs; secondly, PNs allow for automatic analysis and verification of formal properties on the performance of the modeled systems. Available tools such as PIPE [69] or TimeNET [70] are able to check PN properties, both through simulation and Markov Chain analysis.

Due to these facts, PNs have been widely used in literature to model Discrete Event Systems (DES), such as manufacturing systems [71]. The PN-based modeling of robotic tasks started with the pioneer work of Wang *et al.* [72], where PNs were used to implement the Coordination Level of Saridis, a 3-level hierarchy for intelligent machines. In the past few years, multiple approaches to plan generation and representation based on PNs have been proposed, addressing both multi-agent (MAS) and multi-robot systems (MRS).

Action representation using PNs in MAS is proposed, for example, in a work by Celaya *et al.* [73]. This model is limited to purely reactive agents: actions are instantaneous, as they are represented by PN transitions. Instead, places represent the environmental state of the agent. In a MAS framework, typical issues encountered in embodied agents such as non-instantaneous actions or uncertain action effects are not specifically addressed. Interactions among agents have also been modeled by PNs in literature, with a special focus on the formal modeling of conversations using colored PNs [74]. A substantial comparative review of the different approaches, including a colored PN model of multi-agent conversations, where places explicitly represent joint interaction states and messages, can be found in [75]. Poutakidis *et al.* [76] introduced interaction protocols, specified using Agents UML and translated to PNs, to debug agent interaction. The debugger uses the PNs to monitor conversations and to detect when protocols are not correctly followed by the agents. While these works provide formal scalable models of interaction, they do not model actions explicitly, and they are not concerned with commitment issues, which are relevant in applications where the interaction among agents/robots is cooperative.

A first group of works using PNs for designing robotic systems develops ad-hoc models for specific applications rather than providing a formal language for robot programming. For example, in a work by Sheng and Yang[77], PNs are used to model a multi-robot coordination algorithm that is based on an auction mechanism to perform environment exploration. Similarly, [78] shows an agent-based extension of Fuzzy Timed Object-Oriented PNs (proposed in [79]) for the design of collaborative multi-robot systems for a specific industrial application. Another example is [80], where the authors report the use of distributed agent-oriented PNs for the modeling of a Multi-Robot System for playing soccer. These works focus mainly on the PN that controls the robotic system and on its execution, providing no systematic method to program such a controller using PNs and/or for the analysis of the whole system properties.

A second kind of PN-based approaches models robotic systems by representing plans as PNs in order to analyze their properties and/or to synthesize optimal plans from conditional ones. In [81], the authors propose an approach for modeling single-robot systems. In this case, users define several possible plans to carry out a task. Then, a reinforcement learning algorithm is used to select an optimal solution. This approach also exploits formal analysis of the PN models allowing for qualitative evaluation (i.e. stability, controllability and possibility of error recovery). This single-robot approach has been tested in two real world applications regarding manipulators and mobile robots. A follow-up work is presented in [74], but in this case the PN model explicitly includes a representation of the environment. Nevertheless, none of these works provide a formal PN-based language for plan description and composition. Furthermore, no models of cooperation or coordination are proposed.

A third category of works addresses specification and execution monitoring of plans for multi-robot systems using PNs. The compilation of plans for multiple robots into PNs for analysis, execution, and monitoring is proposed by King *et al.* [82]. In this work, plans for each single robot are generated either by using a graphical interface or by using some automated planning method. The operators that are used for the PN representation of the plans are inspired by the STRIPS [83] planning system. Supervisory control techniques are applied to the PN controller in order to identify possible conflicts that may arise due to the presence of shared resources among multiple robots. To deal with unforeseen events, re-planning is used at run-time, which severely limits the applicability of this approach to real-rime systems in dynamic environments. Novel supervisory control techniques are also introduced and applied to simulated and real sensor networks, thereby mixing static and mobile sensors in [84]. Another formal framework for robotic collaboration based on an extension to PNs, known as workflow nets, is introduced by Kotb *et al.* [85] to establish a protocol among mobile agents/robots based on the task coverage they maintain. PNs are used to ensure the soundness of the framework and to quantify task performance and determine goal state reachability. However, none of these works provides a formal PN-based language for plan description and composition.

Out of the large body of work on modeling robot and multi-robot behaviors through PNs, Ziparo *et al.* [86] proposed Petri Net Plans as a possible systematic approach to robot behavior programming and verification. In fact, this approach addresses cooperation in multi-robot teams supported by a well-defined plan specification language.

2.3.2 Task Teaching

The problem of teaching a robot new tasks through the interaction with the user is a new research topic that has emerged in the last decade. Usually, the tasks to be learnt are represented as a composition of primitive behaviors that the user can use as background knowledge to teach the robot. The sequence of such actions is often represented in a graph-like structure that is encoded in a specifically designed language. For example, Matuszek *et al.* [87] and Chen and Mooney [88] use a language called Robot Control Language (RCL) and compound action specifications, respectively, for representing and executing route instructions, parsed from natural language commands. These works, although focusing on understanding and executing the description of a command from the natural language interaction with a user, do not analyze the problem of learning new actions to be used in later stage.

Nicolescu and Matarić [89] face such a problem, by presenting a first and simple method for learning from demonstrations of high level tasks, which are built from a pre-existing behavior knowledge. In this work, the learning approach is divided in two main phases: first, the user gives a demonstration of the action to be learnt, while in a second phase the robot has the opportunity to refine the acquired capabilities, by practicing for a small number of trials under the teacher's supervision.

The problem of teaching new tasks to a robot through natural language is also analyzed in Rybsky *et al.* [90], where the authors introduce a method for teaching tasks in the form of directed acyclic graphs, composed of available action primitives. In their work, the task is verbally described and interpreted through a grammar-free Automatic Speech Recognition (ASR). As in previous works, the task to be learnt is first demonstrated by the user. Next, the robot has the possibility of engaging the human in a spoken language dialog to query any unspecified effects of conditional alternatives. Differently from previous works, the authors focus the learning process on natural language interaction, also allowing the user to teach behaviors that involve conditional branches. A similar contribution is also given by Weitzenfeld *et al.* [91], where they address the problem of teaching soccer skills to a team of robots via spoken language. In this latter approach, the authors design a predetermined set of actions and queries that are associated with a predefined grammar. This grammar specifies all the possible natural language commands understood by the robot. The proposed vocabulary includes a set of soccer behaviors (e.g., shoot and pass), and if-then-else control expressions. An interesting aspect of this framework is the possibility of directly querying

the robot for its internal state. However, the expressive power of this work is limited, being unable to handle loops and not allowing the user to correct previously taught actions.

These two contributions have been presented in Meriçli *et al.* [92], where the authors describe an enhanced approach that allows the user to teach actions involving loops. Their speech interface is based on the Google free-form ASR and they represent actions as Instruction Graphs. This system also enables the user to correct an existing behavior by letting the robot step through the sequence of actions of a chosen task. This revision approach works really well for simple behaviours, but may become cumbersome as task complexity increases.

The above-cited approaches make significant steps forward in extending the language used for task specification. However, they do not address the problem of teaching a robot parametric tasks. It has been shown, by strong linguistic theories based on cognitive studies [33], that when humans talk about an action, they refer to a general structure representing its underlying concept, which is characterized by a set of arguments participating in it. Therefore, when teaching a new task to a robot, in order to make the learning process more intuitive, some authors have suggested referring to general action structures, instead of teaching instances of more general concepts.

For example, Connell *et al.* [93] describe a simple approach for learning parametric actions. By exploiting the Microsoft ASR Engine the authors show how a robot can be taught how to poke a general object. Open parameters are expressed by using chosen keywords and they are specified at run-time. However, the proposed method is limited to a sequence of actions that do not include branches or loops. Moreover, the actions learnt can be characterized only by a single open parameter, leaving the open question of how to generalize the method to multiple parameters.

A final example tackling the problem of learning parametric tasks is presented in She *et al.* [94]. In this work, the authors describe a three-tier representation that supports both the conversion of natural language into robot actions and the application of existing planning algorithms. However, this framework does not allow to represent conditionals and loops and it does not enable the user to revise previously taught tasks.

Differently from these works, in Chapter 4 of this thesis we have considered three problems regarding task teaching through natural language interaction. First of all, we have addressed the problem of allowing a user to teach or specify to a robot new possibly parametrized actions by combining primitive behaviors. Our system makes a significant step forward by allowing the user to refer to multiple parameters during the teaching process, allowing for a rich task specification language, inspired by the Robot Control Language (RCL) [87]. The Task Description Language, in fact, differently from RCL, allows to also represent conditionals, parallel actions as well as variables. Moreover, by associating execution plans in PNP, we rely on an execution semantics that let us capture more expressive task specifications (e.g., action parallelization). This double representation of the tasks

allows us to decouple the problem of capturing the wide variety of linguistic expressions uttered by a user from the problem of defining a clear operational semantics. Additionally, with such a two-fold representation, we are able to perform task revision in a high-level fashion.

The second task-teaching problem addressed in this thesis consists of specifying new tasks involving multi-robot coordination through natural language. Indeed, none of the approaches described above allow a human to teach multi-robot coordination. In the multi-agent planning domain, sparse-coordination has been introduced as a method to overcome planning for large joint state spaces [95]. We apply this same concept to the task teaching domain to separately instruct the role of each robot in a joint plan represented as Instruction Graphs [92].

Finally, we have considered an autonomous robot that persists over time interacting with users and the problem of teaching an additional task to it. We believe that the assumption that a user would know all the tasks previously taught to the robot does not hold. We hence investigate the problem of recognizing when a user is teaching a task similar to one it already knows. We present a graph-based task representation, and contribute algorithms to measure task similarity, to perform task generalization, and to generate task proposals, as a user teaches new task actions to an agent. Tasks are accumulated in a library, and the agent can learn and represent common patterns among tasks.

2.4 Related Work on Qualitative Spatial Reasoning in Robotics

In order to communicate, robots need to be able to understand humans. The conventional numeric approach used in robotics is in fact deeply different from natural language interactions that occur between people. The former is based on precise metric information about already well known environments, in which each element is uniquely specified only through its coordinates. The latter can deal, instead, with ambiguities and spatial uncertainties, which are solved by referring to purely qualitative properties of objects, or relations among them. Many difficulties arise when trying to move from a numeric representation of the world to a qualitative one [96]. Especially if the commands to be executed by the robot are given through speech.

To this end, a considerable amount of effort of the research community has been dedicated to the development of qualitative reasoners. Specifically, in literature qualitative spatial relations have been studied and used in various applications for robotics. For example, in the "CogX"[1] project [97] the spatial relations "in" and "on" are used to define object targets for indirect object search. Instead, Kunze *et al.* [98] use information about landmark objects and their spatial relationship to the target object to show how a searching task can be improved. Hawes *et al.* [99] exploite qualitative spatial reasoning to enable a robot to

[1] http://cogx.eu/

recognize regions of space that are not simply described by the geometry of the environment, but also by their function. McClelland *et al.* [100] use an extension of the *double cross calculus*, introduced by [101], to express robot navigation objectives that include spatial relations in a Mars-like environment. Loutfi *et al.* [102] look at the same problem in the context of perceptual anchoring to provide qualitative relations inferred from observed metric relations.

In general, every qualitative spatial reasoner adopts one or multiple specific frame of references. In literature, three different frames of reference are usually distinguished: *deictic* where the orientation is imposed by the point of view from which the reference object is seen; *extrinsic* where external factors impose an orientation on the reference object; and *intrinsic*, where the orientation is given by some inherent property of the reference object [103]. Various works have proposed approaches to abstractly represent objects and reason about their directions and distances, adopting one of the aforementioned category of reference frames. For example, [104] presents specific systems for composing distances, in addition to two distinct approaches for describing the direction of a point with respect to another reference point in an environment: the *cone-based* approach and the *projection-based* one. A generalization of such direction calculi was proposed by [105]. In this paper, the authors present a method, called *star calculus*, for representing and reasoning about qualitative directions of arbitrary granularity. A further approach, both for orientation and distance reasoning, was instead developed by [101], defining the direction of a located point to a reference point with respect to a perspective point (the so called *double cross calculus*), later augmented with distance information. Finally, other approaches, instead of using points, approximate spatial regions with geometric primitives, often projecting them to the chosen reference axis, resulting in a multidimensional Allen's interval algebra [106]. One of these approaches is the so called *rectangle algebra* [107], where all regions are represented as rectangles whose sides are parallel to the axes determined by the frame of reference.

Qualitative Spatial Reasoning approaches can also be defined as "propositional" or "pictorial" [108]. The former type uses a natural language description of space that however cannot easily express structural properties, being focused on formal properties of the representation itself. The latter, uses a graphical approach to describe space, providing however a low level representation that is not suitable for fast computations. In his work, Hernández suggests using a hybrid representation, interfacing these two categories as separate representations. Inspired by [108] and adopting a similar approach, in Chapter 3 we propose a method for performing qualitative spatial reasoning in robotics, where the interface between the metric information and the symbolic knowledge is represented by a grid-based structure automatically built by our robot. On top of the representation, we adopt a reasoning approach that exploits shapes for distance and orientation qualitative calculus. Finally, in Chapter 5 we exploit qualitative spatial reasoning to learn particular spatial user preferences, which are related to tasks to be accomplished by a robot.

Chapter 3

Acquiring Environmental Knowledge

This work has been published in:

- ◇ *Living with Robots: Interactive Environmental Knowledge Acquisition*. G. Gemignani, R. Capobianco, E. Bastianelli, D. Bloisi, L.Iocchi and D. Nardi. Robotics and Autonomous Systems, RAS 2015.
- ◇ *Approaching Qualitative Spatial Reasoning About Distances and Directions in Robotics*. G. Gemignani, R. Capobianco, and D. Nardi. 14th Conference of the Italian Association for Artificial Intelligence, AI*IA 2015.
- ◇ *Automatic Extraction of Structural Representations of Environments*. R. Capobianco, G. Gemignani, D. Bloisi, D. Nardi, and L. Iocchi. 13th International Conference on Intelligent Autonomous Systems, IAS 2014.
- ◇ *On-line Semantic Mapping*. E. Bastianelli, D. Bloisi, R. Capobianco, F. Cossu, G. Gemignani, L. Iocchi, and D. Nardi. 16th International Conference on Advanced Robotics, ICAR 2013.
- ◇ *Symbiotic Semantic Mapping*. G. Gemignani , D. Nardi, D. D. Bloisi, R. Capobianco, and L. Iocchi. International Symposium on Experimental Robotics, ISER 2014.
- ◇ *Knowledge-Based Reasoning on Semantic Maps*. R. Capobianco, G. Gemignani, D. Nardi, D. Bloisi, and L. Iocchi. Knowledge Representation and Reasoning in Robotics Symposium at AAAI Spring Symposium, 2014.
- ◇ *Knowledge Representation for Robots through Human-Robot Interaction.* E. Bastianelli, D. Bloisi, R. Capobianco, G. Gemignani, L. Iocchi, and D. Nardi. Knowledge Representation and Reasoning in Robotics Workshop at 29th International Conference on Logic Programming, 2013.

When a robot is deployed in an unknown domain, the first type of knowledge that it needs to acquire is the knowledge about the environment it operates in. In fact, with this type of knowledge the robot will be able to navigate and ground the commands to the objects and rooms around it. Because of this fact, in this chapter we present an approach that allows a robot to incrementally learn the knowledge about the environment, by relying on a rich multi-modal interaction with the user. We specifically address the problem of acquiring the knowledge about the environment (i.e., the semantic map) and maintaining it. Compared with previous work, our approach can be seen as an *incremental on-line semantic mapping*, in which a rich and detailed representation of the operative scenario is built with the help of the user. The resulting integrated representation enables the robot to perform topological navigation, understanding target locations and the position of objects in the environment.

The approach described in this paper builds on previous work for off-line construction of semantic maps [52], based on two main components: A component for Simultaneous Localization And Mapping (SLAM), that provides a metric map of the environment; A multi-modal interface that allows the user to point at the elements of the environment and assign them a semantic role. The novel contributions of this work are the following:

- The representation of the environment, which is automatically extracted from the metric map and is labelled through user interaction. On this representation different form of symbolic AI techniques can be applied.
- The process of building and updating the representation, which is done incrementally during the deployment of a robot through a continuous interactive process;

By exploiting the representation of the environment built through this process, we are able to support several forms of reasoning, task execution and complex interactions.

The above listed features have been embedded in a prototype system that has been extensively used over months to validate the proposed approach, in substantially different environments, and with multiple users. This robotic system is a fully functioning prototype able to enter an unknown environment and incrementally create a semantic map that supports the execution of complex tasks in a partially dynamic environment. A semantic mapping approach similar to the one presented here could be deployed on robots that are currently entering the market, such as telepresence robots [109]. In this specific scenario, users could be allowed to give a tour of their homes or offices to the robot, teaching it about relevant objects or rooms in the environment, possibly in different moments. The knowledge acquired through this process could then be used to simplify the teleoperation of a robot by allowing both the remote and the nearby user to command the robot using natural language. For example, users could just instruct the robot to reach a previously learnt room or object, without the need to tele-operate it.

In the remainder of this chapter we will first outline the aim, assumptions and contributions of our work. Then, we will describe the representation of the robot's knowledge. Next, we will show how our robots can acquire and maintain environmental knowledge, followed by an overview of the human-robot interaction mechanisms. Finally, after discussing the system implementation on four different robots, we will illustrate the experiments that we have carried out to validate the proposed approach.

3.1 Aim, Assumptions and Contributions of the Work

Semantic Mapping is the process of gathering information about the environment and creating an abstract representation of it, to support the execution of complex tasks by robots. In literature, semantic maps have been defined as maps containing, in addition to

spatial information about the environment, labelings of mapped features to entities of known classes [3]. Formally, Semantic maps can be defined as a triplet [110]

$$SM =< R, G, P > \tag{3.1}$$

where:

- R is the global reference system in which all the elements of the semantic map are expressed;
- G is a set of geometrical elements obtained as raw sensor data. They are expressed in the reference frame R and describe spatial information in a mathematical form;
- P is a set of predicates that provide for the semantic interpretation of the environment.

As outlined in the previous chapter, semantic maps can be either created in an automatic process or through the interaction with the user. The process of semantic mapping can be formally defined as:

$$f_{SM} :< M, E >\rightarrow SM \tag{3.2}$$

where M is a given metric map and $E = \{e_1, e_2, ..., e_n\}$ is a set of events that occur while the robot is exploring the environment. For instance, for automatic semantic mapping approaches, these events might consist of the recognition of a particular object in the environment. Instead, for human aided semantic mapping, such events might consist of different interactions between the robot and the user. These interactions could be realized through different interfaces, such as natural language, tablets or other forms of human-robot interaction.

Independently from the type of approach chosen, the systems proposed in literature are often designed to learn every possible aspect of the environment. Moreover, semantic mapping is seen as a separate and independent process that needs to be carried out before task execution. We note that robots do not need to gather a complete knowledge of the environment to be able to execute tasks. Thus, we propose a shift in perspective, allowing the user to decide what the robot should and should not know in order to carry out the assigned tasks. Moreover, we note that not only semantic mapping does not have to be carried out in a single and independent step, but also that incremental approaches might better suit semantic mapping techniques that rely on the interaction with the user. Hence, to enable the user to better aid the robot, we propose an incremental semantic mapping mechanism based on a multimodal interaction with the user. Formally, incremental semantic mapping can be defined as

$$\phi_{ISM(t)} :< M, e(t), SM(t) >\rightarrow SM' \tag{3.3}$$

where $e(t)$ is the event occurred at time t, $SM(t)$ is the semantic map built up to time t, and SM' is the new semantic map obtained after applying the function (i.e., after processing $e(t)$).

In order to design a system that supports an incremental semantic mapping, three fundamental questions need to be addressed: i) How to abstractly represent environmental knowledge; ii) How to build this representation; iii) How to use the chosen representation to enable task execution. Solutions to these three questions are present in the literature, but they have been considered as separate components. One contribution of this work is thus a more comprehensive evaluation of an integrated system that fully implements an incremental semantic mapping approach. In the rest of this section, we describe the motivations and the choices made to develop the proposed solutions to the three issues mentioned above, with respect to previous work in the literature. Instead, in the next sections we provide implementation details and discuss additional implementation choices.

3.1.1 Environmental Knowledge Representation

As seen in the previous section, multiple representations have been proposed to represent environmental knowledge. Our semantic map mostly relates to the one described by Goerke and Braun [39]. Specifically, we developed a specific four-layered representation of the environment for the setting considered. In fact, while considering a mobile base that needs to operate in a human-populated environment, we require the robot to navigate to certain locations of the environment. We hence build a 2D grid based representation, borrowing the idea from video-games. Indeed, in video-games, the problem of associating logic, symbols and behaviors to areas of the game-scene has been studied since a long time [111, 112]. However, differently from the top-down approach used in video-games, where the logic influences the scene, we propose a bottom-up representation, in which the environment is strictly conditioning the symbolic layer used by the robot. We device this representation to fully support a rich semantic mapping process. In fact, instead of limiting our robot to annotate objects and areas on the metric map, we develop a four-layer representation, introducing an abstract layer that allows for the use of multiple symbolic AI techniques (such as planning, symbol grounding, and qualitative spatial reasoning). This representation is presented in Section 3.2.

3.1.2 Knowledge Acquisition

Multiple approaches have been proposed in literature to acquire a specific representation of the environment, ranging from fully automatic techniques[45] to human-robot interfaces based on natural language [113] or tablets [114]. Our semantic mapping process mostly relates to the one proposed by Diosi *et. al.* [113]. In this work, the user is exploited to label areas visited by the robot through natural language. By contrast, while aiming at deploying our mobile bases in real human-populated environments, we devised a multimodal interface to enable non-expert users to support the robot in the map acquisition process. Specifically, in this process, the user guides the robot through vocal commands, teaching it about objects

and rooms in the environment with the aid of a laser pointer. Through this mechanism, any non-expert user is able to support our robots, that are in this way able to enter an unknown environment and incrementally create a specific and rich semantic map. Differently from other works discussed in literature, we leverage human-robot interaction, by allowing the user to decide what aspects of the environment the robot should learn. In this process, the user takes an active role by teaching the robot about objects and areas with the aid of a laser pointer. As outlined in Section 3.3, this particular interaction has been devised to be intuitively used by any non-expert users, being them elder or young. Moreover, the interaction with the user enables the system to partially handle the dynamic aspects of the environment. In fact, we allow the user to update the semantic map by moving or removing objects in it.

3.1.3 Knowledge Usage in Common Tasks

In literature, only few works present a fully functioning robotic system able to acquire a semantic map and use it to carry out tasks assigned by users. One of such approaches is presented by Zender *et. al.*[46]. In this work, the authors present a fully functioning system based on natural language interaction. Differently to the approach described in their work, we build a richer four-layered representation that is created incrementally and on-line through human-robot interaction. In our approach, any non-expert user can teach the robot about objects and rooms of the environment through a simple yet effective interaction scheme. The representation obtained through this process is then used to support the execution of complex tasks, issued by the user to the robot through natural language. In fact, we created a fully functional robotic system relying on the semantic map to accomplish tasks assigned by different users. Thanks to our representation, we are able to apply symbolic AI techniques to support the robot during reasoning and planning operations. In particular, by focussing on tasks that require the robot to reach certain positions, we show how the robot can ground such commands into the built semantic map and reason about the objects and rooms described in it. This particular aspect of our prototype is discussed in Section 3.4.

3.1.4 Assumptions

The proposed solution aims at being general and domain independent. However, its implementation is based on some assumptions that are discussed in this subsection. These assumptions have been devised for the specific implementation that we have developed, but they do not represent actual limitations of the general approach presented in this work.

We have developed our approach by considering a mobile base that is required to reach certain locations in the environment as specified by the user through natural language. Hence, a 2D semantic map of the environment is considered expressive enough to support task execution. However, the representation could be extended to 3D to support task

execution for other types of robots. Moreover, we have assumed that users know how to interact with the robot through the developed interface, having been briefly explained before through demonstrative examples. This assumption has been verified during the system evaluation process described in Section 3.6. During such experiments, users only required a brief training and a couple of examples to fully learn how to interact with the system. Additionally, we have assumed that the objects pointed by the user through the laser pointer are distinguishable by either depth or texture and color. Again, such an assumption has been verified in the case scenarios analyzed during the evaluation of our prototype. Finally, we have assumed each object in the environment to have an intrinsic front side. Such a front side is used to establish a common reference frame for the qualitative spatial references used by the user in the natural language interactions with the robot, as explained in Section 3.4. We acknowledge that the assumption that each object has an intrinsic front side is a strong assumption. This assumption has been made observing how users point to objects while interacting with robots. Such an assumption is used in the grounding of the qualitative spatial references found in the command given by the user to the robot. The assumption has been made to establish a common reference frame for the vocal interactions between the human and the robot. A robust object classification and matching with object models would help dropping this assumption, but such an issue was not the focus of this work.

3.2 Representation of the Robot's Knowledge

As previously discussed, how to abstractly represent environmental knowledge is the first issue that needs to be addressed while designing an incremental semantic mapping approach. In our robots, environmental knowledge is divided in two layers: *(i)* the *world knowledge*, which encloses the specific knowledge about the environment acquired by the robot; *(ii)* the *domain knowledge*, consisting in a general knowledge about the domain (Figure 3.1).

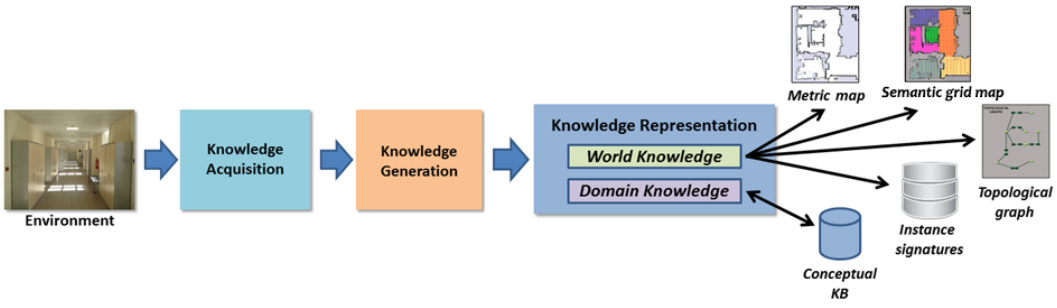


Figure 3.1. Representation of the robot's knowledge.

It is important to point out that, while the two layers may resemble the extensional and intentional components of a classical knowledge base, here they are independent of each other. In fact, the world knowledge may be inconsistent with the domain knowledge

(e.g., the robot may have discovered a fridge in the living room rather than or in addition to the one in the kitchen, while the domain knowledge may state that *"fridges are in the kitchen"*), which is generally used to support the action of the robot, only when specific world knowledge is not available. For example, the robot is only able to reach an object (e.g., the fridge in the living room) if it has been previously discovered; else, if there is no specific knowledge about the position of the fridge, the system will query a general domain knowledge base to find out its possible location (e.g., the kitchen). To better explain this point, the fact that a fridge has been found, for example, in the living room and that the robot knows through human dialog that a particular kitchen has no fridges (in contrast with the conceptual knowledge base that states that *"fridges are located in kitchens"*) does not negatively affect any reasoning of the robot when a fridge is discovered elsewhere.

In this chapter, we will use the term $\mathcal{C}oncept$ to refer to a set of symbols used in the conceptual KB and to denote general concepts (i.e., abstraction of objects or locations). For example, $Fridge$ and $Kitchen$ are concepts in the general domain knowledge base. The term $\mathcal{L}abel$ will be used, instead, to refer to a set of symbols that indicate specific instances of objects or locations. For example, $fridge1$ could be a label denoting a particular fridge, while $kitchen$ could be a label denoting a particular kitchen. In the notation used in this chapter, concept names will be capitalized, while labels will have all lower-case letters, typically followed by a digit. Finally, the associations between labels and concepts will be denoted as $label \mapsto Concept$. For example, $fridge1 \mapsto Fridge$ denotes that the label $fridge1$ is an instance of $Fridge$ (i.e., $fridge1$ is a fridge). Next, we will describe in detail how the aforementioned knowledge structures, pictured in Figure 3.1, are subdivided.

3.2.1 Domain Knowledge

In previous work, domain knowledge has typically been characterized as a conceptual knowledge base, representing a hierarchy of concepts, including their properties and relations, *a priori* asserted as representative of any environment. The conceptual knowledge base usually contains a taxonomy of the concepts involved in the environment tied by an *is-a* relation, as well as their properties and relations [36, 115]. These concepts are used in the world knowledge to characterize the specific instances of the environment, as previously explained. In our representation, three top-most classes have been considered: *Areas*, *Structural Elements*, and *Objects*. *Areas* denote places in the environment (corridors, rooms, etc.), *Structural Elements* are entities that form the environment and that topologically connect areas (windows, doors, etc.), while *Objects* are elements in the environment not related to its structure and located within areas (printers, tables, etc.). A snapshot of a portion of the conceptual knowledge base used in the experiments is reported in Figure 3.2.

Finally, in the conceptual KB we have included synonyms. It is possible, in fact, to refer to the same object with different natural language expressions (e.g., referring to a *"plug"*

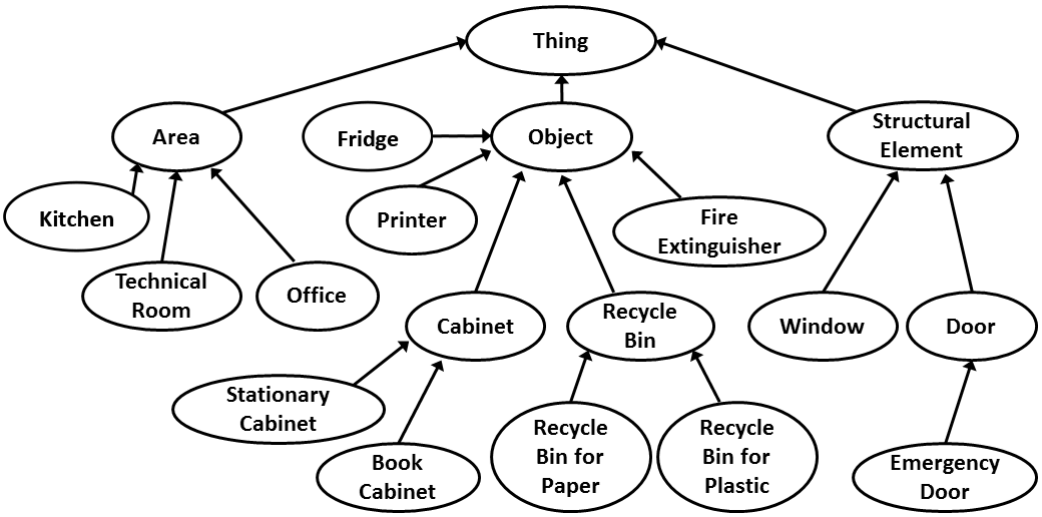


Figure 3.2. A fragment of the conceptual knowledge base used in the experiments.

also with the word *"socket"*). Moreover, we use the structure of the taxonomy during the natural language interactions with a user, by looking at a more specific or more generic concept (e.g., it is possible to refer to a *"book cabinet"* with the word *"cabinet"* and vice versa).

3.2.2 World Knowledge

A semantic map is typically characterized as a low-level representation of the map (i.e., a grid), that is labeled with symbols. Such symbols denote, for example, the area corresponding to a room (e.g., a corridor) or the presence of objects and structural elements of the environment. Our representation builds on a similar structure, supporting however a much more detailed description, based on the interaction and hints provided by the user. Such an environment representation is called World Knowledge and it is composed by the following elements:

Metric map The *metric map* is represented as an occupancy grid generated by a SLAM method. This map has usually a fine discretization and it is used for low-level robot tasks, such as localization and navigation. In Figure 3.3 a metric map with a resolution of 5 cm generated is shown in the background of the image, where black pixels represent occupied cells and grey pixels resemble empty spaces.

Semantic Grid Map The *Semantic Grid Map* is represented as a discretization of the environment in cells of variable size. Each cell represents a portion of a physical area and it is an abstraction of locations that are not distinguishable from the point of view

of robotic high-level behaviors. The Semantic Grid Map also includes a connectivity relation $Connect \subseteq Cell \times Cell$, that describes the connectivity between adjacent cells. In Figure 3.3, the Semantic Grid Map contains multiple cells that are delimited by borders in different colors. Each color corresponds to an area reported in the legend to the right. Connectivity relations between cells are not explicitly shown in Figure 3.3, but they can be derived by looking at adjacent cells.

Topological Graph The *Topological Graph* is a graph whose nodes are locations associated to cells in the Semantic Grid Map and edges are connections between these locations. In Figure 3.3, the Topological Graph is depicted as a graph connecting oval nodes. Dark-colored nodes correspond to static locations, while light-colored nodes correspond to variable locations, which are associated to the main areas of the environment. The static locations denote specific positions that are of interest for the robot tasks (e.g., the position to enter in a room), while the variable locations are used to denote areas, where the instantiation of the position for a navigation behavior is executed at run-time, depending on the current status of the robot and on its goals. Since the Topological Graph is used by the robot for navigation purposes, the edges also contain the specific navigation behavior that is required for the robot to move from one location to another. In this way, the Topological Graph is also used to generate appropriate sequences of behaviors to achieve the robot's navigation goals. For example, by knowing that there is a door that must be traversed to go from a point to another, a particular behavior must be adopted by the robot that will be included in the Topological Graph.

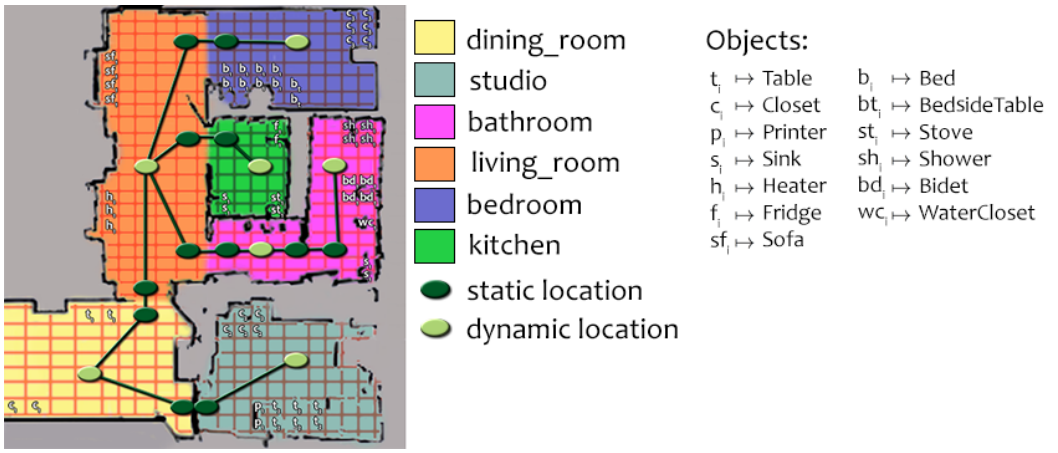


Figure 3.3. An example of world knowledge.

Finally, our specific representation of the environment is populated with **Instance signatures**. The *instance signatures* are represented as a data base of structured data, where each instance has a unique label ($l \in \mathcal{L}abel$), an associated concept ($C \in \mathcal{C}oncept$) such that $l \mapsto C$, and a set of properties expressed as attribute-value pairs. Additionally, every

label is associated to a set of cells ($cells = \{c_1, c_2, ..., c_n\}$) and topological nodes ($nodes = \{n_1, n_2, ..., n_k\}$) associated to them. For example, the green cells labeled with f_1 (top-right corner of the kitchen area) are mapped with the labels $\{f_1, kitchen\}$, that are associated with the concepts $Fridge$ and $Kitchen$, i.e. $f_1 \mapsto Fridge$ and $kitchen \mapsto Kitchen$, respectively. Thus, the corresponding area is characterized as being occupied by a fridge and as belonging to the kitchen. Additionally, f_1 can have the following properties: position = $< x, y, \theta >$, color = $white$, open = $false$.

3.3 Acquisition of the Robot's Knowledge

Once we are able to represent environmental knowledge, we need to address the problem of how to acquire such representation. In our robots, the semantic map is built by using two different modalities: an initialization phase and the on-line incremental extension of the initial knowledge.

The initialization phase aims at creating an initial representation of the robot's knowledge. Such a process is a standard approach used by the majority of the semantic mapping methods found in literature. During this phase we have allowed the user to specify the objects enclosed in the environment while acquiring the metric map. The initialization phase can be divided into two steps: the *Metric Map and Instance Signatures Construction* and the *Semantic Grid Map and Topological Graph Generation*. During the first step, a 2D metric map is generated through a graph-based SLAM approach [116, 117] and the initial set of instance signatures is created. In the second step, a grid-based topological representation, called *Semantic Grid Map*, is obtained by using the 2D metric map and a graph, called *Topological Graph*, needed by the robot to perform high level behaviors is built by processing the instance signatures and the Semantic Grid Map.

The on-line modality can be used to enrich the initial knowledge including additional objects and additional properties. It is worth noting that different users can add knowledge to the system, thus the system has to deal with the possible addition of multiple tags for the same object. The details about the two acquisition modalities are given in the following.

3.3.1 Metric Map and Instance Signatures Construction

In the first step of the initialization phase, the robot is used to navigate the environment in order to create a 2D metric map and to register the positions of the different objects of interest. As already described, during the guided tour of the environment, the user can tag a specific object by using a commercial laser pointer (Figure 3.4a). While the object is pointed through the laser, the user has to name the object, so that the *Vocal Interface* module can register the semantic label that will be assigned to it [52].

The *Dot Detection* module is responsible for detecting the laser dot by using the RGBD

Figure 3.4. Object tagging. a) The object is tagged with the laser pointer. b) The color image from the RGBD sensor is used to locate the laser dot. c) The 3D point cloud is extracted. d) The planes not containing the laser dot are discarded. e) The dot is used as seed point to segment the tagged object.

data. By exploiting the odometry data to detect when the robot is not moving, a set of image samples is collected over 3 seconds in order to generate a background model of the captured scene by means of a multimodal statistical approach [118]. After 3 seconds, the robot communicates to the user that it is ready for acquiring the tag and a background subtraction process is carried out over the current RGB image (see Figure 3.4b), thus obtaining a foreground mask. The background subtraction step allows for filtering out possible false positives (e.g., due to illumination sources coming from windows or ceiling lighting). The foreground pixels are then converted into the HSV color space to search for the specific color values of the light dot generated by the laser. To further refine the results, the depth information is used to discard all the points that are above a certain height (2.00 meters in our case) or that are too far from the camera (3.00 meters). The output of the *Dot Detection* module is the image coordinates (i, j) of the laser dot in the RGB image.

Once the dot is found, the *Object Pose Estimation* module is responsible for finding the 2D position x, y and the bearing θ of the tagged element in the metric map [119]. The object pose in global coordinates (x, y, θ) is calculated by taking into account the normal corresponding to the surface of the segmented object in the reference frame of the RGBD sensor and then applying a first transformation to the reference frame of the robot and a second transformation to the reference frame of the map (this is given by a standard self-localization module, i.e., ROS-AMCL).

The coordinates of the laser dot are also used by the *Object Segmentation* module that aims at segmenting the pointed object in order to extract its width (W), height (H), and depth (D) as well as its color properties. The laser dot is then projected onto the 3D point cloud of the scene (Figure 3.4c). All the planes in the scene are extracted from the 3D point cloud and those that do not contain the dot are discarded (Figure 3.4d). The remaining points are analyzed to segment the shape of the object of interest by using the laser dot as a seed point for the expansion and depth discontinuities as stopping criterion (Figure 3.4e).

The color properties are obtained by analyzing the set of pixels belonging to the segmented object. In Figure 3.5 an example of color information extraction is reported. The set of points in the laser scan corresponding to the tagged object are detected by extracting

the neighbor points around the projection of the laser dot onto the laser scan (Figure 3.5b). In order to segment the tagged object, the point cloud is rotated of an angle equal to $\frac{\pi}{2} - \theta$ around the axis orthogonal to the floor. In this way, all the objects are treated as being acquired by the same angle of view, thus allowing for a normalization of the point cloud of the tagged object. The final poses of the tagged objects are stored as instance signatures together with the corresponding properties (color and size).

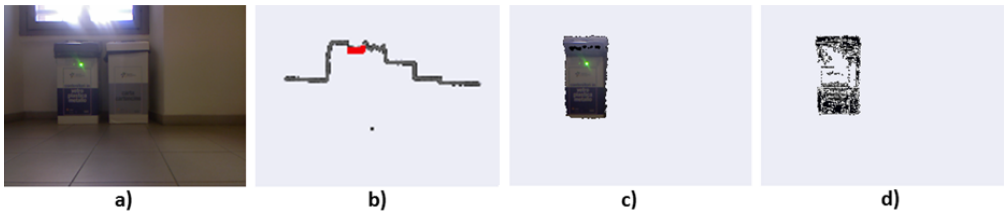


Figure 3.5. Extracting color information from a tagged object. a) The RGB image. b) The laser scan received from the laser range finder. The detected object is highlighted in red. c) The segmented point cloud. d) The color mask corresponding to the blue color.

3.3.2 Semantic Grid Map and Topological Graph Generation

While there are tools and methodologies for building metric maps, and, to some extent, topological graphs, all the knowledge about the environment must usually be provided to the robot in a suitable format by an expert. This is not satisfactory if symbolic and semantic knowledge has to be acquired incrementally or if this process has to be performed on top of a metric map. In fact, neither the occupancy grid, nor the topological graph provide a good representation for adding semantic knowledge. The occupancy grid, in fact, since it has a fine discretization, which is good for a rich description of structural details, does not allow to be easily translated into a symbolic representation, needed for a high-level form of spatial reasoning. The topological graph is suitable for representing the connectivity of the environment with respect to a set of symbols associated to it, but its structure can become very nested and complex, leading to difficulties in physically locating the areas of interest. Moreover, updating the topological graph can become a challenging task, especially if the structure of the environment in the map can change due to the continuous update of the representation.

Given the above considerations, an intermediate abstraction layer, strictly linked to the metric map is indeed useful for both acquiring semantic knowledge and enabling the robot to build a topological graph on top of the actual structure of the building, thus keeping in the high-level description a direct connection with the physical world. Such an abstraction layer can be inserted in a processing chain from the metric map to the symbolic representation of knowledge about the environment, independently both from the mapping methods and from the knowledge acquisition techniques. The Semantic Grid Map and the Topological Graph are generated automatically in the second step of the initialization phase.

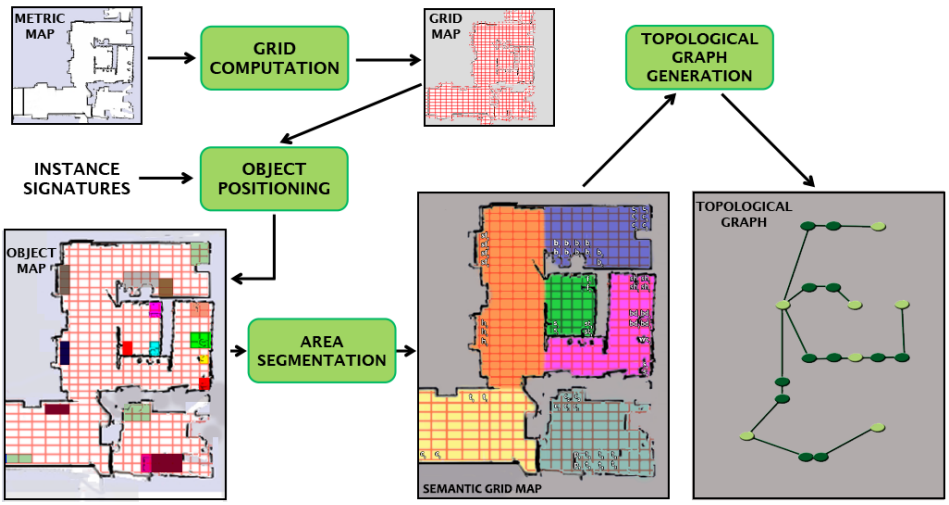


Figure 3.6. Semantic Grid Map and Topological Graph generation. The metric map is enriched with semantic information to obtain the Semantic Grid Map. The instance signatures and the Semantic Grid Map are used to create a Topological Graph.

The Semantic Grid Map contains a high-level description about the regions, structural elements, and objects contained in the environment. It is generated using both the instance signatures and the 2D metric map. The process to generate the Semantic Grid Map is carried out by the modules reported in Figure 3.6 and briefly described below (for details see the article by Capobianco *et al.* [120]).

The *Grid Computation* module is used to generate a rasterization of the 2D metric map into a grid-based topological representation called the *grid map* (not to be confused with the Semantic Grid Map, which is a grid map enriched with semantic information). The operation is accomplished by applying the Hough Transform at different resolution levels to identify the lines passing through the walls. In this way, it is possible to find all the main walls in the map. The cells of the grid have different size with respect to the amount of relevant features in the considered area (e.g., the cells in the corridor are wider then the ones in the offices). In detail, the cells have been chosen to be variable-sized since we want the grid to be built on the basis of the walls and to be consistent with the structure of the environment, which could not be achieved using a fixed-size grid. The cells have a size between $x_{\min} \cdot y_{\min}$ and $2x_{\min} \cdot 2y_{\min}$, where $x_{\min}$ and $y_{\min}$ are calculated by extending the lines generated through the wall detection to the whole image and computing the minimum horizontal and vertical distances. Of course, such a discretization could affect the object localization for small objects, but in general this error is admissible. Specifically, we use the center of the grid cell as a reference for the localization of the objects since it is acceptable both from a qualitative representation and task execution point of view.

The *Object Positioning* module is responsible for placing the representations of the objects and structural elements tagged in the environment into the grid map. The objects'

pose, shape and size, memorized in the instance signature data base in the previous step, are used to label the cells in the grid map. The output is an *object map*, containing labeled cells representing tagged objects.

The *Area Segmentation* module is applied to further enrich the object map, by automatically labeling the different areas of the environment. The object map is divided into different rooms by using the watershed algorithm [121]. The segmentation algorithm is enhanced by considering the tagged objects and structural elements previously included in the environment. For example, the knowledge about the position of doors is used to separate different areas of the environment. The output of the area segmentation module is a *Semantic Grid Map*, containing labeled cells representing tagged objects and areas of the environment.

The *Topological Graph Generation* module allows to complete the initialization phase by creating a *Topological Graph*. Such a graph embodies the information needed to the robot for navigating and acting in the environment and it is generated by exploiting the knowledge contained both in the Semantic Grid Map and in the instance signatures.

3.3.3 On-line Knowledge Acquisition

When the initialization phase is terminated, the robot is ready to incrementally extend its initial knowledge by adding new objects to the Semantic Grid Map. Similarly to the initialization phase, the on-line acquisition is also achieved by tagging objects with the laser pointer (note that we still focus only on static objects). The system operates as for the previous phase, but at the moment of the insertion in the knowledge base, the system updates the symbolic representation accordingly: the Semantic Grid Map and the instance signatures are modified in order to reflect the new situation generated by the addition of a novel object, as discussed in the article by Bastianelli *et al.* [16].

However, since the system is used incrementally and possibly multiple users can interact with it, some difficulties can arise in the maintenance of the knowledge. For example, a user may try to tag an object that is already present in the knowledge representation or he may want to tag an object that is in a cell already occupied by another object. In the first case, the system can detect the problem, since the Semantic Grid Map contains a cell labeled with the same name used by the current user, and it can avoid to add a duplicate label in the Semantic Grid Map. An example for the second case can occur if we want to tag a plug that is located under an already tagged white-board. In this case, since the names of the two objects are different, the system allows to insert two labels in a single cell, leaving to an interaction with the user the choice of keeping or deleting the existing object.

During the on-line acquisition phase, the system can recognize objects already present in the knowledge representation. Considering again a plug as an example, if a user points to it in position (x_1, y_1) and a different plug in position (x_2, y_2) has been previously memorized,

the system tries to recognize the object without the explicit grounding by the user.

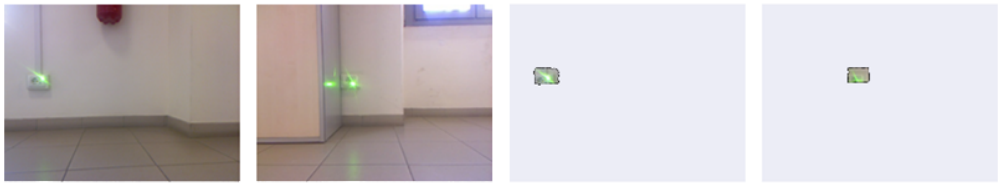


Figure 3.7. The system tries to recognize the class of an object if another instance of the same class has been already memorized by using the SURF features as descriptors.

In Figure 3.7 two plugs are tagged. Since the *Object Segmentation* module transforms each segmented object in the same angle of view, the recognition task is simplified. The two objects are therefore matched by using the SURF features as descriptors and by exploiting the FLANN algorithm to find the correct correspondences.

3.4 Robot Behavior using the knowledge base

The knowledge acquired with the previously described method is used for knowledge maintenance and task execution. These two actions, as well as all the other behaviors needed by the robot to act in the environment, have been modeled using the Petri Net Plans (PNP) formalism [86], based on Petri Nets [68].

3.4.1 Petri Nets

Petri Nets are directed, weighted, and bipartite graphs. A Petri Net has two types of nodes connected by directed weighted arcs, which have a default weight of one. The first type of nodes are called *places* and may contain zero or more *tokens* (Fig. 3.8a and 3.8c). The number of tokens in each place is defined as *marking*, and denotes the state of the system. The second type of nodes are instead called *transitions* and they represent the events modeled by the system (Fig. 3.8b).

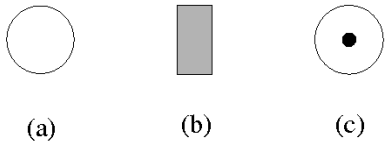


Figure 3.8. Nodes of a Petri Net

Mathematically, a Petri Net can be defined as a tuple $PN = \langle P, T, F, W, M_0 \rangle$, where:

- $P = \{p_1, p_2, \ldots, p_m\}$ is a finite set of *places*.
- $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of *transitions*.

- $F \subseteq (P \times T) \cup (T \times P)$ is a set of edges.
- $W : F \to \mathbf{N}^+$ is a weight function where $w(n_s, n_d)$ denotes the weight of the edge connecting n_s with n_d.
- $M_0 : P \to \mathbf{N}^+$ is the initial marking.
- $P \cup T \neq \emptyset$ and $P \cap T = \emptyset$

Transitions can produce or destroy tokens from places according to the rules that define the dynamic behaviour of the Petri Net. In other words, this behaviour defines how the state of the net changes over time and it is defined by the following *firing rule*:

1. A transition t is *enabled* if each input place p_i is marked with at least $w(p_i, t)$ tokens.
2. If an enabled transition t fires, $w(p_i, t)$ tokens are removed for each parent place p_i and $w(t, p_o)$ are added to each output place p_o.

3.4.2 Petri Net Plans

From Petri Nets, Ziparo *et al.* [86] have defined Petri Net Plans (PNPs). PNPs are particular Petri Nets whose operational semantics are enriched with the use of conditions, which are verified at run-time by querying an external knowledge base. No restriction is imposed on the knowledge base, which is supposed to be updated by other modules according to the agent's perceptions.

PNPs are composed by basic actions that can be combined through a set of possible operators in order to express complex execution paradigms. The basic actions of a PNP are *ordinary actions* (Fig. 3.9a), which represent a durative action, and *sensing actions* (Fig. 3.9b), which represent procedures whose outcomes depend on one or more conditions to be checked in the knowledge base, similarly to if-statements.

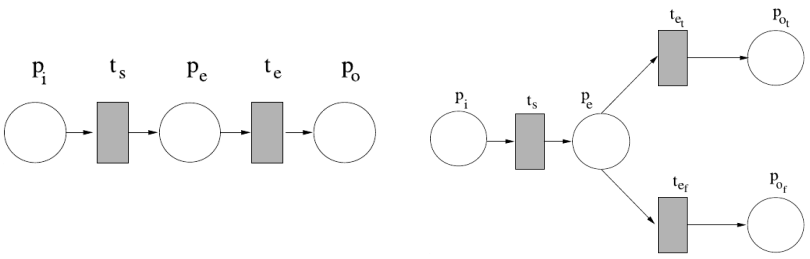


Figure 3.9. Ordinary and sensing actions in a Petri Net Plan

These basic actions can be seen as a specific concatenation of places and transitions of a Petri Net. Each element of an action represents a specific aspect:

- *Input* places (p_i) model the initial configurations of the network before the action has been executed.

- *Execution* places (p_e) model the configurations during which the action is executed. These places are also used to create hierarchical plans by calling a sub-plan while visiting this place.
- *Output* places (p_o) model the final configurations of the network achieved after the execution of the action.
- *Start* transitions (t_s) model the events that trigger the execution of the action.
- *End* transitions (t_e) model the events that trigger the stop of the action. In a sensing action there is a true (t_{e_t}) and false (t_{e_f}) end transition that fire depending on the outcome of the query made to the knowledge base.

These basic actions can be composed using a set of possible operators for single and multi-agent plans. Here we informally introduce some of the single-agent operators, referring the reader to the original paper [86] for a formal and complete definition of the available operators:

- *Sequence*. A sequence is obtained by merging two places of different PNPs. It can be applied to any place except for the execution ones, as shown in Figure 3.10.
- *Interrupt*. Interrupts connect the execution place of an action (or sub-plan) to a non-execution place of another network. It causes the temporary termination of the action (or sub-plan) under anomalous conditions, as shown in Figure 3.11. Interrupts are often used to repeat a portion of a plan that did not realize its post-conditions in order to implement while-loops.
- *Fork* and *Join*. Each token in a Petri Net Plan can be thought of as a thread in execution. Through `Fork` and `Join`, threads can be created and synchronized. An example of a fork followed by a join is shown in Figure 3.12.

Places in the PNP can be used to denote states of execution of the plan, while transitions represent conditions or events that allow for state changes. PNP supports a hierarchical representation, thus execution tokens in one PNP can refer both to atomic actions and to other PNPs (also called sub-PNPs) that can be executed in parallel. Moreover, the presence of markers in a place is used to define a *context* and these contexts are then used for taking contextual decisions. For example, contexts are used for loading the proper grammar at a specific phase of the plan.

In our system we have implemented a simple plan that, based on the input given by the speech component, activates one of the eleven possible actions (Turn, LookAt, Follow, Memorize, Forget, Update, Home, GoToPlace, GetCloser, TellObjectKnown, RecognizeObject). When such an action ends or is interrupted, the system waits for another command given by the user. In the rest of this section, we present two examples of such PNP actions

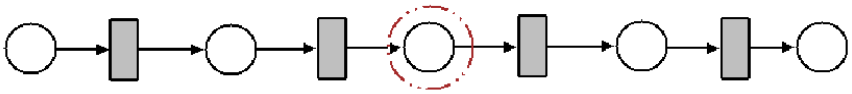


Figure 3.10. The sequence operator

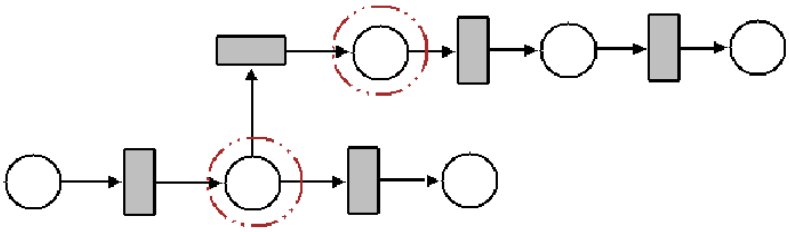


Figure 3.11. The interrupt operator

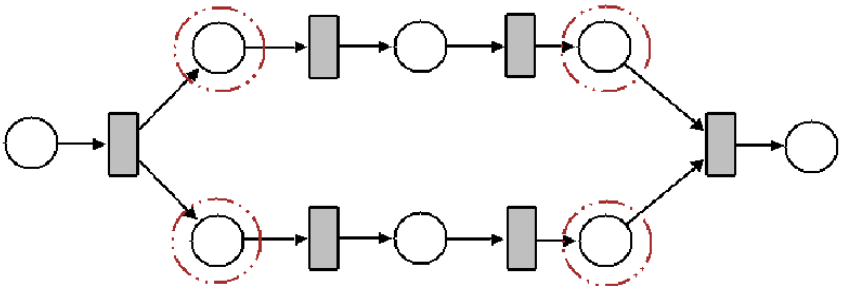


Figure 3.12. The fork and join operators

in order to show how the knowledge can be acquired and used (see PNP[1] for a more detailed description).

3.4.3 Knowledge Base Maintenance

The knowledge acquired by the robot is used, in a first place, for maintaining the knowledge base. In particular, three operations of memorizing, removing, and better specifying the objects found in the environment have been implemented in the system. Figure 3.13 shows a simplified PNP used for memorizing the objects located in the environment through the interaction with the user.

In this plan three `Say`, one `AddObject`, and one `AskForHelp` actions are used to reach the goal. The `Say` actions represent a communication with the user through the text to speech system. `AddObject`, is the action used to store all the information of the

[1] http://pnp.dis.uniroma1.it

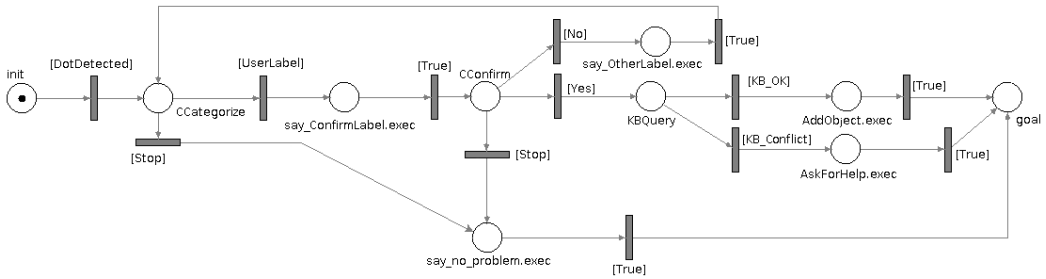


Figure 3.13. Simplified version of the Memorize PNP used in the experiments.

object tagged in the semantic map, while `AskForHelp` represents the action responsible of resolving the possible conflicts generated by the addition of a new object to the semantic map. The conditions used in this plan are the following:

1. `DotDetected`: the user has pointed to the object that needs to be memorized and the laser pointer has been correctly recognized;

2. `UserLabel`: the user has uttered the label for the object to be memorized;

3. `Yes`, `No`: the user has confirmed or not the label understood by the robot for the object to be memorized;

4. `KB_OK`, `KB_Conflict`: possible answers obtained from the query posed to the knowledge base(`KB_OK`: the place/object tagged by the user does not conflict with anything previously memorized; `KB_Conflict`: the place/object tagged by the user conflicts with something previously memorized and the user help is therefore required).

Finally, there are two context places (`CCategorize`, `CConfirm`), that are used to activate contextual behaviors. Specifically, `CCategorize` and `CConfirm` activate special grammars to parse user sentences, representing the label of the object tagged and the feedback to the robot confirmation request, respectively. When a new object is tagged, the knowledge base is queried for other objects located in the same position of the new element. If one or more objects are found, the system tries to resolve the problem by comparing the labels of the new element with those given to the existing ones. If such a label is found to be a synonym or related by a more generic/specific relation to any of the existing ones, the system automatically resolves the conflict by acknowledging the user of the presence of the existing element and by updating the label attached to the object if the new label is more specific than the known one. An example of execution of this plan is given in the following table.

User	*"Memorize."*
Robot	*"I am ready to memorize an object."*
The user points at the object until an auditory feedback is received.	
User	*"This is the book cabinet."*
Robot	*"I already know that a cabinet is here. Now I know that this is a book cabinet."*

The three basic operations (i.e., memorize, remove, and specify) have also been combined in order to create more complex behaviors. For example, when the system cannot resolve the conflict, the user is notified that one or more objects are known to be in the same position of the newly tagged one and she/he is asked whether the new element should be added or substituted to any of the previous ones. An example of such an interaction is given in the following table.

User	*"Memorize."*
Robot	*"I am ready to memorize an object."*
The user points at the object until an auditory feedback is received.	
User	*"This is the fire extinguisher."*
Robot	*"I know that a socket is also here. Should I keep it in my memory?"*
User	*"No."*
Robot	*"Ok, I have removed the socket and I have memorized the fire extinguisher."*

3.4.4 Task Execution

The second case in which the knowledge acquired by the robot is used is task execution. In this case, the knowledge is used for executing the task of moving in front of a specific object by interacting with a user. A simplified version of the GoToPlace sub-PNP relative to such a behavior is given in Figure 3.14.

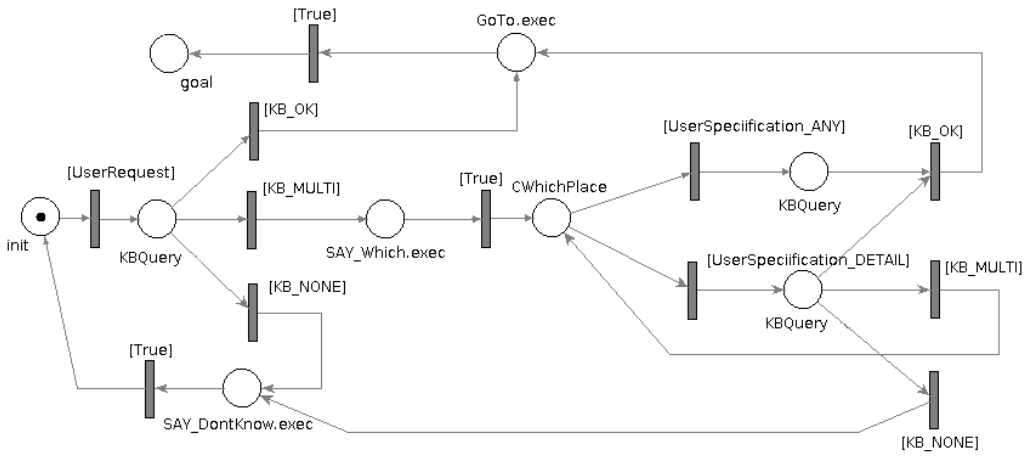


Figure 3.14. Simplified version of the GoToPlace PNP used in the experiments.

Specifically, the plan uses one `GoTo` and two `Say` actions to reach the goal state. The `GoTo` is the action used to move the robot to a particular place while the `Say` action represents a communication with the user through the text to speech system. The conditions in this plan are:

1. `UserRequest`: The user has asked the robot to go to a place/object (the name of the place is stored in the robot's memory during the execution of this plan);

2. `KB_OK`, `KB_NONE`, `KB_MULTI`: Possible answers obtained from the query posed to the knowledge base (`KB_OK` when the place/object given by the user is unique in the knowledge base and its position in the environment is known; `KB_NONE` when there is no place/object with such name in the knowledge base; `KB_MULTI` when there are multiple places/objects with the same name in the knowledge base);

3. `UserSpecification_ANY`, `UserSpecification_DETAILS`: The user has specified which place/object by either answering that any place/object is fine or by giving a more detailed description that is the subject of a new query to the knowledge base.

Finally, there is one context place (`CWhichPlace`) that is used to activate contextual behaviors. In particular, such a context place activates a special grammar used to parse user sentences representing a specification of a place or an object.

The specification of a place or an object can be given either by using spatial relations or synonyms and more general/specific concepts. While the latter specifications can be implemented by embedding a vocabulary in the robot knowledge base, for spatial relations a dedicated spatial reasoner has been devised. In particular, three vicinity relations (*near*, *next to*, and *nearest*), their three opposite relations (*far*, *not next to*, and *furthest*) and four orientations (*behind*, *in front*, *on the right*, and *on the left*) have been implemented. By defining C_{Loc} and C_{Ref} the set of cells belonging to the Semantic Grid Map that include a portion of the objects Loc and Ref respectively, we say that Loc has a vicinity relation with Ref if and only if:

$$d(centroid(C_{Loc}), centroid(C_{Ref})) < t$$

where d is the euclidean distance, t is a threshold constant, and $centroid(x)$ is a function that takes a set of cells x in input and returns the coordinates of its centroid in the metric map coordinate system. By specifying a threshold constant for both the relations *near* and *next to*, t_{near} and t_{next} respectively, we therefore define the six distance relations (the nearest attribute is computed by finding the object that minimizes the above defined distance). In order to define the orientation relations, as in the work by hernandez [108], we exploited the "intrinsic front side" of the objects, identified with the normal of the surface tagged by the user during the learning phase previously described. Specifically, we have used such a

normal to define a forward orientation, later deriving, by rotating clock-wise, respectively the concept of left, backwards, and right regions. By defining the general concept of directions, we adopted the *cone-based* approach [122] to explicate the four directional relations, starting from the centroid of the reference object. By defining A_R^{Ref} the area corresponding to a region in the direction R with respect to the reference object Ref (e.g., $A_{right}^{cabinet}$ is the area on the right of the cabinet), in order for Loc to belong to that particular area, we require that:

$$centroid(C_{Loc}) \in A_R^{Ref}$$

where, again, the $centroid(x)$ is as above defined. Two examples of execution of this plan are given in the following tables.

User	*"Go to the socket."*
Robot	*"There are many sockets in the environment. Which one do you mean?"*
User	*"The one close to the emergency door."*
Robot	*"OK. I am going to the socket."*

User	*"Go to the socket."*
Robot	*"There are many sockets in the environment. Which one do you mean?"*
User	*"Any."*
Robot	*"OK. I am going to the closest socket."*

When the location to be reached is retrieved from the KB and the user is acknowledged with a feedback sentence similar to the ones described above, the system checks whether the robot is located inside the same room where the target is placed. If so, the robot navigates directly to the target without using the Topological Graph, while if not, the system searches the graph for the shortest path to reach the entrance of the *Area* containing the target cell and, from there, it behaves as in the previous case.

3.5 System Description

Our approach has been implemented on a mobile base derived from Segway (Figure 3.15a), on a Videre Design platform (Figure 3.15b), on a Turtlebot (Figure 3.15c), and on a MARRtino, a mobile base built by our students (Figure 3.15d). The standard sensor setting of each mobile base includes a laser range finder for localization and navigation and one (or two) RGBD sensor(s) for the laser dot detection and the object segmentation. The robots carry two additional components, the robot software component and the speech component, which are respectively run by a laptop placed on the robot and by a tablet that can be either held by the user or placed on the robotic platform.

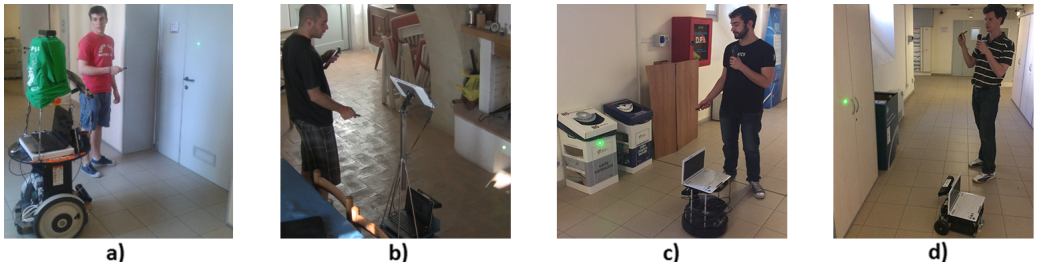


Figure 3.15. Robots on which our system has been deployed. a) Mobile base derived from Segway. b) Videre Design platform. c) Turtlebot. d) MARRtino, a mobile base built by our students.

3.5.1 Robot Software Component

The robot software, including both core robotic functionalities and the system for acquiring and handling the knowledge about the environment, is implemented on ROS[2].

The implementation of navigation relies on standard ROS modules for path planning and obstacle avoidance, that have been specialized for *ad hoc* behaviors, such as "person following", as well as navigation behaviors (e.g., for entering doors), associated with different edges of the topological plan enclosed in the world knowledge.

In order to perform the person following behavior, the detection of the person to be followed is realized by integrating information coming from the laser range finder and the RGBD camera. The poses of these two sensors are calibrated in order to obtain a merged set of range data to process. The detection is based on filtering the current range data with information coming from the localization module and, in particular, the distance map. In this way, it is possible to filter out the range data that are associated to objects that are in the map (low values in the distance map). The remaining data points are assumed to belong to objects not in the map and in particular to the user interacting with the robot. After a clustering and the application of an additional filter on the metric size of the obtained clusters, the closest cluster which is compatible with the typical size of a person is used as target detection for the person following behavior. The procedure is not completely robust to the presence of multiple people in the scene and it may generate false positives when objects that have a similar size of a human (e.g., a plant) appear as new objects in the scene. However, since this work does not focus on people perception and tracking, this simple implementation is robust enough for supporting the semantic mapping task.

The robot can acquire two types of perception data: RGBD data coming from a Kinect sensor and 2D data coming from a laser range finder. The two sources are used to recognize the object that the user wants to insert into the semantic map. In order to do so, the user can point the object through a commercial laser pointer (see Figure 3.15) and provide, through the vocal interface, semantic information about it. A ROS node listening for the data coming from the Kinect has been created for the detection of the laser dot, while a second node is

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

responsible for gathering the 3D points belonging to the tagged object. The second node uses odometry information coming from the robotic platform, in order to calculate the bearing θ of the tagged object. The software for image processing, including the detection of the laser dot, the pose estimation of the tagged object with respect to the position of the robot, the extraction of visual features from the scene, and the matching between objects have been developed using OpenCV and PCL-Point Cloud Library functions.

Multiple ROS services have also been adopted to implement the representation and use of the knowledge about the environment. Such services can be logically divided into two main components:

- The *world representation* component, that includes the knowledge representing the environment and two main processes: one for the construction and update of the knowledge base and the other one for answering queries needed to check the conditions in PNPs, based on the knowledge about the environment. The representation of symbolic knowledge, including the Semantic Grid Map, the Topological Graph, the Taxonomy of conceptual knowledge, as well as the algorithms for solving spatial referring expressions are embedded in a SWI-Prolog module encapsulated within a ROS service.

- The *behavior and dialog maintenance* component, that drives the robot as well as the dialog with the user, has been implemented using our library for Petri Nets Plans[3].

3.5.2 Speech Component

The implemented speech system is a modular component that runs on a dedicated hardware (currently a tablet, a PC or a portable phone). For this component, we have tested different settings (hand-held/omnidirectional microphone, push-to-talk button, wake-up word), that can be easily deployed on each of the four mobile bases. The speech processing takes place on the dedicated device and is divided into two modules: the software dedicated to sending to the robot the result of the interpretation of the user utterances and the software used as a Text-To-Speech engine to vocalize the robot replies.

The speech processing is based on the implementation developed through Speaky for Robot[4] [123]. The language model is specified by defining grammars that drive the recognition process: by attaching a proper semantic output to each grammar rule, we get a representation of the linguistic semantics of a recognized utterance. This representation is based on the *frame* concept, a conceptual structure representing a situation in the world, typically an action, inspired by the notion defined in the *Frame Semantics* linguistic theory [33]. The general meaning expressed by each frame can be enriched by semantic arguments, called *frame elements*, that are part of the sentence and provide additional specification of

[3] http://pnp.dis.uniroma1.it

[4] http://labrococo.dis.uniroma1.it/?q=s4r

the action. The output of the recognition process is then converted to a parse tree containing syntactic and semantic information, that is used to instantiate the associated frame, as it is typically done to represent user commands [124]. As an example, the command *"go to the Phd room"* will be mapped to the MOTION frame, while the sub-phrase *"to the Phd room"* will fill the specific frame element GOAL representing the destination of the MOTION.

The lexical level of the grammars (i.e., the part of the grammar that represents the vocabulary) is organized according to the structure of the conceptual knowledge base. First of all, the words are categorized according to some super-classes (*Areas*, *Objects* and *Structural Elements*). Inside each of these super-classes, the different words are grouped into more specific classes, and subdivided according to their functionalities. In this approach, words like *"table"* belong to the *Furniture* class, while words like *"cabinet"* will belong both to the *Furniture* class and the *Container_furniture* subclass. This organization of the lexical level is useful to specify the set of words that can fill a certain frame element of a frame, according to their semantic function (e.g., for the frame ENTERING the frame element GOAL, i.e. the place where the subject is entering, can be filled only by the words belonging to the *Area* class, that are rooms, or to the *Structural Element* class, such as doors). In this work, we are assuming that the names of the objects and locations not present in the map are known to the speech recognition module and that they can be understood when the user refers to them. However, different speech processing chains that rely on general purpose Automatic Speech Recognition systems (ASRs) and automatic algorithms for querying the web could be used to overcome the need of such an *a priori* knowledge [125].

A request by the user may start a dialog that aims at acquiring additional information when the grounding of an object or a location can not be resolved by the system. During this process, the user can guide the system with voice commands such as *"turn right"*, *"follow me"* or *"go to the Phd room"*. For the references related to objects in the environment, when the robot is in front of the new targets to be grounded, the user can point to the object and tell the robot the natural language reference for it, as *"this is the emergency door"*. A specific CATEGORIZATION frame is then associated with the description command during the grammar decoding phase. In the same way, the CATEGORY frame element, which represents the description or the specification of the subject, is instantiated with the referenced object (i.e., *"the emergency door"*). The lexical representation expressed in the CATEGORY frame element is then linked to the pointed position in the space by using the information obtained through the dot detection system.

The sentences that are recognized by the system belong to different categories including commands and object descriptions. In order to have a better performance of the ASR in the implementation of the dialog, the grammars are loaded dynamically, contextually with the robot's behavior. The dialog maintenance is part of the robot software and it is embedded within the Petri Net Plans module, which is used to specify the robot's behaviors.

3.6 Evaluation

The main goal of the experiments reported in this section is to show the feasibility, the effectiveness, and the robustness of the proposed approach for on-line acquisition of knowledge through human interaction. In order to validate our approach, we have carried out experiments both on single components and on the whole system. These experiments have been conducted by executing the robot behaviors described in the previous section, across different settings, including different robots, several users, and two very different kinds of environments: a home environment and our Department. Multiple videos showing the execution of these behaviors can be found in the dedicated web page[5].

3.6.1 System Component Evaluation

The Semantic Grid Map generation process, the object segmentation module, the speech recognition unit, and the spatial-reasoner are the four main components of our system. Such components have been quantitatively evaluated in order to measure the performance of our implementation.

Semantic Grid Map Generation In order to evaluate the Semantic Grid Map representation, two experiments have been conducted. In the first experiment the publicly available Radish Data Set[6] has been used with the purpose of evaluating, in terms of compactness of the representation, the effectiveness of the proposed discretization, when dealing both with regular and irregular indoor environments. In particular, six 2D metric maps[7], obtained from different SLAM methods, have been processed to extract the Semantic Grid Map on a number of different occupancy grids. In addition, four maps generated by our robots have been processed, for a total of ten different environments. Qualitative results are shown in Figure 3.16. A very good correspondence between the real environment and the structured information within the Semantic Grid Map is achieved. Even if the Semantic Grid Map generation process has been developed for dealing with ordinary (regular) buildings, it provides good results also in environments with irregular edges (e.g., see the last row of Figure 3.16). Quantitative data about the reduction in terms of the size of the representation allowed by our approach are reported in Table 3.1. The results underline that the Semantic Grid Map representation is more compact than the corresponding bitmap (about two orders of magnitude, depending on the complexity of the map).

When objects are located in the Semantic Grid Map, errors in their positions and dimensions are introduced because of the discretization of this map. A second experiment

[5] `www.dis.uniroma1.it/~gemignani/Articles/LivingWithRobots.html`

[6] `http://radish.sourceforge.net/index.php`

[7] The following maps have been used: *albert-b-laser*, *ap_hill_07b*, *ubremen-cartesium*, *intel_lab*, *belgioioso* and a portion of *hospital_floorplan_fort_sam_houston*

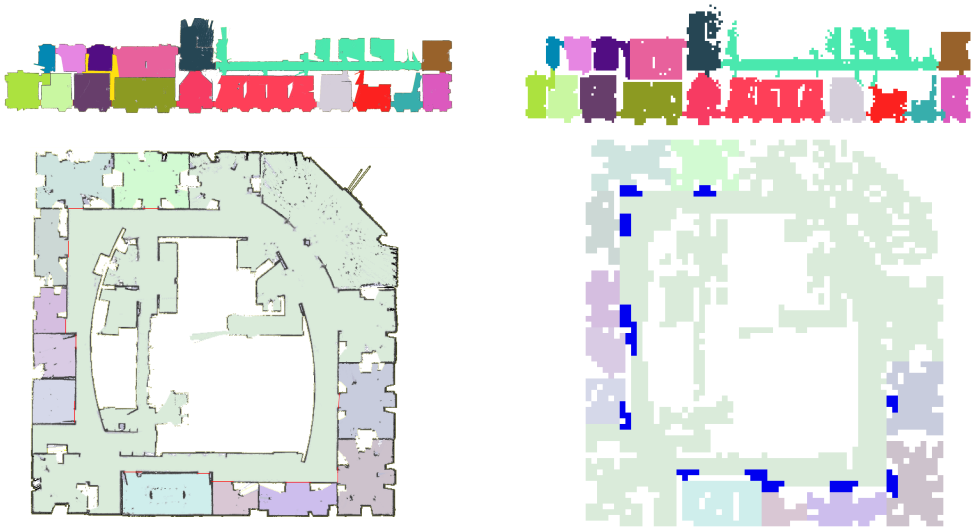


Figure 3.16. Representation obtained for two different maps from the Radish Data Set.

was thus performed in order to evaluate such errors. To this end, we considered 11 instances of 3 different categories of objects in our department. To avoid the errors introduced in the perception layer, we measured the ground truth of objects' position and size and we evaluated their error when represented in the Semantic Grid Map. By overlapping the object representations from the Semantic Grid Map (SGM) onto the metric map (i.e., drawing the occupied cells on the metric map), we compared their area and size in terms of pixels with respect to a ground truth map (GT), thus measuring the error introduced by the use of the Semantic Grid Map. In detail, we measured the errors for the width (W), the depth (D), and for the full area of each object:

$$e_W = \frac{|W_{SGM} - W_{GT}|}{W_{GT}} \qquad e_D = \frac{|D_{SGM} - D_{GT}|}{D_{GT}} \qquad e_{area} = \frac{|area_{SGM} - area_{GT}|}{area_{GT}}$$

The results obtained (reported in Table 3.2) naturally depend on the size of the considered object and on the granularity of the discretization in the portion of the map where the object is located. For example, the representation error for a cabinet, which is big in size and can be easily detected from the metric map, is usually small since it is extracted by the wall detection algorithm. This is true especially on the W dimension, since the metric map of our department is regular. The error in the case of a recycle bin is instead higher, since four cells are required to fully represent such an object. Indeed, even if a single cell representation would decrease the error, it would not be suitable for safe navigation. In the case of the fire extinguisher, the error strictly depends on the granularity of the grid, since this object always requires one single cell to be represented. In general, even if the error for the object area can reach values around 3, losing precision is still acceptable from the point of view of the task execution, since after reaching the desired location on the semantic map, an accurate localization of the objects is performed through perception. Overall, the results show that

Table 3.1. Comparison between the pixels of each processed metric map and the cells of the corresponding Semantic Grid Map.

Map	Pixels	Cells
BelgioiosoCastle	768 792	11 600
dis-B1	1 080 700	10 290
dis-B1-part	501 840	7372
dis-Basement	992 785	13 455
FortAPHill	534 520	7878
Freiburg	335 248	4794
HospitalPart	30 000	285
Intel	336 399	4473
scheggia	92 984	1116
UBremen	831 264	10 962

Table 3.2. Error evaluation for the size and the area of the objects in the Semantic Grid Map with respect to ground truth values.

Object	Cells	$\mathbf{e_W}$	$\mathbf{e_D}$	$\mathbf{e_{area}}$
Cabinet1	3	0.31	0.1	0.18
Cabinet2	3	0.31	0.1	0.18
Cabinet3	6	0.32	0.3	0.71
Cabinet4	6	0.32	0.5	0.97
Cabinet5	3	0.31	0.1	0.18
FireExt.1	1	0.8	0.4	1.52
FireExt.2	1	1.4	0.8	3.32
FireExt.3	1	1.2	0.8	2.96
RecycleBin1	4	0.89	0.88	2.54
RecycleBin2	4	0.34	0.13	0.5
RecycleBin3	4	0.89	1.13	3.01
RecycleBin4	4	0.45	1.13	2.06

the proposed representation lightens the computational load in exchange for a small error that does not affect task execution.

Object Segmentation A quantitative evaluation for the object segmentation process has been carried out by considering the same objects used for the Semantic Grid Map evaluation. In particular, we have evaluated the accuracy of our approach at segmenting multiple instances of three different classes of objects in our knowledge base (i.e., fire extinguishers, cabinets, and recycle bins) as shown in Figure 3.17.

Table 3.3 reports the results of the image segmentation process in terms of Detection Rate (DR) and False Alarm Rate (FAR). DR and FAR are computed as follows:

$$DR = \frac{TP}{TP + FN} \qquad FAR = \frac{FP}{TP + FP}$$

where TP are the true positives, i.e., correctly segmented pixels, FN are the false negatives, i.e., the number of object points detected as background, and FP are the false positives, i.e., the number of background points detected as object points. Lower values for DR are mainly caused by holes in the depth data, especially along the borders of the objects. Higher values for FAR are mainly caused by a slight misalignment between the RGB image and the depth map provided by the sensor. The highest FAR value is obtained in the case of *RecycleBin3* since part of a cabinet alongside the tagged recycle bin is incorrectly segmented as part of it.

Since the final goal of our framework is to acquire knowledge for generating an accurate semantic map, we evaluate also the precision of our segmentation method at extracting the

Figure 3.17. Three classes of objects have been selected for the quantitative evaluation of the object segmentation module: fire extinguisher, cabinet, and recycle bin. In the first column the images of the objects are reported, while the manually obtained ground truth images for the silhouettes of the objects are shown in the second column. The third column contains the results of the segmentation process.

width (W size) of the tagged objects. The results are reported in Table 4. The error e_W is calculated as follows:

$$e_W = \frac{|detected_W - GT_W|}{GT_W}$$

where $detected_W$ is the width detected by our segmentation algorithm and GT_W is the ground truth width. The analysis of the results suggests that the proposed approach can recover the W size of the tagged objects with an acceptable error e_W. The highest e_W value is caused by the erroneously segmented *RecycleBin3*. It is worth noting that in such a case, the system memorizes the tagged object. However, since the W value for *RecycleBin3* is not coherent with the object properties stored in the conceptual KB, a clarification dialog has been implemented to flag this error.

Speech Recognition The speech component has been designed mainly as a support for the *Augmented Mapping* task experiment described in this chapter. For this part, we aimed at having a robust system, covering a controlled language with a low error rate in terms of transcription ability, instead of trying to deal with a wide range of linguistic phenomena. We evaluated the performance of the Speech component considering two aspects: the quality of the transcription of user utterances and the command interpretation process.

The first has been evaluated in terms of the Word Error Rate (WER) [126]. We measured a WER of 0.258 on the transcription of commands uttered during the experiments. Since

Table 3.3. Error for the Object Segmentation module in terms of Detection Rate (DR) and False Alarm Rate (FAR).

Object	DR	FAR
Cabinet1	0.865	0.055
Cabinet2	0.946	0.010
Cabinet3	0.622	0.000
Cabinet4	0.841	0.037
Cabinet5	0.911	0.022
FireExt.1	0.621	0.151
FireExt.2	0.677	0.151
FireExt.3	0.795	0.280
RecycleBin1	0.892	0.195
RecycleBin2	0.839	0.119
RecycleBin3	0.900	0.502
RecycleBin4	0.628	0.022

Table 3.4. Error in extracting the width (W size) of the tagged object.

Object	Ground Truth	Detected	Error
Cabinet1	100 cm	96.56 cm	0.034
Cabinet2		76.03 cm	0.239
Cabinet3		79.16 cm	0.208
Cabinet4		138.20 cm	0.382
Cabinet5		80.50 cm	0.195
FireExt.1	15 cm	11.29 cm	0.247
FireExt.2		11.72 cm	0.218
FireExt.3		15.71 cm	0.047
RecycleBin1	38 cm	44.30 cm	0.165
RecycleBin2		29.25 cm	0.230
RecycleBin3		79.30 cm	1.086
RecycleBin4		34.85 cm	0.082

we use the Microsoft Speech Recognition Engine, the WER is actually measuring the performance of an off-the-shelf component. Note that this factor biases the outcome of the final interpretation, since a wrong transcription affects the recognition of the frame and the related arguments.

The evaluation of the performance of the interpretation process is inspired by a typical measurement specific of the *Semantic Role Labeling* task [127]. *Semantic parsing*, that is the process of assigning a frame structure to a sentence representing its meaning in terms of *semantic frames*, is subdivided into three phases. First, the *Frame Prediction* task, that is the ability of the system to recognize a frame associated with the command expressed in an utterance. Second, *Boundary Detection* is the task of recognizing the span of the arguments of a frame in a sentence. Finally, *Argument Classification* is the task that aims to assign a label to each recognized argument span from the Boundary Detection step. In our case, the interpretation of a recognized utterance in terms of frames and frame elements is produced simultaneously during the recognition process. The implemented grammars present a verb-arguments structure, where each verb is linked to the corresponding frame, as well as the arguments to the relative frame elements. Each recognized chunk of a sentence is then tagged as a frame element, preventing the possibility of having not labeled chunks. According to the nature of this *semantic parsing* procedure, we decided to define two measures inspired by the methodology deriving from the Semantic Role Labeling, the *Action Recognition* and the *Full Command Recognition*. The first measures the ability of the system

of assigning the right frame (thus understanding the right action) to a spoken command, exactly as it is done for the Frame Prediction task in the Semantic Role Labeling. The second refers to the capability of recognizing the arguments of a given action, and corresponds to the measure of the Argument Classification task. Table 3.5 reports the performance of these two measures in terms of Precision (P), Recall (R) and F1-Measure (F1), obtained from a dataset of 250 interactions.

Table 3.5. Performance of the Speech Recognition component.

Metric	P	R	F1
Action Recognition	89.47	80.63	84.82
Full Command Recognition	75.43	67.98	71.51

Spatial Reasoning To demonstrate the improvements that qualitative spatial reasoning can determine in grounding commands, as well as the effectiveness of our approach on a real robot, two different kinds of experiments have been carried out.

The purpose of the first experiment was to evaluate the impact of a qualitative spatial reasoner on an agent whose amount of knowledge continuously grows, as well as the influence of the already available knowledge on such a reasoning. Such an evaluation has been carried out considering the number of unambiguous and ambiguous commands (i.e., commands referring to more than one object with a specific spatial property) grounded by the agent. Indeed, when full knowledge about the environment is available, grounding ambiguous commands would mostly lead to the execution of an action that does not match the user expectation, while all the unambiguous commands are likely to be correctly grounded. We therefore analyzed first the impact of the presence or absence of the qualitative spatial reasoner (QSR) and then the impact of the amount of knowledge available to the agent. In detail, we first asked 26 students to provide a set of three commands containing spatial relations between objects, by looking at pictures of the test environment. Then, from the 78 acquired commands, we extracted two types of tasks: 28 ambiguous and 50 unambiguous. By gradually adding knowledge about the objects inside the knowledge base of the agent, we therefore measured how many commands were grounded. We repeated the experiment for both categories of commands, with or without the qualitative spatial reasoner. Since the curves depend on the order of the objects inserted in the knowledge base, the experiment has been performed five times in order to obtain its average trend (Figure 3.18). In case the QSR was not present (red curve), only the objects in the environment, whose category has a unique member, were correctly identified. For example, since we had two cabinets in the test environment, there was no way of distinguish them without exploiting spatial relations. By comparing the two curves in the image, it can be noticed that the presence of the QSR does not greatly affect their trend when a little amount of knowledge is available,

due to the absence of exploitable spatial relations between objects. On the other hand this is not true when substantial environmental information is available. Note that when complete knowledge about the relevant elements of the environment is known by the robot, the number of grounded commands, as expected, is equal to the number of unambiguous phrases (50 commands) present in the adopted set of commands.

The second experiment performed is aimed at understanding the limitations of the proposed approach. To this end, we measured the agreement between the user's expectations and the grounding performed by the robot. In particular, we first produced a Semantic Grid Map by driving the robot on a tour of the environment and tagging 23 objects within an office environment, as well as the doors and the functional areas in it. Then, we asked 10 different non-expert users to assign 10 distinct tasks to the robot, additionally asking them to evaluate whether the robot correctly grounded their commands, thus meeting their expectations. The commands have been directly acquired through a Graphic User Interface, in order to avoid possible errors due to misunderstandings from the speech recognition system. In detail, the users had the possibility to choose the action to be executed by specifying the located object, the reference object and one of the 10 spatial relations implemented in our reasoner. Table 3.6 shows that approximately 80% of the given commands have been correctly grounded. The remaining 20% of the wrongly grounded commands where due to two different phenomena: *(i)* the command given was ambiguous, requiring other properties, in addition to direction and distance, to identify the object; *(ii)* the users did not behave coherently during the interaction with the robot, by varying their concept of vicinity or by adopting different reference frames.

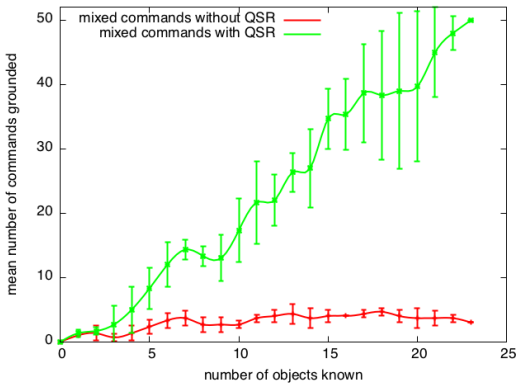


Figure 3.18. Mean number of grounded commands with respect to the number of objects known in the environment, added in a random order.

Table 3.6. Number of correctly and wrongly grounded commands with respect to the expectations of the users.

User	Correct	Wrong
1st	7	3
2nd	8	2
3rd	10	0
4th	6	4
5th	8	2
6th	8	2
7th	10	0
8th	7	3
9th	9	1
10th	8	2
Total	81	19

Table 3.7. Result obtained from the test performed on the whole system. The position of the tagged objects is compared with the one obtained from a manually generated ground truth by calculating the distance between the two points.

Distance Thresholds	Average Percentage	Experts	Non-Experts
≤ 0.1m	18%	20%	16%
≤ 0.2m	42%	37%	47%
≤ 0.3m	48%	46%	50%
≤ 0.4m	76%	72%	80%
≤ 0.5m	88%	94%	82%

3.6.2 Whole-system Evaluation

In this last set of experiments, our goal was to evaluate the whole system in a real environment during a typical task executed by the robot. For this reason, we deployed our robot in an office environment and asked both expert and non-expert users to drive the robot around using the vocal interface and to tag various objects present in the environment. To test the robustness of our system in a noisy environment, we carried out a data collection during a public opening of our department asking 10 visitors, in addition to all of the authors of this work (for a total of 16 users), to take part in the following experiment. The robot started with no knowledge about the objects enclosed in the environment and each user, after being explained for a minute the commands understood by the robot, had to drive, using the vocal interface, the mobile platform in front of a desired object and teach the robot its position and name. Having memorized different objects, the user had to ask the robot to move in front of them in order to demonstrate that the learning process had been carried out successfully. In this experiment all the users have been able to successfully memorize an object. After collecting the data, we calculated the distance between the position of the centroid of the learned objects with the one belonging to a ground truth manually created. The result of such a comparison is shown in Table 3.7. From the table it can be seen that almost 90% of the objects were placed with an error less than 50 cm. The remaining objects where placed instead at a distance between 50 cm and 1.5 m, due to errors resulting from the object segmentation component, and the Semantic Grid Map Generator. It can also be noticed that the precision does not vary between the expert and non-expert users, thus confirming that this system does not require a specific training to be used. Overall, the evaluation of the performance shows that the system can effectively acquire knowledge about the environment, allowing for the representation in the semantic map of a wide variety of elements. Finally, the results of the final experiment with the users show that the approximations that have been introduced in the representation do not affect the execution of the task, thus providing some evidence of a good balance between abstraction and accuracy reached in our representation.

3.7 Chapter Summary and Discussion

In this chapter we have presented a new approach for acquiring and representing the knowledge about arbitrary indoor environments. The knowledge about the environment that is gathered with the help of the user is turned by the system into a layered representation (i.e., a semantic map), which allows for high-level interaction with the user. In the development of our representation, we focused on the acquisition of knowledge about the specific environment, so that the behaviors of the robot over time are supported by grounded facts, rather than by general world knowledge. Additionally, we have shown how human-robot collaboration can play a key role, being the robot instructed by a user through a simple interaction. The resulting semantic map provides a compact and expressive representation, which is used to handle dialogs about the objects and locations in the environment and to ground complex user commands referring to spatial locations. Such maps can be used, for example, to drive the actions of a remote robot through speech, operating in a home or office environment.

The implemented system has been tested with four different robots, in different environments and with many users, showing a good performance over extended periods of time. In particular, the system has been experimentally evaluated as a whole, as well as in all its main components. The results of the experiments show that, despite some approximations in the construction of the representation, the knowledge acquisition process is robust and easy to be performed by non-expert users and on different robotic platforms.

While the implemented prototype shows the capabilities that can be achieved through the integration of AI techniques (NLP, KR, spatial reasoning, perception, etc.), the implementation can be improved in multiple aspects and several research topics may be addressed. First of all, a better integration with state-of-the-art techniques for object detection and classification may enable a more proactive role of the system in building the map, by matching both acquired models of objects and general model of object categories. A better integration of the domain knowledge with external resources, possibly including the web, may also be exploited to support the learning capabilities of the system in grounding the objects of the environment to new concepts and to their linguistic counterparts. Moreover, we plan to improve the whole system to address more cognitive issues related to the construction of the semantic map. Among them, the characterization of changes in the environments by analyzing the evolution of the knowledge base over time and the ability to properly handle multiple instances of objects in the same category. Finally, we are considering the extension of the Semantic Grid Map to 3D, which would bring about a new set of spatial relations among the objects.

Chapter 4

Acquiring Procedural Knowledge

In the previous chapter we have seen how a robot can incrementally acquire environmental knowledge through the interaction with the user, in order to adapt to the environment it is deployed in. Once a robot is able to acquire such knowledge, it should also be able to use it to carry out tasks given by the user. However, such tasks often vary from user to user and cannot be foreseen at design time. To this end, in this chapter we address the problem of learning from the users new complex tasks. With the term *complex* we refer to tasks that are compositions of predefined robotic *action primitives* (i.e., atomic behaviors that are not further decomposable in terms of simple actions). Once learnt, during execution a complex task will instantiate a specific plan and, for this reason, in the rest of the section, we will refer to complex learned actions either as *tasks* or *plans*, while their composing actions will be called *primitives* or *primitive actions*.

In the rest of this chapter we first discuss an approach for allowing non-expert users to teach parametric tasks to a robot. With the term *parametric* we refer to tasks that contain references to abstract concepts that are instantiated at execution time. An example of such type of task could be represented by the plan of "bringing an object to a location". In this case, the concepts "object" and "location" are understood by the robot in an abstract way. Such concepts are instantiated during execution with one of the instances of the semantic map acquired with the approach previously shown. Next, we discuss an alternative method for learning parametric tasks. By analyzing a library of previously taught tasks, we show how it is possible to extract common parametric tasks. These parametric tasks are then used for supporting the user in the teaching process. Finally, we consider the problem of teaching multiple coordinating robots. By leveraging the concept of sparse-coordination, we show how we can enable a user to teach to multiple robots new tasks that require them to coordinate.

4.1 Parametric Task Teaching

This work has been published in:

◇ *Teaching Robots Parametrized Executable Plans Through Spoken Interaction*. G. Gemignani, E. Bastinaelli, and D. Nardi. 14th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015.

In this section, we attempt to tackle some challenges associated to task teaching, by presenting a novel approach for learning, through the interaction with the user, task descriptions that are defined as a combination of primitive actions. To achieve this goal, we decouple the problem of learning the semantic meaning of the commands from the problem of executing them. Specifically, we first use a grammar-based approach to acquire the description of the actions, which are represented through a novel language representation designed to capture the multiple expressions of the language used during the interactions. This language representation is then converted into plans readily executable by the robot. The action learnt by the robot in this way can involve sequences of actions, conditional branches and iterations that can be characterized by multiple open parameters specified at run-time. The two main contributions of this work are the following:

- A novel approach for learning the syntactic structure of the taught actions, represented as parametric tasks that can be instantiated at run-time;
- A new language representation, called Task Description Language, mapped into the Petri Net Plans (PNP) formalism, which has been described in the previous chapter. This mapping allows us to give a clear execution semantics and to express complex execution paradigms, such as parallel actions.

In addition, we show how to revise the tasks learned, allowing the user to refer to their composing actions in a high-level fashion.

In this section, we first describe in detail the proposed approach and how it has been deployed on a service robot. Next, we present a set of tests designed to validate our method and a brief discussion of the results obtained.

4.1.1 Approach

Our approach allows to dynamically teach a robot new complex task descriptions through user interactions. Unlike previous works, in our approach the task learned by a robot take as input arguments that are not statically associated to a specific instance, e.g., *the red book*, but they are filled with possible semantic categories, e.g., *object*. For example, a possible implementation of a *bringing* task could be represented by the general concept of *bringing* an *object* to a *location*. In each task, the arguments can assume different values every time a command is given to the robot, e.g. "*bring the book to the office*".

Our learning process is based on a spoken dialog with a user that incrementally explains to the robot how the task is composed in terms of primitives. To this end, the Task Description Language (TDL) has been designed in order to give a structure to the task descriptions, enabling for the use of different patterns of execution of the involved primitives, e.g. conditional branches or while loops. The interaction with the user is mainly based on spoken inputs, however the proposed approach also allows for the use of examples to drive the learning process (e.g., it is possible to tell the robot *"go to the location for example the office"* to check if the robot correctly executes the command). Moreover, we also investigate the possibility of using dialogs to update a previously learned task by removing, inserting and replacing some of its parts with new TDL constructs. Generally, our interactive and action learning approach takes place in three consecutive phases: Processing verbal instructions; Generating executable plans; Updating plans. Figure 4.1 reports the interaction scheme of the first two processes that we designed.

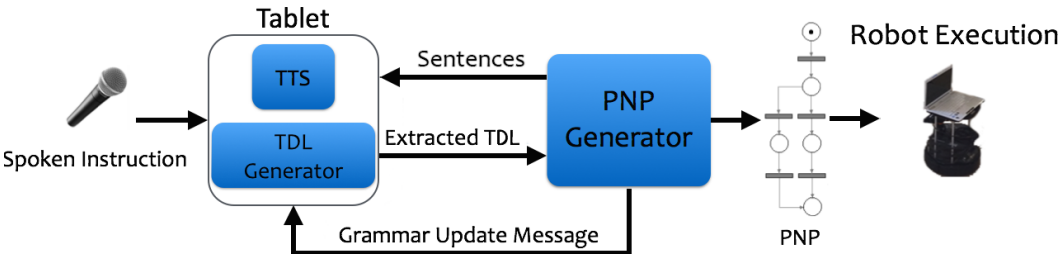


Figure 4.1. Interaction scheme designed for processing verbal instruction and for generating executable plans.

Processing verbal instructions

The first step in learning new actions through user interaction consists of acquiring a natural language description of the command and its interpretation, associated with a structured representation. Specifically, taking inspiration from the Robotic Control Language proposed in [87], we designed a Task Description Language (TDL), to create a first representation of the tasks to be learnt.

TDL is a high-level representation for complex procedure specifications, described using natural language. TDL acts as a bridge between the instructions as expressed by humans and the final plan of the robot, and it is specified by the grammar reported in Table 4.1. In TDL, tasks are described as a composition of procedural components that can be primitive actions or structural elements describing different execution patterns, that embed other TDL constructs in a recursive way. Starting from the capabilities of our robotic platforms, we first define a set of primitive actions so that every complex TDL structure is expressed as a function of them. Below, we list the primitives we defined to test our approach:

- `goTo`: the action of going from a point to another point in the space. It takes one

Table 4.1. The constructs defined in Task Description Language.

	TDL Form
Condition	*<Condition>:[Parameter List]*
Primitive Action	*<Action>:[Parameter List]* *<Variable>=<Action>:[Parameter List]*
Sequence of Actions	*do-sequentially* $<TDL_1>$ ... $<TDL_n>$
Conditional	*If <Condition> then* $<TDL_1>$ *else* $<TDL_2>$
Counting Loop	*do-n-times <TDL> <Integer>*
Do-until Loop	*do <TDL> until <Condition>*

parameter: the destination.

- `follow`: the action of following someone or something in front of the robot, in an open-loop fashion.
- `takePicture`: the action of taking a snapshot of the scene, acquired through some visual sensing device. It outputs the name of the picture taken, which can be stored in a variable (`@picture`) to be recalled in the dialogue by the user.
- `say`: tells the robot to call the Text-to-Speech service to synthesize a string into voice. Its parameter is the sentence to be said.
- `sendEmail`: the action of sending an email to an address with a content. Its two arguments are: the content of the mail and the address where to send it.
- `pickUp`: the action of picking up an object. It takes the object to be picked up as a parameter.
- `drop`: the action of releasing an object.

Additionally, we defined two conditions to be checked in the `if-then-else` and `do-until` constructs:

- `in`: to check whether the robot is located in a particular location or not.
- `isPerceived`: to verify whether an object is perceived by the robot or not. If the *anyone* parameter is passed as an argument, a person detector functionality is activated.

The list of parameters (see Table 4.1) passed to the primitives can either hold variables, denoted with an @ character followed by a string, or instances, represented by strings. Such lists are used to instantiate the arguments of the primitive actions, and are bounded to the parameters of the task being described. This feature is fundamental for teaching the robot general concepts of actions, where arguments are only referenced as possible semantic categories and not as specific instances of them.

For example, consider the case of teaching a robot the general concept of the *bringing* action. We can assume that, for a particular instance, this action will require two arguments: an *object* to be brought and a *location* where the *object* must be brought. This general conceptual structure of the action is instantiated at the beginning of the learning interaction, after the user tells the robot a phrase like "*I'll teach you how to bring an object to a location*". Consequently, two variables are created (i.e. `@object` and `@location`) for the current task. During the explanation, the user can refer to these variables as arguments of prompted primitive actions. Let us assume that the task corresponding to the *bringing* action is composed by the sequence of four primitives: a `goTo` action needed to reach the object to be brought, followed by a `pickUp` action, then another `goTo` action to reach the final location and, finally, a `drop` action to release the object. When referring to a primitive that requires one or more arguments, it is possible to bind them with the task variables, e.g. `goTo:[@location]`. The final TDL structure for this particular instance of the *bringing* action therefore will be:

```
( do − sequentially
        ( goTo : [@object] )
        ( pickUp : [@object] )
        ( goTo : [@location] )
        ( drop : [@object] ) ).
```

In order to acquire such a description of the action, we propose an interaction scheme that is based on a spoken dialog with a user that prompts step by step the elements composing the final TDL. The process is structured in three steps:

- Automatic Speech Recognition to translate the user vocal input into text.
- Interpretation, where the transcribed input is translated into a TDL construct.
- Decision about how to proceed in the dialog flow (e.g. asking for confirmation about the learned action or possible clarification about the input received).

The ASR phase is realized using a grammar-based engine that allows us to implement specific grammars for describing the TDL. The grammars drive the recognition process: by attaching a proper semantic output to each grammar rule, we obtain a representation of the linguistic semantics of a recognized utterance, which is later translated to a specific TDL. This representation is based on the *frame* concept inspired by the notion defined in the *Frame Semantics* linguistic theory [33]. The general meaning expressed by each frame can be enriched by semantic arguments that are part of the sentence and provide additional meaning to the action. As an example, the command *"go to the office"* will be mapped to the MOTION frame, while the sub-phrase *"to the office"* will fill the specific

frame element GOAL representing the destination of the MOTION. The representation will then be translated in the following TDL structure:

(`goTo` : [`office`]).

The adopted ASR provides also for the possibility of using special grammar rules that trigger a free-form dictation ASR engine. We use this feature to recognize the names of new tasks that are not predefined in the grammar, as well as messages to be uttered when performing the `say` primitive.

The process of learning new tasks does not only consist of converting spoken instructions to TDL and consequently instantiating a new plan. In fact, the acquisition of knowledge about new tasks also involves the learning of new linguistic forms in order to verbally recall them. To this end, after a new plan has been correctly built, the ASR grammars are augmented with new syntactic-semantic structures corresponding to the learned task, e.g. *bring the @object to the @location* for the task “bring the object to the location".

Generating Executable Plans

Once the spoken description of the task given by the user has been converted in its corresponding TDL structure, it is processed to obtain an executable plan described with the Petri Net Plans (PNP) formalism [86]. PNP, is a plan-representation framework based on Petri Nets, which includes a rich set of features suitable for expressing executable actions: non-instantaneous actions, sensing and conditional actions, action failures, concurrent actions, interrupts, and action synchronization in a multi-agent context are among the aspects that can be described by such a formalism. This versatility allows us to represent and precisely characterize within PNP not only the multiple details of the plan uttered by the user, but also every action and dialog between the user and the robot. Moreover, with this additional representation we are able to decouple the problem of understanding user utterances from the issue of executing a task on the robot.

Starting from the TDL structure representing the plan extracted from the utterances of the user, in order to obtain an executable plan, a tree structure is created by recursively decomposing it in its elementary parts. Such tree is composed of nodes labeled with the TDL forms contained in the input TDL and by edges representing the constituency relation (see Figure 4.2). Once the tree has been generated, starting from its leaves, each TDL construct is replaced by the corresponding PNP structure. Figure 4.3 shows the PNP structure corresponding to each TDL constructs shown in Figure 4.1. By querying a KB that holds the information about the primitive actions (i.e., their name and their input and output semantic categories), each component of the PNP structures is then renamed to form a correct parametric plan. When dealing with a parametric primitive action, during execution the system is able to query the KB in order to bind the variable used in the obtained PNP with an instance of the semantic map. By converting the TDL structure into a specific PNP,

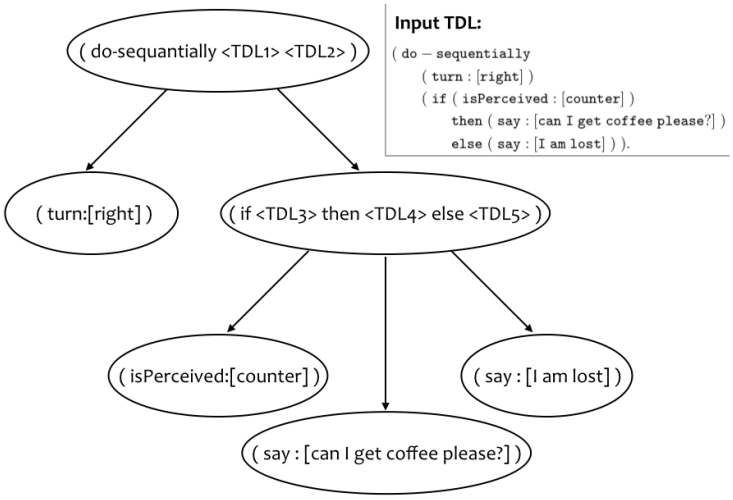


Figure 4.2. Tree structure created from an example input TDL.

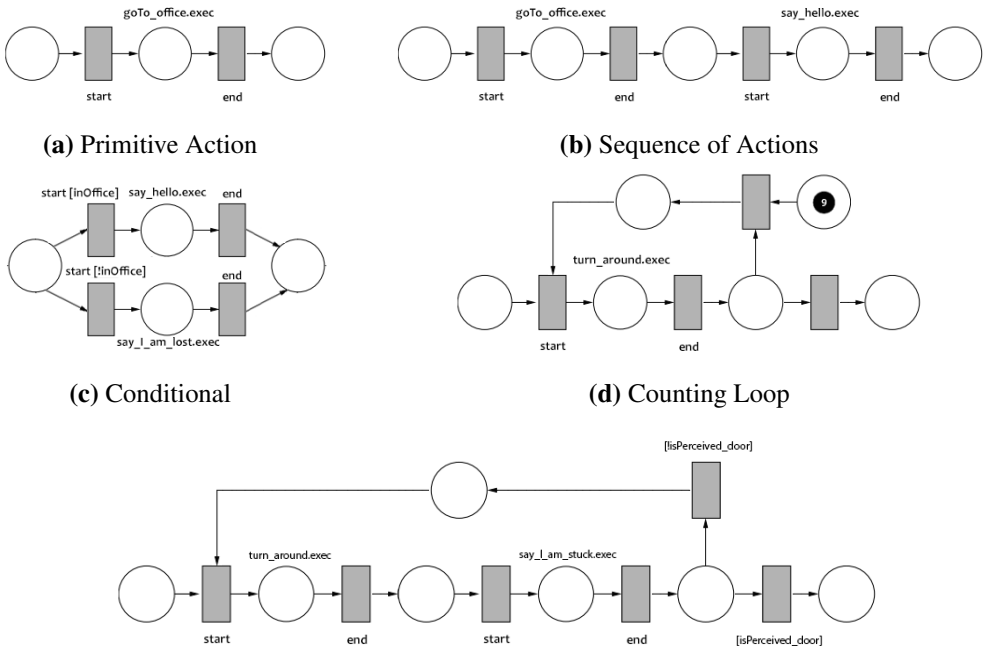
an exact mapping between natural language and executable PNPs is obtained, thus providing a suitable execution semantics.

Once the PNP representation of the action has been created, an update message is sent to the speech recognition module and to the KB to flag that the new plan has been learnt. Specifically, this message is used to store in the KB the number of variables used by the learnt plan, their name and the associated TDL structure needed during the update phase, as described in the following section.

Updating Plans

A feature of our approach consists of allowing the user to correct parts of the taught plan as desired. The work by [92] is the only example of plan revision in a task teaching framework: the authors propose a task correction mode, during which the robot describes the action sequence and asks, at each step, whether the user wants to replace a primitive action with another one. We propose a way for referring to each primitive action composing the newly learnt plan in a high-level fashion, allowing the user to add, replace or delete a particular action in a task.

To update a previously taught task, the user can verbally select it and subsequently refer to its primitive actions. By naming the action that needs to be modified, the desired operation, and the optional parameters needed, the plan is modified accordingly. For example, assuming that the user needs to replace a `say` action with a `goTo` action, this can be achieved by telling the robot: *"I want you to perform the go to location action instead of the say action"*. When multiple actions of the same kind are instead present in a plan to be modified, the user can distinguish them by referring to their parameters (e.g., *"the go to location"*) and/or by specifying the ordinal number with which they appear in the TDL structure (e.g., *"the*



(a) Primitive Action

(b) Sequence of Actions

(c) Conditional

(d) Counting Loop

(e) Do-until Loop

Figure 4.3. Examples of PNP structures for each TDL construct.

second go to"). Once the updating command has been acquired by the robot, the old plan TDL structure is retrieved from the KB. If the command univocally identifies an operation in the plan (i.e., there are not any ambiguities in the given command) the old TDL structure is modified accordingly, otherwise a clarification dialogue is started by the robot. Finally, the new TDL is translated into its corresponding PNP to enable its execution on the robot.

4.1.2 System Deployment

Our goal in this section is to demonstrate the effectiveness of the approach, showing that our teaching methodology can be adopted to learn high-level parametric plans, additionally allowing to represent multiple execution paradigms expressed using natural language. To this end, we deployed our system on a robotic platform to show the advantage of using an intermediate representation, namely TDL, together with the PNP formalism to model the learning, revision and execution processes. To ground the semantic categories expressed in TDL, a semantic map built with the aid of the user has also been given to the robot [16]. The proposed approach has been implemented on a Videre Design platform equipped with a Hokuyo laser range finder and a Kinect. For localization and navigation, standard software packages of the ROS framework have been adopted. Finally, for the ASR engine, we used the Microsoft Speech Platform running on an external tablet with a push-to-talk microphone, used to implement the connected speech interface.

Figure 4.4. The Videre Design platform used in the experiments interacting with a user.

Figure 4.4 shows an image of the robot interacting with a user.

Interaction Example

In order to better understand the process of learning a complex plan, we report a video showing an example interaction that guides the teaching process of the *bringing* an *object* to a *location* plan[1].

H: *I'll teach you how to perform the "bring an object to a location" plan.*
R: *Ok, tell me how to perform this action.*
H: *First, go near the object, for example "the DVD".*
R: [moves near the DVD]
R: *did I perform the right action?.*
H: *Yes.*
R: *Should I do anything else afterwards?.*
H: *Yes, pick up the object.*
R: *Ok, can you put the DVD on the tray?*
R: [waits for the user]
U: [confirms through a dialog box]

... [The action is added to the task and the interaction continues]

H: *bring the book to the office*

[1] `https://vid.me/HXnG`

R: [performs the new instance of the action]

The interaction starts with a sentence that instantiates the structure of the plan, defining its general name (e.g. *bring*), and its arguments, (e.g. *object* and *location*). Afterwords, the explanation phase begins: the use of predefined keywords enables for recalling constructs of TDL associated with them. For example, the word "*first*" at the beginning of the second sentence uttered by the human specifies to the system that the prompted primitive is part of a sequence. The resulting interpretation of this statement will be a `do-sequentially` construct, where the first action to be performed is a `goTo:[@object]`:

```
( do − sequentially
      ( goTo : [@object] ) ).
```

In this example, an instance of the semantic category object, e.g. the *DVD*, is provided so that the robot can show the user that the action he prompted has been correctly understood. The action is then executed using the example as input argument. Note that instead of using predefined keywords to point out the end of a TDL construct, e.g. *endif*, we rely on a question-answer modality to determine its scope boundaries. In this example, the question "*Should I do anything else afterwards?*" is used to understand if the next instruction belongs or not to the current TDL construct, i.e. the `do-sequentially`. The interaction continues with the second action. This time the user refers only to the semantic category, but because of the binding between variables and instances provided in the previous example, the robot asks directly the user to put the *DVD* on the tray, as the `@object` has been locally instantiated before. Doing so, the `pickUp` primitive is simulated and the learning process continues by inserting it in the current TDL construct, e.g.

```
( do − sequentially
      ( goTo : [@object] )
      ( pickUp : [@object] ) ).
```

It is important to underline that the use of examples while explaining a task is not mandatory. In fact, it is possible to refer only to the semantic categories involved in the task, without using direct instances and skipping the demonstration about the action to perform. Using or not using examples during the teaching phase does not affect the final plan instantiation.

At the end of the interaction, the final TDL structure is processed yielding the PNP plan corresponding to it. Accordingly, the language of the robot is augmented by inserting in the ASR grammar the semantic-syntactic structure corresponding to the command "*bring an @object to a @location*". Now it is possible to call the plan for any location and any object defined in the ASR grammar.

Table 4.2. Results obtained for the evaluation of the proposed task teaching framework.

Task	Min	# Err	# Mis	# Cor	AT No Err	AT w Err
enter @location	4	0.2	0.4	0.4	29	38
take an @object	5	0.6	0	0	41	52
bring an @object to a @location	7	0.2	0.6	0.6	57	74
give @person an @object	6	1	0.4	1	49	79
check an @object for a @person	5	0.6	0	0	55	69

4.1.3 Evaluation

In order to validate the system, three different kinds of analysis have been carried out. First, we taught our robot five different tasks. During this teaching phase we measured the accuracy of the system as well as the time needed for teaching each task. We then counted the number of non parametric tasks that should have been taught to act on a particular set of instances, comparing it with the number of parametric tasks needed to act on the same set of targets. For example, following an unparameterized approach, in order to learn the *bring an object to a location* task valid for every object and location in the environment, one should have taught the robot a single task for each combination of instances of *object* and *location*. Next, we replicated the tests performed on different mobile bases by the authors of related works. This test aimed at verifying the expressive power of our approach with respect to the others found in literature. Finally, since we adopted a formalism able to represent a wide variety of execution paradigms, we attempted to show how other kinds of tasks could be learned by the robot. In particular, we successfully managed to teach the robot a task involving parallel actions. This is a first example of the several possible extensions that are achievable by creating new TDLs and providing them with a suitable execution semantics using PNP.

Expressivity and Effectiveness Evaluation

The aim of our work is to create an interactive task teaching framework, based on a natural language interaction with the user; however, we do not evaluate our approach through a user study, since usability is not the purpose of this work. Rather, we make an assessment of the expressivity and effectiveness of our approach. In particular, to understand the impact of learning parametric actions and to measure the efficiency of our approach, we first taught our robot five basic tasks, each involving one or two parameters. The five tasks we selected for the evaluation are just illustrative examples of what our system can learn, and they are reported in the following list:

- *enter @location*: this action allows to add another possible linguistic reference for the `goTo:[@location]`.

- *take an @object*: This task is represented by `goTo:[@object]` followed by `pickUp:[@object]`. As the `pickUp` primitive, this task requires the robot to be near the target object.
- *bring an @object to a @location*: (see above).
- *give @person an @object*: this task represents the action of bringing an object that the robot already carries to a person, assuming that the position of the person is known.
- *check an @object for a @person*: this task consists of going in front of an object, taking a snapshot of it and sending an email with the picture to the person.

In order to show the overall behavior of the system while learning this set of tasks, we asked 5 users not familiar with robots to test the teaching framework. Each user was first given a brief explanation of the task teaching framework and the semantic map and primitives available to the robot. Next, the user was asked to teach the robot each of the previously described actions in a random order. For each task we measured the following quantities:

- The minimum number of instructions required to teach a specific task (**Min**).
- The average number of instructions not recognized by the automatic speech recognition (**Err**). In this measure, only the instructions that the ASR could not process were considered.
- The average number of instructions misrecognized (**Mis**) by the natural language understanding.
- The average number of corrections needed to modify a wrongly learned task (**Cor**).
- The average time in seconds needed to teach a specific task when no errors or misrecognitions were encountered (**AT no Err**).
- The average time in seconds needed to teach a specific task when errors or misrecognitions were encountered (**AT w Err**).

Table 4.2 reports the numerical results obtained for the analysis. Notice that the time measured refers only to the interaction with the user, as we avoided examples requiring the action of the robot.

A first fact that can be noticed from the table is the cost, in terms of time, of a correction process. When one or more corrections have been required, the average time needed to correctly teach a task increases by $16.2s$ on average. Additionally, it can be noticed that the longer a task is, the higher is the probability of having a misrecognition. This expected result is in fact underlined by the highest number of misrecognitions obtained for the two longest tasks (e.g., *bring an @object to a @location* and *give @person an @object*). For

these particular tasks we also obtained the highest difference between **AV no Err** and **AV w Err**, respectively of $17s$ and $30s$. The effects of a misrecognition can be rather different: in the starting phase of a procedure it may not require a correction (e.g. for the *enter* task), while in other cases one or more corrections might be needed (e.g for the *bring* and *give* tasks).

Finally, in order to give an idea of the teaching steps saved with this parametric approach, we counted the number of non parametrized tasks that should have been taught to the robot in order to cover the same exact set of possible instantiated tasks. This particular analysis has been carried out considering the scenario presented in [16], built through the direct interaction with a user. Such an environment consists of 23 objects and 10 persons, located in 10 different rooms. In this specific environment, in order to obtain the equivalent of the five parametric tasks, we should have taught the robot 723 single tasks by combining all the instances of the involved semantic categories for each task. The gain in terms of number of interaction is straightforward, especially if we consider the possibility of dynamically extending the number of objects, or in general the knowledge about the environment.

Comparison with Related Works

A second validation of the learning ability of our system has been carried out by acquiring the same tasks that systems presented in other related works learned. The *dinner is ready* task in the article by Rybski *et al.* [90] is used to test the presented approach. Such a task is composed by a sequence of `goTo` and `say` primitives, interleaved with some conditional branches regarding the presence of specific persons in the environment. We taught our robot the exact same task, resulting in the following TDL:

```
( do − sequentially
      ( goTo : [dining_room] )
      ( if ( isPerceived : [jeremy] )
            then ( say[“Jeremy set the table”] )
            else ( say[“cannot find Jeremy”]) )
      ( goTo : [living_room] )
      ( if ( isPerceived : [kevin] )
            then ( say[“Kevin come to dinner”] )
            else ( say[“cannot find Kevin”] ) )
      ( goTo : [bedroom] )
      ( say[“turn off the television”] )
```

(4.1)

```
( goTo : [living_room] )
( say : ["task complete"] ) ).
```

The average training time for this particular task was 185 seconds. Even though the environment we used for reproducing the task was different, our system was able to learn the same sequence of actions learnt by the other robot. Such a result demonstrates the learning ability of the system, independently from the actual scenario.

In a similar way, we managed to interactively teach the *get coffee* task presented in the work by Meriçli *et al.* [92] to our robot. In this case, we had to slightly adapt the task learnt as the set of primitives provided by the CoBot platform was different from ours. Indeed, due to the lack of specific primitives, we exploited the *do-until* loop for tasks that required quantified parameters, e.g. *move forward 5.4 meters*, terminating the loop with a specific perception condition. Instead of referring to generic numbered landmarks, instead, we used objects on the map to trigger some perception checks. The average training time for this task was 92 seconds. The following TDL reports the result obtained:

```
( do − sequentially
    ( do ( getCloser : [] )
    until ( isPerceived : [door] ) )
    ( turn : [right] )
    ( if ( isPerceived : [counter] )
        then ( say : [can I get coffee please?] )
        else ( say : [I am lost] ) ).
```

Teaching a More Complex Task

In order to asses the expressive power of our approach, we performed a third validation of the system. Specifically, we used TDL to capture an additional execution paradigm already available in the PNP formalism. To this end, we instructed our robot to execute a complex task involving primitive actions to be performed in parallel. This pattern has been realized using an ad-hoc TDL construct. We therefore designed a non parametric task to *patrol* a wing of our department. Such a task consists of visiting three different rooms, while continuously checking for the presence of persons. If a human is detected, the robot sends an email sent to the security staff. The loop is ended when the robot reaches the final

destination. The TDL structure representing the task is the following:

```
( do
    ( do − sequentially
        ( goTo : [office] )
        ( goTo : [storage_room] )
        ( goTo : [printer_room] ) )
 and
    ( do
        ( if ( isPerceived : [anyone] )
            then ( sendEmail : [security] )
            else ( continue ) )
      until ( in : [printer_room] ) ) ).
```

The resulting PNP is instead shown in Figure 4.5. According to this specific definition of the patrolling task, the two branches are performed in parallel, by exploiting one of the key features of the PNP formalism.

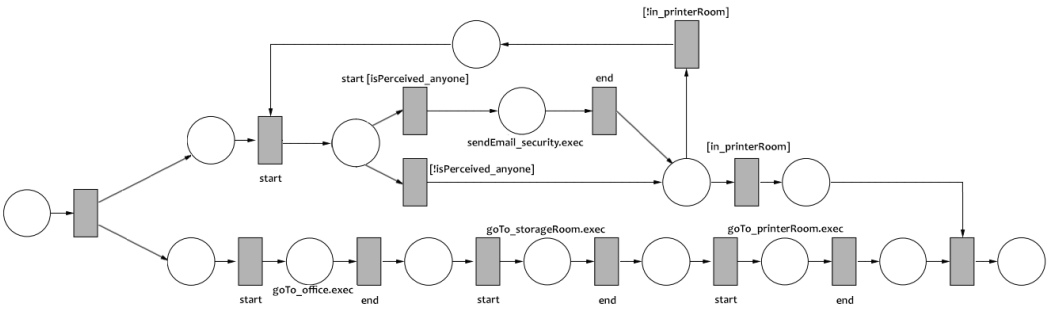


Figure 4.5. The resulting PNP for the patrolling task.

4.1.4 Section Summary and Discussion

In this section, we presented a novel approach for learning executable plans through the natural interaction with the user, described as a combination of primitive actions. Specifically, our system uses a grammar-based approach to first acquire the description of the action, later converted into an executable Petri Net Plan, passing through an intermediate representation expressed in a specifically developed language called Task Description Language. The proposed approach takes a significant step forward by making task descriptions parametric with respect to domain specific semantic categories. Moreover, by mapping the task representation into a plan representation language, we have been able to express

complex execution paradigms and to revise the learnt plans in a high-level fashion. The teaching framework has been deployed on a Videre design robot, showing the advantage of using an intermediate representation, namely TDL, together with the PNP formalism to model the learning, revision and execution processes. With a double task representation we have shown how it is possible to decouple the problem of capturing the wide variety of linguistic expressions uttered by the user and the problem of defining a clear operational semantics.

For future works, a number of additional features need to be addressed. We are first investigating the issues that arise during the possible failure of an action. We would like in fact to enable the user to teach the robot how to recover from a specific action failure. This can achieved by exploiting the PNP formalism that allows for the implementation of actions interrupts based on particular conditions. Moreover, we are investigating the possibility of specifying optional parameters during the teaching phase when using examples. Instead of just telling the robot if it performed the correct action or not, the user could indeed exploit this particular phase for specifying optional parameters specific to the taught plan (e.g., by saying *"Yes but stop a little bit closer"* for the `goTo` action). Finally, we are planning to deploy and monitor our robot in our department for an extended period of time, by letting it freely interact with the users present in the environment.

4.2 Task Generalization and Proposal

This work has been published in:

- ◇ *Graph-Based Task Libraries for Robots: Generalization and Autocompletion*. S. D. Klee, G. Gemignani, D. Nardi, and M. Veloso. 14th Conference of the Italian Association for Artificial Intelligence, AI*IA 2015.
- ◇ *On Task Recognition and Generalization in Long-Term Robot Teaching (Extended Abstract)*. G. Gemignani, S. Klee, M. Veloso, and D. Nardi. 14th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015.
- ◇ *Task Recognition and Generalization in Long-Term Robot Teaching*. G. Gemignani, S. Klee, M. Veloso, and D. Nardi. Autonomous Robots and Multirobot Systems, ARMS 2015.

In this section, we address the problem of a user teaching an *additional* task to a service robot that is deployed for an extended period of time. In such a scenario, the user is unlikely to know all of the tasks previously taught to the robot. Therefore, we want the robot to recognize when it is being taught a task similar to one that it already knows. In these cases, the robot will use this information to propose the next steps of a task to the user, reducing the necessary number of human-robot interactions.

For a long-term deployed robot, we assume that its task library will become very large. So, it is not feasible to compare a newly taught task to every past entry. Since many tasks have similar patterns of action and sensing primitives, instantiated with different parameters, we instead focus on learning a library of general parametric tasks. Compared with the related

work presented in the previous section, in this case we do not require the user to remember how to teach parametric tasks to the robot. Instead, the robot extracts parametric tasks from the already known and instantiated tasks. We show that by creating a parametric or general task library we can perform online generalization and task step proposals for multiple given libraries of tasks. Moreover, these parametric tasks can be referenced at teaching-time like programming functions, instead of creating a specific instance of the task from scratch. This greatly reduces the overall size of each task being taught if parts of it belong to a general class.

In this work, we contribute an approach for measuring task similarity, performing task generalization, and proposing future steps of a task during the teaching process. In this work, we represent tasks as graph-based structures called Instruction Graphs [92]. We adopt this alternative representation due to specific tools readily available while researching this topic. However, the concepts presented in the rest of this chapter can be applied to the TDL and PNP representation presented in the previous section. From a given Instruction Graph task library, we perform frequent labeled subtree mining [128] to extract general classes and use a structure-based similarity metric to propose the most likely task being taught. The main contributions of this work are:

- An algorithm to generalize similar tasks.
- A structure-based approach for recognizing task similarity in task teaching frameworks.
- A method for robots to propose the next steps of a task during teaching.

In the rest of the section, we first briefly summarize Instruction Graphs, in order to later describe our Task Generalization and Proposal approach, showing all of our contributions thoroughly. Next, we discuss the evaluation process undertaken to validate the approach.

4.2.1 Instruction Graphs

In the Instruction Graph representation, vertices contain robot-primitives, and edges represent possible transitions between vertices. Mathematically, an Instruction Graph is a graph $G = \langle V, E \rangle$ where each vertex v is a tuple:

$$v = \langle id, InstructionType, Action \rangle$$

where the *Action* is itself a tuple:

$$Action = \langle f, P \rangle$$

where *f* is a function, with parameters *P*. The function *f* represents the action and sensing primitives that the agent can perform, such as grasping or detecting a color. We introduce

parameter *types* related to their purpose on the robot. For instance, on a manipulator, the function *set_arm_angle* has parameters of type *Arm* and *Angle*, with valid values of $\{left, right\}$ and $[0, 2\pi]$ respectively. The primitives and parameter types are robot-specific.

Each Instruction Graph is executed starting from an initial vertex, until a termination condition is reached. During execution, the *InstructionType* of the vertex describes how the robot should interpret the output of the function f in order to transition to the next vertex. The Instruction Graph framework defines the following types:

- Do and DoUntil: Used for actuation primitives. The output of f is ignored because there is only one out-edge. From now on, we will refer to both of these types of nodes simply as Actions.
- Conditional: Used for sensing actions. The output of f is interpreted as a boolean value used to transition to one of two children.
- Loop: Used for looping structures. The output of f is interpreted as a boolean value, and actions inside of the loop are repeated while the condition is true.
- No-Op: Used to delimit the end of a loop or conditional branch.

Additionally, the *Reference InstructionType* are introduced for specifying hierarchical tasks.

- References: Used to execute one Instruction Graph inside of another to express hierarchical tasks. The output of f is interpreted as a reference to the other task.

Figure 4.6 shows an example node with id 4, *InstructionType* Action, function *set_arm_angle*, and parameters *left* and π. Figure 4.7 shows instead a possible Instruction Graph representing an action for a manipulator robot. For a more detailed overview of Instruction Graphs, we refer the reader to the original publication [92].

Figure 4.6. Example of Instruction Graph node with id 4, *InstructionType* Action, function *set_arm_angle*, and parameters *left* and π.

4.2.2 Approach

This work considers a robot with modular primitives that represent its action and sensing capabilities. We assume that the robot has a library of common tasks, where each task is composed by some of these primitives. Our goal is to identify common frequent subtasks and generalize over them with limited user assistance. In this section, we present an in-depth description of the generalization and task autocompletion algorithms.

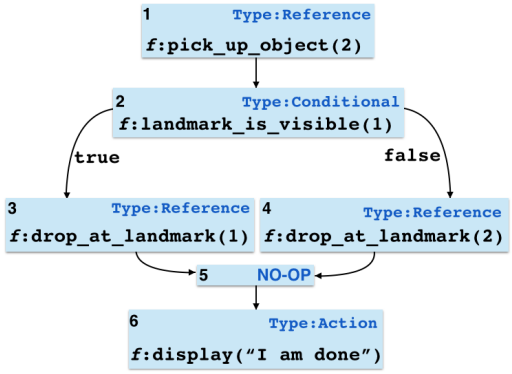


Figure 4.7. Example of Instruction Graph representing a plan for picking up a block and place it at a particular location depending on the visibility of a certain landmark.

Generalizing Tasks

We define a general task as a *Generalized Instruction Graph* (GIG). In a GIG, the parameters of some actions are ungrounded. In such cases, we know the type of these ungrounded parameters, but not their value. So, for each parameter we associate a distribution over all known valid groundings. For instance, in the case of a grounded parameter, the distribution always returns the grounded value. Formally, a GIG is also a graph $GIG = \langle V, E \rangle$ where each vertex v is a tuple:

$$v = \langle id, InstructionType, GeneralAction \rangle$$

$$GeneralAction = \langle f, P, \Phi \rangle$$

where $\phi_i \in \Phi$ is a distribution over groundings of the parameter $p_i \in P$.

These distributions are learned during task generalization and are used to propose initial parameters during task autocompletion. A GIG can be instantiated as an IG by grounding all of the uninstantiated parameters. This process consists of replacing any unspecified p_i with an actual value.

Our approach generates a library of GIGs from a library of IGs, as shown in Algorithm 1. The general problem of finding labeled subgraph isomorphisms is NP-Hard. However our problem can be reformulated into the problem of finding common labeled subtrees in a forest of trees. To this end, we create a tree representation of each IG. As the first step, we define a mapping from IGs to Trees (*T*):

$$toTree : IG \rightarrow T$$

and its corresponding inverse:

$$toIG : T \rightarrow IG$$

The function *toTree* computes a labeled spanning tree of an input Instruction Graph (line 3). Specifically, *toTree* creates a spanning tree rooted at the initial vertex of the input IG, by

Algorithm 1 Task Generalization

```
1:  procedure GENERALIZETASKS(IGs, σ, L)
2:    // IG library is converted to trees
3:      IGTrees ← {toTree(g) | g ∈ IGs}
4:    // Tree patterns are found by a tree mining algorithm
5:      tp ← ftm(IGTrees, σ)
6:    // Mapping from tree patterns to IGs is created
7:      igp ← {⟨p, toIG(T)⟩ | ⟨p, T⟩ ∈ tp}
8:    // Filters remove unwanted tree patterns
9:      igp ← filter_not_exec(igp)
10:     igp ← filter_by_length(igp, L)
11:   // Tree patterns of full tasks are reintroduced
12:     igp ← add_full_igs(IGs, igp)
13:   // Vertices and edges of the GIGs are constructed
14:     gigs ← create_ugigs(IGs, igp)
15:   // Parameters and distributions are computed
16:     gigs ← parametrize(IGs, igp, gigs)
17:     return gigs
18: end procedure
```

performing a depth-first search and by removing back edges in a deterministic manner. This ensures that instances of the same GIG map to the same spanning tree.

Each node in the tree is labeled with the *InstructionType* and function *f* of the corresponding node in the IG. In this label, we do not include the parameters because we eventually want to generalize over them.

Next, we use a labeled frequent tree mining algorithm to find frequently occurring tree patterns (line 5). A frequently occurring tree pattern is a subtree that appears more than a threshold σ, called the *support*. A tree-mining algorithm *ftm* takes as input a set of trees and the support. As output, it provides a mapping from each tree pattern to the subset of trees that contain it. Then, since each tree pattern is associated to a set of trees and each tree corresponds to a specific IG, we can create a mapping directly from tree patterns to IGs (line 7). We denote this mapping as *IGP*.

A tree mining algorithm will return many tree patterns. In particular, for any tree pattern, any subtree of it will be returned. This is because each subtree will have a support at least as large as its parent. Rather than keeping all these patterns, we focus on storing those that are the most applicable. There are many possible ways to filter the patterns. We propose several heuristic filters that select patterns based on their *executability*, *frequency*, and *usefulness*.

- **Executable** patterns are those that the robot can run.

- **Frequent** patterns are statistically likely to appear in the future.
- **Useful** patterns reduce many interactions when correctly proposed.

Each filter is formally defined as a function:

$$\mathit{filter} : IGP \rightarrow IGP$$

We first filter patterns that cannot be executed by the robot. In particular, we remove patterns with incomplete conditionals and loops (line 9).

Then, there is a tradeoff between highly frequent patterns and highly useful patterns. Patterns that occur with a large frequency are typically smaller, so they provide less utility during task autocompletion. Larger patterns provide a lot of utility, but they are usually very specific and occur rarely.

Less frequent patterns are already filtered out by the tree-mining algorithm when we provide it a minimum support σ. To optimize for larger patterns that save more steps during autocompletion, we also remove patterns that are shorter than a threshold length L (line 10). This ensures that we do not keep any pattern that is too small to justify a recommendation to the user.

Finally, since we are dealing with autocompletion for robots, one desirable feature is to be able to propose entire tasks. Even if a full task has a low support, or is below the threshold length, there is value in being able to propose it if a reparameterized copy is being provided to the robot. Consequently, we reintroduce the tree patterns corresponding to full IGs (line 12).

Even with these filters, we still keep some tree patterns that are complete subtrees of another pattern. In practice, many of these patterns provide useful task autocompletion suggestions. However, in memory-limited systems, we suggest also filtering them.

Finally, the algorithm processes the filtered set of tree patterns to create GIGs by creating vertices and edges from the tree pattern and then parameterizing the vertices. First, to create the GIG's vertices and edges, we copy the subgraph corresponding to the tree pattern from any of the IGs containing the pattern (line 14). This gives us a completely unparameterized GIG (*uGIG*), with no parameter distributions. Next, we determine which parameters are grounded in the GIG, and which are left ungrounded. A parameter is instantiated if it occurs with the same value, with a frequency above a given threshold, in all corresponding IGs. Otherwise, the parameter is left ungrounded with an empirical distribution.

This process is repeated for every subtree pattern not removed by our heuristic filters, creating a library of GIGs (line 16). When this algorithm is run incrementally, this library is unioned with the previous library. Such a mechanism represent an alternative to the approach presented in Section 4.1. Indeed, in the previous section, the robot was able to learn parametric tasks by directly interacting with the user. Instead, in the scenario here

presented, the robot is able to learn parametric tasks by analyzing all the instantiated tasks previously learned through the interaction with the user.

Figures 4.8a and 4.8c show example IGs for a task that picks up an object, and drops it at one of two locations. Figures 4.8b and 4.8d depict their corresponding spanning trees. Finally, Figure 4.9 shows the general task that is extracted. In this GIG, the parameters in nodes 3 and 4 are kept instantiated, since they were shared by the two original IGs. The others parameters instead are left ungrounded. These parameters have a type *id*, and a distribution over the landmark ids $\{1, 3\}$, which were extracted from the IGs in Figure 4.8.

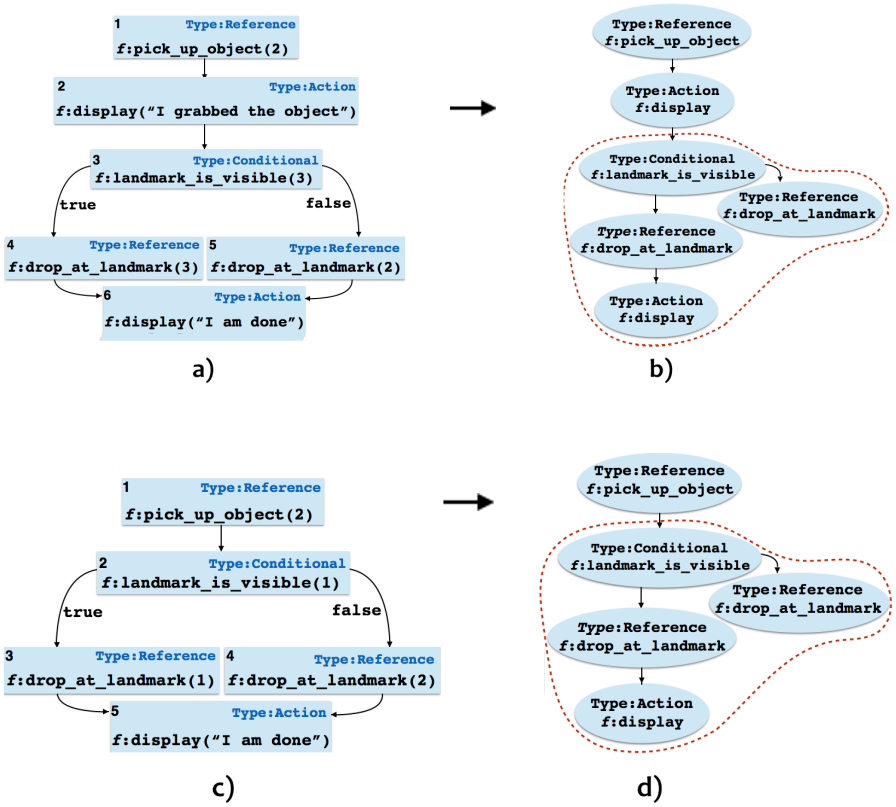


Figure 4.8. Example of two Instruction Graphs (a, c) converted into their corresponding spanning trees (b, d). The tree pattern shared between them is circled in red.

Task Autocompletion

We now consider an agent that is provided a task incrementally through a series of interactions. Each interaction consists of adding a vertex to the graph or modifying an existing vertex. At any step of this process, the agent knows a *partial task*. After each interaction, this partial task is compared against the library of GIGs to measure task similarity and perform autocompletion.

The algorithm performs this comparison by checking if the end of the partial task being

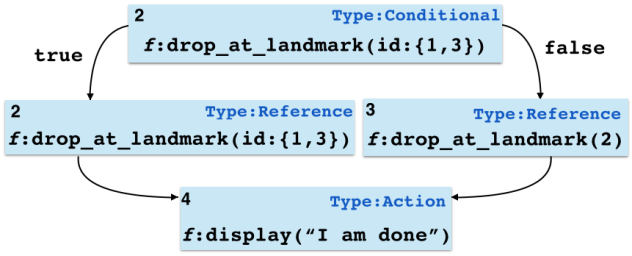


Figure 4.9. Example GIG that is extracted from the graphs in Figures 4.8a and 4.8c. The parameters in nodes 3 and 4 are instantiated, since they were shared by the two original IGs. Instead, the parameters in nodes 1 and 2 are left ungrounded.

provided is similar to a GIG (Algorithm 2). Specifically, we keep a set of candidate proposals, denoted *props*, that match the final part of the partial task. When the partial task changes (line 2), we first update this set to remove any elements that no longer match the task being taught (line 3). Then, we add new elements for every GIG that starts with the new vertex (line 4). When a threshold percentage τ of one or more GIGs in this set matches the current partial task, the robot proposes the largest GIG and breaks ties randomly (lines 6 and 7).

Algorithm 2 Task Autocompletion

```
1:  procedure AUTOCOMPLETE(GIGs, ig, props, τ)
2:      if hasChanged(ig) then
3:          props ← deleteNotMatching(ig, props)
4:          props ← addNewMatching(ig, props, GIGs)
5:          (best, similarity) ← bestMatch(ig, props)
6:          if similarity ≥ τ then
7:              propose(best)
8:          end if
9:      end if
10: end procedure
```

When a specific proposal is found, the robot displays a representation of the GIG and asks for permission to demonstrate the task. Having previously filtered all the incomplete GIGs, all the proposals can in fact be executed. When granted permission, the agent demonstrates an instance of the GIG, noting when a parameter is ungrounded.

At the end of the demonstration, the agent asks if the partial task should be autocompleted with the demonstrated task. If so, the agent asks for specific values for all of the ungrounded parameters. At this stage, the agent suggests initial values for each ungrounded parameter p_i by sampling from its corresponding distribution ϕ_i.

After all of the parameters are specified, the nodes matching the general task in the partial task are replaced with one *Reference* node. When visited, this node executes the

referenced GIG, instantiated with the provided parameters. With this substitution, the length of the task is reduced.

Figure 4.10 shows a sample task acquisition for a Baxter manipulator interacting with a user. After the first command, the robot finds at least one general task starting with $display\_message$. However, none of the GIGs recognized surpass the similarity-threshold τ. When the third instruction is given to the agent, this threshold is surpassed, and the autocompletion procedure is started. First, the robot asks permission to perform a demonstration of the general task. After completing the demonstration, the robot asks if the autocompletion is correct. If so, it also asks for ungrounded parameters to be specified and suggests values using each parameter's distribution.

```
Example Interaction

U: Open Gripper
R: I will open my gripper.
R: What should I do next?
U: Display message "Hello".
R: Ok, what should I do next?
U: Set your left arm to 80 degrees.
R: I think you are teaching me
   something similar to: GIG_14.
R: Can I demonstrate it to you?
U: Yes.
R: First I will display the message "Hello".
R: Then I will set my left arm to 80 degrees.
R: Now I will set my right arm to 90 degrees (open).
R: This is my full suggestion.
R: Would you like to use it?
U: Yes.
[User specifies open parameters]
[User can rename the GIG]
```

Figure 4.10. Sample autocompletion interaction during task acquisition.

4.2.3 Evaluation

Several of our robots can perform generalization and autocompletion, including a manipulator and a mobile base. In order to demonstrate the value of our approach, we define two sets of tasks. Intuitively, the first set of tasks represents a robot that is still acquiring completely new capabilities. Instead, the second set of tasks represents a robot that is acquiring instances of tasks that it already knows. More formally, in the first set, S_d no tasks are repeated. They share only a small fraction of similar components that can be generalized. To show that

generalization takes place, we use a second set, S_r consisting of two repetitions of each of elements in S_d with different parameters. We see that the algorithm recognizes and autocompletes the second instance of each task.

An additional benefit of this approach is that we we can keep one common library for all of our robots. If the robots have different primitives, their tasks are automatically generalized apart. In fact, tasks belonging to different robots will not share any node to the different primitives available. However, if two robots share primitives, our approach can learn subgraphs common to both of them. In the rest of this section, we show in detail experiments run on a Baxter manipulator robot.

Experiments with Baxter

Baxter has two 7 degree of freedom arms, cameras on both arms, and a mounted Microsoft Kinect. The robot-primitives on Baxter manipulate the arms, open their grippers, display messages, and sense landmarks. The frequent subtree mining algorithm we employ is an open source version of SLEUTH [2].

The tasks that Baxter can perform range from waving to making semaphore signs to pointing at landmarks. Many of Baxter's tasks involve picking up an object and moving it to another location. For instance, Figure 4.11 shows Baxter searching for a landmark to see if a location is unobstructed to drop a block. Without task generalization and autocompletion, a new task must be provided for each starting location and ending location in Baxter's workspace. With generalization and autocompletion, these locations become ungrounded parameters that can be instantiated with any value.

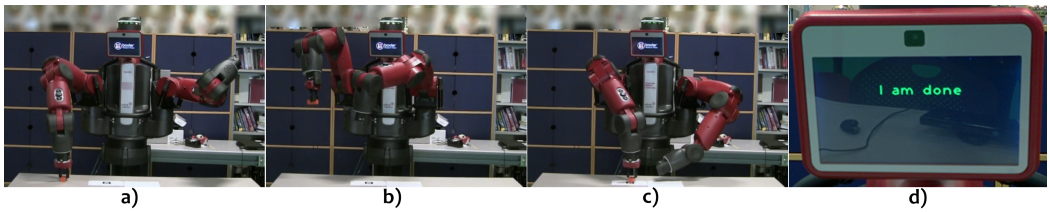


Figure 4.11. Baxter performing an instance of the GIG shown in Figure 4.9. First, Baxter picks up the orange block (a); then Baxter checks if a location is unobstructed with its left arm camera (b); Since the location is unoccupied, Baxter drops the Block. (c); Finally, Baxter says that it is done (d).

For this experiment, 15 tasks were taught by two users familiar with robots but not the teaching framework. These tasks ranged from waving in a direction, to pointing to visual landmarks, to placing blocks at different positions, to performing a series of semaphore signals to deliver a message. S_d has 15 distinct tasks and S_r has 30 tasks. The average length of a task in both sets is 9.33 nodes.

[2] www.cs.rpi.edu/~zaki/software/

Experimental Results

As we accumulate the library incrementally, the order in which tasks are provided affects the generalization. To account for this, we ran 1000 trials where we picked a random ordering to our sets and had a program incrementally provide them to Baxter. The GIG library was updated after each task was provided. At the end of every trial, we measured the number of steps saved using autocompletion compared to providing every step of the task. This measurement includes steps added due to incorrect autocompletion suggestions. For this particular experiment, the support was fixed to 2, the minimum GIG length to 2, and task autocompletes were suggested if $\tau = 30\%$ of a GIG matched the partial task.

We compare our results to an upper bound on the number of steps saved by a specific autocompletion algorithm. For any set of tasks that take n steps to be provided, and are only proposed once τ percent of a task matches a GIG, we have:

$$OPT \leq (1 - \tau) \cdot n$$

This corresponds to generalizing every task after τ percent of it has been acquired. For S_d and S_r we have:

$$OPT_d \leq 92$$

$$OPT_r \leq 184$$

Table 4.3 reports the result of the experiment. Specifically, in this table we report the following measures:

- **Maximum Steps Saved** $(\%)$**:** maximum percentage of steps saved over all permutations, in comparison to the theoretical upper bound.
- **Average Steps Saved** $(\%)$**:** average percentage of steps saved over 1000 permutations, in comparison to the theoretical upper bound.
- **Average Partially Autocompleted:** average number of tasks that were *partially* autocompleted with a GIG.
- **Average Completely Autocompleted:** average number of tasks that were *completely* autocompleted with a GIG.

As expected, S_r benefits the most from the autocompletion method. Specifically, the average percentage of steps saved compared to OPT_r is approximately 82%. For S_d, the average compared to OPT_d is 33%. In the former case, the robot is able to leverage the knowledge of the similar tasks it already knows. Indeed, our approach meets the theoretical upper bound when provided tasks from S_r in the optimal ordering. This fact is additionally underlined by the number of tasks in which the robot suggested any correct GIG. In particular, on average the robot proposed a correct autocompletion suggestions for 65% of graphs in

Table 4.3. Results obtained for the two sets taught to the Baxter robot.

	1st Set (S_d)	2nd Set (S_r)
Max. Steps Saved	70.65%	100%
Avg. Steps Saved	$33.44 \pm 14\%$	$81.92 \pm 7.05\%$
Part. Autocompleted	4.72 ± 1.56	4.70 ± 1.58
Compl. Autocompleted	0 ± 0	15 ± 0

the second set, and 30% in the first set. Also, the 15 IGs added to the second set are all completely autocompleted from their other similar instance. Furthermore, this happens with a statistically insignificant change to the effectiveness of the partial autocompletions.

Finally, the size of the GIG library for the first set was 21 and that the size of the GIG library for the second set was 45. This shows that the heuristic filters we proposed achieves a good balance between saving steps and library size.

4.2.4 Section Summary and Discussion

In this work, we considered autonomous robots that persist over time and the problem of providing them additional tasks incrementally. Compared with the problem addressed in Section 4.1, in this section we proposed an alternative approach for enabling a robot to learn parametric tasks. In fact, instead of requiring a user to directly teach parametric tasks to a robot, in this approach we considered using previous task knowledge to enhance the capability of a robot. To this end, we presented an approach for generalizing collections of tasks taught to a robot over an extended period of time. This bottom-up technique allows an agent to learn parameterized tasks from collections of specific examples, based on task structure. The main contributions are a method for finding generalized tasks using frequent labeled subtree mining algorithms, and an approach for agents to propose future steps of a task to reduce human-robot interactions and the size of each task. By mapping our tasks to trees, we are able to handle the difficulty normally associated with finding subgraph isomorphisms. Our generalization and autocompletion algorithms have been successfully deployed on multiple robots, acquiring large task libraries. Our experiments report in-detail the effectiveness of our contributions on a Baxter manipulator robot for two sets of tasks. With both sets, we found a significant reduction in the number of steps needed for Baxter to acquire new tasks.

In terms of future work, there may be other applicable filters for deciding which tree patterns should be converted to GIGs. Furthermore, structure-based generalization is just one way for a robot to express its capabilities. Future research may look at domain-specific forms of task generalization. For instance, we might use the motion patterns of a manipulator as a measure of task similarity. Finally, we are working to allow users to actively query the robot for the tasks it knows. These queries may be goal-based or involve domain-specific

information associated to each task.

4.3 Multi-Robot Task Teaching

This work has been published in:

◇ *Multi-Robot Task Acquisition through Sparse Coordination*. G. Gemignani, S. Klee, M. Veloso, and D. Nardi. International Conference on Intelligent Robots and Systems, IROS 2015.

In this third section, we extend the previous works by considering a group of robots. Specifically, we consider the problem of *separately* and *incrementally* providing tasks to a group of autonomous robots that need to infrequently coordinate. The robots may have very different internal representations of their state, actuation capabilities, and sensing capabilities. Furthermore, they may acquire their tasks in different manners. For example, a manipulator may be taught by a human through natural language, and a mobile base may acquire tasks from a planner. However, to pick up a package and deliver it, the two robots must work together. In these kind of tasks, we note that the robots do not need to coordinate at every decision step. In fact, much of their tasks can be completed entirely independently. In literature, this concept of coordinating when necessary is known as *sparse-coordination* [95]. In terms of task representation, sparse-coordination represents the joint state space only when the robots need to coordinate.

Our goal is to enable heterogeneous robots, acquiring tasks from different providers, to solve problems requiring sparse coordination. We first note that coordinating different robots requires a common means of communication. To this end, we contribute a task representation for a robot to incrementally acquire a task with preconditions and effects represented in a shared domain language. Then, we present an approach to sparsely coordinate robots using this task representation. Specifically, each agent conditions on the state of other robots to sparsely coordinate.

Tasks are represented again as Instruction Graphs composed by action and sensing primitives, conditionals, and loop structures. The preconditions and effects of the actions are written using a common domain language. Robots keep track of their own state, and condition on the state of each other by sending queries to interact. By only representing the coordination between robots when necessary, this approach is partially immune to the combinatorial explosion in the number of states found in other representations.

Our contribution has been used to coordinate several robots, including a Baxter and a CoBot robot. Baxter is an industrial manipulator robot able to perform complex manipulation tasks. CoBot is instead an omnidirectional mobile service robot equipped with a variety of sensors including a laser range finder, microphones, a camera, and Microsoft Kinect sensors [129]. We first show an example of how the two arms of a Baxter manipulator can be treated as two autonomous agents and coordinated. We then show a more complicated example involving a CoBot and the two arms. Figure 4.12 shows both robots.

Figure 4.12. Baxter manipulator and CoBot mobile service robots coordinated with the proposed approach.

Next, we introduce the technical details of our approach, present demonstrative examples, and discuss the contributions with some hints on future work.

4.3.1 Approach

We consider several autonomous robots with primitives that represent their actions and sensing capabilities. Each robot acquires its task separately and incrementally by interacting with a provider, be it a user or a planner. Our goal is to allow these robots to sparsely coordinate to complete their tasks. In this part of the thesis, we present an extension of the Instruction Graphs formalism that allows us to represent and query robot states.

Sparse-Coordination Instruction Graphs

To sparsely coordinate, robots must keep track of their state and be able to query the state of one another. We define Sparse-Coordination Instruction Graphs (SCIG) as graphs $G = \langle V, E \rangle$ where each vertex v is a tuple:

$$v = \langle id, \mathit{InstructionType}, f, P, \mathit{Prec}, \mathit{Eff} \rangle$$

where the additional elements *Prec* and *Eff* respectively represent sets of preconditions and effects of the function *f*. More generally, each function *f* has an associated set of literals $\mathscr{L}_f$ that represents its preconditions and effects. Thus, we define:

$$\mathscr{L} = \bigcup_{\forall f} \mathscr{L}_f$$

as the common domain language used by all of the robots. We represent each literal using STRIPS semantics [83]. In particular, each action adds or removes positive literals from

the robot's current state. While robots may represent their internal state differently, their primitives express this state in terms of the common set of strips literals, $\mathscr{L}$.

To associate these preconditions and effects to actions, each robot sensing and actuation primitive is defined in the Planning Domain Definition Language (PDDL) [130].[3] For example, a Baxter manipulator may have an action *pick_up(object_id)*, to pick up an object with a given ID. Internally these objects are represented as a 3D point in space and bounding boxes. However, the effects of the action are to remove the literal *hand_empty*, and then add the literal *holding(object_id)*. Figure 4.13 shows an example PDDL definition for Baxter's "pickup" action. Currently, we assume that all changes in state are captured by the robot primitives and that each robot can only modify its own state.

```
(:action pickup
 :parameters (?x)
 :precondition (and (OBJECT ?x)
                    (hand_empty))
 :effect       (and (holding ?x)
                    (not (hand_empty)))
```

Figure 4.13. Example PDDL definition for the primitive "pickup". As preconditions, its parameter must be an object, and the hand must be empty. The effects are that the hand is no longer empty, and the robot is holding an object.

During execution, each robot keeps track of its own state. Specifically, the state predicates are either appended or deleted from the robot's state according to the effects of the executed action. We introduce a special function *check_literal*, used in Conditional and Looping vertices that can condition on the state of any agent. The *check_literal* function takes as input a unique robot identifier and a query. In our framework the query is represented as a set of STRIPS predicates, possibly composed with the *and*, *or* and *not* operators. The query is routed to the the robot with the corresponding identifier.

When a robot receives a query, it is evaluated against its current state. Each robot adopts a closed-world assumption when responding to queries. In particular, the robot checks that positive literals are present in its state and that negated literals are absent in its state. The result of this query is returned to the requesting robot. In this way each robot has only a representation of its own state, and makes no assumptions about the state of another.

Figure 4.14 shows a partial example of a SCIG for a CoBot mobile base, where the *check_literal* function is used to condition on the state of a Baxter manipulator. Specifically, the CoBot will perform the *move_to* action if Baxter's state does not contain *hand_empty*.

With this approach we are able to implement coordination at a high level. In particular, we define several useful coordination actions from Loops and Conditionals:

[3] Although we represent actions using PDDL, any other language could be used to define the common domain language.

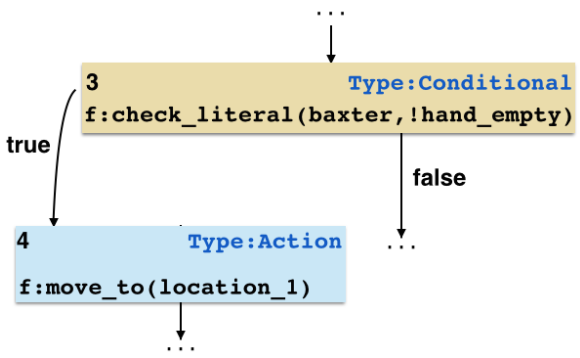


Figure 4.14. Partial example of a SCIG for a CoBot conditioning on the state of a Baxter manipulator. If Baxter's state does not contain *hand_empty*, CoBot will perform the *move_to* action.

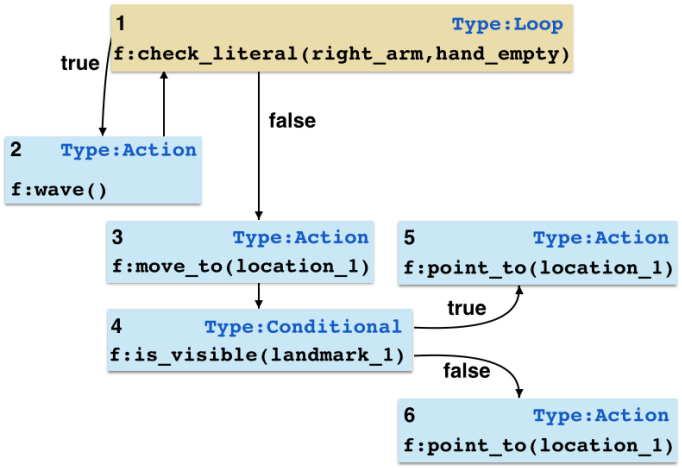
- **Wait Until**: The robot waits until another robot is in some state. This is implemented with a Loop.
- **Act Until**: The robot repeats some actions until another robot is in some state. This is implemented with a Loop.
- **Ask**: The robot conditions on the state of another robot. This is implemented with a Conditional.

We provide examples of each of these forms of coordination in the next sections.

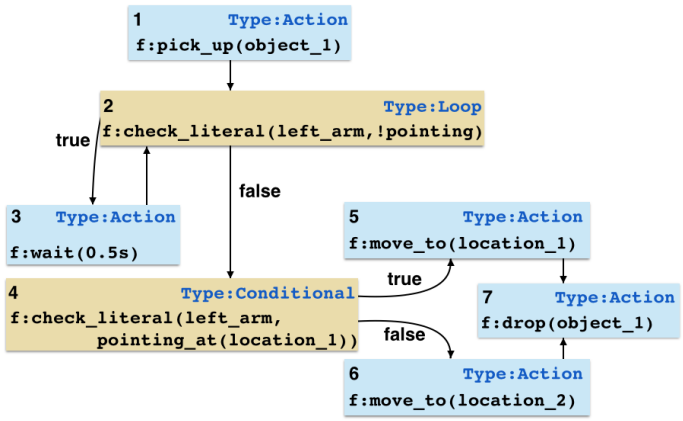
We note that for sparse-coordination, many of typical problems of multi-robot communication do not arise. For instance, consistency is not an issue, because the robots keep track of only their own state, and directly query each other's state as needed. Since coordination is infrequent, and at a high-level, the robots can also cooperate in environments with high-latency and low-bandwidth. In this work, we do not address the problem of faulty sensors or non-deterministic action effects. For now, we assume that the robot-primitives are all fault tolerant.

4.3.2 Demonstrative Examples

We coordinated several robots with the presented approach, including a manipulator and a mobile base. In this section we show how multiple robots can be taught to perform different tasks that involve sparse-coordination. In particular, we show how a Baxter robot can be instructed to perform a manipulation task during which its two arms need to sparsely interact with each other. Then, we extend this example by considering also a CoBot mobile base. In this case, we show how the two arms and the mobile service robot can be instructed to deliver and store an object.



(a)



(b)

Figure 4.15. Sparse-Coordination Instruction Graphs extracted from the text in Figure 4.16. Specifically, (a) represents the graph for the left arm, while (b) represents the graph for the right arm. The nodes in yellow represent the vertices that require a query to another robot's state.

Store Task

We first show how the two arms of a Baxter manipulator can be treated as separate agents and coordinated. We denote the arms as *left_arm* and *right_arm*. Table 4.4 shows the set of robot primitives for both arms, with their associated preconditions and effects. Our task is to have the left arm assist the right arm in finding an unobstructed location to place an object.

In this example, a user describes the task to each agent in two separate teaching sessions through natural language. Specifically, the left arm is instructed to wave until the right arm picks up an orange wooden block (Figure 4.19a). At this point, the state of the right arm is changed to *holding(object_1)* and the function *check_literal(right_arm, hand_empty)* returns false. After realizing this fact, the left arm starts checking if a landmark can be detected at the drop position (Figure 4.19b). In the case a landmark is detected, the left arm points at it,

```
Left arm:

wave while right hand is empty
move to location 1
if landmark 1 is visible
point to location 1
otherwise point to location 2

Right arm:

pick up object 1
wait while left arm is not pointing
if left arm is pointing at location 1
move to location 1
otherwise move to location 2
end if
drop object 1
```

Figure 4.16. Natural language input provided to the two arms. The names of the agents are shown in red, and their states are shown in blue.

reaching the *pointing_at(location_1)* state. The function *check_literal(left_arm, !pointing)* now returns false and the right arm drops the block at *location_1* (Figure 4.19c). Instead, when the landmark is not detected the left arm points at an alternative location (*location_2*) where the orange block can be dropped (Figure 4.19d).

Figure 4.16 shows a natural language description of the task provided to the two arms. Instead, Figures 4.15a and 4.15b show the corresponding Sparse Coordination Instruction Graphs. In this specific case, the natural language description of the tasks was parsed through the aid of specifically developed parsers, similarly to a previous work [32]. The descriptions were grounded to objects and locations of a knowledge base containing a high-level description of the environment and the robot primitives. In this example we assume that the two robots have the same high-level representation of the environment. In other words, we assume that the two robots agree on the position of the two possible locations. The execution of the task can be seen in the video attached.[4]

Deliver and Store Task

Next, we extend the previous example by considering also a CoBot robot [129]. CoBot must coordinate with the Baxter's arms in order to deliver and store an object. To this end, we modify the previous example by making the *right_arm* wait for CoBot to be at a specific location. Note that the *right_arm* is instructed to wait just for simplicity. It could also be instructed to do other work while waiting for CoBot's arrival. When CoBot reaches the state

[4] http://youtu.be/4nLjuLhFUvk

Table 4.4. Baxter arms primitives with associated preconditions and effects.

*A*ction Primitives	Preconditions	Effects
wave()	-	-
wait(time)	-	-
is_visible(landmark_id)	-	-
move_to(location)	-	-pointing, -at(old_location), +at(location)
pick_up(object_id)	hand_empty	-hand_empty, +holding(object_id)
drop(object_id)	holding(object_id)	-holding(object_id), +hand_empty
point(location_id)	-	-at(old_location_id), +at(location_id), +pointing, +pointing_at(location_id)

at(location_3), the *right_arm* will pick up the delivered object from CoBot's basket, storing it at the location pointed out by the *left_arm*. For CoBot, we defined the robot primitives shown in table 4.5. For Baxter we used the previously described primitives.

Table 4.5. CoBot primitives with associated preconditions and effects.

*A*ction Primitives	Preconditions	Effects
Say(message)	-	-
Move_to(location)	-	-at(old_location_id), +at(location_id)

The user teaches the tasks to the robots in three separate teaching sessions. Figure 4.18 shows a natural language description of the tasks provided to CoBot and to the *right_arm*. The task given to the *left_arm* is the same as in the previous example. Figure 4.20 depicts the Sparse Coordination Instruction Graph extracted for CoBot. Since the SCIG of the right arm is almost identical to Figure 4.15b, we omit it.

Discussion

Sparse-Coordination Instruction Graphs allow users to teach a wide-variety of tasks that require multi-agent coordination. In particular, they are well suited to tasks that require high-level cooperation between robots. We have found the approach especially effective with robots that have separate goals. For instance, our fleet of CoBots perform many tasks, such as escorting people and picking up objects. Some of these tasks require brief interaction

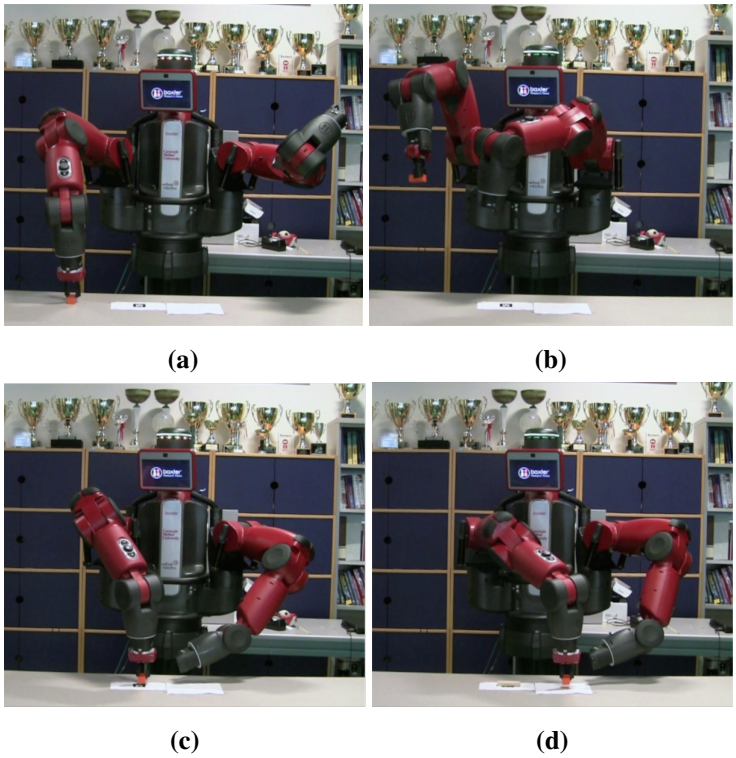


Figure 4.17. (a) The left arm is shown waving until the right arm picks up an orange wooden block. (b) The left arm checks if the drop position is open. (c) Since this position is open, the left arm points at it where the object will be placed. (d) In the second run, since the drop position is not open, the left arm points at an alternative position.

with Baxter, which has its own goals to accomplish. The robots coordinate infrequently because their goals require a limited amount of interaction.

We can also represent joint-plans with goals that require a tight coupling of robot actions at each decision step. An example of such a task is the bimanual manipulation of a large object. However, each robot will need to make many queries to represent most of the joint-state space before acting. Thus, it is often impractical for a user to teach tightly coordinated tasks to the robot.

4.3.3 Section Summary and Discussion

In this section, we extended previous related works by considering multiple robots. In particular, we addressed the problem of multiple robots separately and incrementally acquiring tasks that require coordination. By leveraging principles from sparse-coordination we enable robots to acquire and represent joint-robot plans compactly. Specifically, we introduced Sparse-Coordination Instruction Graphs, which encapsulate robot-primitive preconditions and effects. The robots act independently, and only coordinate when necessary by querying each other's state. We demonstrated this approach with two examples. First we treated both

```
CoBot:

move to location 3
say ``I am here to deliver a package"
wait while left arm is not pointing
move to location 4

Right arm:

wait while CoBot is not at location 3
pick up object 1
wait while left arm is not pointing
if left arm is pointing at location 1
move to location 1
otherwise move to location 2
end if
drop object 1
```

Figure 4.18. Natural language input provided to CoBot and the *right_arm*. The description of the *left_arm* task instead is the same shown in Figure 4.16. The names of the robots are shown in red while their states are shown in blue.

arms of a Baxter robot as separate agents and had them coordinate to store an object at an unobstructed location. Then, we extended this example by having a CoBot mobile base deliver the object that the arms stored.

As a future work, we are investigating extensions to represent tasks that require tight coordination more compactly. We are also interested in how our coordination approach can be used with knowledge-acquiring actions. For instance, in cloud robotics a robot-primitive may request information, or even queue a task on another robot.

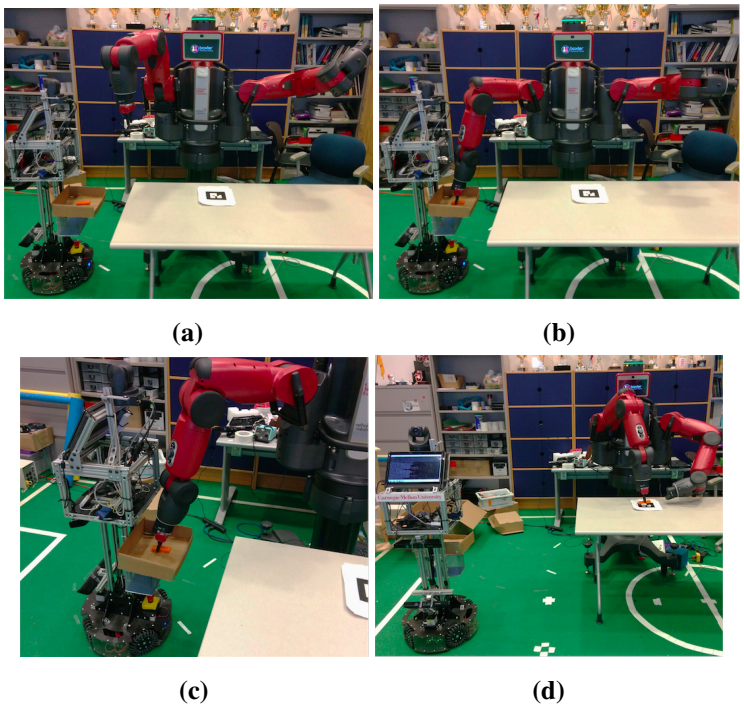


Figure 4.19. (a) CoBot reaches the pick up location while the right arm is waiting and the left arm is waiving. (b) The right arm can now pick up the object. (c) Detailed view of Baxter picking up the object. (d) The left arm points at location 1, the right arm drops the object, and CoBot leaves.

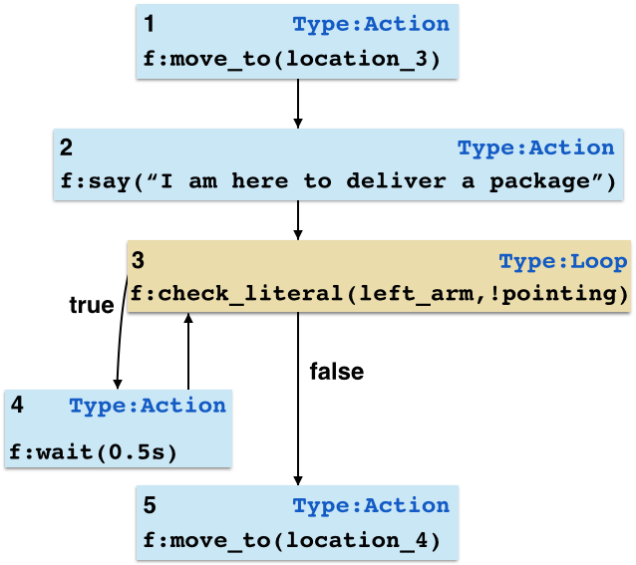


Figure 4.20. SCIG extracted from the CoBot's task description in Figure 4.18. Node 3 requires a query to another robot's state, in this case Baxter's left arm.

4.4 Chapter Summary and Discussion

In this chapter, we have presented multiple approaches for acquiring procedural knowledge. Such approaches allow a non-expert user to specify new tasks based on the action primitives already known by the robot. In the development of these approaches, we focused on the knowledge about specific tasks, so that the robots are able to cope with specific user needs, rather than always performing preprogrammed tasks. Additionally, we have again shown how human-robot collaboration can play a key role, enabling even non-expert user to instruct a robot through simple interactions. The designed approaches have been tested with three different robots, in different environments and with multiple users, showing the overall generality of the contributions proposed. In particular, these contributions have been experimentally evaluated on two kinds of mobile bases as well as a manipulator robot. The results of the experiments show that, despite some artificiality of the language used during the interaction, the knowledge acquisition process is robust and easy to be performed also by non-expert users and on different robotic platforms.

Chapter 5

Acquiring User Knowledge

The acquisition of the two types of knowledge presented in the previous chapters is carried out through the interaction with the user. However, each user is a unique person, with distinctive understandings of the world and different ways of talking about them. In this chapter we address two problems regarding preferences of users interacting with the robot. Specifically, we first consider the problem of learning how specific users communicate with robots. To this end, we describe an approach that enables a robot to learn new expressions used by a user to refer to objects in the operational environment. Additionally, this approach allows a user to recognize the robot understanding capabilities, enabling him or her to learn how to interact with an unknown robot. Next, we consider the problem of learning how specific users understand and reason about qualitative spatial relations. To this end, we present an approach that allows a robot to learn user preferences related to regions in the environment used during task executions. This learning is based on the feedback given by the human to the robot after the completion of each task.

5.1 Learning to Understand Each Other

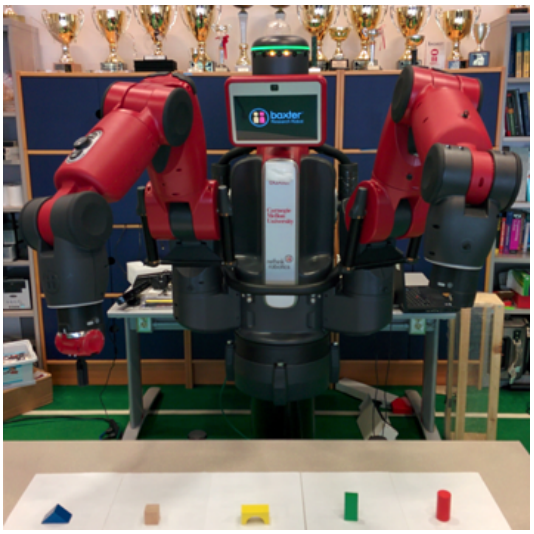
This work has been published in:

- ◇ *Language-Based Sensing Descriptors for Robot Object Grounding*. G. Gemignani, M. Veloso, and D. Nardi. 19th Annual RoboCup International Symposium, 2015. **Best Paper Award**.

As a first setting, we consider the scenario in which a human needs to instruct an autonomous robot through a natural language interface. We assume that this robot is provided with a specific internal representation of the environment that is unknown to the user. For example, a robot might be able to understand colors but not orderings. Also, it may be able to recognize shapes but may not be able to resolve spatial referring expressions. In this scenario, we address the problem of allowing a robot to recognize what object properties can or cannot be grounded with its current sensing capabilities. Moreover, we address the

problem of learning new ways of referring to objects by exploiting past interactions with the user. While addressing these problems, our goal is to enable an untrained user to understand, through the interaction with the system, which object properties the robot can understand. These interaction can then be used to enhance the grounding capabilities of our robots. Note that in this section, we will use the term grounding to refer to the concept of “physical symbol grounding" as defined by Vogt [131].

To this end, we contribute a novel approach that enables the robot to recognize unknown objects properties contained in the received commands and warn the user about them. We note that the majority of the techniques proposed in literature make the implicit assumption that if a robot can semantically parse an utterance, then it will be able to ground it. We believe that this assumption may not always hold, since while a robot may be able to correctly parse a sentence and extract its semantics, it may not be able to ground it due to a missing sensing capability. Hence, we internally represent sensing capabilities through *sensing descriptors* and use them to recognize unknown object properties. At this point, the robot can notify the user and request an alternative command. In addition, the robot can learn new object properties by leveraging these interactions with the user. After learning, the robot is able to execute the natural language commands, as in Figure 5.1. Our contribution has been used to instruct several robots, including a Baxter manipulator able to perform complex manipulation tasks. In this section, we describe all the components of our approach along with in depth illustrative examples with the Baxter manipulator robot.



Commands
- pick up the cubic block
- grab the yellow block
- touch the second block
- point at the left block
- take the narrow block

Figure 5.1. Baxter manipulator robot used in our experiments and examples of commands that our approach is able to successfully execute.

In the remainder of the section, we provide an overview of our natural language approach describing all of our contributions thoroughly. Then, we present an application of the approach to the case of a Baxter manipulator. This setting is then used to quantitatively

evaluate the proposed approach.

5.1.1 Approach

In this section, first we motivate and introduce the concept of *sensing descriptors*. Next, we present our approach for human-robot natural language interaction based on such a concept. Finally, we show how the system can leverage previous interactions with users to learn previously unknown referring expressions for the objects perceived.

Sensing Descriptors

Usually, when dealing with robots and natural language user commands, a standard processing chain is adopted to decouple the semantic parsing problem from the grounding problem [28, 29, 25, 32]. First, the natural language utterances are converted into text through an automatic speech recognition (ASR) system. Next, the text is converted into a specific representation that captures the semantic meaning of the uttered command. This conversion is carried out either through grammars or probabilistic approaches. The obtained representation is then "contextualized" in the operational environment through a grounding process. The final result is an executable function and a set of parameters passed as input.

In general, during this process each natural language command is grounded through a combination of sensing actions and queries to a given knowledge base. However, this approach does not take into account the sensing capabilities of the robot. In fact, we note that approaches proposed in literature often assume that if the robot can semantically parse an utterance, then it will be able to ground it. However, a robot may be able to correctly parse a sentence and extract its semantics without being able to ground the command due to a missing sensing capability. Hence, we propose to explicitly represent in the knowledge base these capabilities and use them to recognize parts of the commands that could only be grounded through a sensing ability not available to the robot. To this end, we introduce the concept of *sensing descriptor*.

Each sensing operation carried out by a robot can be defined as a function that takes as input a particular type of sensed data and outputs a value expressed in the internal representation of the robot. This value will be an instance of a sensing descriptor. Formally, a sensing operation can be defined as:

$$f_{sensing} : D \rightarrow SD$$

were *D* is the particular type of data sensed and SD is a specific sensing descriptor. As an example, let's consider the operation of sensing the color of a particular object. The input will be the RGB values of the pixels sensed by a camera. The output will be one or more instances of the sensing descriptor *color* (e.g., $[255, 0, 0]$ or red depending on the internal representation of the robot). These sensing descriptors can be used to check if the utterances

received from a user can be grounded with the current capabilities of a robot. We perform this check as an intermediate step between the semantic parsing and the grounding process, as explained in the next section.

Human-Robot Natural Language Interaction

Figure 5.2 shows an overview of our processing chain. Specifically, this processing approach is divided in four consecutive steps. First, speech is converted into text using a free-form speech-to-text engine. Text from speech is confirmed by the user. Thus, without loss of generality, the input of the system is established as natural language text.

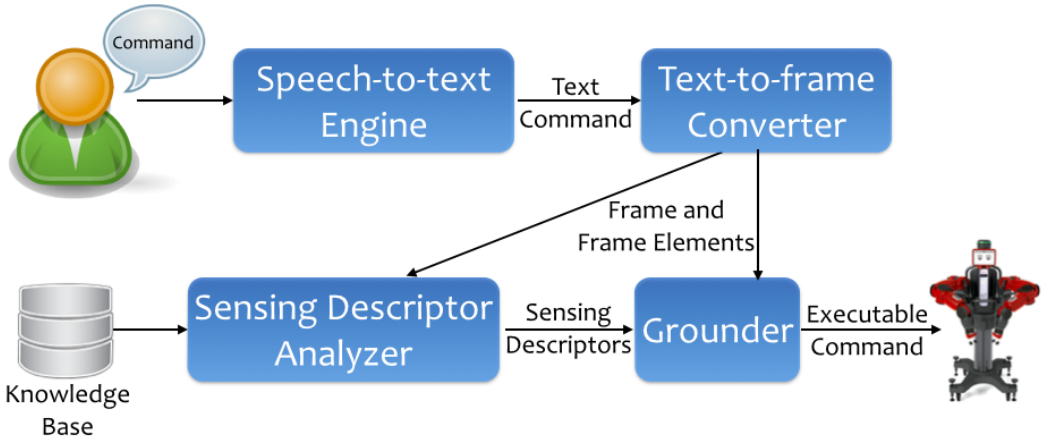


Figure 5.2. Overview of our natural language processing chain. Instead of directly grounding the frames extracted from the commands, we perform an additional step that analyzes the sensing descriptors included in the frame elements.

Next, the text is converted into a specific representation characterizing the semantic of the sentence. This step is performed through the aid of specific grammars that drive the recognition process by attaching a proper semantic output to each grammar rule. The output has the form of a *semantic frame* representing a "situation" in the world (typically an action) inspired by the notion defined in the *Frame Semantics linguistic* theory [33]. The meaning of each frame can be enriched by semantic arguments, called *frame elements*, that are part of the input sentence. The output of the recognition process is then converted to a parse tree that contains syntactic and semantic information. This information is used to instantiate a frame, similarly to the work by Kollar *et al.* [124]. As an example, the command "pick up the red block" will be mapped to the GETTING frame. The sub-phrase "the red block" will instead represent the specific frame element THEME, which represents the target of the GETTING action.

At this point, instead of directly grounding the frames in the internal representation of the robot we explicitly represent each sensing descriptor that can be recognized and grounded by the robot, also defining the range of values that it can assume. Formally, in

our knowledge base we represent every sensing descriptor SD_i that can be handled by a robot, also representing all its possible known instances $sd_j \in SD_i$. We use these sensing descriptors to check if the obtained frame elements can be grounded with the current sensing capabilities of the robot. Hence, we define *sensing descriptor extractor* a function ψ able to extract from each frame element all the contained instances of sensing descriptors. Formally, if we define FE the frame element type, the *sensing descriptor extractor* can be specified as:

$$\psi : FE \rightarrow \{SD_1, SD_2, ...SD_n\}$$

where SD_i is a specific sensing descriptor extracted from the given frame element.

There are many possible ways to implement this function. In our approach, the sensing descriptor extractor is represented as a parser that exploit grammatical rules to carry out its task. In fact, we note that particular grammar elements are associated to referring expressions that require sensing capabilities to be grounded. Hence, for our specific case, we propose an heuristic rule that selects all the adjectives found in the frame elements. This rule is used to handle element frames such as "the big red and cylindric block" where the word "and" may or may not be used and where the words "big", "red", and "cylindric" need to be extracted. The words extracted represent the sensing descriptor instances that will be checked in the knowledge base. If all the instances are found to belong to a particular sensing descriptor expressed in the knowledge base, the system will proceed to ground the command, otherwise we either leverage dialog or adopt a probabilistic approach to resolve this issue.

Handling Unknown Sensing Descriptors

When an instance of a sensing descriptor is not found in the knowledge base two different scenarios may occur:

- The referring expression belongs to an unknown sensing descriptor and it has never been used by a user;
- The referring expression belongs to a sensing descriptor not available to the robot but it has been previously used to refer to a particular object.

In the first case, the robot asks the user to provide an alternative referring expression to the object, while keeping track of all the referring expressions used in the different interactions. These expressions are in fact the unknown sensing descriptors found in the frame elements that are not represented in the knowledge base of the robot. Since there is a limited amount of sensing properties that can be expressed, eventually the user will refer to the object in a way that the robot can understand, enabling the robot to associate all the previously used referring expression to the grounding found. To this end, we explicitly represent this association in the knowledge base by using the binary logic predicate *sd_grounding(X, Y)*. In this predicate, *X* represents the unknown instance of a sensing

descriptor, while *Y* represents the grounding found through the multiple interactions with the user.

For example, let us consider a robot that is only able to recognize colors. Additionally, let us assume that a user needs to refer to a cylindric red object. At a first interaction a user might refer to the object as "the cylindric block". When warned by the robot that the term "cylindric" can not be understood, the user will provide a command with an alternative referring expression. Eventually, the user will refer to the object as "the red block", enabling the robot to correctly ground the expression and assert *sd_grounding(cylindric, block_1)* in his knowledge base. Figure 5.3 shows an example of a dialog between a manipulator robot and a user that our system is able to understand and the information that the robot is able to extract and store in the knowledge base.

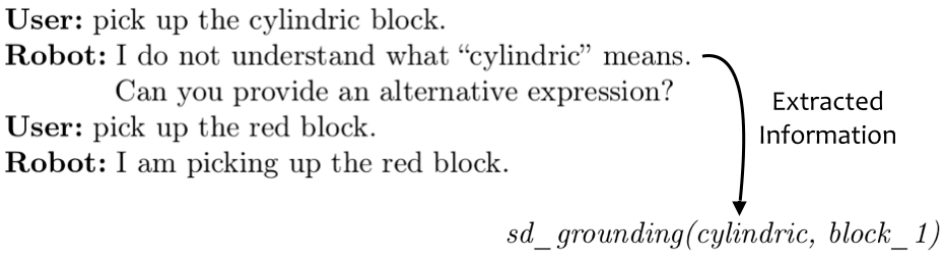


Figure 5.3. Example of a dialog between the robot and a user that our system is able to understand and the information extracted and stored in the knowledge base.

To each association instance in the knowledge base, a number is also attached to keep track of how many times the referring expression has been used to refer to a particular object. This counter is needed to handle the alternative scenario that may occur. In this second scenario, a referring expression belonging to an unknown sensing descriptor has been previously used to refer to a particular object. In this case, we adopt a probabilistic approach to ground the expressions. Specifically, if we define KB as the knowledge base available to the robot, R the referring expression being analyzed and G the possible groundings for it, we can obtain the most probable grounding by selecting the one that maximizes Bayes rule:

$$p(G|R; KB) = \frac{p(R|G; KB) \cdot p(G; KB)}{\sum_R p(R|G; KB) \cdot p(G; KB)}.$$

Here, the prior over groundings $p(G; KB)$ is computed by looking at the counts of each element of G in the knowledge base. The other term $p(R|G; KB)$ is instead obtained by counting the number of times a particular referring expression has been used to refer to a particular grounding, and dividing by the overall number of referring expressions used for the same grounding. Formally, if we define *count* the function that returns the number of times that a particular association has been encountered, we can compute $p(R|G; KB)$ as:

$$p(R|G; KB) = \frac{count(association(R, G))}{\sum_R count(association(R, G))}.$$

After having grounded the expressions, we allow the user to give feedback to the robot to update the counter attached to each association instance. Algorithm 3 reports the overall natural language processing approach. Specifically, the algorithm takes as inputs the natural language command expressed as text and a specific knowledge base. The command is first analyzed to obtain its representation in terms of frames and frame elements (line 3). Next, the sensing descriptor instances are extracted from each frame elements through the sensing descriptor extractor ψ (line 5). Once extracted, the instances are checked against the available knowledge base to find any that cannot be grounded with the current sensing capabilities of the robot (line 8). If an unknown instance is found, the robot exploits dialog and the previous knowledge acquired to assign a grounding to the referring expressions (line 10). Otherwise, the command is grounded into the knowledge base available to the robot to obtain the final executable function (line 13).

Algorithm 3 Ground Command

```
1:  procedure GETEXECUTABLEACTION(Text command C, knowledge base KB)
2:      // Extract frames and set of frame elements
3:      f, FE ← extractFramesAndFrameElements(C)
4:      // Extract the set of sensing descriptor instances
5:      SD ← ψ(FE)
6:      // Select unknown sensing descriptor instances
7:      USD ← selectUnknownInstances(SD, KB)
8:      if USD ≠ {} then
9:      // Exploit Dialog and Previous Experience to ground command
10:         Φ ← handleUnknownSensingDescriptors(USD, SD, KB, f, FE)
11:     else
12:     // Otherwise normally ground command
13:         Φ ← ground(f, FE)
14:     end if
15:     return Φ
16: end procedure
```

5.1.2 Evaluation

In this section we describe in detail how the presented approach has been deployed on a Baxter manipulator robot able to manipulate a set of blocks placed in front of it. This setting has been used to quantitatively evaluate our proposed approach. Since the evaluation space of the experiment was large and generating results with humans was extremely time consuming, the experiments were conducted by using a simulator faithful to the chosen

setting[1]. A representative sample of the scenarios described in this section was successfully run on the manipulator interacting with humans, achieving results that are consistent with those reported in the following sections.

Setup

Baxter has two 7 degree of freedom arms, cameras on both arms, and a mounted Microsoft Kinect. Baxter has been programmed to perform the actions touch, grab, move, point to, and push. These primitives are used to manipulate a set of blocks located on a table in front of the robot. The manipulated blocks have different shapes and colors. Additionally, each block has a unique id, associated with a specific QR code. Given this setting, we considered the sensing descriptors shown in Table 5.1. Specifically, five different blocks were considered:

- A short, wide, triangular, blue block;
- A short, narrow, cubic, brown block;
- A short, wide, bridge-shaped, yellow block;
- A tall, narrow, rectangular, green block;
- A tall, narrow, cylindric, red block.

Additionally, these blocks were associated with the number one through five, respectively. Figure 5.1 shows the described scenario.

Table 5.1. Sensing descriptors considered in the chosen scenario and possible values.

*S*ensing descriptors	Possible values
color	{blue, brown, yellow, green, red, orange, purple}
shape	{triangular, cubic, bridge-shaped, rectangular, cylindric}
block id	{first, second, ..., fifth}
height	{short, tall}
width	{narrow, wide}
spatial location	{left, center, right}

Before accepting commands, the robot was allowed to analyze the scene in order to accumulate knowledge about the operational environment. This knowledge was stored in the form of logic predicates in a knowledge base. The spoken commands given to the robot were converted into text through a free-form ASR[2]. For this particular scenario, a dedicated grammar was developed to convert the natural language commands to the previously described frame representation. To extract the sensing descriptors from the frame elements, a POS Tagger[3]

[1] https://github.com/RethinkRobotics/sdk-docs/wiki/Baxter-simulator

[2] The Google free-form ASR has been used.

[3] We exploited the Stanford POS Tagger to extract the sensing descriptor instances from the frame elements.

was used to grammatically analyze the words in the command. Particularly, we adopted the heuristic of extracting the adjectives related to target objects, considering them instances of a specific sensing descriptor. With this approach we were able to allow users to understand how to instruct the robot while interacting with it.

Approach Evaluation

In order to show the effectiveness of our algorithm, we compared our approach with an algorithm commonly used in the literature. Specifically, the chosen two-step approach first converts the received commands to frames exploiting grammars. Then, it directly grounds the commands without exploiting any information about sensing descriptors. When the algorithm receives a command that can be grounded to multiple targets (e.g., "touch the narrow block" in this scenario), it selects a random target between the possible ones.

The two approaches have been tested by first generating all the possible commands that can be given to the robot in this setting. Figure 5.1 shows some example commands generated. Next, 50 commands were randomly chosen and incrementally given in input to the robot. When the robot wasn't able to understand an object attribute, the property was changed with another one not yet used. This process was repeated until the robot understood the command. Such an operation has been carried out for both approaches and averaged for 100 times by varying the number of sensing descriptors known by the robot. For each run we measured the cumulative number of interactions needed to execute all the 50 commands. Figure 5.4 shows the results obtained in the experiment.

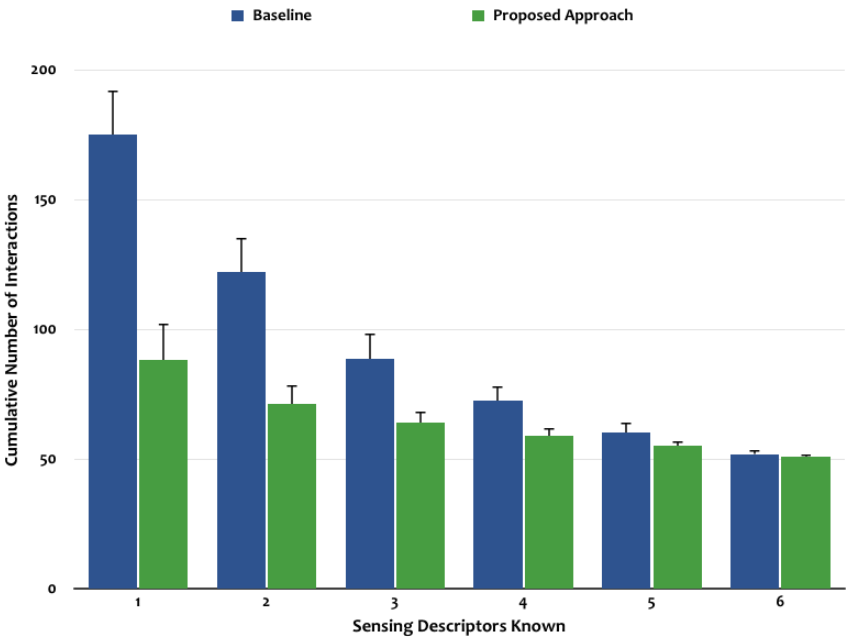


Figure 5.4. Results for the experiment performed on both processing chains averaged for 100 times by varying the number of sensing descriptors known by the robot.

From the graph, it can be noticed that on average, our algorithm required significantly less interactions to ground the randomly chosen commands. Moreover, it is worth noticing the effects of the different available levels of information on the two approaches. In fact, when the two robots were capable of understanding and grounding most of the used sensing descriptors, the two approaches had a comparable result. Instead, when a lower amount of information was available, our approach greatly outperformed the other one, leading to a decrease in interactions needed to understand the command, up to approximately 50% in the chosen scenario.

5.1.3 Section Summary and Discussion

In this section, we considered an autonomous robot provided with an internal representation of the environment, unknown to a user interacting with it through natural language. In this setting, we addressed the problem of allowing humans to understand the internal representation of the robot through dialog. Moreover, we enabled our robot to learn previously unknown object properties leveraging the past interactions with the user. We successfully deployed our approach on a Baxter manipulator robot able to carry out tasks assigned by several users through natural language. Specifically, our experiments report the performance of our algorithm in this scenario, suggesting an improvement in the grounding effectiveness compared to another commonly used approach.

As a future work, we are studying extensions of the proposed approach. In fact, as a long term goal, we would like to generalize the approach allowing our robots to not only recognize unknown object properties but also every unknown concept contained in the received commands.

5.2 Learning Spatial Preferences for Task Execution

As a second issue about user profiling[4], we consider the problem of learning different qualitative spatial understandings of users through human-robot interaction. Specifically, we aim at enabling a robot to learn regions of space used in user commands that are not simply described by the geometry of the environment, but also by their function (or semantics more generally). Such regions are called Context-Dependent Spatial Regions (CDSRs) [99]. Considering these regions, we study the scenario in which a robot needs to carry out a task. Specifically, we focus on tasks that involve reaching a position specified by a user through a spatial relation. An example of this type of task is represented by the command of delivering an object to a person. In this case, the task involves reaching a point near the position usually occupied by the person to which the object needs to be delivered. In this scenario we assume

[4]The ideas described in this section have been developed in collaboration with Dr. Nick Hawes, Senior Lecturer at University of Birmingham

that the position of the person is known and fixed in time. Additionally, we assume that a default probability density function for each spatial relation is given a-priori to the robot. An example of such density function in the case of the spatial relation *near* is shown in figure 5.5.

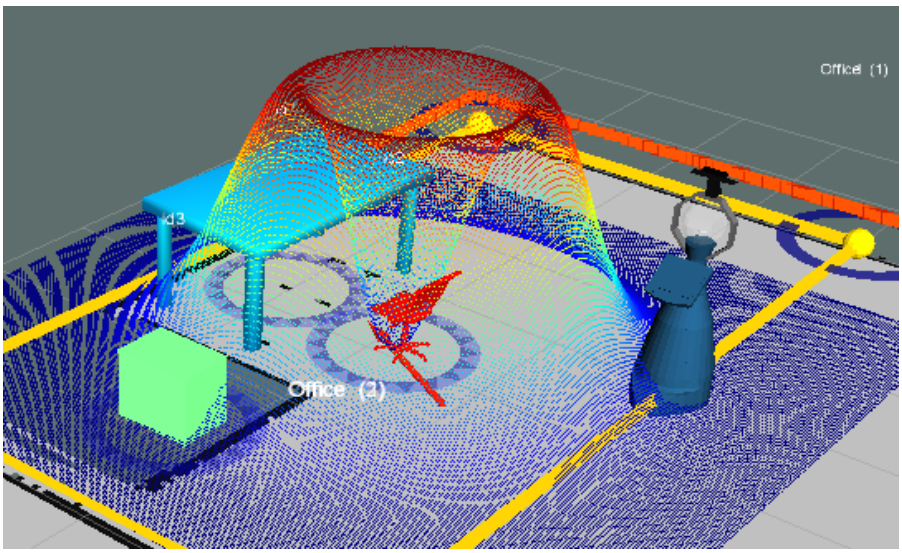


Figure 5.5. Possible default probability density function for the spatial relation *near*.

In this setting, we address the problem of learning which particular position (i.e., CDSR) the user prefers the robot to be in, while carrying out the given task. We make the assumption that this preference is defined, in part, by the arrangement of the objects in the environment. The preferences are represented as Gaussian Mixture Models (GMMs) that are modified, starting from a default model, based on the Boolean feedback provided by the user after the execution of each task. In addition, we address the problem of transferring the knowledge acquired in previous scenarios to new settings, in order to speed up the learning process. In fact, we hypothesize that such a transfer will allow a robot to locate more quickly the positions that satisfy a user for a given task. The transfer is carried out by first finding a mapping between the objects located in the two rooms. Then, by abstracting the components of the preference model through QSRs, we produce a qualitative description of their position with respect to the objects in the environment. Such qualitative descriptions are finally used to perform the transfer.

In the remainder of the section, we provide an overview of our approach describing all of our contributions thoroughly. Then, we present preliminary results obtained in a specific case scenario to better show the functioning of our approach.

5.2.1 Approach

We consider an autonomous robot that has been assigned a task to be carried out. We assume that such a task involves moving to a certain location expressed through a spatial relation. For example, a task of such kind could consist of delivering an object to a location or retrieving an object from a person. In fact, both tasks involve getting close to a certain target that will either load or unload the robot. In this scenario, our goal is to enable the user to teach the robot which position they prefer the robot to be in, in order to carry out the assigned task. In this section, we describe how the user preferences are modeled and how such models are refined through vocal human-robot interaction. Next, we will show how such preferences can be transferred from one target location to another, exploiting qualitative spatial reasoning.

Modeling User Preference

We model the preferences of the user as probability distributions. Such distributions are used to choose the final destination to be reached by sampling from them and removing the points occupied by obstacles. Starting from a default probability distribution $DPD(\mathbf{x}|r)$ assigned to each spatial reference r, the probability distributions are modified based on the Boolean feedback provided by the user. Such Booleans inform the robot whether the user likes or does not like the final position occupied. The Booleans are used to create a Gaussian Mixture Model that is combined with the given default model in order to obtain the model that will be used in the next task execution.

A Gaussian Mixture Model is a parametric probability density function represented as a weighted sum of Gaussian component densities. GMMs are commonly used as a parametric model for the probability distribution of continuous measurements. Mathematically, GMMs are defined as the weighted sums of M component Gaussian densities:

$$GMM(\mathbf{x}|\lambda) = \sum_{i=1}^{M} w_i g(\mathbf{x}|\mu_i, \mathbf{\Sigma}_i) \tag{5.1}$$

where $\mathbf{x}$ is a D-dimensional continuous-valued data vector, w_i are the mixture weights such that $\sum_{i=1}^{M} w_i = 1$, and $g(\mathbf{x}|\mu_i, \mathbf{\Sigma}_i)$, are the component Gaussian densities. Each component density is a D-variate Gaussian function with the following form:

$$g(\mathbf{x}|\mu_i, \mathbf{\Sigma}_i) = \frac{1}{(2\pi)^{\frac{D}{2}} |\Sigma_i|^{\frac{1}{2}}} \cdot exp\left(-\frac{1}{2}\frac{(\mathbf{x}-\mu_i)}{\sum_i(\mathbf{x}-\mu_i)}\right) \tag{5.2}$$

with mean vector μ_i and covariance matrix $\mathbf{\Sigma}_i$. The complete Gaussian mixture model is parameterized by the mean vectors, covariance matrices and mixture weights from all the component densities. These parameters are collectively represented by $\lambda = \{w_i, \mu_i, \mathbf{\Sigma}_i\}$.

In our scenario, each Boolean is associated with the position p_i assumed by the robot during the execution of a given task. Each of these Booleans is associated to a Gaussian

component density created with a mean vector $\mu_i = p_i$, a fixed covariance matrix, and a weight $w_i = 1/N$, where N is the number of Booleans gathered by the robot for a given task. Note that for each task and each parameter we store two specific Gaussian Mixture Models, one for good feedback and one for bad, represented by the parameters λ. Once the Gaussian Mixture Model is created based on the feedback of the user, it is then used in the next execution of the specific task. In fact, by overlaying these GMMs over the default model, we obtain a new distribution, that is sampled in order to locate the new target position to be reached. Such overlaying is mathematically carried out in the following way:

$$p(\mathbf{x}|r, \lambda_{good}, \lambda bad) = \mathit{DPD}(\mathbf{x}|r) \cdot [\mathit{GMM}_{good}(\mathbf{x}|\lambda_{good}) - \mathit{GMM}_{bad}(\mathbf{x}|\lambda_{bad})] \,. \quad (5.3)$$

Thus, over subsequent trials, the region preferred by the user gets iteratively defined.

Transferring User Preference Models

After showing how to model user preferences, in this part we tackle the problem of transferring such models to unknown users and rooms. We make the assumption that these models are partially defined by the arrangement of objects in the target room. To this end, we perform the transfer based on the previously built models in three steps. First, we choose the best previously learned model to transfer from. Then, we abstract the GMM components through qualitative spatial reasoning by analyzing the spatial relations between the object in the environment and the chosen model. Lastly, we create a qualitatively similar CDSR in the target room based on the result of the previous step.

Choosing the Best Model When the robot carries out a task in a novel environment or for a new user, two possible scenarios can be identified. The same task has been previously carried out in a similar scenario or the specific assigned task has never been considered before. In the latter case, the robot exploits the default model associated to the spatial relation used in the command to choose the destination position. From that default model, the robot builds a new preference model as described in the previous section. In the other scenario the robot transfers a similar and previously built model to the new setting, obtaining a new and transferred CDSR for the specific task to be carried out.

We assume that the position and shape of the transferred CDSR depend on personal user preferences and on the operational environment. Since user preferences are personal and therefore often unique, we rank the possible starting models based on the room type and on the objects within them. Specifically, starting from all the available models, we first discard all the ones that are associated to a different type of task. Then, we select the models associated to the same type of room. If there are multiple possible models, we choose the one associated with the room that has the higher number of objects in common with the target room, breaking ties randomly. If there are no models of such type, we use the default model as in the previous case.

Algorithm 4 GMM Abstraction

```
1:  procedure ABSTRACTMODEL(room, relations, feedback)
2:    // Initialize empty result list
3:      result ← []
4:    // Cycle over common objects and considered relations
5:      for commonObject in room do
6:          for relation in relations do
7:            // Convert feedback to a specific metric relative to object and relation
8:              metrics ← toSpatialRelaiton(commonObject.pose, relation, feedback)
9:            // Instantiate a new GMM
10:             classifier ← GMM()
11:           // Run the Expectation-Maximization Algorithm on metrics
12:             classifier.fit(metrics)
13:           // Predict the posterior probability
14:             predictions ← classifier.predictProbability(metrics)
15:           // Save the result if the relation is meaningful
16:             if predictions < THRESHOLD then
17:                 result.append([commonObject,relation,classifier])
18:             end if
19:         end for
20:     end for
21:     return result
22: end procedure
```

Abstracting GMM Components Once we have selected the preference model to transfer from, we analyze the associated feedback given by the user (Algorithm 4). In particular, we cycle through multiple spatial relations (i.e., distance and relative angle) and every object in common to the two rooms, in order to differentiate between meaningful and not meaningful relations objects and poses (line 5). To this end, we first convert the feedback poses to the dimension metric relative to the object and relation considered (line 8). For example, if we are considering the relation "near", we convert the feedback pose to a distance relative to the position of the object considered. Then, we use these metrics as input training data for an Expectation-Maximization Algorithm (line 12). Such algorithm is used to fit different GMMs to various dimensions of the feedback poses. The obtained GMMs are then used to predict the posterior probability of the data under each Gaussian in the model (line 14). Such predictions are finally used to differentiate between meaningful and not meaningful relations (line 16). At the end of this process, we obtain a list of meaningful relations, each associated to a specific object and a classifier (line 21).

Transferring Models At this point, we have an abstract description of the starting user preference model that needs to be transferred to the new target scenario. In order to do so, based on the obtained abstract model we create a qualitatively similar CDSR in the target room. This new CDSR is then used as the default model for the new task, allowing the user to change it expressing their preferences as previously described.

The creation of the qualitatively similar CDSR is a straight-forward process when there is an exact match between the object in the base and target rooms and their spatial layout. In fact, in this scenario the abstract model is reprojected in the target room to form the new default preference model. However, when the object types, or their spatial relations differ, the problem becomes more complicated. For example, let's consider the simple case of two offices that are one the mirror image of the other. If we directly apply the abstract model extracted from one room to the other, then the obtained CDSR appears in a wrong position.

In this work, we have taken into account only similar rooms and rooms that are mirror images of one another, leaving more complicated transformations for future work. For this scenario, we build a qualitative relation graph of the objects inside each room. In particular, this graph contains vertices that are labeled with the type of each object in the room. Instead, the edges are labelled with the qualitative spatial relation that relate the two objects connected. For our purpose we limit such relations to near/far and left/right/behind/front. With the graphs of the two rooms we determine the transformations that needs to be applied to the transferred model. This produces a CDSR in the correct location in the target room. The next section illustrates in detail an example to better clarify our approach.

5.2.2 Demonstrative Examples

In this section, we show how a mobile robot can learn CDSRs through the interaction with a user. In particular, we show how a simulated SCITOS G5 robot can learn a CDSR related to a deliver task involving a specific user. Additionally, we show how this model can be transferred to another room, in order to decrease the user interactions needed to learn a new model.

Simulation In order to simulate the robot and the interaction with the user, we adopted the MORSE simulator [132], which is able to simulate realistic small and large environments, as well as user interactions. We adopted standard ROS modules for navigation and localization, while relying on PNPs for modeling robot behaviors. In particular, Figure 5.6 shows the Petri Net Plan used to model the deliver behavior of the robot used in this setting. Specifically, in this plan, the robot first queries the internal database to retrieve the ID of the target to be reached. Next, it computes the best target pose based on the preference model of the user built so far. Finally, after moving to the final position the robot asks the user for feedback. The interactions with the users have been implemented with a vocal interface, composed

by the Google ASR coupled with a parser based on grammars and implemented in Prolog. As the majority of the other works presented in this thesis, we represented the linguistic semantics through frames that activate the transitions of the PNPs. Finally, the robot was provided with a semantic map represented through the SOMA ROS package developed in the STRANDS project[5].

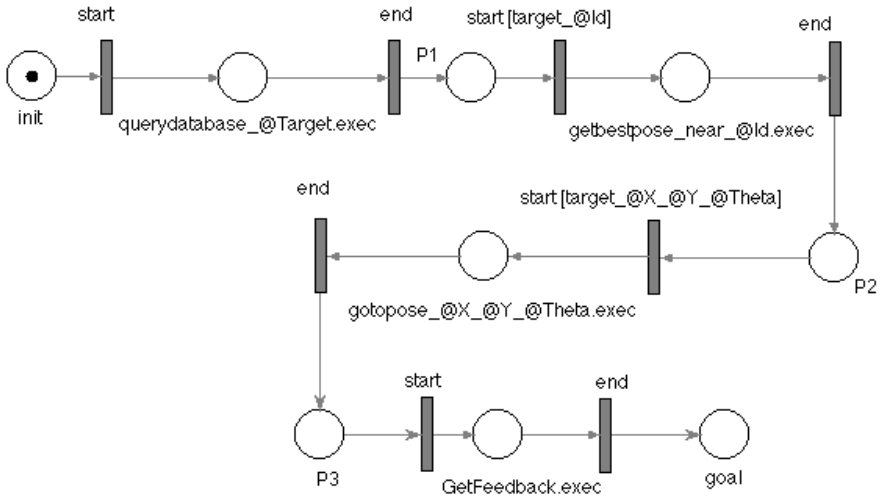


Figure 5.6. Petri Net Plan used to model the deliver behavior of the robot used in the example.

Example In order to demonstrate the functioning of our approach, we describe in detail the learning mechanism in a specific scenario. In particular, we consider the previously mentioned environment composed by two offices. Such offices hold a desk, a chair and a cabinet positioned as depicted in Figure 5.7. Note that, in this setting, the offices are each other mirror images.

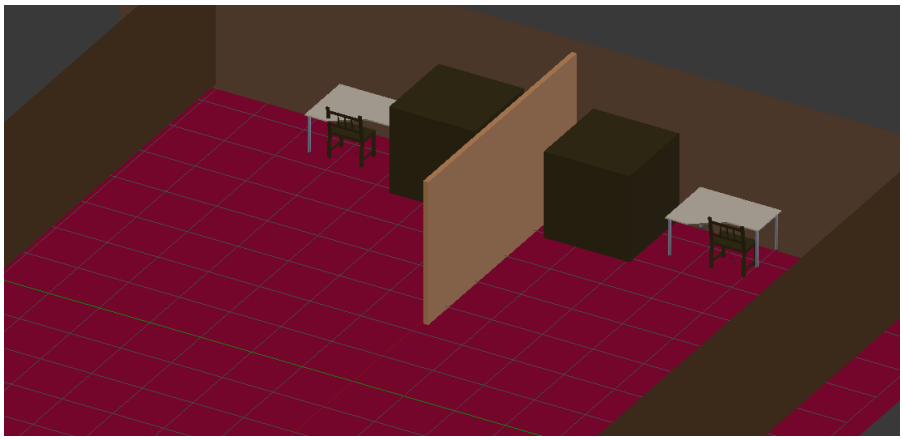


Figure 5.7. The simple environment composed by two offices and used in the example.

[5] https://github.com/strands-project/soma

In this environment, we first commanded multiple times the robot to deliver a book to the person located in the room to the right. After each task execution, we gave feedback to the robot for the location chosen to deliver the book. Figure 5.8 shows the changing spatial model over time as the robot receives different feedback. In particular, Figure 5.8a shows the initial near model, Figure 5.8b depicts the model after receiving two negative feedback, while Figure 5.8c illustrates the final preference model learned by the robot for this specific scenario. As requested, the user was able to teach the robot the preferred Context-Dependent Spatial Region.

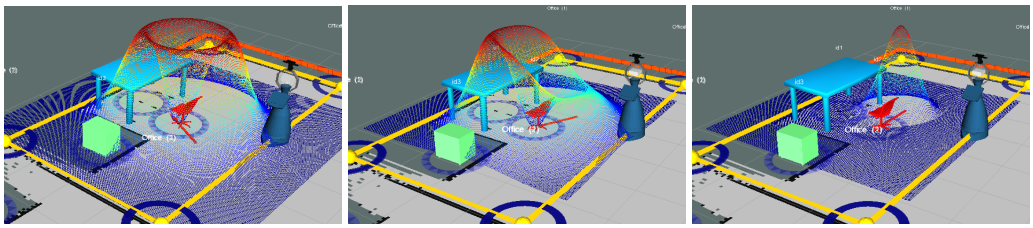


Figure 5.8. The changes of the spatial model in time as the robot receives different feedback during the example described.

Once the first spatial model was learned, we ordered the robot to deliver the book to the other office, which was never used before as a target. In this occasion, the robot was able to transfer the model learned from the first room in order to learn more quickly the new spatial model. In particular, the transfer has been performed by first clustering the feedback given by the user assuming a single good position. Next, this clustering has been used to describe the position in terms of spatial relations with respect to all the objects in the room, as described in the previous section. Having this qualitative description of the position, it has been transferred to the target room (discarding non-discriminative relations) to create a new preference-based default model. Note that in this process, the abstract model has been reflected in order to compensate the different positions of the objects in the two rooms. Figure 5.9a shows the transferred model in this particular example, Figure 5.9b depicts the model after the robot has gathered some negative feedback, while Figure 5.8c illustrates the final preference model learned by the robot for this specific room.

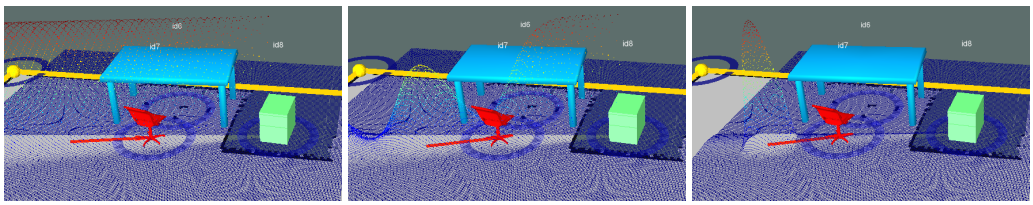


Figure 5.9. The changes of the transferred spatial model in time as the robot receives different feedback from the user.

5.2.3 Section Summary and Discussion

In this section, we considered an autonomous robot that needs to learn new Context-Dependent Spatial Regions defined by user preferences. In this setting, we have developed a mechanism that allows the user to provide Boolean feedback to the robot about the position assumed during the execution of the commanded task. Based on these feedback we have proposed an approach to learn CDSRs and transfer them to new scenarios to speed up the learning process. We successfully deployed our approach on a simulated mobile base robot able to carry out tasks assigned by users through natural language. Specifically, we have described a demonstrative example to better explain our algorithm in an office environment, pointing out possible advantages that can be achieved through our contribution.

As future work, we would like to quantitatively evaluate our approach and extend it in two ways. In fact, we would like to address the problem of how to identify differences between two rooms and how to express them in transformations that can be applied to the transferred model. Finally, we would like to enable the user to also express feedback in more complex ways beyond Booleans. To this end, we are thinking about allowing users to adopt natural language expressions to better describe their spatial preferences.

5.3 Chapter Summary and Discussion

Given the uniqueness of every human, in this chapter we have presented two approaches for learning user preferences. In particular, since users understand the world and talk about it in different ways, we have considered two different scenarios that involve user profiling. In particular, we have presented two approaches that allow a robot to learn new user referring expressions to objects and user context-dependent spatial region preferences. In the development of these approaches, we focused on the knowledge about users, so that the robots are able to adapt to the unique aspects of every human, rather than always assuming to be interacting with the same user. Additionally, we have again shown how human-robot collaboration can play a key role, enabling even non-expert users to change how a robot behaves around them through simple interactions. The designed approaches have been tested with two different robots, in different environments and with multiple users. In particular, these contributions have been experimentally evaluated on a simulated SCITOS G5 mobile base as well as on a Baxter manipulator robot. The results of the experiments show that, through simple vocal interactions between the robot and the user it is possible to enhance the agent capabilities, adapting it to the peculiarity of every user.

Chapter 6

Conclusion

In this thesis, we focused on robots that operate in environments populated by humans. While considering the current problems that withhold the deployment of such robots in our homes, we identified as one of the main issues to be tackled the inability of adapting to multiple scenarios encountered during operation. In fact, robots are currently unable to enter an unknown environment and autonomously learn its relevant aspects. Moreover, they are currently often unable to carry out the various tasks assigned by the users due to the differences of every human's need and the complexity of the environment they operate in. Finally, robots are still hardly accepted into our homes due to their lack of understanding of how each human talks and thinks differently about the world.

To overcome these limitations, multiple researchers have proposed to represent robotics deficiencies explicitly. With such representations robots can ask for human help when recognizing a scenario not manageable autonomously. In this thesis, we embraced this philosophy to allow robots to effectively adapt and operate in human-populated environments. Instead of relying on general world and user models that support robot operation, we hence exploit the interaction with the user to learn the specifics of every encountered scenario and adapt to it.

In particular, first we addressed the problem of acquiring specific knowledge about the environment a robot is deployed in. To this end, in Chapter 3 we presented an approach that exploits multi-modal user interaction to acquire a semantic map of the operational environment incrementally and on-line. Specifically, with the aid of a laser pointer and a vocal interface, the user is able to tour the robot around an indoor environment and teach it about the multiple objects and room found inside it. Through these interactions, the robot is able to acquire specific models and labels later used to support task execution. In particular, the semantic map created through this process has been used to support various forms of linguistic and spatial reasonings to carry out tasks given to different mobile robots. The proposed approach improves over existing methods by enabling robots to effectively enter unknown environments and incrementally build rich and accurate semantic maps that

support reasoning and task execution.

Once we were able to effectively build semantic maps, we addressed the problem of learning to carry out specific tasks described by the users as a combination of the available robotic primitives. In fact, through the natural language interaction with non-expert users, robots can adapt to the unique needs that each human has and cannot be foreseen at design time. To this end, in Chapter 4 we considered three open issues regarding robotic task teaching. In particular, we first described a mechanism to teach a single robot complex parametric tasks, which are instantiated during execution. By decoupling the problem of understanding the user description of the tasks from the problem of executing them, we presented a two-layered representation that effectively supports single robot task teaching. Next, we considered the problem of using previous knowledge about tasks to come up with suggestions to support the user during the teaching phase. In particular, by mapping our task graph representation into a tree representation, we showed how such trees can be mined to generate general tasks. These tasks are then used to provide autocompletion suggestions to the user during the teaching of new tasks. Finally, we considered the problem of teaching multiple robots tasks that require them to coordinate. In particular, by allowing robots to query one another's states, we showed how a user can be enabled to teach multiple robots how to sparsely-coordinate.

As a final topic, we considered the problem of learning how humans think and talk about the environment they are placed in while interacting with a robot. To this end, in Chapter 5 we first discussed an approach that enables a robot to learn new referring expressions to objects used by the user while interacting with it. While learning such new expressions the robot is able to enhance its grounding capabilities and inform the user about unknown terms used in the commands uttered. In this way, the user is also able to learn how to communicate with a possibly unknown robot. Finally, we considered the problem of learning the real meaning of different qualitative spatial relations used by the user in the commands given to the robot. To this end, we presented a preliminary study on this topic. Specifically, in this study, starting from a default model associated to each spatial relation, for each user, task, and target location we modified the default model to come up with a personalized one. By acquiring a boolean feedback from the user after each execution of a task, we combined the default model with a new feedback gaussian mixture model. Through this mechanism, the robot is able to adapt to the requests of the user by changing its model of the space.

The above mentioned contributions have been validated on multiple robots interacting with various users in different environments. Through such experiments we have demonstrated that by leveraging natural language human-robot interactions, our general approaches enable different kinds of robots to adapt to new environments, task requests and user peculiarities. From the evaluations carried out on the robots we have verified that multiple contributions presented in this thesis could be applied to robots and technologies already present on the market. Specifically, with the aid of an engineering and optimization process,

a semantic mapping approach similar to the one presented in Chapter 3 could be used to enhance the capabilities of telepresence robots that are currently entering the market [109]. For instance, users could be allowed to give a tour of their homes or offices to the robot, teaching it about relevant objects or rooms in the environment. These knowledge could then be used to simplify the teleoperation of a robot by allowing both the remote and the nearby user to command the robot using natural language. For example, users could just instruct the robot to reach a previously learnt room or object, without the need to teleoperate it. Additionally, some of the task teaching approaches presented in Chapter 4 could be used to specify tasks to be accomplished both by telepresence robots or by novel voice command devices that are being introduced on the market (e.g., Amazon Echo [133]). Indeed, such approaches could enable these technologies to perform novel tasks specified by the users as a combination of basic capabilities of the given platforms. Finally, some of the techniques presented in Chapter 5 could be applied to the previously mentioned agents, as well as to different vocal interfaces that are increasingly being deployed on computing devices (e.g., Siri [134], Google Now [135], or Cortana [134]). Such approaches could in fact allow a more advanced personalization of these technologies.

Despite these encouraging results, several areas for long-term research still exist. In particular, our semantic mapping approach should be extended to 3D in order to support more advanced spatial reasoning and task executions. In fact, for tasks that require a more fine grained precision, novel and more accurate perception capabilities should be integrated as they are developed by the research community. Leveraging such enhanced capabilities, robots should be enabled to adopt more proactive behaviors such as suggesting possible object categorizations or autonomously searching and discovering objects similar to the known ones. Additionally, natural language interactions should allow users to adopt a more flexible language while interacting with the robots. Consequently, the representations used to internally ground such commands should be extended accordingly. Finally, we have just started to scratch the surface of the research topic of user profiling through robot interaction. We believe that this research area will become increasingly more investigated as robots will start entering our homes.

Bibliography

[1] S. Rosenthal, J. Biswas, and M. Veloso, "An effective personal mobile robot agent through symbiotic human-robot interaction," in *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2010.

[2] T. Fong, C. Thorpe, and C. Baur, "Robot, asker of questions," *Robotics and Autonomous systems*, 2003.

[3] J. Hertzberg and A. Saffiotti, "Using semantic knowledge in robotics," *Robotics and Autonomous Systems*, 2008.

[4] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov, "Scalable object detection using deep neural networks," in *Computer Vision and Pattern Recognition (CVPR)*, 2014.

[5] S. Schiaffino and A. Amandi, "Intelligent user profiling," in *Artificial Intelligence An International Perspective*, 2009.

[6] J. Mccarthy, "Programs with common sense," in *Semantic Information Processing*, 1963.

[7] N. J. Nilsson, "Shakey the robot," tech. rep., SRI Artificial Intelligence Center, 1984.

[8] B. Kuipers and Y.-T. Byun, "A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations," *Journal of Robotics and Autonomous Systems*, 1991.

[9] H. Levesque and R. Reiter, "High-level robotic control: Beyond planning. a position paper," in *AIII 1998 Spring Symposium: Integrating Robotics Research: Taking the Next Big Leap*, 1998.

[10] S. Coradeschi and A. Saffiotti, "Symbiotic robotic systems: Humans, robots, and smart environments," *IEEE Intelligent Systems*, 2006.

[11] T. Winograd, "Procedures as a representation for data in a computer program for understanding natural language," tech. rep., Research and Teaching Output of the MIT Community, 1971.

[12] X. Zuo, N. Iwahashi, K. Funakoshi, M. Nakano, R. Taguchi, S. Matsuda, K. Sugiura, and N. Oka, "Detecting robot-directed speech by situated understanding in physical interaction," *Information and Media Technologies*, 2010.

[13] J. H. Connell, "Extensible grounding of speech for robot instruction," *Robots that Talk and Listen: Technology and Social Inpact*, 2014.

[14] T. Kollar, S. A. Tellex, and N. Roy, "A discriminative model for understanding natural language route directions," in *AAAI Fall Symposium Series*, 2010.

[15] G. Kruijff, H. Zender, P. Jensfelt, and H. Christensen, "Situated dialogue and spatial organization: What, where... and why," *International Journal of Advanced Robotic Systems*, 2007.

[16] E. Bastianelli, D. D. Bloisi, R. Capobianco, F. Cossu, G. Gemignani, L. Iocchi, and D. Nardi, "On-line semantic mapping," in *International Conference on Advanced Robotics (ICAR)*, 2013.

[17] D. Gouaillier, V. Hugel, P. Blazevic, C. Kilner, J. Monceaux, P. Lafourcade, B. Marnier, J. Serre, and B. Maisonnier, "Mechatronic design of nao humanoid," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[18] D. Sallé, M. Traonmilin, J. Canou, V. Dupourqué, *et al.*, "Using microsoft robotics studio for the design of generic robotics controllers: the robubox software," in *ICRA Workshop on Software Development and Integration in Robotics*, 2007.

[19] A. Bannat, J. Blume, J. T. Geiger, T. Rehrl, F. Wallhoff, C. Mayer, B. Radig, S. Sosnowski, and K. Kühnlenz, "A multimodal human-robot-dialog applying emotional feedbacks," in *Social Robotics*, 2010.

[20] R. Nisimura, T. Uchida, A. Lee, H. Saruwatari, K. Shikano, and Y. Matsumoto, "Aska: receptionist robot with speech dialogue system," in *International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[21] R. Stiefelhagen, H. K. Ekenel, C. Fugen, P. Gieselmann, H. Holzapfel, F. Kraft, K. Nickel, M. Voit, and A. Waibel, "Enabling multimodal human–robot interaction for the karlsruhe humanoid robot," *IEEE Transactions on Robotics*, 2007.

[22] M. E. Foster, M. Giuliani, A. Isard, C. Matheson, J. Oberlander, and A. Knoll, "Evaluating description and reference strategies in a cooperative human-robot dialogue system," in *International Joint Conference on Artificial intelligence (IJCAI)*, 2009.

[23] S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, A. G. Banerjee, S. J. Teller, and N. Roy, "Understanding natural language commands for robotic navigation and

mobile manipulation.," in *Association for the Advancement of Artificial Intelligence (AAAI)*, 2011.

[24] P. E. Rybski, K. Yoon, J. Stolarz, and M. M. Veloso, "Interactive robot task training through dialog and demonstration," in *ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2007.

[25] G. Gemignani, E. Bastianelli, and D. Nardi, "Teaching robots parametrized executable plans through spoken interaction," in *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2015.

[26] G. Gemignani, S. D. Klee, M. Veloso, and D. Nardi, "On task recognition and generalization in long-term robot teaching," in *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2015.

[27] S. Lemaignan and R. Alami, "Talking to my robot: From knowledge grounding to dialogue processing," in *ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2013.

[28] T. Kollar, S. Tellex, D. Roy, and N. Roy, "Toward understanding natural language directions," in *ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2010.

[29] M. MacMahon, B. Stankiewicz, and B. Kuipers, "Walk the talk: connecting language, knowledge, and action in route instructions," in *National Conference on Artificial intelligence*, 2006.

[30] S. Chernova, J. Orkin, and C. Breazeal, "Crowdsourcing hri through online multiplayer games," in *AAAI Fall Symposium Series*, 2010.

[31] J. Dzifcak, M. Scheutz, C. Baral, and P. Schermerhorn, "What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[32] T. Kollar, V. Perera, D. Nardi, and M. Veloso, "Learning environmental knowledge from task-based-robot dialog," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[33] C. J. Fillmore, "Frames and the semantics of understanding," *Quaderni di Semantica*, 1985.

[34] A. Nüchter, H. Surmann, K. Lingemann, and J. Hertzberg, "Semantic scene analysis of scanned 3d indoor environments," in *International Fall Workshop on Vision, Modeling, and Visualization*, 2003.

[35] O. Grau, "A scene analysis system for the generation of 3-d models," in *International Conference on Recent Advances in 3-D Digital Imaging and Modeling*, 1997.

[36] C. Galindo, A. Saffiotti, S. Coradeschi, P. Buschka, J. Fernández-Madrigal, and J. González, "Multi-hierarchical semantic maps for mobile robotics," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2005.

[37] S. Coradeschi and A. Saffiotti, "An introduction to the anchoring problem," *Robotics and Autonomous Systems*, 2003.

[38] S. Coradeschi and A. Saffiotti, "Anchoring symbols to sensor data: preliminary report," in *National Conference on Artificial Intelligence and Conference on Innovative Applications of Artificial Intelligence*, 2000.

[39] N. Goerke and S. Braun, "Building semantic annotated maps by mobile robots," in *Conference Towards Autonomous Robotic Systems*, 2009.

[40] E. Brunskill, T. Kollar, and N. Roy, "Topological mapping using spectral clustering and classification," in *IEEE/RSJ Conference on Robots and Systems (IROS)*, 2007.

[41] S. Friedman, H. Pasula, and D. Fox, "Voronoi random fields: Extracting the topological structure of indoor environments via place labeling," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.

[42] J. Wu, H. I. Christenseny, and J. M. Rehg, "Visual place categorization: Problem, dataset, and algorithm," in *IEEE/RSJ Conference on Robots and Systems (IROS)*, 2009.

[43] O. M. Mozos, H. Mizutani, R. Kurazume, and T. Hasegawa, "Categorization of indoor places using the kinect sensor," *Sensors*, 2012.

[44] A. Hermans, G. Floros, and B. Leibe, "Dense 3d semantic mapping of indoor scenes from rgb-d images," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[45] M. Gunther, T. Wiemann, S. Albrecht, and J. Hertzberg, "Building semantic object maps from sparse and noisy 3d data," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[46] H. Zender, O. Martínez Mozos, P. Jensfelt, G. Kruijff, and W. Burgard, "Conceptual spatial representations for indoor mobile robots," *Robotics and Autonomous Systems*, 2008.

[47] A. Pronobis and P. Jensfelt, "Large-scale semantic mapping and reasoning with heterogeneous modalities," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[48] J. Peltason, F. H. Siepmann, T. P. Spexard, B. Wrede, M. Hanheide, and E. A. Topp, "Mixed-initiative in human augmented mapping," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[49] C. Nieto-Granda, J. G. R. III, A. J. B. Trevor, and H. I. Christensen, "Semantic map partitioning in indoor environments using regional analysis," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[50] G. Kruijff, H. Zender, P. Jensfelt, and H. Christensen, "Clarification dialogues in human-augmented mapping," in *ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2006.

[51] S. Hemachandra, T. Kollar, N. Roy, and S. Teller, "Following and interpreting narrated guided tours," in *IEEE International Conference on Robotics and Automation*, 2011.

[52] G. Randelli, T. M. Bonanni, L. Iocchi, and D. Nardi, "Knowledge acquisition through human–robot multimodal interaction," *Intelligent Service Robotics*, 2013.

[53] C. Kemp, C. Anderson, H. Nguyen, A. Trevor, and Z. Xu, "A point-and-click interface for the real world: laser designation of objects for mobile manipulation," in *International Conference on Human Robot Interaction (HRI)*, 2008.

[54] E. Gat, "ESL: A language for supporting robust plan execution in embedded autonomous agents," in *IEEE Aerospace Conference*, 1997.

[55] R. Simmons and D. Apfelbaum, "A task description language for robot control," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 1998.

[56] R. J. Firby, *Adaptive Execution in Complex Dynamic Worlds*. PhD thesis, Yale, 1989.

[57] K. Konolige, "COLBERT: A language for reactive control in saphira," *Lecture Notes in Computer Science*, 1997.

[58] K. Konolige, K. Myers, E. Ruspini, and A. Saffiotti, "The saphira architecture: A design for autonomy," *Journal of experimental & theoretical artificial intelligence*, 1997.

[59] M. Loetzsch, M. Risler, and M. Jungel, "Xabsl-a pragmatic approach to behavior engineering," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2006.

[60] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE Transactions on Robotics*, 2009.

[61] E. Gat, "Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots," in *National Conference on Artificial Intelligence*, 1992.

[62] A. S. Rao and M. P. Georgeff, "Modeling rational agents within a bdi-architecture," *KR*, 1991.

[63] M. P. Georgeff and A. L. Lansky, "Procedural knowledge," *Proceedings of the IEEE*, 1986.

[64] M. Tambe, "Towards flexible teamwork," *Journal of Artificial Intelligence Research (JAIR)*, 1997.

[65] G. A. Kaminka and I. Frenkel, "Flexible teamwork in behavior-based robots," in *National Conference on Artificial Intelligence*, 2005.

[66] P. R. Cohen and H. J. Levesque, "Teamwork," *Special Issue on Cognitive Science and Artificial Intelligence.*, 1991.

[67] J. Ferber, *Multi-Agent Systems*. Addison-Wesley, 1999.

[68] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, 1989.

[69] N. Akharware and M. Miee, "Pipe2: Platform independent petri net editor." `http://pipe2.sourceforge.net`, 2005. Accessed: 07-09-2015.

[70] A. Zimmermann and J. Freiheit, "Timenet ms-an integrated modeling and performance evaluation tool for manufacturing systems," in *IEEE International Conference on Systems, Man, and Cybernetics*, 1998.

[71] N. Viswanadham and Y. Narahari, "Performance modeling of automated manufacturing systems," *NASA STI/Recon Technical Report A*, 1992.

[72] F. Y. Wang, K. J. Kyriakopoulos, A. Tsolkas, and G. N. Saridis, "A petri-net coordination model for an intelligent mobile robot," *IEEE Transactions on Systems, Man and Cybernetics*, 1991.

[73] J. R. Celaya, A. A. Desrochers, , and R. J. Graves, "Modeling and analysis of multi-agent systems using petri nets," in *IEEE International Conference on Systems, Man and Cybernetics (ISIC)*, 2007.

[74] R. S. Cost, Y. Chen, T. Finin, Y. Labrou, and Y. Peng, "Using colored petri nets for conversation modeling," in *Issues in Agent Communication*, 2000.

[75] G. Gutnik and G. A. Kaminka, "Representing conversations for scalable overhearing," *Journal of Artificial Intelligence Research*, 2006.

[76] D. Poutakidis, L. Padgham, and M. Winikoff, "Debugging multi-agent systems using design artifacts: The case of interaction protocols," in *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2002.

[77] W. Sheng and Q. Yang, "Peer-to-peer multi-robot coordination algorithms: petri net based analysis and design," *IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, 2005.

[78] D. Xu, R. Volz, T. Loerger, and J. Yen, "Modeling and verifying multi-agent behaviors using predicate/transition nets," in *International Conference on Software Engineering and Knowledge Engineering*, 2002.

[79] C. Maier and D. Moldt, "Object colored petri nets-a formal technique for object oriented modeling," in *Concurrent object-oriented programming and petri nets*, 2001.

[80] C.-H. Kuo and I.-H. Lin, "Modeling and control of autonomous soccer robots using distributed agent oriented petri nets," in *IEEE International Conference on Systems, Man and Cybernetics*, 2006.

[81] D. Lima and P. Milutinovic, "Petri net models of robotic tasks," in *International Conference On Robotics And Automation (ICRA)*, 2002.

[82] K. J., R. K. Pretty, and R. G. Gosine, "Coordinated execution of tasks in a multiagent environment," *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 2003.

[83] R. E. Fikes and N. J. Nilsson, "STRIPS: a new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, 1972.

[84] V. Giordano, P. Ballal, F. Lewis, B. Turchiano, and J. B. Zhang, "Supervisory control of mobile sensor networks: math formulation, simulation, and implementation," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 2006.

[85] Y. T. Kotb, S. S. Beauchemin, and J. L. Barron, "Petri net-based cooperation in multi-agent systems," in *Canadian Conference on Computer and Robot Vision (CRV)*, 2007.

[86] V. Ziparo, L. Iocchi, P. Lima, D. Nardi, and P. Palamara, "Petri Net Plans - A framework for collaboration and coordination in multi-robot systems," *Autonomous Agents and Multi-Agent Systems*, 2011.

[87] C. Matuszek, E. Herbst, L. Zettlemoyer, and D. Fox, "Learning to parse natural language commands to a robot control system," in *Experimental Robotics*, 2013.

[88] D. L. Chen and R. J. Mooney, "Learning to interpret natural language navigation instructions from observations.," in *Association for the Advancement of Artificial Intelligence (AAAI)*, 2011.

[89] M. N. Nicolescu and M. J. Matarić, "Natural methods for robot task learning: Instructive demonstrations, generalization and practice," in *International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2003.

[90] P. E. Rybski, J. Stolarz, K. Yoon, and M. Veloso, "Using dialog and human observations to dictate tasks to a learning robot assistant," *Intelligent Service Robotics*, 2008.

[91] A. Weitzenfeld, C. Ramos, and P. F. Dominey, "Coaching robots to play soccer via spoken-language," in *RoboCup 2008: Robot Soccer World Cup XII*, 2009.

[92] Ç. Meriçli, S. D. Klee, J. Paparian, and M. Veloso, "An interactive approach for situated task specification through verbal instructions," in *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2014.

[93] J. Connell, E. Marcheret, S. Pankanti, M. Kudoh, and R. Nishiyama, "An extensible language interfacefor robot manipulation," in *Artificial General Intelligence*, 2012.

[94] L. She, S. Yang, Y. Cheng, Y. Jia, J. Y. Chai, and N. Xi, "Back to the blocks world: Learning new actions through situated human-robot dialogue," in *Annual Meeting of the Special Interest Group on Discourse and Dialogue*, 2014.

[95] F. S. Melo and M. Veloso, "Decentralized mdps with sparse interactions," *Artificial Intelligence*, 2011.

[96] S. Schiffer, A. Ferrein, and G. Lakemeyer, "Reasoning with qualitative positional information for domestic domains in the situation calculus," *Journal of Intelligent & Robotic Systems*, 2012.

[97] K. Sjöö, A. Aydemir, and P. Jensfelt, "Topological spatial relations for active visual search," *Robotics and Autonomous Systems*, 2012.

[98] L. Kunze, K. K. Doreswamy, and N. Hawes, "Using qualitative spatial relations for indirect object search," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[99] M. Klenk, N. Hawes, and K. Lockwood, "Representing and reasoning about spatial regions defined by context," in *AAAI Fall 2011 Symposium on Advances in Cognitive Systems*, 2011.

[100] M. McClelland, M. Campbell, and T. Estlin, "Qualitative relational mapping for planetary rovers," in *Association for the Advancement of Artificial Intelligence (AAAI)*, 2013.

[101] K. Zimmermann and C. Freksa, "Qualitative spatial reasoning using orientation, distance, and path knowledge," *Applied Intelligence*, 1996.

[102] A. Loutfi, S. Coradeschi, M. Daoutis, and J. Melchert, "Using knowledge representation for perceptual anchoring in a robotic system," *International Journal on Artificial Intelligence Tools*, 2008.

[103] D. Hernández, *Qualitative representation of spatial knowledge*. 1994.

[104] A. U. Frank, "Qualitative spatial reasoning about distances and directions in geographic space," *Journal of Visual Languages & Computing*, 1992.

[105] J. Renz and D. Mitra, "Qualitative direction calculi with arbitrary granularity," in *Trends in Artificial Intelligence, 8th Pacific Rim International Conference on Artificial Intelligence (PRICAI)*, 2004.

[106] J. F. Allen, "Maintaining knowledge about temporal intervals," *Communications of the ACM*, 1983.

[107] P. Balbiani, J.-F. Condotta, and L. F. del Cerro, "A new tractable subclass of the rectangle algebra," in *nternational Joint Conference on Artificial Intelligence (IJCAI)*, 1999.

[108] D. Hernández, "Diagrammtical aspects of qualitative representations of space," *AAAI Spring Symposium on Reasoning with Diagrammatic Representations*, 1992.

[109] A. Kristoffersson, S. Coradeschi, and A. Loutfi, "A review of mobile robotic telepresence," *Advances in Human-Computer Interaction*, 2013.

[110] R. Capobianco, J. Serafin, J. Dichtl, G. Grisetti, L. Iocchi, and D. Nardi, "A proposal for semantic map representation and evaluation," in *Proceedings of the 7th european conference on mobile robots (ecmr)*, 2015.

[111] J. DeNero and D. Klein, "Teaching introductory artificial intelligence with pac-man," in *Proceedings of the Symposium on Educational Advances in Artificial Intelligence*, 2010.

[112] T. S. Standley, "Finding optimal solutions to cooperative pathfinding problems," in *AAAI*, 2010.

[113] A. Diosi, G. Taylor, and L. Kleeman, "Interactive slam using laser and advanced sonar," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2005.

[114] M. Schwarz, J. Stückler, and S. Behnke, "Mobile teleoperation interfaces with adjustable autonomy for personal service robots," in *Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction*, 2014.

[115] O. Martínez Mozos and W. Burgard, "Supervised learning of topological maps using semantic information extracted from range data," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2006.

[116] G. Grisetti, R. Kuemmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based SLAM," *IEEE Intelligent Transportation Systems Magazine*, 2010.

[117] R. Kuemmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g2o: A general framework for graph optimization," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[118] D. Bloisi and L. Iocchi, "Independent multimodal background subtraction," in *International Conference on Computational Modeling of Objects Presented in Images: Fundamentals, Methods and Applications*, 2012.

[119] T. M. Bonanni, A. Pennisi, D. D. Bloisi, L. Iocchi, and D. Nardi, "Human-robot collaboration for semantic labeling of the environment," in *Workshop on Semantic Perception, Mapping and Exploration (SPME)*, 2013.

[120] R. Capobianco, G. Gemignani, D. Bloisi, D. Nardi, and L. Iocchi, "Automatic extraction of structural representations of environments," in *Intelligent Autonomous Systems 13*, 2016.

[121] L. Najman, M. Couprie, and G. Bertrand, "Watersheds, mosaics, and the emergence paradigm," *Discrete Applied Mathematics*, 2005.

[122] A. U. Frank, "Qualitative spatial reasoning with cardinal directions," in *Austrian Conference on Artificial Intelligence*, 1991.

[123] L. Aiello, E. Bastianelli, L. Iocchi, D. Nardi, V. Perera, and G. Randelli, "Knowledgeable talking robots," in *Artificial General Intelligence*, 2013.

[124] B. J. Thomas and O. C. Jenkins, "Roboframenet: Verb-centric semantics for actions in robot middleware," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[125] R. Basili, E. Bastianelli, G. Castellucci, D. Nardi, and V. Perera, "Kernel-based discriminative re-ranking for spoken command understanding in hri.," in *Conference of the Italian Association for Artificial Intelligence*, 2013.

[126] M. Popović and H. Ney, "Word error rates: Decomposition over pos classes and applications for error analysis," in *Proceedings of the Second Workshop on Statistical Machine Translation*, 2007.

[127] D. Croce, G. Castellucci, and E. Bastianelli, "Structured learning for semantic role labeling," *Intelligenza Artificiale*, 2012.

[128] C. Jiang, F. Coenen, and M. Zito, "A survey of frequent subgraph mining algorithms," *The Knowledge Engineering Review*, 2013.

[129] J. Biswas and M. M. Veloso, "Localization and navigation of the cobots over long-term deployments," *The International Journal of Robotics Research*, 2013.

[130] M. Ghallab, C. Knoblock, D. Wilkins, A. Barrett, D. Christianson, M. Friedman, C. Kwok, K. Golden, S. Penberthy, D. E. Smith, *et al.*, "Pddl-the planning domain definition language," tech. rep., Yale Center for Computational Vision and Control, 1998.

[131] P. Vogt, "The physical symbol grounding problem," *Cognitive Systems Research*, 2002.

[132] S. Lemaignan, G. Echeverria, M. Karg, J. Mainprice, A. Kirsch, and R. Alami, "Human-robot interaction in the morse simulator," in *ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2012.

[133] Amazon, "Amazon echo web page." `http://www.amazon.com/Amazon-SK705DI-Echo/dp/B00X4WHP5E`, 2015. Accessed: 07-09-2015.

[134] J. Aron, "How innovative is apple's new voice assistant, siri?," *New Scientist*, 2011.

[135] J. Schalkwyk, D. Beeferman, F. Beaufays, B. Byrne, C. Chelba, M. Cohen, M. Kamvar, and B. Strope, ""your word is my command": Google search by voice: A case study," in *Advances in Speech Recognition*, 2010.