Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Informatica

Design and implementation of a system for mutual knowledge among cognition-enabled robots

Laureando:
Michele Palmia

Relatore:
Ch.mo Prof. Enrico Pagello

Anno accademico 2013/2014

Abstract

The progressive integration of robots in everyday activities is raising the need for autonomous machines to reason about their actions, the environment and the objects around them. Only by acquiring these capabilities, they might improve in understanding vaguely described tasks, in reacting to changes in their surroundings and in many more areas. The KnowRob knowledge processing system is specifically designed to bring these competences to autonomous robots, helping them to acquire, reason about and meaningfully store knowledge. This system, the most comprehensive of its kind, is capable of greatly enhancing the possibilities of autonomous robots, but research about it has only been focused on single machines independently executing specific tasks. Sharing information and data about themselves with other knowledge-enabled machines in the environment, however, would prove very useful for robots. With the help of modules leveraging on the knowledge about their teammates, they could easily coordinate and collaborate with other machines.

This work presents a framework for enhancing the KnowRob system with mutual knowledge acquisition and reasoning among knowledge-enabled robots. The first chapter presents an overview of KnowRob, together with RoboEarth and Rapyuta, respectively a world wide web for robots and a platform-as-a-service system for offloading robot computation to the cloud. While being projects with very different scopes and aims, they still all collaborate in the task of allowing efficient knowledge-enabled robots to work and to exchange knowledge. In the second chapter we give an overview of the experimental setup we used, especially describing a robot model we developed for our simulations. We finally present, in the third chapter, the architecture we propose together with a simple prototype.

Sommario

Il progressivo coinvolgimento di robot autonomi in attivitá quotidiane sta accrescendo la necessitá che tali macchine possano ragionare riguardo le loro azioni, l'ambiente e gli oggetti attorno ad esse. Capire indicazioni vage o poco accurate, e reagire prontamente a modifiche dell'ambiente circostante sono solo alcune delle innumerevoli possibilitá che tali abilitá potrebbero rendere immediate. Il sistema di elaborazione della conoscenza KnowRob é stato progettato per permettere a robot autonomi di acquisire e immagazzinare conoscenza, e poter ragionare su di essa. Questo sistema é in grado di migliorare notevolmente le abilitá di tali robot, ma la letteratura al riguardo si é unicamente focalizzata sul migliorare le abilitá individuali di singole macchine. Permettere ai robot di condividere dati e informazioni riguardo se stessi con i loro simili presenti nello stesso ambiente, potrebbe peró rivelarsi estremamente utile. Con l'aiuto di moduli di controllo programmati appositamente per utilizzare la conoscenza acquisita, puó essere facilmente reso possibile a diversi robot autonomi coordinarsi e collaborare.

Questa tesi presenta un sistema che permette a robot autonomi equipaggiati con il sistema KnowRob di sfruttare la mutua conoscenza per eseguire azioni in maniera coordinata. Il primo capitolo presenta una introduzione a KnowRob, insieme ad una breve panoramica su RoboEarth e Rapyuta, rispettivamente un world wide web per robot e un sistema platform-as-a-service per alleggerire i carichi computazionali piú complessi dai robot, spostandoli in sistemi remoti. Nonostante questi ultimi due progetti abbiano scopi diversi da KnowRob, essi mirano comunque alla creazione di un efficiente sistema attraverso il quale i robot possano gestire al meglio la conoscenza. Il secondo capitolo offre una panoramica dell'impostazione sperimentale utilizzata; in particolare, si sofferma su un modello di robot che abbiamo creato per le nostre simulazioni. Il terzo e ultimo capitolo, infine, suggerisce una proposta di architettura per il sistema introdotto nel precedente paragrafo, insieme ad un prototipo dello stesso.

Contents

List of Figures

Algorithms and Listings

Chapter 1

Introduction

Autonomous robots are expected, in the very near future, to get more independent in managing their tasks, to become better and faster in understanding the environment around them and to improve in the challenge of taking informed decisions about their actions while performing them [Bischoff and Guhl, 2009, Bicchi et al., 2007]. Moreover, these machines will soon be requested to collaborate with non-technical staff, thus needing to interpret vaguely described tasks and to translate abstract solutions to problems into a real plan. Satisfying these requirements would mean for the robots to be able to fully, precisely and appropriately parametrize control programs given very little explicit information.

Robots cannot be expected to store all the necessary knowledge and data in order to perform these tasks at all times. Moreover, they will soon be requested to be able to switch between different tasks very quickly, making it completely unfeasible to store all the potentially useful knowledge at any given moment. They finally will not be allowed to ignore the presence of other machines in the environment they are working in, and should even be able to leverage on their presence in certain cases.

This work tackles the challenge of mutual knowledge exchange among autonomous robots for coordination and collaboration purposes. In particular, these aspects are studied in the context of complex knowledge reasoning and representation systems, whose aim is to solve many of the problems presented in the previous paragraphs.

In this chapter we will present three projects that address three different areas discussed in the previous paragraphs. The first introductory section is completely dedicated to KnowRob, the knowledge engine that forms the basis for our work. The next two sections respectively deal with RoboEarth, a world wide web for robots, and Rapyuta, a Platform-as-a-service framework for offloading heavy computations from robots.

Please note that in the remainder of this thesis we will extensively use the basic concepts found in the Robot operating system[1], in particular in its communication and build systems. If not comfortable with ROS topics and services, for example, the reader might be interested in getting a general ROS overview before continuing to read. Many other basic computer engineering's subjects are used through the paper without previous definition.

When citing the KnowRob system, we will refer to what described in section 1.1. On the other hand, when talking about the KnowRob framework (or the RoboEarth framework), we will refer to the whole set of researches that has been carried on by the RoboEarth consortium[2] and by other scientists, specifically at Technische Universität München[3] and at Universität Bremen[4] within the KnowRob and RoboHow European programmes. While being very diversified, the projects born from these programmes all collaborate to the implementation of autonomous intelligent robots. The three packages discussed in this chapter form the core of this infrastructure. Most of the illustrations about these projects found in this work come from articles and papers published by the cited institutions.

1.1 KnowRob

KnowRob's creators expect the ability *to infer what is meant from what is described* to soon become a strong prerequisite for taskable robotic agents [Tenorth and Beetz, 2013b]. Human communication is indeed a very laconic and often inaccurate way to express tasks, mostly because we usually expect our communication partners to have some commonsense knowledge. We also usually talk with people sharing a knowledge base with us: if not, we can still teach them the basic concepts they need to understand us. Finally, we expect our partners to be able to reason about a given task, eventually solving any problem that might arise or asking for help before finally giving up. Robotic agents behaving this way will thus require both a strong body of knowledge and a knowledge processing mechanism.

In order for control programs to respect these requirements, they have to be written in a *knowledge-enabled* manner, querying the knowledge base

[1]ROS - http://www.ros.org
[2]http://www.roboearth.org/collaborators
[3]https://ias.cs.tum.edu/
[4]http://ai.uni-bremen.de/research/ias

every time a decision has to be taken. While keeping a known structure for the program and using known data structures for queries and responses, robot actions will thus be bound to the robot's own knowledge.

Having low level routines directly interfacing with the knowledge base, it is possible to create plans that automatically adapt to the changing environment. While the routine itself is clearly defined and doesn't change, the knowledge of the robot about its surroundings and the objects in it might change: this approach allows the robot to adapt to its environment without the need for external support. At the same time, this approach also helps decoupling the routines from a specific environment, allowing strong code reuse. With small modifications to object properties and inference rules, the same code potentially works on similar tasks.

KnowRob is a knowledge processing system that allows to efficiently acquire data from many different sources. It allows knowledge storing and reasoning leaving to programmers the possibility to include custom modules for enhancing specific parts of the system. Strong coupling between the reasoning system and sensors means inference can be carried on over information that is always up to date. KnowRob provides to the robot's modules a knowledge base containing a complex semantic taxonomy reflecting real relations between objects that can be customized to allow specific tasks or functionality. Interface with perception modules also allow to include an environment representation in the knowledge base, allowing the robot to reason about its surroundings.

KnowRob is part of a wider spanning project called Cognitive Robot Abstract Machine (abbreviated CRAM), "a software toolbox for the design, the implementation, and the deployment of cognition-enabled autonomous robots performing everyday manipulation activities" [Beetz et al., 2010a]. It is divided in two main parts: the CRAM Plan Language (also called CPL) and KnowRob itself. As this work focuses on the core infrastructure, we decided not to dig deeper into CPL, leaving its analysis to future works.

In the following subsections, the various parts of the KnowRob system will be analyzed in more detail. We will first describe the design rules that the team behind this project took as paradigm. We will then proceed discussing about the formalization used to represent the knowledge inside robots and discuss the reasoning engine and the way it interfaces with the outer world. We will finally discuss the various kinds of entities on which robots can reason using the proposed system, and how abstract actions can be transformed in actual robot movements.

1.1.1 Design and paradigms

When developing control modules that need to interface with KnowRob, programmers use a very traditional structures for the software. The modules should be designed so that while the main structure of the programs remain exactly the same, the output changes with different settings of the environment. Let us consider an example for explaining this concept very simply. A robot is asked to put the utensils it finds on a working table at their respective locations. A simple routine would first establish which objects are on the table then, for each object, would establish its likely destination and finally move the robot repeatedly from the table to each object's inferred location, in order to store it where it belongs.

In this example, the first two steps are based on the robot's knowledge base. The answers to the decision problems seen in the previous paragraph are based on the robot perceptions and knowledge about the environment wherein it moves. While the structure of the program is simple, linear and static, the output will depend on the specific setting the robot will find around itself.

The fact that programmers designing a plan know the structure and type of the queries to be used brings a few advantages. First, queries can be optimized exactly as it would be done for SQL queries; second, the inference method to be used can be tailored to the needs of each specific query, resulting in more accurate results. Last, the internal representation for the knowledge base could be optimized for faster answering specific recurrent queries. Details about each of these subjects will be provided in the next sections.

Common properties for knowledge processing systems

While describing KnowRob, Tenorth and Beetz [2013b] outline the following nine properties, considered important for a knowledge processing system to have in order to become a useful resource for an autonomous robot. We propose them here as a way to help the reader understand the properties needed by such a system to actually be effective and efficient.

1. *It must provide a tell-ask service in which the robot can record experience and beliefs, and from which it can query information that was inferred from the stored knowledge.* In order to account for the non static nature of the robot's knowledge, the system must support updates. Moreover, it has to be thought as an interactive service that the robot can consult whenever needed.

2. *It must operate effectively and efficiently as part of the robot's control system.* On the one hand, this means data from the various robot's components should be easily integrated with abstract information from the knowledge base. On the other hand, it most importantly means that answers to queries should be returned fast enough so as to avoid slowing down the robot actions. As we will see in subsequent sections, this often means some inference techniques will be realistically available only in some cases, and expressiveness will often have to be sacrificed in favor of practical usability.

3. *It must provide the difference between the information provided by a natural task specifications and the knowledge a robot needs for successfully carrying out a task.*

4. *It has to provide an encyclopedic knowledge base that defines and specifies an appropriate conceptualization of the information needed for autonomous robot control.* While some encyclopedic knowledge bases have been designed, as in Cyc [Lenat, 1995] and SUMO [Niles and Pease, 2001], they usually miss important information needed by common robot manipulation tasks. As an example, the authors notice how upper ontologies as the ones cited just before correctly specify eggs as products of birds and fishes , but lack manipulation information such as the fact that eggs breaks easily. Encyclopedic knowledge bases for robots should be very rich in the way they represent actions, events, situations, and in general event-related information.

5. *It must provide the robot with self-knowledge.* Given an action specification, the robot should be able to determine whether it is able to carry it on or if it is missing required hardware or software modules. Within the KnowRob framework, Kunze et al. [2011] proposed a method to combine robot's kinematic structures with semantic annotations to make them available to abstract reasoning. This robot representation, called Semantic robot description language (SRDL), is used within the KnowRob framework to ensure robots can potentially complete an action recipe before starting to work on it.

6. *It must make the robot knowledgeable about its actions.* That is, it should be able to predict the outcome of an action, and state its prerequisites and outcomes. Tenorth and Beetz [2012] analyze this property in its details and explain how the KnowRob system implements this requirement.

7. *It must make the robot capable of using its control and perception systems as knowledge sources.* Most of the time, the information needed by the inference process is already ready and usable either as part of the control system for the robot or from the robot's sensors. The knowledge base needs to act as a parasite of both these systems and to dynamically update its data as new information becomes available.

8. *It needs to provide methods for (semi-)automatically acquiring and integrating knowledge from different sources.* It is impossible to think of a robot embodying all the possible knowledge about objects, actions and places. As this kind of systems scale up, it simply becomes impossible to store all the information needed by the robot inside a default knowledge base. It is therefore necessary to equip machines with methods to import new knowledge. The RoboEarth sytem [Waibel et al., 2011], discussed in the next section, is one such example. Other possibilities proposed by the KnowRob creators are provided by modules for including human activity observation and interpretation [Beetz et al., 2010b] and robot activities logged data [Mosenlechner et al., 2010] as a knowledge source. Acquiring knowledge from the surroundings is another strong trend in literature [Tenorth et al., 2010, Pangercic et al., 2012].

9. *It should exploit problem properties to make inference tractable.* As noted in point 2, a trade-off has to be made between accuracy and speed. For some specific classes of problems though, it might be possible to exploit hard inference techniques surprisingly fast: a knowledge base system should account for this and let programmers build queries that leverage on these particular properties.

The world as a virtual knowledge base

In order to deal with information from external sources, the common procedure in literature is to abstract data and assert it in the knowledge base, then only perform inference on these abstract concepts [Daoutis et al., 2009, Lemaignan et al., 2010]. The biggest issue with this approach is that all the relations that could possibly be of interest have to be computed at once, because the original data is then discarded. The robot is thus only left with qualitative information.

The approach proposed in KnowRob is substantially different, and has been regarded as *the virtual knowledge base paradigm* [Tenorth, 2011]. Within the KnowRob system, data is stored as is, and no relations are computed in advance. Conversely, the knowledge base is free to compute relations from existing data when needed and to query sensors and controllers in case they have relevant data.

This concept is extremely important for the KnowRob system as it allows to answer queries with the most up to date information and only abstracting to the level that is most appropriate in the situation. This way, the outer world can be regarded as a virtual knowledge base to which the robot's knowledge base can send queries in terms of perception tasks.

1.1.2 Formalizing representation

First-order logic, and Description Logic (DL) in particular, is the representational formalism that has been chosen for KnowRob. Description logic is a family of logical languages for knowledge representation, consisting of several dialects with different expressiveness, most of which are a decidable subset of first-order logic. In particular, the Web Ontology Language (OWL, Motik et al. [2009]) has to be used in the proposed system for storing Description Logic formulas in an XML-based file format. The semantic web was the first project to use OWL to represent knowledge, but the language has ever since been used for very different projects and has been adopted by various knowledge representation systems.

Description Logics distinguishes between the terminological knowledge, called the TBOX and the assertional knowledge, called the ABOX. While the TBOX defines hierarchically arranged concepts such as *Table, Screw, Moving*, the ABOX contains instances of these concepts. In the KnowRob system and in modeling knowledge for robotics applications, the ABOX usually describes detected object instances, observed actions or perceived events. Classes of object, actions and events are, in contrast, described by the TBOX.

An ontology is defined as a taxonomy of concepts and the relations between them. In OWL members of the ABOX are called instances, while members of the TBOX are called objects. Properties can be appended to both instances and objects: in the first case to better describe an individual, in the second case to restrict the extent of a class to individual having certain properties.

1.1.3 Storing knowledge and reasoning about it

Prolog [Sterling et al., 1986] has been chosen in KnowRob as the central system to store and reason about knowledge. As a language combining

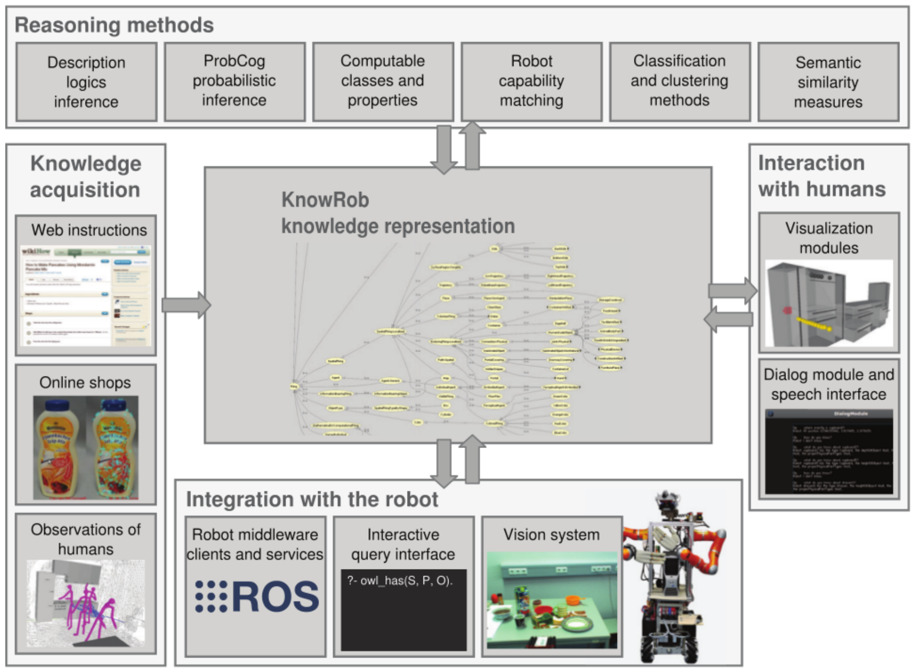
Reasoning methods
Description logics inference
ProbCog probabilistic inference
Computable classes and properties
Robot capability matching
Classification and clustering methods
Semantic similarity measures
Knowledge acquisition
Web instructions
Online shops
Observations of humans
KnowRob knowledge representation
Interaction with humans
Visualization modules
Dialog module and speech interface
Integration with the robot
Robot middleware clients and services
ROS
Interactive query interface
?- owl_has(S, P, O).
Vision system

Figure 1.1: KnowRob can be enhanced with custom modules implementing specific parts of the system.

a procedural interpretation with a declarative reading, it easily allows the programmers to inspect the content of the knowledge base. The use of Prolog, a full fledged programming language, also allows to improve the system's capabilities beyond simple inference. Moreover, it makes it easy to include different knowledge sources as external modules.

Simpler and less expressive representations such as RDF [Beckett and McBride, 2004] or OWL-lite (practically, a simplified version of OWL), allow much faster and efficient reasoning, but do not ensure the possibility to express more complex relations. Cycl [Matuszek et al., 2006] and Scone [Fahlman, 2006], on the other hand, are example of knowledge representation languages the can potentially model everything that can be expressed in natural language, with the drawback of very poor support for automated inference. Prolog has been chosen, among this other representations, as a language of medium expressiveness.

The central KnowRob component is the knowledge representation system, that provides the mechanisms to store and query information about object classes, instances and their properties. The open-source library SWI

Prolog [Wielemaker et al., 2012] is used for this purpose. The KnowRob ontology and its rules are represented in this core system.

Extensions of the system fall into three different categories:

- *additional knowledges* - When new classes or properties are needed, custom OWL files can be included in the system. This way KnowRob can deal with novel application fields. New measurement units can also be included as additional knowledge.

- *knowledge acquisition modules* - As stated in point 8 within the list of paradigms of subsection 1.1.1, any knowledge base should be able to provide ways to enhance its knowledge. Custom modules can be included in the KnowRob system for filling the knowledge base with both information extracted from other sources and from the robot's control system.

- *extensions of the reasoning system* - Modules providing different inference techniques or simply adding procedures for specific tasks can be programmed and interfaced through Prolog.

As we have just seen, the central store for the robot's knowledge is the SWI Prolog system. While additions might be made to the knowledge itself in the form of custom ontologies, the core storage system cannot be changed. On the other side, KnowRob allows different inference engines to be used. Leveraging on such an hybrid system architecture, it allows selected queries to be answered using specific techniques. While systems able to combine multiple inference methods into one exist [Getoor and Taskar, 2007], they cannot guarantee the soft real-time constraints (see point 2, page 2) given by a knowledge base system.

Most DL resoners mantain a fully classified knowledge base in memory. While maintaining an up-to-date memory state makes these reasoner efficient, it's computationally impossible for robots to update their knowledge base at the needed frequency, as this operation can require significant time for large knowledge bases. KnowRob takes this into account and uses Prolog as its default inference system for this reason. OWL statements are internally represented as Prolog predicates (thanks to the SWI Prolog library), to which common Prolog inference can be applied. Since the search-based inference in Prolog is not affected by changes in unrelated parts of the knowledge base, it can be kept up to date with minimum overhead.

Inference techniques different from the previous one can be added to the base system integrating external modules or libraries when requested

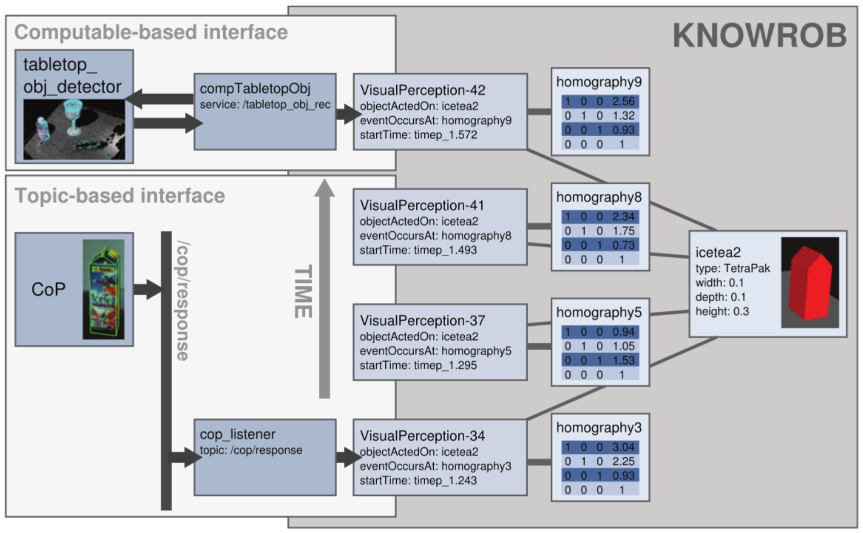

Figure 1.2: Perception systems can interface to the knowledge base in two different ways. In the upper part, the schema represent an on-demand perception system, that can be invoked by the reasoning engine; in the bottom, a continuous perception system keeps storing object perceptions. Both systems produce the same results and store the same kind of representation into the system.

by specific tasks. As a simple example, probabilistic inference can be used when uncertain information has to be represented, as first-order logic can leverage on a great expressiveness but misses this kind of probabilistic representation. Statistical relational models [Getoor and Taskar, 2007], well suited to describe uncertain information, can become very hard, often intractable, when they get too big and contain complex relations among a great number of instances. Programmers thus have to only specify inside selected queries that this inference technique has to be used, they have to write and train the model, and finally select an appropriate inference algorithm. In KnowRob, the implementation of these such models is realized by integrating the ProbCog library[5].

[5]http://ias.in.tum.de/research/probcog

1.1.4 Interfacing perception

Perception systems, particularly for machines powered by ROS, may work in two distinct ways:

on-demand if the communication with the knowledge base is performed synchronously, using a request-response scheme. The perception system works as a server, receiving requests, and the knowledge base sends perception queries when needed, as the client;

continuous if the communication is performed asynchronously, with the knowledge base subscribing to a specific topic on which detections by the perception system are published.

The earlier case is managed in KnowRob using computables. Computables are of two kinds: computable classes are able to create instances of the classes they are associated with —their target classes— while computable properties compute relations between instances. Computable properties are moslty used to extract information from existing data associated with the instances in the knowledge base. On the other hand, computable classes are used to call specific on-demand robot perception systems when answers about the environment are needed.

Continuous perception is managed using external modules that simply listen to the published perceptions and add them, into the knowledge base, to the appropriate instance. The difference in the way these two systems work and the similarities in the way the data they produce is treated can be found in figure 1.2

1.1.5 Representing knowledge

Up to now, we only discussed the theoretical foundations on which the KnowRob system is built. In the next few paragraph actions, object models and environment models will be discussed. These three data types, organized in a topological structure, are the operational foundations of KnowRob, and every customized application has to be built using them.

Actions and tasks

Lists of actions to be executed, specified with their properties, are called either action specifications or action recipes. In OWL, a class can be defined restricting a specific property to have a specific value, a specific range of values, a specific cardinality and so on; these such classes are called restrictions. Restrictions can be combined to form complex classes by intersection,

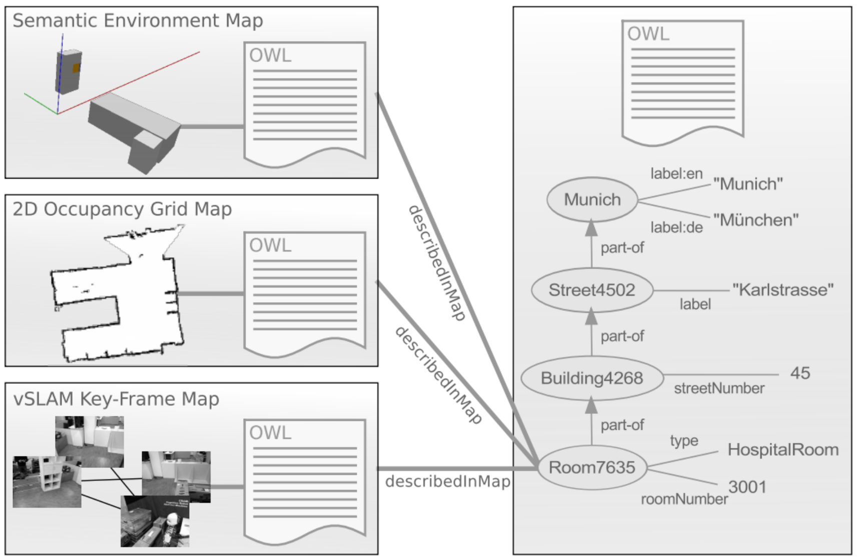

Figure 1.3: Different kinds of environment maps are treated in the same way if they provide a structured OWL file containing data about the map's type and properties. The *describedInMap* property links the description of an environment to the actual maps.

union or complement. Action specifications in KnowRob are defined as intersections of restrictions over the subAction property, with each restriction specifying one of the subactions making up the actual action specification. The ordering with which the restrictions are written in the OWL file does not imply any actual ordering, so that an action specification must also intersect restrictions on the orderingConstraints properties to ensure that subAction properties can be given a specific ordering.

Custom actions specified as objects of the subAction properties are usually themselves intersections of known action classes and restrictions over some of their properties. While the KnowRob ontology presents around 130 actions that can be readily used to build action recipes, new ones can very easily be added writing customs OWL files.

Objects

Objects in KnowRob's ontology represent everything that is neither an action nor an environment model. Multiple inheritance within the Web ontology language allows to account for the different aspects of an object.

In order to better specify particular classes, restrictions on properties are sometimes used as well. Object recognition models are not stored by the knowledge base itself, but their presence, and the robot's ability to recognize specific objects, can be stored in the KnowRob reasoning system.

Articulation models are also taken into account, and programmers can leverage on specific queries to solve problems involving boxes and containers that have to be opened using such models.

Environment models

In order to allow inference over elements of the environment, maps should be written in OWL as well. Semantic maps constist of localized object instances and can be both reasoned upon and updated by the robot perception system. An important note has to be made about the possibility for the knowledge storage system to provide other kinds of environment models even though it cannot reason about them. Each map has to be composed of an OWL description of its type and properties and, optionally, a binary file. While semantic maps are completely described in the OWL file, 2d occupancy grids and vSLAM key-frame maps, for example, are bounded to their binary data file and are treated as black boxes by the system, that nonetheless knows about them and can provide them upon request. This approach reflects the similar behavior of which object recognition models are subjects. Figure 1.3 illustrates this structure.

1.1.6 Executing action specifications

It should first be noted that no actual code nor any package is provided directly with the KnowRob system in order to implement action specifications' execution. While other modules within the KnowRob infrastructures are supposed to provide some of the functionality described in the rest of this section, their use is, in our opinion, not always clear and straightforward. The architecture illustrated in the remainder of this section has to be implemented completely by the keen programmer.

As a first step before executing an action recipe, the robot has to check whether its current software and hardware assets correspond to what is needed to complete the task. For this purpose, SRDL files — cited before in section 1.1.1 — can be used. The matching process, that should be implemented by the programmer, should check for all the recipe requirements to be available on the robot, and only go on with the actual execution in case all needs are satisfied.

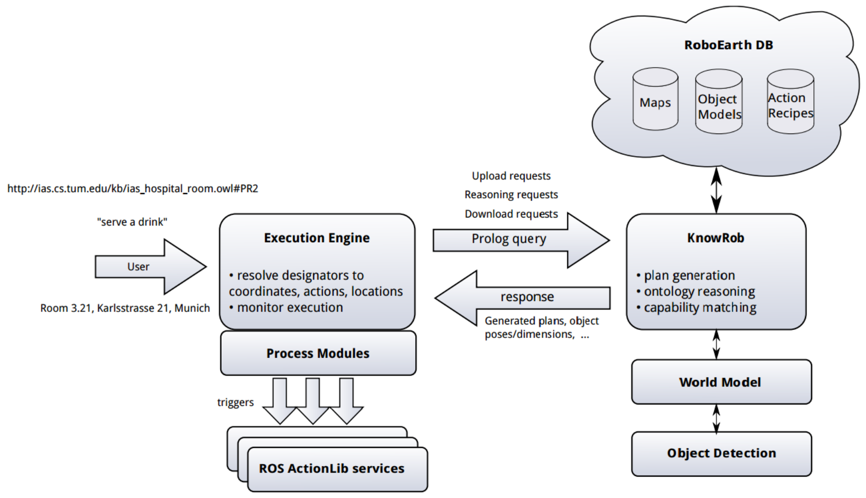

Figure 1.4: A high level overview of the interaction between the different parts of a system involved in transforming action recipes in actual robot movements.

KnowRob provides a few different methods for retrieving subactions for a given action specification. These calls are discussed in the system's documentation and will not be proposed here. Programmers, anyway, still have to autonomously choose how to implement the gap between these abstract actions classes returned by the Prolog system and actual physical robot movements. Every action can be recursively decomposed in order to get to the atomic actions that, when executed one after the other, will result in the intended goal. On the other hand, they could be left as they are if the abstraction level is considered deep enough. No fixed level of granularity is set at which transition from abstract to physical should take place.

The approach suggested [Beetz et al., 2010a, Tenorth and Beetz, 2013b] by the KnowRob creators is to translate the action recipe into a CRAM plan and write a CRAM executive module for taking care of the transition with the help of the CRAM plan language cited before. A second, simpler method is to use the providedByMotionPrimitive property to clearly state which robot routine should be used to carry on that action [Tenorth et al., 2012]. Potentially, only a few motion primitives could be needed, as using robot routines for moving the robot base and arms should suffice for most machines.

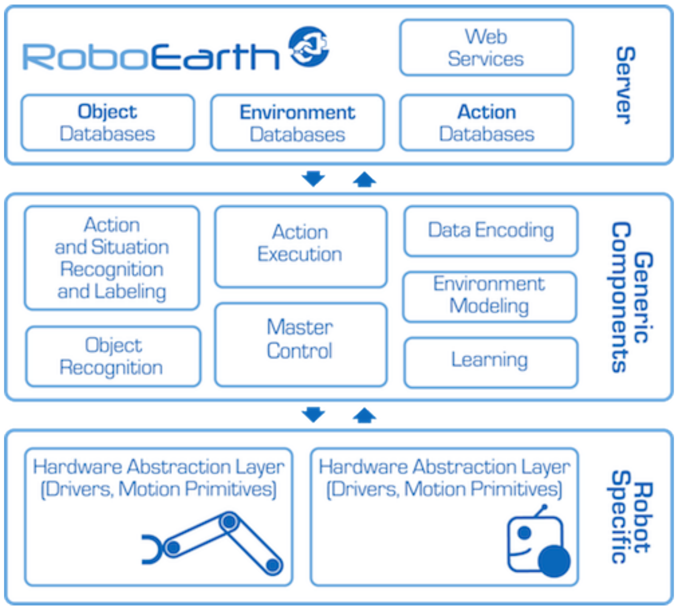

Figure 1.5: A high level overview of the RoboEarth's three-layered architecture.

1.2 RoboEarth

This section describes RoboEarth, a worldwide open-source platform that, in the words of the creators, "allows any robot with a network connection to generate, share and reuse data" [Waibel et al., 2011]. The RoboEarth project addresses the reuse and sharing of knowledge among autonomous robots, with the goal of fostering code and data reuse among different machines [Tenorth and Beetz, 2013a].

RoboEarth allows robots to share reusable knowledge independent from their hardware and configuration. In the idea of the developers, on being assigned a specific task a robot can download both the action recipe and the objects that are needed to carry on the task from the RoboEarth database. The robot can then ground the independent data to its own configuration and execute the task. Finally, the robot may upload its knowledge back to RoboEarth, allowing other robots to benefit from a wider experience.

1.2.1 The three-layered architecture

The core of RoboEarth's architecture is a *server* layer that holds the database. It stores reusable data in the form of actions (i.e., action recipes), environments (e.g., semantic maps, occupancy grid maps) and objects (e.g., images, point clouds, models), and it is accessible through common web interfaces. These kinds of objects, the same described in subsection 1.1.5 are returned to queries coming from lower levels.

The second layer is composed of *generic components*, part of the robot's control software, whose goal is to query the web interface for action recipes and for the objects needed to carry them on, and transform them in actual robot actions. KnowRob can be made part of this layer by allowing RoboEarth to be one of the external knowledge sources. When KnowRob needs a specific object model, for example, it can simply include the call to the RoboEarth database in its query and obtain the appropriate result in case it exists. Sensing and mapping, for example, might also be part of this layer.

The third layer is *robot specific*, and provides a standard interface to a robot's specific, hardware-dependent functionalities. Programmers needing to reproduce the same action recipes on different robots only have to reproduce the functionalities in the third layer across different robots, while the rest of the architecture can remain unchanged.

1.2.2 KnowRob integration

Figure 1.4 explains what the RoboEarth architecture looks like after integration with KnowRob. The execution engine, a custom module able to communicate with both KnowRob and with the robot's services, is an implementation of the executor described in subsection 1.1.6. As KnowRob is directly connected to the RoboEarth database, it can query for missing elements such as semantic maps or object recognition models. Moreover, it can query for assets it cannot manage, such as occupancy grids or images, that it will treat as black boxes directly passing them to the executor, that will then forward them to the correct robot module.

1.3 Rapyuta

Myiazaki's castle in the sky gave its toponym for this project, also more traditionally named RoboEarth Cloud Engine (RCE). Unlike for the movie's castle, lost among the clouds and full of robots, the aim of Rapyuta is to provide a Platform-as-a-service for running heavy computations for robots in the cloud and, at the same time, connecting them easily and efficiently [Hunziker et al., 2013].

Computing power is a key enabler to solve many of the challenges the robotic world is facing right now, and complex reasoner such as KnoRob, analyzed in section 1.1, are a clear example of modules requiring great computing power while serving robots in ways that could not be imagined before. Anyway, using on-board power sources to perform this kind of computation could lead to much shorter battery life, thus reducing operating duration and increasing costs. Most of the heavy tasks carried on by robots are anyway not bound to be executed on the robot itself, and don't have strong real-time requirements. They could thus be easily carried on in a cloud infrastructure [Hu et al., 2012], with requests and results being transported back and forth through a wireless network.

Rapyuta is based on an elastic computing model that dynamically allocates secure computing environments for robots on the cloud. Each robot can count on a virtual counterpart, or clone, to which it can delegate computation. These environments are implemented using Linux Containers[6], which provide isolation of processes and system resources within a single host. Computing environments in Rapyuta are set up to run any process that is a ROS node, and all processes within a single environment communicate with each other using ROS inter-process communication. No modifications are needed on the packages by developers.

These environments are tightly interconnected, allowing robots to share some of their services and information with the other robots. Rapyuta's communication protocols are split into three parts. The *internal communication protocol* covers all communications between the Rapyuta processes. The *external communication protocol* defines the data transfer between the physical robots and the cloud infrastructure on which Rapyuta is running. Last, communications between the Rapyuta system and the nodes running inside the containers are carried on using ROS, as stated before.

Rapyuta is also shipped with all the necessary controller modules which take care of establishing containers, nodes inside them and set up communications. New robots connecting to the system have to provide an extensive configuration in order to run correctly, but the use of ROS communication infrastructure and the fact that nodes do not need any modifications to run in the containers makes it very easy to dislocate heavy computations in cloud-based containers.

[6]http://lxc.sourceforge.net/

Chapter 2

Experimental setup

While the very first goal has been to thoroughly study the KnowRob and RoboEarth frameworks, we soon realized a realistic experimental setup was needed to understand the full potential of the various analyzed modules. As our aim has specifically been to understand how multiple robots might be enabled to collaborate, we decided to use a simulator in order to compensate for our lack of physical equipment. Moreover, we had to find a meaningful goal for out work. Before proceeding with our proposal, we give here a general overview of the setup we used while designing and developing our system.

We decided to use the RoCKIn competition as an example setting. An overview of the reasons for this choice will be provided in the next section. We choose Kuka's YouBot as the robot for our experiments, as done by most of the competitors in the more famous RoboCup, in the @Work [Kraetzschmar et al., 2013] section. The description of the YouBot model we designed is provided in the second section below. The final section gives a general overview of the set up we used to program and test our collaboration framework with knowledge-enabled robots.

2.1 RoCKIn @Work

RoCKIn is "a EU-funded project aiming to foster scientific progress and innovation in cognitive systems and robotics through the design and implementation of competitions" [Ahmad et al., 2013]. As the laboratory in which we carried out our experiment is involved in the design of the first edition of this event, we had direct access to proposals and ideas circulating around this new project.

The first characteristic that caught our attention was the project's consideration for general solutions to the challenge problems. In the proposed challenges, general soluions should score better than approaches that work only in narrowly defined contexts. As we have seen in the introductory chapter, the two main goals of the KnowRob framework are to allow physically different robots to share information and to enable robots to quickly obtain new knowledge about diverse tasks. Solving a challenge using the tools described in chapter 1 would thus be the perfect fit, favoring a very general and broad approach over a more specific solution.

Another aspect of the challenge that we found very important is that proposed problems look very similar to tasks that a human would be given, like cooking pasta in a restaurant's kitchen or dealing with package returns in a warehouse. Once again we have seen in the previous chapter that the aim of the whole KnowRob infrastructure is demonstrating that robots can be allowed to participate in this kind of works, usually performed by humans alone.

As the design of the competition is still in a very early stage, we also identified the possibility to propose the inclusion of a problem involving a coordination challenge to the list of proposed ones. This would allow to prove the architecture proposed in the following chapter during a real competition, with all the consequences that such a commitment would provide.

We would finally like to stress the fact that our goal with this work has not been to completely code a program able to succeed in one of the challenges proposed by the RoCKIn team. Conversely, we analyzed the competition as a way to obtain enough background to purposefully decide which robot to use and how to set up our ROS environment. Moreover, this work is thought as a single part of a much wider effort in setting up a system to take part in the competition.

2.2 A simple YouBot simulation

The YouBot, produced by KUKA, is a small mobile manipulator. Its arm is mounted on a mobile base and has five degrees of freedom. The base moves thanks to four mechanum wheels, allowing it to translate in any direction. It by default hosts a small plate, that can also be replaced with a second arm. Both the arm and the plate are independently mounted on the base and can be replaced and exchanged as needed. Figure 2.1 shows the default YouBot setup.

The laboratory that hosted our experiments at University of Padua does

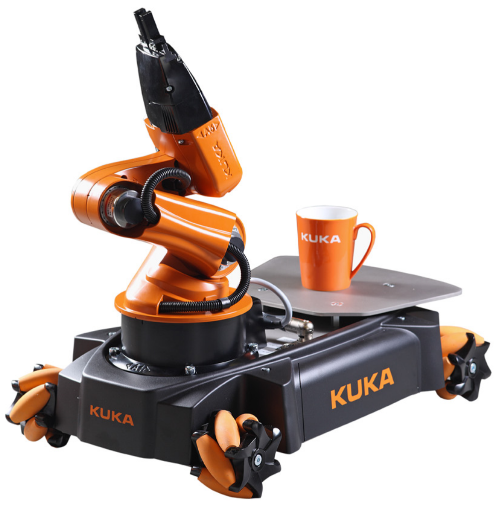

Figure 2.1: KUKA's YouBot equipped with a single arm and a plate.

not own any YouBot. It has thus been impossible to carry out any actual experiment during the development process. Anyway, even having the possibility to work with one physical machine would not have been enough, since our goal was to understand robot coordination. Gazebo is the official robot simulator for the ROS platform, but the only YouBot Gazebo model we could find[1] is out of date and not usable with the latest version of the simulator. As KUKA provides the three-dimensional meshes[2] for the robot, we decided to work for a few weeks rebuilding the obsolete model and equipping it with up to date navigation and teleoperation modules.

We used the git version control system to keep track of our work on the YouBot model, that we organized in a package called youbot_ ros_ tools. All the code and the documentation can be found online[3], and change proposals can be submitted by anyone. As stated before, the model has been coded under *ROS Hydro Medusa* and *Gazebo 1.9*. The package is divided in five parts, each of which deals with a very specific section of the simulation. The following list describes the various implemented sections of the model, while implementation details and more information about each part's specific content can be found in the following sections.

[1]https://github.com/WPI-RAIL/youbot_description
[2]http://www.youbot-store.com/youbot-developers/
[3]https://github.com/micpalmia/youbot_ros_tools

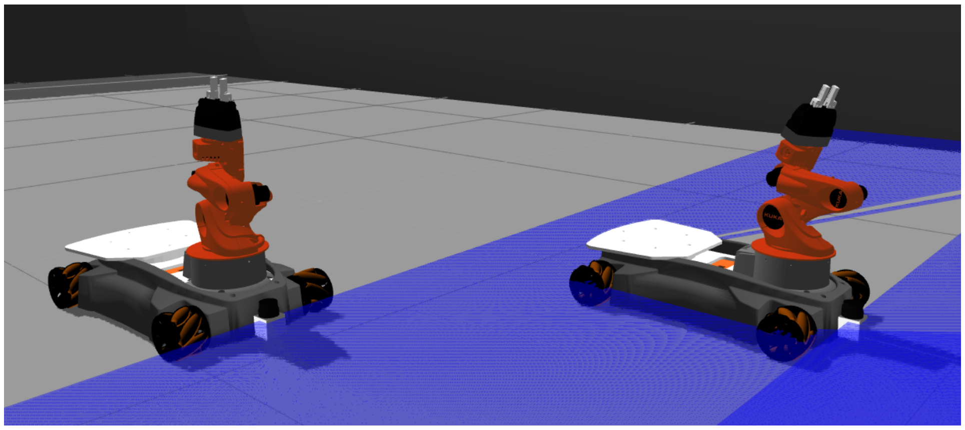

Figure 2.2: A Gazebo simulation showing two robots in an empty room. The blue layer is a graphical representation of the laser signal.

youbot_description is the core of the model. It is programmed in self-contained modules, that can be composed in different ways to set up various robot configurations. The wheels are included in the simulation but their physics is not taken into account because of the many challenges posed by simulating this kind of wheels. Gazebo's ros_ planar_ move plugin, however, allows the robot to move in all directions as if mechanum wheels were actually operating. A simple Hokuyo laser is attached to the front of the robot by default. The plugins for controlling the base and for managing the laser data are the only ones included into the model by default.

youbot_gazebo only contains utility launchers for spawning the robot model inside a new Gazebo environment. The simulated map for an empty room is also included.

youbot_control contains the configuration for the arm controllers and the utility launchers for the appropriate plugin setup.

youbot_teleop provides a simple script for remotely operating the robot once the model and the controllers have been started. Methods are provided to move the base, the arm and the gripper. The base can be moved using commands that require translation values relative to the robot's current position. The arm and the gripper, on the other hand, can also be moved passing values indicating the absolute position of the joints.

youbot_2dnav contains the configuration for the YouBot to work with the ROS navigation stack. The default launcher starts a map server with the default empty room map, the amcl localization module and the move_base module.

Xacro[4] (from XML Macros) is an XML macro language. It allows to write concise robot models in an improved XML syntax that automatically expands customized expressions after compilation. The Xacro compiler produces URDF files, that Gazebo can read. The content of the youbot_gazebo module is almost entirely written using this language. Xacro allows to include Gazebo-specific tags to specify the plugins to be used on particular parts of the model, such as done for the base movement or for the laser scanner.

The laser equipped by default on the robot is an Hokuyo URG-04LX-UG01. A number of other sensors (Asus Xtion, Microsoft Kinect and lifecam) are usually used on competition robots, and the code to include them is ready to be used in the public repository.

The empty room map model for Gazebo has been created with the building construction tool in Gazebo's graphical editor itself. It is included in the youbot_gazebo module together with the corresponding occupancy grid. The grid has been automatically obtained running the slam_gmapping[5] package, providing simultaneous localization and mapping, while the robot was being manually moved around in the simulated room using the youbot_teleop module. Having both the room simulated in Gazebo and its occupancy grid available for robots allows the youbot_2dnav module to work properly.

While the Gazebo model can be used for the physical simulation, the occupancy grid is a component used by robot's modules. In particular, the move_base plugin, the default ROS package for providing robot navigation, requires the amcl module to be running on the robot as well. This latter module is a probabilistic localization system for robots moving in 2D, and requires a map of the environment to be available. As the move_base package is the core of our youbot_2dnav module, having both the simulated environment and he occupancy grid is necessary in order to be able to use the proposed navigation module with the simulated robot.

[4]http://wiki.ros.org/xacro

[5]http://wiki.ros.org/slam_gmapping

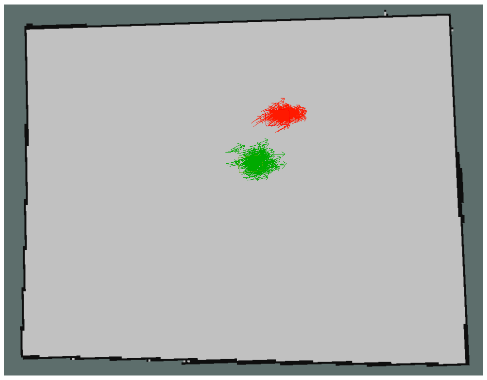

Figure 2.3: The empty room occupancy grid overlayed with the current approximate positions for two YouBot robots. This map was corresponds to the setting of figure 2.2. Positions are represented as a cloud of arrows beacuse of the probabilistic nature of the localization module.

2.3 Multi-robot setup

This section will analyze the setup we used while developing and testing our proof of concept. Since the implementation of our prototype is the subject of the next chapter and has not yet been discussed in its details, we will forcibly have to anticipate some decisions that will be explained in chapter 3. We will discuss how multiple simulated robots can be run in the same environment, and how they can be allowed to communicate with each other. We will also discuss more precisely about how we set up a reasoner for each machine.

We have seen in the introductory chapter that a robot having to execute generic tasks and to reason about requests can be equipped with an execution engine powered by the KnowRob system and enhanced with an ecosystem of custom modules and plugins that enable specific functionality. RoboEarth and Rapyuta, while being themselves completely autonomous projects, can be considered as part of this wide KnowRob ecosystem. While bringing undisputed benefits, anyway, these modules make the development and testing process slower and more complex, adding different layers of functionalities that are not strictly requested in a simple prototype such as the one described in this work. We thus decided to only use KnowRob in our implementation, leaving the addition of other modules to later works on the subject.

All of the packages discussed in the previous paragraphs have default launchers, very easy to customize. They all start the corresponding nodes, topics and services in the namespace specified at launch time. Each namespace represents a single robots. For a full set up, each module has to be started once in each namespace. This is a very elegant way to deal with multiple robots in a simulated environment, as it allows the complete separation of the simulated machines while maintaining all the communications local. This way, it is possible to easily establish inter-robot communication without setting up other packages.

Communication between robots, with this setup, does not need any additional module and can be implemented using default ROS services and topics. We would anyway like to stress the fact that in a competition scenario our suggestion would be to use the power provided by the Rapyuta platform presented in section 1.3. This would allow greater autonomy time for the robots, faster processing of hard real time tasks, that would have more resources available, and easy communication.

The json_prolog ROS module is the core of any smart robot built with KnowRob. It takes as input the name of a ROS package and starts a KnowRob reasoner that can be queried passing Prolog queries to a specific ROS service. It automatically loads the init.pl file provided in the prolog folder in the specified package. This file usually contains directives for parsing OWL files containing project-specific knowledge and prolog custom rules.

We noted before that a single KnowRob reasoner is thought as the knowledge storage for one single robot. It contains information about object positions, recognition models etc. relative to one single machine and, if configured, it can be directly connected to its sensors. For this purpose, the json_prolog package also has to be started for each and every robot, passing the appropriated namespace at launch time. At any given time, the number of active reasoner is the same as the number of working robots.

Our very simple setup of the init.pl file used to customize the reasoner engine will be discussed in the next chapter, together with the rest of the details regarding the architecture we proposed and the simple implementation we delivered.

Chapter 3

Design and implementation

After describing the projects and the frameworks we decided to test, and discussing the way we set up the simulation, this chapter will continue with our suggestions about how collaboration and coordination can be made possible by mutual knowledge exchange in a system that uses the KnowRob system to guide knowledge-enabled machines.

In the following sections, we will first briefly discuss previous experiments of this kind within these systems —or better, the lack of such experiments— and the motivations that brought us to producing this thesis. Second, we will state the hypotheses behind our work. We will then explain the general architecture of a system leveraging on KnowRob to offer knowledge about other machines in the same environment to robot control software. Fourth, we will give an overview of a simple implementation, the issues it presented and the problems it posed. The closing section will finally propose a set of enhancements that would make the proposed system more robust, efficient and elastic; the possibility to relax some hypotheses will also be discussed.

3.1 Motivation

Knowledge representation and sharing among autonomous robots, along with studies about reasoners for robots, come a long way from the early days of robotics [Nilsson, 1984]. A number of articles about the different aspects of the problem have been written in past years, and the subject is now starting to get attention and consideration in the scientific community, as artificial intelligence is being applied more and more to robotics. Tenorth and Beetz [2013b], in section 11, presents a thorough literature review on the subject. As we have seen in section 1.3, systems that efficiently share information among cloud-powered robots have also been successfully implemented, and ROS even provides this as a basic functionality, without the need for additional packages to be installed and configured.

The KnowRob system seems, from our research, to be the most comprehensive knowledge representation and reasoning system available to the scientific community now. As mentioned before, some experiments have been carried on to test its capabilities both by the consortium of universities that is working on designing and building it and by other institutions. Among these experiments, anyway, we have not been able to find any test involving robots with mutual knowledge about each others existence. The only test we could find in which two different robots were operating in the same environment [Beetz et al., 2011] did not seem to consider any kind of mutual consideration between robots.

The knowledge representation system and the reasoner discussed in chapter 1 are built to only serve one single machine, and we have discussed some of the reasons for this before. But could the problem of mutual knowledge be solved designing a single knowledge representation and reasoning system serving multiple robots? Designing and implementing a platform of this type would pose many difficult choices and implementation problems. As an example, should perceptions be stored in one single world representation or should each robot use its own, or both? Communication problems affect the system described in the previous chapters, but robots can always reason on the knowledge they obtained before being disconnected, even when they are offline. On the other hand, what would happen to robots that completely depend on a central reasoning engine to work? A centralized reasoning system from which the robots could have to disconnect would make the single clients completely useless when offline. It seems very likely that a system of this kind would not be the best implementation choice.

Even though the reasoning system only serves one robot, this does not mean that independent and autonomous machines should ignore each other. First, being aware of the presence of other robots can avoid collisions and result in better coordination. Second, complex systems could leverage on this knowledge in order to solve problems that cannot be solved by one robot alone, to ask for help avoiding any human intervention and so on. For these reasons, we think that meaningfully representing the presence of other robots in the environment would greatly enhance the possibilities available to each of them.

Modern robot coordination algorithms require each machine to know the current positions and final goals for each and all of the robots involved [Boscolo, 2012]. With this data, each robot can autonomously navigate to the goal without colliding with coworkers. Robots keeping a semantic representation of their teammates in their reasoning engine could obtain these information with very little effort, when needed.

3.2 Hypotheses

In section 2.3 we expressed the motivations that led us not to directly use RoboEarth and Rapyuta in this project. While the hypothesis presented there form a valid starting point, a few others are needed before continuing with the sections about architecture and implementation.

Robots can usually move different parts of their structure. They usually can move their base, but they sometimes can also move an arm or two, maybe even more. Using these parts, they can operate at different heights, towards specific directions, doing different things. The YouBot, after moving from one point to another, could for example start picking up objects and putting them on its plate. All of these movable parts can interfere with other robot's components in the environment. For the sake of our proof of concept, we decided to stick with base movements only. We will be thus practically only using the base of the YouBot without its arm. As a direct consequence, the only possible action recipes are the ones only made up of navigation tasks. We also avoided having coincident goal positions for different machines, but this is a very weak hypothesis and could easily be removed.

Equipping robots with knowledge not just about their environment but also about their coworkers would allow action recipes to contain directives directly involving other robots. Since this would open a completely new field of possibilities, we decided not to proceed in this direction. Future works might consider studying the consequences to this improvement.

Finally, we decided to start with a known set of robots: in our experiments, each machine has a unique identifier and all of the robots in the same environment know each others' unique names.

A direct consequence of the hypotheses expressed in section 2.3 is that machines have no limit on computation, and power consumption is not an issue. While these and many other problems could be solved in a real setup using RoboEarth and Rapyuta, we would like to stress the fact that we preferred to complete a very essential prototype showing our approach in a simple and linear way instead of using all possible additional packages. This way, we avoided cluttering the project with functionalities that would make it hard to identify the real contributions brought by the design presented in the next sections.

3.3 Architecture

It should be clear by now that a system storing and reasoning about other robots in a semantic way has to be divided between a group of modules adding semantic features to incoming knowledge and meaningfully storing it, and another group of modules using this knowledge to avoid collisions with other machines in the same environment.

The execution engine and the KnowRob system itself (recall figure 1.4) are the only parts of the system we can manipulate in order to achieve our goal. In particular, the coordination modules has to be made part of the earlier, while custom modules for the latter have to be programmed in order to integrate knowledge into the reasoning system as it becomes available.

With our hypotheses and restrictions, the only situation in which robots may be requested to coordinate with each other would be during movement from one location to another. Normally, in this case, the execution engine would simply call the appropriate actionlib service, that would carry on the task without further interaction with the knowledge base. Our approach to coordination is to implement the execution engine so that it will query KnowRob for other robot's data before actually moving the machine.

Obviously, some new semantic entities and properties will be needed to store useful information in the system. The types of data about other robots could span from simple mechanical data such as the position of their joints or their position in space to more high level information such as the task they are currently executing or their current state relatively to some motion parts (e.g., still or moving). This knowledge can be obtained by the robots using sensors or directly receiving it over the network through specific topics. These approaches can also be combined to obtain a more robust system.

In a system implementing this architecture, any single robot has to continuously broadcast information about its state. It also needs to subscribe to the same broadcasts from other robots in order to update its reasoning system. Some types of information could also be obtained from sensor data. One or multiple modules, integrated with the KnowRob system, should take care of transforming other robot's data into meaningful information to be stored in the knowledge base.

Changes in the current world state, detected by the aforementioned component, should also be promptly communicated to interested modules, such as the coordination one. For this purpose, these modules shall publish a service hook to be called whenever a change is recorded in the world state. While this is a very implementation-dependent feature, we believe that leaving to the logger module the task to wake up interested modules is the best way to deal with this problem; querying the knowledge base is quite fast and efficient, but a busy wait on interesting features mostly appears as a waste of computational resources.

This description purposefully left out the core part of the execution engine, the program transforming action recipes into actual robot movements. As stated before, in our system the only kind of actions allowed to robots are the ones of type Navigation-Translation event. With this constraint, the only purpose of the core execution module is to call the coordination module with the parameters directly taken from the actions list. The coordination engine would then take care of querying the reasoning system calling the navigation server, returning the control to the execution module only when the task has been completed. Such a system does not pose any interesting design challenges and has already been described in literature [Marcato, 2012, Beetz et al., 2011]. We will thus avoid dealing with it in the remainder of this thesis.

3.4 Implementation

In the previous section we discussed the proposed architecture at a very high level. While this is necessary to clarify all aspects of the proposed infrastructure, developing a working prototype is very useful in various different ways. First, it helps correcting ingenuous errors in the design process. Moreover, it allows to clarify the design itself, both while in the process and in further iterations. Finally, it is a rather accurate proof of the fact that the proposed architecture works as expected.

A simple proof of concept implementing the described architecture is forcibly bound to a specific coordination algorithm. Specific enhancements to the knowledge base and the actual implementation of the coordination algorithm necessarily have to be tailored to a defined routine that machines can follow when multiple robots want to move at the same time.

In the remainder of the section we will first present the very simple and naïve algorithm we used for our implementation. Second, we will describe our enhancements to the default knowledge base. Finally, we will describe the two parts of the ROS package we implemented, following the directives expressed in section 3.3.

The result of the work described in this section is available on-line in an open source repository. As the json_prolog library, used to connect external modules to the KnowRob reasoner, is still using the rosbuild system, we could not create a catkin package. We thus had to stick with the old build system as well. Documentation and more details on the actual use of the package can be found online in the same repository.

3.4.1 A naïve coordination algorithm

Very accurate coordination algorithms are now available to the scientific community [Pagello et al., 1999, Kowalczyk, 2001]. They vary in the parameters they require as inputs, and produce as output optimal paths for robots to avoiding collisions. The max-sum multi-robot coordination algorithm, for example, solves this problem efficiently and without the need for a central server to calculate the trajectories [Boscolo, 2012]. We conversely decided to implement a very naive algorithm for collaboration in order to be able to concentrate on the implementation of the rest of the infrastructure instead of focusing on the coordination algorithm itself, not the center of this work.

We choose to implement a very naïve algorithm as the baseline for our prototype. The algorithm is described in the remainder of this paragraph, while a pseudo-code implementation can be find in the listing for algorithm 3.1. Each robot needs to be associated with a unique priority value, known to all other robots as well. Only one robot at a time is allowed to move in the shared environment at any given moment. The robot with the highest priority expressing the request to navigate always have to be granted permission to move, even at the cost of robots with lower priorities having to be left waiting. These requirements are respected without the help of a controlling authority, and all the machines in the environment respect these rules in a completely distributed manner. The algorithm allows each robot to assess in a completely autonomous way its right to move.

Being able to obtain the state of its teammates at any given moment, any single robot can decide whether its time to move has come or not. The information needed for each machine by this particular algorithm is only limited to the navigation state of the particular machine. Each robot can be *still*, standing in its position and not expressing the desire to move, or *moving*, navigating in the environment or waiting to do so[1]. The next section will describe how these states can be represented in the robot's knowledge base and how the algorithm is implemented within the KnowRob system.

[1]These last two situations could be divided in two different states, but as we limit the cardinality of robots moving at the same time to one it is straightforward that the only machine actually moving at any given time is the robot with state *moving* and the highest priority.

```
$self.stop()$
$self.rightToMove \leftarrow true$
$movingRobots \leftarrow query(type = robot \cap status = moving)$
for $robot \in movingRobots$ do
    if $robot.priority < self.priority$ then
        $self.rightToMove \leftarrow false$
        break;
    end
end
if $self.rightToMove$ then
    $self.move(goal)$
end
```

Algorithm 3.1: Naïve coordination — Whenever a goal is assigned to a machine, or a change in the status of another robot is detected, this algorithm is triggered. *self* refers to the robot itself, the *query* methods refers to the KnowRob system. The robot first stops (in case it was moving), then assesses its own right to move, and only proceeds if it is allowed to.

3.4.2 Knowledge base custom classes and properties

The KnowRob system comes with a very essential knowledge base. Users willing to implement specific modules on top of the system or use robots for any specific task have to write their own ontologies. In section 2.3 we stated that KnowRob starts up reading a single Prolog file. In our case, the file only contains a single directive for including a custom ontology and adding its statements to the default one. Such a custom ontology is defined in a OWL file, and its content is defined below.

As seen in section 1.1.6 when discussing about action recipes execution, actions are transformed into actual robot movements by the execution engine, that calls the appropriate actionlib service for each action type. The default ontology, anyway, does not contain any way to define which actionlib service should be called for a specific action entity when transforming it into actual robot movement. The idea behind the lack of this feature is that a CRAM executor should be invoked to implement action recipes, and action entities should be transformed into actual robot movements during this stage. As we decided not to go into the details of CPL, a solution on this side was needed.

Since the only allowed action for our robots is moving to one specific

Listing 3.2: The GoToPoint class

```
<owl:Class rdf:about="&move;GoToPoint">
  <rdfs:subClassOf rdf:resource="&knowrob;Translation−LocationChange"/>
  <rdfs:subClassOf>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:onProperty rdf:resource="&move;providedByMotionPrimitive"/>
          <owl:hasValue rdf:resource="&move;move_base"/>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty rdf:resource="&move;destXValue"/>
          <owl:cardinality rdf:datatype="&xsd;decimal">1</owl:cardinality>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty rdf:resource="&move;destYValue"/>
          <owl:cardinality rdf:datatype="&xsd;decimal">1</owl:cardinality>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </rdfs:subClassOf>
</owl:Class>
```

point from the current location, we defined a GoToPoint class, subclass of the Translation-LocationChange action from KnowRob. Every and each action included in an action recipe valid in our implementation will thus have to be a subclass of this class. We also defined the providedByMotionPrimitive property, and bound all subclasses of our GoToPoint to have move_base as a value for this property. The code for the class can be found in listing 3.2. With this setup, the move_base service is set up to be called whenever an action of type GoToPoint has to be transformed by the execution engine.

In this specific listing, resources are appended to a prefix that can either be &move; or &knowrob;. In XML syntax, these prefixes are a shorthand notation that avoid having to write the full domain to which a resource belong to. While the knowrob keyword refers to the default ontology, move references the custom ontology we designed. While the instances of the GoToPoint class have a very strict bound on the value of the providedByMotionPrimitive property, the only bound on the other two custom properties, destXValue and destYValue is cardinality. Naturally, a single instance of this class should only have one single goal, giving these properties a cardinality

Listing 3.3: Possible states for robots

```
<owl:Class rdf:about="&move;MovingStatus"/>

<owl:Class rdf:about="&move;Still">
  <rdfs:subClassOf rdf:resource="&move;MovingStatus"/>
</owl:Class>

<owl:Class rdf:about="&move;Moving">
  <rdfs:subClassOf rdf:resource="&move;MovingStatus"/>
</owl:Class>
```

of one.

The other classes we added to the improved ontology are quite straightforward. The YouBot class, subclass of Robot, can be instantiated so that pose data can be attached when needed. We also added the hasRobotId property for robots, so that the execution engine could establish id and priority[2] of the various robots instantiated in the knowledge base. The algorithm described in subsection 3.4.1 also requires the state of the robots to be represented in the reasoning system. We thus added the isInMovingStatus property to the Robot class, and listing 3.3 shows the definitions for the range of possibilities this property can be set to. As noted in the previous section, Still refers to machines standing in their position and not expressing their need to move. If a robot is Moving, on the other hand, it expressed its desire to move and if it has the highest priority among its teammates can actually do so.

3.4.3 Algorithms and data structures

We developed a simple system implementing the architecture described in section 3.3 in order to both clarify its structure while we were designing it and show its effectiveness. Because of these reasons and of the limited amount of time available, we kept the implemented infrastructure very simple. The resulting prototype, programmed completely in Python, is divided into two modules. A first module can be considered part of the KnowRob core system and its aim is to log all data coming from other robots. The second module, on the other hand, ensures smooth coordination among the various machines in the environment using the simple algorithm proposed.

[2]During our tests, we set identifiers to be in the form *name* + *unique number*. Robots with a lower id number have higher priority.

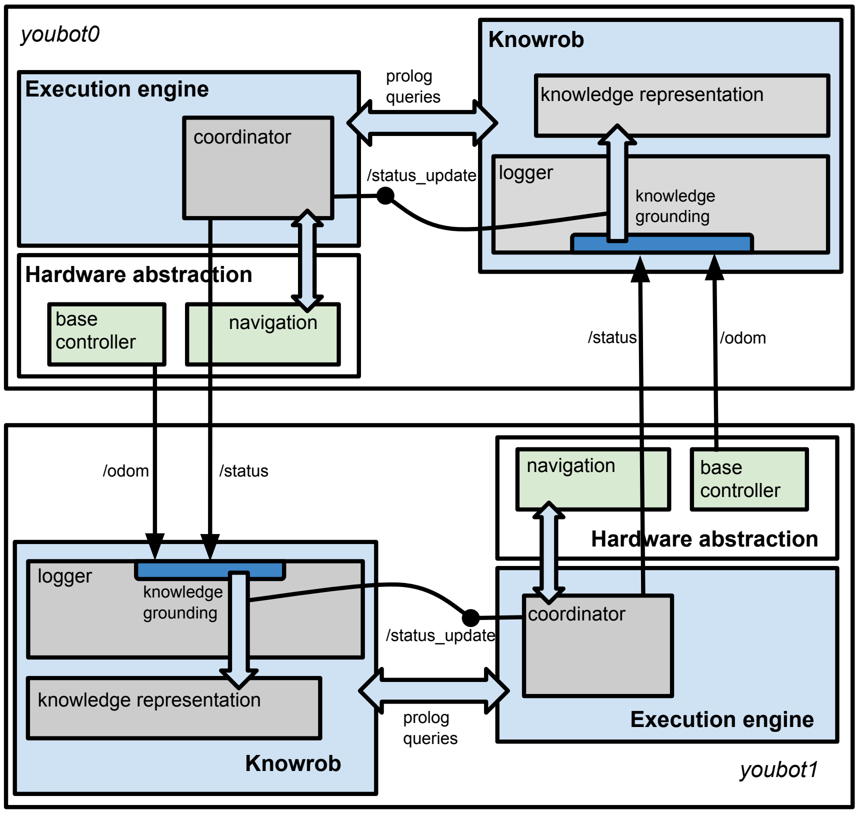

Figure 3.1: The internal architecture is represented here for two robots sharing data about themselves with their teammate, using the design proposed in this chapter. The */odom* and */status* topics are published by the base controller and by the coordination modules respectively. The logger grounds this knowledge and stores it in the knowledge base, triggering the */status_update* service when needed. Only relevant modules and connections are displayed.

In our proof of concept, all robots publish two ROS topics in order to allow other machines to stay up to date about their respective state:

/odom messages are automatically published by the Gazebo plugin for the control of the base. They are of the standard Odometry type and are produced at a rate that can be adjusted when programming the simulation.

/status has custom type Status, defined as a single string. Robots publish a message on this topic whenever they change their state with regard to base movement. The possible options corresponds to the ones defined in listing 3.3.

The first module, the logger, requires as input the identifiers of the robots involved in the simulation and its own identifier. It first waits for the reasoning engine to be up and running. Immediately afterwards, it subscribes to both the /odom and the /status topics. Upon registering the first information received, it creates a new YouBot instance in the knowledge base and associates it to the right identifier. After this first step, all data received is logged and automatically associated with this instance as soon as it is received. Upon receiving messages on the topic regarding the moving status of other robots, after updating the related property for the appropriate instance, the logger also calls the /status_update service advertised by the coordination module described in the next paragraph.

The coordination module is built on top of the move_base action library. It provides a /goal service, of custom type Goal, to which the x and y values representing the goal to be reached have to be passed[3]. The module also provides an empty service (i.e., it does not require any input and does not return any value) called /status_update that has to be called by the logger upon updating the robots' status as seen in the previous paragraph. Finally, the module advertises the /result topic on which it publishes a message on completing the current goal and the /status topic, on which it advertises changes in the navigation state as discussed before.

Upon receiving a new goal, the coordination module sets the state of the robot as Moving and publishes this information on the appropriate topic, thus indirectly updating the knowledge base of all other robots as well. It then queries KnowRob for the list of machines currently moving and decides whether the goal should be passed to the actionlib server and actually executed or not. In either cases, receiving a status update fires a reassessment of priorities, and the robot can either be kept moving, started or stopped, depending on the previous state. When the result for the current navigation task is received from the actionlib server, an update on the robot state topic is published containing the keyword Still. The rest of the robots can thus reassess their own priorities and move on with their suspended tasks.

Figure 3.1 represents the implementation discussed in this section, only representing the communication paths and modules useful to describe it.

[3]We only used x and y values for the goal instead of a complete pose message for simplicity.

Topics are represented as arrows going from the publisher to the subscriber, and the /status_update service is represented as a dot coming out of the responsible module. The graph only represents two robots, but the same structure is valid for multiple machines

3.4.4 Results

It is not very easy to uniquely test a design proposal of this kind and hope to find meaningful results showing its correctness. Moreover, finding useful metrics requires a very long time for this kind of systems, especially because implementation of analogous platforms is required for comparison. The proposed architecture shows one way to allow *knowledge-enabled* machines to obtain and use knowledge about their teammates, thus fostering coordination and collaboration in systems implemented using this technology. As this seems to be the first and only proposal for implementing coordination in these systems, it should be considered as a simple (and working) starting point, but should not be taken as a robust or full fledged implementation.

We tested our prototype both in an automated and a manual manner, in order to assess all possible situations. We first equipped two simulated robots with different action recipes that would make them move around the empty room with continuous risk of collision. Robots behaved as expected and did not have any problem in communicating their state or grounding the received knowledge.

In order to push the limits imposed by using fixed action recipes, we also manually tested our proof of concept manually calling the ROS services provided by the collaboration module described in section 3.4.3. These calls correspond to the calls that would be made by the execution engine. These experiments also showed that the architecture is working as expected and does not seem to present any particular issue.

3.5 Possible improvements

The hypotheses in section 3.2 strongly restricted the span of our proposal and implementation. While this restriction was needed to focus more clearly over a specific problem and its solution, removing some of the hypotheses would result in a more complete and interesting system. In the remainder of this section, the most relevant improvements will be discussed.

Configuration

In our proof of concept, we embedded in the program's code the list of topics to which the robot had to subscribe. Each robot involved in the simulation published the same data. They also only notified the coordination module upon receiving updates on a specific topic. This made it very hard to quickly edit the configuration for the module, with the source code actually having to be modified to change the configuration.

This problem can be solved with relatively little efforts by designing a simple way to configure the module before launching it. Passing parameters to the program at launch time might not be a feasible option. A probably better option would be to set up a configuration file containing all the information needed by the module: the identifiers of the robots involved and the robot's own name, the topics published by each of the robots involved and how to map them to semantic knowledge, the modules to be updated when receiving updates from specific topics, as done for the coordination module in the proposed system.

Multiple algorithms

The proposed solution simply considered one single algorithm as the only possibility. Our naive algorithm is called whenever a movement goal is given to the robot, and it takes care of managing priorities querying the knowledge base when needed.

Allowing the presence of multiple coordination algorithms on the robots would be hard but certainly very interesting. Robots, advertising the different options they have, could be able to establish which algorithm to use, together with the machines they have to coordinate with. Unlike for the previous improvement, this would require careful considerations and a completely new design.

Not just translation

Obtaining and storing knowledge about other robots allows many other kinds of interaction in addition to the translation events described in this chapter. In order to keep the prototype simple and straight to the point we only allowed the coordination module to deal with this kind of interactions, but this does not mean it is the only possibility. We devise two important improvements that could be made.

First, robots should be allowed to coordinate with each other while using any movable part of their body. The could, for example, find themselves working with utensils next to each other at the same desk. This could lead to very dangerous collisions between the two arms if mutual presence was not taken into account. Second, more complex algorithms could leverage on knowledge about other machines, evaluating the respective capabilities, in order to ask for help on tasks it cannot complete for some reason.

This is for sure, out of the three, the most complex of the improvements. Its implementation is not only bound by issues with the design of such a complex system but also by the perception of such a system in society and in common working environments. KnowRob and the RoboEarth environment in general seem to open many possibilities and allow processes that we could not even think of before, but advances in technology also require society to be ready for change. We will discuss this issue after analyzing the conclusions of our work in the next chapter.

Conclusions

In this thesis we presented a proposal for the design of a system allowing multi-robot coordination and collaboration among knowledge-enabled machines. We first presented KnowRob, RoboEarth and Rapyuta, three projects regarding different aspects of knowledge reasoning, representation and sharing, all contributing to the goal of empowering robots that can, without fear of exaggeration, be considered smart. We proceeded with a short explanation of the material we needed to actually present our work. The architecture of the proposed system and the design of a simple prototype finally compose the third chapter.

During the first phase of our research, we focused on studying and analyzing the KnowRob framework, trying to understand how the many different projects developed around it are connected and integrated. While starting to study this very vast body of knowledge, we realized that the amount of work produced on the subject was definitely too vast to be analyzed in its entirety. We thus only proceeded in deeply studying and testing the three projects presented in the introduction.

In order to be able to carry on our first tests, we also started developing the YouBot model presented in the second chapter. As both ROS and Gazebo are still in a somewhat early development stage, we encountered many problems while developing the package. In particular, the lack of clear documentation posed quite a hard challenge, that we fortunately faced and overcame with the help of the helpful online community.

While busy with our research in the Intelligent autonomous systems (IAS) laboratory at the University of Padua, a team of students and researchers started thinking about participating in the RockIn @Work competition. This interest was mostly fostered by the participation of a student to a workshop organized by the universities that are currently designing the challenge. After listening and debating about the outcome of the workshop, we decided to use the challenge as the background for our work, as explained in chapter 2.

Having a clear view of the structure of KnowRob and being able to carry on experiments with realistic simulations, we could finally tackle the task of allowing robot coordination and collaboration through mutual awareness. We designed the architecture proposed in the third chapter while implementing the corresponding prototype, quickly iterating between design and development to obtain the best result we could.

As noted at the end of the previous chapter, the developed proof of concept is very simple. It can only be used to demonstrate that the approach suggested in chapter 3 works. It is nonetheless a very important proof of the fact that mutual knowledge can be used for coordination and collaboration during task execution. Future works might start from the notes of section 3.5, but many other are the possibilities to enhance what already done.

Bibliography

Aamir Ahmad, Iman Awaad, Francesco Amigoni, Jakob Berghofer, Rainer Bischoff, Andrea Bonarini, Rhama Dwiputra, Giulio Fontana, Frederik Hegger, Nico Hochgeschwender, et al. Specification of general features of scenarios and robots for benchmarking through competitions - RoCKIn. 2013.

Dave Beckett and Brian McBride. Rdf/xml syntax specification (revised). *W3C recommendation*, 10, 2004.

Michael Beetz, Lorenz Mosenlechner, and Moritz Tenorth. CRAM — A cognitive robot abstract machine for everyday manipulation in human environments. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1012–1017. IEEE, 2010a.

Michael Beetz, Moritz Tenorth, Dominik Jain, and Jan Bandouch. Towards automated models of activities of daily life. *Technology and Disability*, 22 (1):27–40, 2010b.

Michael Beetz, Ulrich Klank, Ingo Kresse, Alexis Maldonado, L Mosenlechner, Dejan Pangercic, T Ruhr, and Moritz Tenorth. Robotic roommates making pancakes. In *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*, pages 529–536. IEEE, 2011.

Antonio Bicchi, Rainer Bischoff, Fabio Bonsignorio, et al. Research roadmap - Technical report DR.1.3. *EURON - European Robotics Network*, 2007.

Rainer Bischoff and Tim Guhl. Robotic vision to 2020 and beyond-the strategic research agenda for robotics in europe. *European Robotics Technology Platform (EUROP), Brussels, Belgium*, 2009.

Nicolò Boscolo. *A distributed approach for safe kinodynamic motion planning in multi-robot systems, based on a Social Force Model.* Dissertation, University of Padova, Padova, 2012.

Marios Daoutis, Silvia Coradeshi, and Amy Loutfi. Grounding commonsense knowledge in intelligent systems. *Journal of Ambient Intelligence and Smart Environments*, 1(4):311–321, 2009.

Scott E Fahlman. Marker-passing inference in the scone knowledge-base system. In *Knowledge Science, Engineering and Management*, pages 114–126. Springer, 2006.

Lise Getoor and Ben Taskar. *Introduction to statistical relational learning.* The MIT press, 2007.

Guoqiang Hu, Wee Peng Tay, and Yonggang Wen. Cloud robotics: architecture, challenges and applications. *Network, IEEE*, 26(3):21–28, 2012.

Dominique Hunziker, Mohanarajah Gajamohan, Markus Waibel, and Raffaello D'Andrea. Rapyuta: The roboearth cloud engine. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA), Karlsruhe, Germany*, 2013.

Wojciech Kowalczyk. Multi-robot coordination. In *Robot Motion and Control, 2001 Proceedings of the Second International Workshop on*, pages 219–223. IEEE, 2001.

Gerhard Kraetzschmar, Walter Nowak, Nico Hochgeschwender, Rainer Bischoff, Daniel Kaczor, and Frederik Hegger. RobocCup @Work - Rule book. 2013.

Lars Kunze, Tobias Roehm, and Michael Beetz. Towards semantic robot description languages. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5589–5595. IEEE, 2011.

Séverin Lemaignan, Raquel Ros, L Mosenlechner, Rachid Alami, and Michael Beetz. Oro, a knowledge management platform for cognitive architectures in robotics. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 3548–3553. IEEE, 2010.

Douglas B Lenat. Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995.

Flavio Marcato. *Integration of Roboearth in ROS for the sharing of knowledge and robot actions.* Dissertation, University of Padova, Padova, 2012.

Cynthia Matuszek, John Cabral, Michael J Witbrock, and John DeOliveira. An introduction to the syntax and content of cyc. In *AAAI Spring Symposium: Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*, pages 44–49. Citeseer, 2006.

Boris Motik, Peter F Patel-Schneider, Bijan Parsia, Conrad Bock, Achille Fokoue, Peter Haase, Rinke Hoekstra, Ian Horrocks, Alan Ruttenberg, Uli Sattler, et al. Owl 2 web ontology language: Structural specification and functional-style syntax. *W3C recommendation*, 27:17, 2009.

Lorenz Mosenlechner, Nikolaus Demmel, and Michael Beetz. Becoming action-aware through reasoning about logged plan execution traces. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2231–2236. IEEE, 2010.

Ian Niles and Adam Pease. Towards a standard upper ontology. In *Proceedings of the international conference on Formal Ontology in Information Systems-Volume 2001*, pages 2–9. ACM, 2001.

Nils J Nilsson. Shakey the robot. Technical report, DTIC Document, 1984.

Enrico Pagello, Antonio D'Angelo, Federico Montesello, Francesco Garelli, and Carlo Ferrari. Cooperative behaviors in multi-robot systems through implicit communication. *Robotics and Autonomous Systems*, 29(1):65–77, 1999.

Dejan Pangercic, Benjamin Pitzer, Moritz Tenorth, and Michael Beetz. Semantic object maps for robotic housework-representation, acquisition and use. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4644–4651. IEEE, 2012.

Leon Sterling, Ehud Y Shapiro, and David HD Warren. *The art of Prolog: advanced programming techniques*, volume 1994. MIT press Cambridge, 1986.

Moritz Tenorth. *Knowledge Processing for Autonomous Robots.* Dissertation, Technische Universität Mnchen, Mnchen, 2011.

Moritz Tenorth and Michael Beetz. A unified representation for reasoning about robot actions, processes, and their effects on objects. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 1351–1358. IEEE, 2012.

Moritz Tenorth and Michael Beetz. Exchanging action-related information among autonomous robots. In *Intelligent Autonomous Systems 12*, pages 467–476. Springer, 2013a.

Moritz Tenorth and Michael Beetz. Knowrob: A knowledge processing infrastructure for cognition-enabled robots. *The International Journal of Robotics Research*, 32(5):566–590, 2013b.

Moritz Tenorth, Lars Kunze, Dominik Jain, and Michael Beetz. Knowrob-map-knowledge-linked semantic object maps. In *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*, pages 430–435. IEEE, 2010.

Moritz Tenorth, Alexander Clifford Perzylo, Reinhard Lafrenz, and Michael Beetz. The roboearth language: Representing and exchanging knowledge about actions, objects, and environments. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1284–1289. IEEE, 2012.

Markus Waibel, Michael Beetz, Javier Civera, Raffaello D'Andrea, Jos Elfring, Dorian Galvez-Lopez, Kai Haussermann, Rob Janssen, JMM Montiel, Alexander Perzylo, et al. Roboearth. *Robotics & Automation Magazine, IEEE*, 18(2):69–82, 2011.

Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjörn Lager. Swi-prolog. *Theory and Practice of Logic Programming*, 12(1-2):67–96, 2012.

Ringraziamenti

Questo lavoro rappresenta per me la chiusura di un percorso iniziato cinque anni fa e portato a termine con l'aiuto e il supporto di tante persone e molti amici. Non sarebbe stato possibile giungere a questo bel traguardo senza di loro, e vorrei sfruttare questa pagina per ringraziarli.

Alla richiesta di una proposta di tesi non banale, ed anzi stimolante, il Professor Enrico Pagello ha saputo propormi un lavoro particolarmente interessante e molto in linea con la mia specializzazione. Vorrei ringraziarlo per la possibilitá che mi ha dato e per il valore che ha poi posto sul mio lavoro quando, una volta completato, ha subito fatto in modo che altri lo portassero avanti per tenerlo in vita. Un ringraziamento anche ai giovani dottorandi e assegnisti del laboratorio di robotica, specialmente a Mauro Antonello, che durante l'estate non hanno mai esitato ad aiutarmi e a consigliarmi nei momenti di difficoltá.

Grazie a tutti gli amici con i quali ho condiviso questi cinque anni di universitá, specialmente Marina, Edoardo, Federica e Renato che, ognuno in maniera differente e probabilmente in modo inconsapevole, hanno reso i primi tre anni particolarmente interessanti e che, in quel periodo, mi hanno aiutato a cambiare tante cose di me che non mi piacevano.

I would also like to thank the people that made my year in Cork an unforgettable one. First, the lads and lassies of the UCC Canoe Club, one of the craziest bunch of students I have ever met. Just to cite the most innocent of my memoirs, I'll not forget spending St. Patrick's day kayak surfing in the ocean. Second, the people of the Choir Society, that despite needing some time to acquaint themselves with my presence (in typical Irish style) resulted to be the most interesting Irish friends I've met, and appreciated my R pronunciation as nobody had done before. Last but not the least, thanks to all the other international students: the excitement of meeting new people and embarking with them on any kind of incredible adventures is something I'll never forget. Citing you all would take the whole page, but you know who you are: thank you from the bottom of my heart.

It still feels incredible, but IBM also needs its own paragraph in these acknowledgments. Paul, Tony, Laura, you taught me what it means to really work in a team, to be responsible for what you do, for the code you write, for the designs you propose, and for this I can only be grateful. When I came back to Cork at the beginning of the summer and nobody was there I felt I made the wrong decision, but you made me change my mind from day one. Thank you to Ian, Eric, Nathalia, Anthony and Ken, for supporting us and leading us through the three months to a wonderful expo in Nice. Thank you for giving us this opportunity. A special thank to William, for the sincerity with which he discussed with me.

Meriterebbero ognuno un paragrafo, alcune persone che negli ultimi anni sono state particolarmente importanti. Alberto, per esserci sempre, per essere una persona incredibile. Chris, per aver passato dei momenti bellissimi a Cork, per essere una persona cosí diversa da me, ma in fin dei conti cosí simile. Mira, per essere stata al mio fianco durante la stupenda estate irlandese, per avermi fatto conoscere Berlino e per aver vissuto con me un bel pezzo di vita. Grazie.

Gli amici polesani, che mi accompagnano da un tempo ben piú lungo di quello degli studi universitari, non possono mancare a questa lista. Sono tanti e avró modo di ringraziarli di persona. Grazie, sinceramente, di avermi sopportato cosí a lungo. Grazie specialmente alle persone con le quali in questi anni ho mantenuto un rapporto personale stretto, con le quali ho scambiato idee ed impressioni, con le quali sono cresciuto.

Grazie al gruppo degli under, che ignari di quanto stavano facendo mi hanno permesso di vivere questo ultimo anno con una gran voglia di andare avanti. Mi hanno contagiato con il loro entusiasmo e la loro voglia di vincere, permettendomi allo stesso tempo di imparare tantissimo. Grazie a Roberto, Alberto, Francesco, Marco Jacopo, Dest, Ale, Fabio e Lorenzo, e un grazie forte a Massimo che con la sua difficoltá ad accettarmi mi ha costretto a riflettere su tanti aspetti del mio rapporto con loro. Grazie a Biri, il mio riferimento in tutto questo percorso. Grossi o no, andate avanti cosí.

Infine un grazie alla mia famiglia, alla mamma e al papá che sono sempre vicini e che mi supportano (e sopportano) come nessun altro, e che mi hanno guidato fin qui dandomi un esempio impareggiabile; a Francesco e a Maria che, purtroppo, devono troppo spesso sorbirsi la parte di me piú nervosa e antipatica, ma che sanno sempre dimostrarmi il loro affetto; a tutto il parentame, il migliore che avrei potuto desiderare. Questo traguardo é anche vostro.