ICT Call 7
ROBOHOW.COG
FP7-ICT-288533

Deliverable D1.2:

Knowledge representation, reasoning, and the transformation of robot skills into executable programs

January 31st, 2013
Project acronym: ROBOHOW.COG
Project full title: Web-enabled and Experience-based Cognitive Robots that Learn Complex Everyday Manipulation Tasks

Work Package: WP 1
Document number: D1.2
Document title: Knowledge representation, reasoning, and the transformation of robot skills into executable programs
Version: 1.0

Delivery date: January 31st, 2013
Nature: Report
Dissemination level: Restricted (RE)

Authors: Moritz Tenorth (UNIHB)
Michael Beetz (UNIHB)

The research leading to these results has received funding from the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement n° 288533 ROBOHOW.COG.
Contents

1 System Aspects of Knowledge
   Representation and Reasoning for Robots 5

2 Representations for Robot Knowledge 31
Summary

In this deliverable, we discuss different issues related to the representation of a robot’s knowledge and the integration of this knowledge into the task execution. The presented methods form the backbone of the representation and reasoning infrastructure in RoboHow, which serves as a kind of “semantic integration layer” to integrate information from the various information sources (visual observation of human activities, object perception, Web instructions, kinesthetic teaching, simulation games, etc) in multiple different modalities.

This deliverable consists of two chapters, each of which is constituted by a journal paper. The first part discusses the systems aspects: Which kinds of representation formalisms are suitable for robot knowledge processing, how can the formal representations be integrated with lower-level information, which capabilities does the addition of formally represented knowledge enable? The second chapter focuses on the representational aspects (i.e. how knowledge about actions, objects, the environment, the robot itself etc. be described inside the system described in the first part) on the example of complementing underspecified instructions and making them executable.
Chapter 1
System Aspects of Knowledge Representation and Reasoning for Robots

This chapter discusses requirements on knowledge representation and reasoning systems for robots. As robots have to reason about problems in the physical world based on perceived sensor information, the problems to be addressed are different from classical, purely symbolic, disembodied knowledge bases. We present different aspects of the KnowRob knowledge processing system that serves as the basis for knowledge-related tasks in RoboHow, including the core ontology, the different inference mechanisms, integration with the robot’s control system and perception methods, tools for acquiring knowledge from online sources and observation and methods for exchanging information with other robots.

A core concept of KnowRob are virtual knowledge bases that can be defined over different kinds of (possibly external) knowledge sources. To the reasoning system, they provide a query interface in terms of Prolog predicates which are, however, internally computed using various inference methods or even evaluated using queries to the robot’s perception system. This allows very elegant integration of different kinds of information into a coherent semantic representation.

The knowledge provided by KnowRob can be accessed by the robot’s executive during task execution and is used for taking control decisions and inferring suitable action parameters. This tight coupling between knowledge representation and task execution is important for achieving flexible and adaptable behavior and for translating high-level information into executable robot control programs.

This chapter is based on the following publication that has been published in the International Journal of Robotics Research:

KnowRob: A knowledge processing infrastructure for cognition-enabled robots

Moritz Tenorth and Michael Beetz

Abstract

Autonomous service robots will have to understand vaguely described tasks, such as “set the table” or “clean up”. Performing such tasks as intended requires robots to fully, precisely, and appropriately parameterize their low-level control programs. We propose knowledge processing as a computational resource for enabling robots to bridge the gap between vague task descriptions and the detailed information needed to actually perform those tasks in the intended way. In this article, we introduce the KnowRob knowledge processing system that is specifically designed to provide autonomous robots with the knowledge needed for performing everyday manipulation tasks. The system allows the realization of “virtual knowledge bases”: collections of knowledge pieces that are not explicitly represented but computed on demand from the robot’s internal data structures, its perception system, or external sources of information. This article gives an overview of the different kinds of knowledge, the different inference mechanisms, and interfaces for acquiring knowledge from external sources, such as the robot’s perception system, observations of human activities, Web sites on the Internet, as well as Web-based knowledge bases for information exchange between robots. We evaluate the system’s scalability and present different integrated experiments that show its versatility and comprehensiveness.

Keywords

knowledge representation, knowledge bases for robots, grounded reasoning methods

1. Introduction

Future service robots are expected to be robot assistants, companions, and (co-)workers (Bicchi et al., 2007; Bischoff and Guhl, 2009; Hollerbach et al., 2009) that are to perform tasks such as setting the table, cleaning up, and making pancakes. In order to understand and execute informal commands such as “set the table”, they need to infer the missing pieces of information that are not spelled out explicitly in these vague instructions. To set a table, for instance, a robot has to determine which items are needed for the meal, where in the kitchen they can be found and where they shall be placed.

We expect that this ability to infer what is meant from what is described will be a ubiquitous prerequisite for intuitively taskable future robotic agents. Most instructions given by humans are incomplete in some sense because humans expect the communication partner to have some amount of commonsense knowledge. Competently detecting these information gaps and deciding on how to obtain the missing information and how to make use of it in the task execution context will require robotic agents to have substantial bodies of knowledge and powerful knowledge processing mechanisms.

To include these reasoning techniques into the task execution, we propose to implement the robot’s control programs in a knowledge-enabled manner. This means that decisions are formulated as inference tasks which can be answered by the robot’s knowledge base. Figure 1 gives an example of a routine for fetching objects that is compactly specified as: “Find the object at its most likely storage location. If the object is inside a container, open the container in the appropriate manner using the articulation model of the container object”. When writing a plan, a programmer usually knows which information is required to take a decision and can therefore decide on the structure of these queries, like the query for the most likely object locations in the previous example. While the structure of the queries and the results is known, the actual result set will be determined based on the robot’s knowledge at execution time.

Universität Bremen, Bremen, Germany

Corresponding author:
Moritz Tenorth, Universität Bremen, Am Fallturm 1, 28359 Bremen, Germany.
Email: tenorth@cs.uni-bremen.de
Fig. 1. Example of a knowledge-enabled robot control program (left). Control decisions, e.g. where to search for an object, are formulated as inference tasks to be answered by the robot’s knowledge base.

Using queries to the knowledge base as interface allows the creation of robot plans that automatically adapt when the robot’s knowledge changes. If the robot in the example explores its environment and detects a cupboard with rice and spaghetti, this information shall be considered from that time on when computing where to search for, e.g., macaroni pasta. Additional knowledge can also lead to the selection of different inference techniques: searching for objects at places where similar objects are stored (Schuster et al., 2012) requires information about other objects in the environment, using a utility-based approach that considers the distance to the locations (Kunze et al., 2012) requires information about the robot’s position. These methods will therefore only generate hypotheses if the respective information is available. In addition to improved adaptability, knowledge-enabled plans are also less dependent on a concrete domain or environment since the knowledge about object properties, the spatial arrangement of these objects in the environment, and the inference rules that compute likely positions based on this knowledge can all be changed independently without adapting the robot plan.

However, to make use of these inference results during task execution, the robot’s knowledge base needs to be closely integrated with its sensing and acting components. Inference needs to be performed on information that was perceived from the outer world and that has been abstracted to the most appropriate level, while still maintaining the link to the original percepts. This integration, including issues such as the grounding of symbolic knowledge in sensor data or the ability to handle regularly updated (and sometimes wrong) information, is one important aspect that distinguishes robot knowledge processing from common (disembodied) knowledge-based systems.

In this paper, we present KNOWROB, a knowledge base specifically designed for autonomous robots. As part of the CRAM framework (Beetz et al., 2010a), it is integrated with the robot’s control program and its perception system to realize knowledge-enabled control programs. KNOWROB is one of the most comprehensive knowledge processing systems for robots to date, providing the most diverse set of knowledge types, inference methods, and integration with the control program (see Section 11 for a comparison with related systems). We believe that it is this combination of a variety of methods in a coherent framework that brings us closer to equipping robots with all of the knowledge they actually need for accomplishing complex realistic tasks. We do not see any single method that covers all required aspects of such a system while still being effective as part of the robot’s control program, that is, while computing answers fast enough during the operation of the robot.

The main contributions of this work are the following: we describe the KNOWROB architecture and the design decisions that resulted in a knowledge processing system that can operate in practical applications as part of the robot’s control system. The extensive KNOWROB ontology conceptualizes the robotics and household domains and provides the vocabulary to describe in a coherent format events, temporal information, actions, composite tasks, action parameters, objects, spatiotemporal information, processes, and robot components and capabilities. We propose the concept of “virtual knowledge bases” as a technique to create a symbolic layer on top of the robot’s internal data structures and its perceptual information. It allows logical inference while keeping the original data structures in the background to ensure that the abstract concepts are grounded in the lower-level data. We further present different knowledge acquisition techniques to fill the knowledge base with information derived from observations of human activities or from sources on the Internet.

To assess its performance, we consider a knowledge processing system as a resource that enables robot control programs to take better decisions. We therefore measure its quality in terms of the range of queries it can answer, the significance of the results, and the delay induced on the
The following sections give an overview of the main components and the design considerations and introduce the “virtual knowledge base” paradigm that underlies the system. We therefore exclude aspects such as the generation and execution of plans, the computation of motions, or the generation of suitable locations where the robot shall stand or where it shall put down objects. These issues are addressed by other components of our CRAM system. We then discuss general design considerations for the knowledge base that strongly influenced the overall design of the system. We start by explaining with an example scenario how the knowledge base is used in our system, since many design decisions are inferred based on the robot’s knowledge of the task execution: For example, the actual set of objects to be put away will not be known before the robot had a look at the table. The target location for these objects needs to be selected based on the types of objects, their state (e.g. clean or dirty) and the context during this step of the task (e.g. before or after a meal). If the task parameters and control decisions are inferred based on the robot’s knowledge base and belief state, they can be based on the most recent information.

In this section, we first explain use cases for a robot knowledge base. Other related papers discuss the acquisition of knowledge from the Internet (Tenorth et al., 2011) and from observations of human activities (Beetz et al., 2010b). To keep the paper to a reasonable length, we cannot provide detailed technical descriptions of every aspect, but refer to prior publications about individual components and to the documentation of the released source code for more in-depth technical information.

In the remainder of this paper we proceed as follows. We start by explaining with an example scenario how the knowledge base is used in our system, since many design decisions have been influenced by this. We then discuss design considerations and introduce the “virtual knowledge base” paradigm that underlies the system. The overall structure of a generic plan for tidying up can be described as: “For all objects on the table, put them where they belong”. The following plan in pseudo-code illustrates this example; the queries to the knowledge base are highlighted:

```
routine clear-table(table)
  forall(obj ← on-top-of(obj, table)) do:
    to-loc ← likely-location(obj)
    base-loc ← to-reach(to-loc)
    pick-up(obj)
    navigate(base-loc)
    if in-container(loc, container)
      then traj ← articulation-model(container)
      open-container(container, traj)
      put-down(obj, to-loc)
  done
```

While being very general, such an instruction is also very vague and contains lots of unspecified aspects. However, the missing information can often only be acquired during task execution: For example, the actual set of objects to be put away will not be known before the robot had a look at the table. The target location for these objects needs to be selected based on the types of objects, their state (e.g. clean or dirty) and the context during this step of the task (e.g. before or after a meal). If the task parameters and control decisions are inferred based on the robot’s knowledge base and belief state, they can be based on the most recent information.

In this example, the robot would drive to the table, detect a bottle of milk and infer that this is one of the “objects on the table” that need to be put away. It would reason about an appropriate location for the object and come to the conclusion that milk needs to be cooled and thus belongs in the refrigerator. Since the target location is inside a container, the routine for putting objects away requires information...
about the container's handle and its articulation model that are read from the semantic environment map.

Since the domain- and environment-specific aspects are factored out of the robot plans, the same task definition using the same queries can be used for different kinds of tidy-up tasks, be it in a kitchen, a supermarket or a mechanical workshop. The questions asked by the control program are always the same: Which are the objects that need to be put away? Where does this object belong? How can I open that container? What changes is the context that influences the set of results and how these results are generated (e.g., different sets of rules for inferring the target location for food, books, or hand tools).

As a result, the types and the structure of queries are known when a programmer designs a plan, which has several implications: first, queries can be optimized, e.g. by ordering them in such a way that the most restrictive part of a conjunction comes first (similar to re-ordering in database query optimization); second, the inference mechanisms that are to be used can be selected by choosing appropriate query predicates, so the programmer can decide whether expensive techniques such as probabilistic inference shall be used or not; and, finally, the internal representations of the knowledge base can be optimized for common query types using, e.g., indexing techniques.

We therefore use the knowledge base largely as a tool for information retrieval, similar to SQL (ISO/IEC, 2008) or Datalog (Ceri et al., 1989) queries, to infer values for action parameters, in contrast to other knowledge bases such as ORO (Lemaignan et al., 2010) that regularly compute the complete deductive closure of the available knowledge and whose main task is to check the properties of states, i.e. if a state falls into a certain category.

This approach results in more predictable usage patterns and often rather shallow backtracking. It can lead to a massive speed-up that makes it possible to use the knowledge base as part of the robot's control program in (soft) real-time settings, and also allows the use of expensive inference techniques for selected problems because the programmer has control over which techniques are to be used for answering a query.

### 2.2. Design considerations

In the following, we will identify several properties we consider important for a knowledge processing system to have in order to become a useful resource for an autonomous robot and explain how we address these aspects. This paper concentrates on architecture-related issues and the integration of the knowledge base with the robot control system. A knowledge processing system for autonomous service robots should fulfill the following criteria.

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.** The ask-interface is used during task execution for taking decisions and for inferring action parameterizations, while the tell-interface serves as a semantic memory, enabling the robot to reason about its experiences. Since a robot's knowledge is not static, the system must support regular updates of the knowledge base.

2. **It must operate effectively and efficiently as part of the robot's control system.** On the one hand, this refers to the integration of data obtained from the robot's components with abstract information in the knowledge base. On the other hand, this also means to generate answers fast enough not to slow down the operation of the robot. To achieve these (soft) real-time capabilities, expressiveness and effectiveness of the reasoning methods need to be carefully balanced, which often means to sacrifice expressiveness in favor of practical usability. For example, we chose a rather shallow integration of probabilistic inference techniques with the (otherwise deterministic) knowledge base (see Section 6.3 for details). While full-fledged hybrid inference would allow the flexible combination of probabilistic knowledge with all deterministic knowledge the robot has, it is also an extremely hard problem and cannot efficiently be used in a real-time context. We therefore chose the less expressive, but more pragmatic approach to read information from the KNOWROB system, perform probabilistic inference in this flat knowledge base, and transfer the results back to KNOWROB.

3. **It must provide the difference between the information provided by natural task specifications and the knowledge a robot needs for successfully carrying out a task.** For example, to correctly execute a command to flip a pancake using a spatula, the robot must know that the handle of the spatula is to be grasped, that only its blade is to be pushed under the pancake, and that the motion has to be parameterized in a way that the pancake can be lifted safely afterwards. All of this information is implicit and therefore has to be inferred by the robot. While this knowledge in between actions and objects is partly similar to affordances (Gibson, 1977), it also includes symbolic knowledge about object parts, geometric knowledge to select grasps, perceptual knowledge to recognize the objects, as well as experience- and action-related knowledge to choose suitable parameters. Besides symbolic knowledge, a robot also needs continuous-valued information: it is often not sufficient to infer that an object can be reached, but the robot needs coordinates to drive to in order to reach it. All of these pieces of information are interrelated and need to be combined to answer the robot’s queries, so they need to be represented in a coherent format.

4. **It has to provide an encyclopedic knowledge base that defines and specifies an appropriate conceptualization of the information needed for autonomous robot control.** The conceptualization differs from other conceptualizations, which have primarily been created for natural
language interpretation, in terms of which concepts are included and how they are described. Upper ontologies such as Cyc (Lenat, 1995) and SUMO (Niles and Pease, 2001) specify eggs as products of birds and fish and maybe include their nutrition facts, but lack knowledge that is necessary for manipulation such as the information that eggs break easily. A knowledge representation for robots must include such action-related knowledge. Thus, knowledge representations for autonomous robots must be particularly rich in the way they represent actions, events, processes, situations, action effects and consequences, failures, knowledge preconditions of actions, etc. More recent approaches to automatically extract ontologies from the Web, such as NELL (Carlson et al., 2010) and TextRunner (Banko et al., 2007), or from encyclopedias such as Wikipedia, e.g. YAGO (Suchanek et al., 2007) and DBpedia (Auer et al., 2007), share the same problem because this kind of knowledge is often so obvious to humans that it is not spelled out explicitly and is therefore not on the Web. Attempts to collect this commonsense knowledge from Internet users by questionnaires (Gupta and Kochenderfer, 2004) are useful for some aspects, but are still neither comprehensive nor deep enough.

5. It must provide the robot with self-knowledge. Self-knowledge includes knowledge about the robot’s body with its sensors and actuators, its capabilities, actions, and plans. Given an action specification, the robot should be able to decide whether it is capable of performing this action and, if a capability is found to be missing, if it can be obtained in any way. In contrast to work on bottom-up learning of sensor models (Pierce and Kuipers, 1997) or of the robot’s kinematic configuration (Sturm et al., 2009) without initial knowledge, we try to make use of existing information whenever possible. For many robots, detailed engineering models exist that describe their kinematic structure very precisely (Arnaud and Barnes, 2006; Meuissen et al., 2009). Kunze et al. (2011) proposed a method to combine the geometric descriptions with semantic annotations to make them accessible for abstract reasoning.

6. It must make the robot knowledgeable about its actions. This includes forward models to predict the outcome of an action as well as declarative specifications of an action’s prerequisites and effects. These models are to support the robot with planning its actions, with projecting future world states, and with reasoning about the changes created by actions (Tenorth and Beetz, 2012b). They need to include knowledge about how actions are to be performed, which failures can occur, and how these can be detected and resolved. This aspect is related to work on action planning, but most planning systems focus on determining a sequence of actions to perform to attain a specified goal state, while the problems of determining how to perform the actions and how to determine the goal state from incomplete instructions often require further inferences and additional knowledge.

7. It must make the robot capable of using its control system and perception system as knowledge sources. Often, information that is needed for abstract reasoning is already available in some form in the robot’s internal data structures, such as the robot’s pose estimate, or can be acquired from its components, such as the perception system. To re-use this information, the robot can “listen” to the control program and log the dynamic data structures as a dynamic and virtual knowledge base (Mösenlechner et al., 2010). The knowledge processing system thus acts as a kind of “parasite” that is reusing and abstracting data structures which were originally created for action execution for the purpose of reasoning. Since the knowledge is generated just in time from the data structures that are used for controlling the robot, the abstract representations are inherently grounded in these data structures.

8. It needs to provide methods for (semi-)automatically acquiring and integrating knowledge from different sources. We expect that manually encoding the vast amounts of knowledge needed to scale towards realistic applications will not be feasible, so tools to translate and import existing knowledge will be highly important. Possible knowledge sources include human activity observation and interpretation (Beetz et al., 2010b), logging of robot activities (Mösenlechner et al., 2010), sources on the Web such as task instructions and product catalogs (Tenorth et al., 2011), collections of commonsense knowledge such as the Open Mind Indoor Common Sense database Kunze et al. (2010), and knowledge sharing techniques for robots such as the RobodgEarth system (Waibel et al., 2011). To ensure high-quality data, some degree of human supervision and adaptation will probably be necessary for aligning imported ontologies, correcting mistakes in task descriptions, and for selecting data sources to be imported; however, we expect this to be much less effort compared with completely manual knowledge acquisition.

9. It should exploit problem properties to make inference tractable. Many inference problems a robot needs to solve are hard, if not infeasible, to solve in their most general form. While many of these problems can be described in expressive formalisms such as Markov logics (Richardson and Domingos, 2006) or partially-ordered Markov decision processes (POMDPs) (Kaelbling et al., 1998), they can often be solved more efficiently using specialized inference methods that exploit specific problem properties. We therefore combine general-purpose inference methods with special-purpose inference and procedural attachments, and use Prolog as an interlingua. The query-oriented processing
and recurring inference schemata allow the creation of optimized data structures that support fast inference on common problems.

2.3. The virtual knowledge base paradigm

In KNOWROB, the abstraction of data into symbolic concepts is not performed before the data enters the knowledge base, as it is the common approach in literature (Daoutis et al., 2009; Lemaigran et al., 2010). Instead, the abstract representation is computed on demand once it is needed to answer a query. This allows us to perform abstraction based on the most current information (at query time), and up to the level that is most appropriate in this situation. We do this by enabling the knowledge base to compute information on demand, either using information that already exists in the knowledge base or by forwarding queries to other robot components which are expected to provide better answers.

The perception system, for example, can provide the most recent information about which objects are currently in front of the robot. By including such external knowledge sources, the robot’s knowledge is extended beyond the content of its knowledge base at query time to further include those statements that can be computed from the available knowledge or that can be acquired from external sources.

We call those parts of the knowledge that are not explicitly stored but computed on demand “virtual knowledge bases”. The outer world can be regarded as a “virtual knowledge base” to which the robot can send “queries” in terms of perception tasks. Also the robot’s internal data structures can serve as virtual knowledge bases: much of the information that is needed for inference is already available in some form in the data structures that are used for controlling the robot. This information is integrated into the reasoning process such that the symbolic knowledge is transparently extracted from the underlying data structures.

Let us consider the example in Figure 2. The robot has an estimate of its position in form of a probability distribution, but it may need to evaluate it in terms of symbolic concepts, e.g. when a decision depends on the reliability of the current position estimate. In KNOWROB, this is realized using procedural attachments to classes and properties in the knowledge base, called “computables”; that describe how to evaluate the respective concept on the underlying data structures. Since the abstract concepts are directly computed from these data structures, they are inherently grounded, i.e. the abstract and the sub-symbolic representations are closely coupled and cannot diverge. Section 6.2 describes in more detail how computables are integrated into the inference process.

This concept is similar to the concept of views in database design which can be defined to provide simplified access to complex data tables. In our case, the symbolic knowledge is an abstract view on the original data structures that allows logical reasoning. One consequence of this approach is that we do not assume to always have a complete axiomatization of the world in the knowledge base since much information will only be computed when an inference process asks for it.

3. The KnowRob knowledge processing system

The knowledge content and functionality of the KNOWROB core system can be extended with different modules. Figure 3 gives an overview of the system structure. Its central component is the knowledge base that provides the mechanisms to store and retrieve information about actions, objects, processes, temporal events, their properties, and relations. We use the open-source SWI Prolog (Wielemaker et al., 2012) as central knowledge store. The KNOWROB ontology is represented in this core system and provides the general vocabulary and representation into which other representations and inference methods can be incorporated.

The core system can be augmented with extensions that provide different kinds of functionality. First, they can contain additional knowledge in terms of “micro-theories”, which are extensions of the KNOWROB ontology that provide additional vocabulary and predicate definitions. Micro-theories are for example used to adapt the system to a novel application domain, or to include new kinds of information such as measurement units. Second, there are extensions of the reasoning system, providing new inference techniques. They specify how queries have to be described, how context is represented, how the results are returned, and which procedures are to be called to perform the
The KNOWROB system provides several components for knowledge acquisition and representation, for reasoning about this knowledge, and for grounding it in the robot’s perception and action system.

actual inference. Third, there exist extensions for knowledge acquisition, i.e. for filling the knowledge base with information extracted from other sources. Further extensions provide interfaces to the robot’s control system to update the belief state inside the knowledge base and to offer knowledge and reasoning services. Tools for visualizing knowledge and communicating with humans allow introspection and debugging of the knowledge base’s content.

4. Prolog as an interlingua for robot knowledge processing

KNOWROB uses Prolog for storing and reasoning about the robot’s knowledge. There are various logical languages for describing first-order relations, ranging from generic predicate logics (Frege, 1879) over logical programming languages such as Prolog (Sterling and Shapiro, 1994) to languages that combine logical with probabilistic representations (Getoor and Taskar, 2007). All these logical dialects differ in what they can describe and how elegantly different kinds of facts can be expressed. Selecting a representation with the right expressiveness is an important design decision for a knowledge representation and reasoning system. In general, simpler and less expressive representations such as RDF (Becket, 2004) or OWL-lite (W3C, 2009) allow more efficient reasoning, often guaranteeing desirable properties such as decidability. Their drawback is that more complex relations may not be expressible in these languages, or at least not in an elegant way. On the other hand, very expressive representations such as CycL (Matuszek et al, 2006), Scone (Fahlman, 2006), or Topic Maps (ISO/IEC, 1999) are able to model almost everything that can be expressed in natural language, but often have poor support for automated inference.

In KNOWROB, we have chosen Prolog as a language of medium expressiveness which combines a procedural interpretation with a declarative reading that enables a programmer to inspect the content of the knowledge base. The possibility to write procedures in Prolog can massively speed up inference and extend the range of what can be computed beyond pure logical inference. It also facilitates the integration of external information sources or reasoning engines. Another advantage is that Prolog is a rather widely used standardized language for which textbooks, course material, and industrial-strength implementations exist. Since most inference tasks in KNOWROB are queries for action parameters, our use of Prolog is in some way similar to query languages such as SQL (ISO/IEC, 2008) or Datalog (Ceri et al., 1989). This means that many use cases do not require very deep backtracking and complex search procedures, which is one of the reasons why Prolog can be used efficiently in the robot’s control loop.

5. Encyclopedic knowledge for robots

Encyclopedic knowledge is the kind of knowledge commonly found in a dictionary: definitions of types of actions and objects, such that cups are a kind of container, have
a handle, and are used for drinking liquids. From a systems point of view, encyclopedic knowledge provides a common, formal, well-defined vocabulary for representing knowledge that can be used by the different components of the robot. Sharing a common representation is necessary to make sure that information generated by one component can be used and interpreted by a different component.

The encyclopedic knowledge is complemented by commonsense knowledge that provides additional information that most humans immediately associate with the concepts, such as the fact that cups can break or that coffee may be spilled if a full cup is moved too quickly. Since these facts appear to be so obvious, they are normally not spelled out explicitly. There are, however, some initiatives such as the Open Mind Indoor Common Sense (OMICS) project that try to collect this knowledge from voluntary Internet users. Kunze et al. (2010) describe techniques to convert this information from natural language into a logical representation that can be used in a formal knowledge base such as KNOWROB.

In KNOWROB, we chose Description Logic (DL) as a formalism to represent encyclopedic and commonsense knowledge, in particular the Web Ontology Language (OWL) (W3C, 2009), which stores DL formulas in an XML-based file format. OWL was originally developed for representing knowledge in the Semantic Web (Lee et al., 2001), but has since become a commonly used knowledge representation format. An extensive overview of the concepts behind DL can be found in Baader et al. (2007), with a shorter introduction given by Baader et al. (2008). The OWL files are loaded into the Prolog-based representation and can be queried using Prolog predicates (see Section 6.1 for details on the internal representation of OWL triples in KNOWROB).

DL distinguishes between terminological knowledge, the so-called TBOX, and assertional knowledge, the ABOX. The TBOX contains definitions of concepts, for example the concepts Action, SpatialThing, PickingUpAnObject or TableKnife. These concepts are arranged in a hierarchy, a so-called taxonomy, using subclass definitions that describe for instance that a TableKnife is a specialization of SilverwarePiece. The ABOX contains individuals that are instantiations of these concepts, e.g. knife1 as an instantiation of the concept TableKnife. In robotic applications, the ABOX typically describes detected object instances, observed actions and perceived events. The TBOX, in contrast, describes classes of objects and types of actions.

Roles describe the properties of an individual or the relation between two individuals. They can be used in concept definitions to restrict the extent of a class to individuals having certain properties. For example, the concept OpeningABottle can be described as a subclass of OpeningSomething with the restriction that the objectActedOn has to be some instance of a Bottle:

\[
\text{OpeningABottle} \subseteq \text{OpeningSomething} \cap \exists \text{objectActedOn}.\text{Bottle}
\]

A knowledge representation that consists of a taxonomy of concepts and relations between these concepts is called an “ontology”. The layout of the upper levels of the KNOWROB ontology, including many classes, their hierarchy, and properties, has been adopted from the OpenCyc ontology (Lenat, 1995). OpenCyc has emerged as quasi-standard for robot knowledge bases to which we remain compatible, facilitating the exchange of knowledge with other systems. There are also many tools and links between OpenCyc and other knowledge bases, for example the links to the WordNet lexical database that were used by Tenorth et al. (2010b). These links can help to integrate knowledge from different sources that refer to different upper ontologies.

However, since OpenCyc was developed as a general upper ontology with the intention of understanding natural language, it covers a broad range of human knowledge, but often lacks domain-specific knowledge needed by robots, for example in the areas of mobile manipulation and human everyday activities. We therefore extended Cyc with more detailed descriptions of concepts such as everyday tasks, household objects and robot parts. Figure 4 visualizes the uppermost levels of the KNOWROB ontology. The most important branches are the TemporalThings, containing descriptions of Events (yellow) and, as an important subclass, Actions (green), and the SpatialThings (blue), describing abstract spatial concepts such as Places and object classes. Most objects in the robot’s environment as well as pieces of furniture or body parts are subsumed under the HumanScaleObject class. Other notable branches are MathematicalObjects such as Vectors or CoordinateSystems (red).

6. Inference mechanisms in KnowRob

KNOWROB uses a hybrid system architecture that combines multiple general- and special-purpose inference methods. At first glance, it may look more elegant to use a single very expressive technique for storing all of the knowledge and drawing all inferences. Unfortunately, there is not one universal technique that is equally suited for all kinds of problems and that is at the same time expressive enough to provide the robot with useful information, scalable enough to store large amounts of knowledge, and fast enough to be effective as part of the robot’s high-level control loop. Statistical relational models Getoor et al. (2007) such as Markov logic networks (Richardson and Domingos, 2006) and Bayesian logic networks (Jain et al., 2009) combine probabilistic and deterministic information in a single knowledge base, but learning and inference in these models is usually too complex to use them as the sole representation method for a large-scale robot knowledge base. On the other hand, many problems in robotics require probabilistic representations, so a completely deterministic representation will be restricted in many senses.
We therefore decided to integrate multiple techniques into a coherent framework and to use the more computationally expensive ones only for selected problems. Since most inference in KNOWROB is performed to answer queries that have been designed by a programmer along with the robot plan, it is possible to select which inference techniques shall be used for which part of the query, and to optimize its structure for fast inference. In the system, the DL knowledge base serves as conceptualization of the domain and backbone for all inference methods. Other inference techniques are embedded into it, i.e. they accept parameters described in terms of the concepts in the DL knowledge base, and return their results in this format as well. Computable classes and properties allow the transparent injection of externally-computed results into the DL inference procedure. The probabilistic inference methods can read evidence data from the DL knowledge base; they can either be called using specialized query predicates (outside of the DL infrastructure) or be embedded using computables.

6.1. DL inference using Prolog

Many inferences in DL can be reduced to the problems of classification (computing which classes are sub-classes of other classes) and realization (computing the most specific classes an individual belongs to). These tasks are usually performed by DL reasoners such as Racer (Haarslev and Müller, 2001), Pellet (Sirin et al., 2007), or HermiT (Shearer et al., 2008) that maintain a fully classified knowledge base in memory. What is problematic for robotics applications is that changes in the knowledge require a re-classification that can take significant time for large knowledge bases. Since a robot’s knowledge base constantly changes, this can lead to severe scalability issues. On the other hand, many of the inferred relations will never be needed and therefore should not be computed after all.

We therefore opted for a rule-based inference method using Prolog: OWL statements are internally represented as Prolog predicates to which the common Prolog inference methods can be applied. OWL inference schemata
are implemented as Prolog predicates that can for example check whether an individual complies with all restrictions defined on a class. Since the search-based inference in Prolog is not affected by changes in unrelated parts of the knowledge base, it gracefully handles updates to the knowledge. External information from “virtual knowledge bases” can easily be included into the inference as new branches in the solution tree that are automatically covered by Prolog’s backtracking mechanisms.

The implementation in KNOWROB is based on the SWI Prolog Semantic Web library (Wieliemaker et al., 2003) for loading and storing RDF triples, and the Thea OWL parser library (Vassiliadis et al., 2009) that provides OWL reasoning on top of these representations. An efficient internal triple store forms the backbone of the Semantic Web library. It combines several indexing schemes (by subject, predicate, object and combinations thereof), and scales very well up to large knowledge bases (according to Wieliemaker (2009) up to approximately 300 million triples on a computer with 64 GB of memory). A set of increasingly complex query predicates operates on top of the internal triple store and adds functionality with each layer. The advantage of having this stepwise increase in complexity is that one can select how much inference is to be included by choosing the appropriate predicate. In particular, computables that forward queries to components such as the perception system can slow down the inference process and should be included with care. Often, it may for example be sufficient to query the perception system only once in the beginning of the query, while later parts that, e.g., check properties of the perceived objects operate on the (then updated) static content of the knowledge base. The following hierarchy of predicates allows to query for individuals and their properties. We have extended the built-in set of predicates with the rdf_triple predicate for including computables into the inference procedure.

- rdf(S, P, O) returns only exactly matching triples from the internal triple store.
- rdf_has(S, P, O) also takes the subPropertyOf relation into account, returning matches for all specializations of P.
- rdf_reachable(S, P, O) further considers transitivity of the predicate P.
- rdf_triple(P, S, O) additionally includes results generated by computable classes and properties.
- owl_has(S, P, O) also returns results of the OWL inference process (e.g. the class membership of an individual inferred from OWL restrictions).

An important difference to common DL reasoners is that KNOWROB adopts Prolog’s closed-world assumption: everything that is not known to be true is assumed to be false, whereas the usual DL semantics make the open-world assumption that everything that is not explicitly known to

be false is considered to be true. This proved to be useful for the implementation of robot programs in which the non-availability of a piece of information is by itself important information. With closed-world semantics, the non-existence of something does not have to be extensively described but can be concluded from the fact that its existence cannot be proven (negation as failure). For example, if the robot has to decide whether it can perform a task or if some component is missing, it can simply check whether all required components are known to be available and decline otherwise.

6.2. Computable classes and properties

Computables are procedural attachments to OWL classes or properties. They are realized as hooks into the DL-based inference to which external reasoning methods can be attached. Both the arguments they accept and the results they generate are part of the DL knowledge base which results in a complete and transparent integration into the DL inference. There are two kinds of computables: Computable classes, which create instances of their target class, and computable properties, which compute relations between instances.

Computables can be attached to any OWL relation without changing its semantic description. In Figure 5, the relation objectActedOn that links an ActionOnObject to a SpatialThing is modeled as usual by its domain and range. One or multiple computable properties, which are linked to the property to be computed by their target property, can define how to compute this relation by specifying a binary Prolog predicate as command. Attaching computables via the target property separates the semantic properties of the OWL relations to be computed from implementation aspects that define how these relations are finally computed. It also contributes to the modularity of the system because computable definitions can easily be loaded and unloaded during runtime.

An important use case of computables is the computation of new statements from existing information, for instance
of inference methods that have been developed for these underlying representations.

While the combination of high expressiveness with the representation of uncertainty makes statistical relational models well-suited for the uncertain, partially observable, and dynamic environments autonomous robots are acting in, inference in such models can become very hard, often even intractable, when the models get large and contain complex relations between a lot of instances. Exact inference is rarely tractable, and though there are various approximate inference techniques (see Bishop (2006) for an introduction), their applicability depends on the inference problem and the model structure. For these reasons, statistical relational models are in KNOWROB only used for selected problems that require the representation of uncertain relations. It remains the task of the programmer to design and train a suitable model, select and test an appropriate inference algorithm, and specify in the queries that the probabilistic model shall be used.

The implementation of the statistical relational learning methods in KNOWROB is realized by integrating the PROBCOG library. PROBCOG provides a variety of learning and inference algorithms for both Markov logic networks and Bayesian logic networks that are integrated in a common framework. Figure 6 describes how the integration is realized technically. The statistical relational models in PROBCOG each form a probabilistic knowledge base that can be accessed from KNOWROB. Integrating these knowledge bases with KNOWROB is a non-trivial task: the system needs to determine which statistical model is to be used to answer which query (a predicate may appear in different models, although some may be more appropriate for inferring its value). Predicates in both knowledge bases may have different identifiers and semantics, and the inference results need to be processed in some way to be used in the KNOWROB context, for example by selecting the most likely solution or by iterating over all results sorted by their probabilities. In the proposed solution, these aspects can be defined in a flexible way using Prolog predicates. Each PROBCOG model has a corresponding KNOWROB module in which the model designer can encode which queries can be answered, how predicates are translated between the models, which model is to be used, and if open- or closed-world inference shall be applied. In a simple case, the mapping between both sets of predicates may be trivial and a default model can be used for all inferences. In more complex cases, the mapping can become quite complex, and the selection of a suitable model can take the set of query predicates and the execution context into account.

Queries from KNOWROB are sent using the probcog_query predicate (upper left block in Figure 6). Before starting the inference, the abstract statistical relational model is instantiated with a concrete domain, and evidence values are set. This information is read from KNOWROB using wrapper predicates that translate the predicates in the PROBCOG model into queries to the
Fig. 6. Integration of the ProbCog inference system into the KNOWRob knowledge base. Two sets of query predicates translate between the predicates in both systems, define parameters for the probabilistic inference and determine how to proceed with the results.

**7. Interfacing perception**

Whenever robots interact with objects, they need to reason about their properties, infer where to find them, how to manipulate them, or where to put them. Taking these decisions requires a robot to apply its abstract knowledge about object types to the physical objects present in the environment. The knowledge base thus needs to be linked to the perception system in order to be informed about which types of objects have been detected at which poses.

Different kinds of robot perception systems exist that are interfaced in different ways: some perception methods perform recognition on demand, others operate continuously on the incoming streams of sensor data. In the former case, the communication with the knowledge base is performed synchronously using a request–response-based scheme, in the latter one asynchronously by passively listening to the published object detections. Methods for realizing these interfaces can be found in the knowrob_perception package.

Figure 7 visualizes the two kinds of interfaces. In the case of synchronous communication (upper left), the integration...
with KNOWROB is solved using computables: a computable class with an appropriate target is defined which is called automatically whenever a query involves objects of that respective type. The computable then sends a request to look for this kind of object to the perception system, and creates the object representation for all detected objects returned in the result set. While the target of the computable is often a rather generic class, which can be useful to detect several different kinds of objects at once, the resulting object instances will have the specific types determined by the object recognition system. This kind of interface is for example used to read information from the tabletop_object_detector by Willow Garage.

The second kind of interface, using asynchronous communication, listens to all perception results and adds them to the knowledge base. This listener is running in a separate thread in parallel to the KNOWROB engine, receives all object detections that are published on the topic and creates the respective perception instances for them (lower left block in Figure 7). This second kind of interface is for example used to read information from the CoP vision system (Klank et al., 2009), the K-COPMAN system (Pangerlic et al., 2010), and the ROBOEARTH object recognition component (Marco et al., 2012).

In addition to the poses of recognized objects, the knowledge base should also provide information about which objects can be recognized at all, namely those for which the robot has appropriate recognition models. KNOWROB maintains an internal representation of the set of available object recognition models that allows to reason about which objects can be recognized. Being aware of these recognition capabilities is important to assess whether a plan can be executed successfully or whether additional models need to be acquired.

Note that KNOWROB does not trigger perception mechanisms that require the robot to move. We decided that only the robot's executive is allowed to call actions that change the physical state of the robot since it can best assess whether side-effects of an action would interfere with another action that is being executed. For example, the robot should not be commanded to look away while performing an important manipulation task. While such active perception actions cannot be actively started by KNOWROB, it can still process the results of actions started by the executive (if they are published in a way that they are accessible) and include them into the knowledge base.

8. Acquiring knowledge

The previous sections presented KNOWROB’s knowledge processing infrastructure and its representation and reasoning methods. In order to use these methods for robotic applications, one has to fill the representation with content and provide the vast amounts of knowledge a robot needs to competently accomplish everyday manipulation tasks. Manually encoding this knowledge for large-scale systems that are to operate under real-world conditions would be a daunting task. We therefore explore methods for automating the acquisition of knowledge from existing sources.

One useful resource are Web pages, originally created for humans, from which information can be extracted. For example, wikihow.com and ehow.com provide thousands of
step-by-step instructions for all kinds of everyday tasks from cleaning up to setting a table. Recipe databases such as epicurious.com contain a large number of cooking recipes. On-line shops provide information about the appearance and properties of objects. Enabling robots to make use of these knowledge sources will help them to scale towards realistic tasks in natural human environments more quickly. In this section, we will briefly describe some techniques realized in KNOWROB and refer to the respective publications for more detailed descriptions.

8.1. Import of natural-language task instructions

Tenorth et al. (2010b) describe a method for translating natural-language task instructions into robot plans that consists of the following steps. First, the sentences are parsed using a common natural-language parser (Klein and Manning, 2003) to generate a syntax tree of the instructions. The branches of the tree are then recursively combined into more complex descriptions to create an internal representation of the instructions describing the actions, the objects involved, locations, time constraints, the amount of ingredients to be used, etc. The words in the original text are resolved to concepts in the robot’s knowledge base by first looking up their meanings in the WordNet lexical database (Fellbaum, 1998), and by then exploiting mappings between WordNet and the Cyc (Lenat, 1995) ontology. Large parts of the KNOWROB taxonomy are compatible with the Cyc ontology, so these mappings can directly be exploited to link natural-language words to concepts in KNOWROB.

The automated import of task specifications helps to quickly acquire formal descriptions of the actions and objects needed to perform a task. Although the often similar structure of instructions as imperative statements facilitates the automated processing in many cases, a complete understanding of these instructions would require a solution to the (very difficult) general natural language understanding problem. Especially long sentences and complex grammatical structures still cause problems in the initial parsing step as well as during subsequent processing stages. We therefore consider these methods mainly as an aid for programmers to quickly generate a first draft specification from existing sources which may then be extended and debugged. In this case, the programmer can also select which Web sites and which instructions are to be imported to ensure data quality. The source code of the system is available in the comp_ehow package.

8.2. Generation of product knowledge bases

To execute these abstract task descriptions, a robot needs to ground the contained object references to physical objects in its environment. This requires information about their appearance for recognition purposes, but also information about their properties to locate them in the environment and handle them correctly. Information about both the appearance of objects and about their properties can be extracted from on-line shops, whose product catalogs list large numbers of objects with pictures and a structured description. Pangercic et al. (2011) describe methods for recognizing objects based on the product pictures, Tenorth et al. (2011) describe the creation of a product ontology from an online shop. Some manual adaptation is often required to align the generated ontology with the existing one, but at least for the use cases where detailed branches of existing classes are created, this only consists of adding a few sub-class assertions.

8.3. Extracting knowledge from observations of humans

While the World Wide Web is very useful for information that can be expressed verbally, other kinds of knowledge that are more situated (e.g. locations in a specific environment) or continuous-valued (e.g. motions) need to be obtained from other sources. Often, observations of humans that perform similar activities in the environment can provide such information. From their motions and the objects they interact with, a robot can learn about how they perform a task, which locations they use, or how they move during an action. Figure 8 exemplarily shows information that can be extracted, such as human poses, segmented and classified hand trajectories, locations that play a certain role in a task, or objects a human interacts with. To integrate the observations with other knowledge, it is however necessary to set them in relation to the symbolic information in the robot’s knowledge base.

Beetz et al. (2010b) present techniques for tracking human motions and for segmenting them into fragments that are represented as instances of action classes in the robot’s knowledge base. This formal action representations can be abstracted into hierarchical task models by applying knowledge about the composition of actions, creating a structured and semantically annotated representation of the observed activities. Since the observations are described using the same ontology that is also used for the robot’s actions (as instances of the action classes used for robot
9. Exchanging knowledge

Despite the efforts presented in the previous section, the acquisition of knowledge remains a complex procedure that can often only partly be automated. However, once a piece of knowledge is available in a format that robots can understand, it should be feasible to exchange it and make it usable for other robots. Establishing a Web-based knowledge base that can be used for the exchange of information between robots is the goal of the RoboEarth project (Waibel et al., 2011). The KnowRob knowledge base is a central component of that system providing knowledge representation and reasoning services. The libraries for interacting with the RoboEarth system are also available as open-source software.

Expressive representations are needed to make the autonomous exchange of information possible: first of all, they need to encode the information to be exchanged, i.e. the description of an action or an object class. In addition, they also need to support robots with finding and selecting information: before downloading a task description, the robot should make sure it has all required capabilities and check whether it needs additional components such as object recognition models. Automating this information selection process requires formal representations and annotations that describe which conditions need to be fulfilled for a piece of information to be usable. A robot can match the requirements of a task specification in RoboEarth against a formal model of its own components and capabilities to verify that all required components are available. If software components are missing, the robot can for instance load more detailed task specifications, object models, or an environment map. While this process cannot guarantee successful task execution, which can still fail for various reasons, it can at least compute whether the robot has all required information based on all information about the task that is available.

10. Evaluation and experiments

Evaluating a knowledge processing system for robots is a complex task for which there are neither established benchmarks nor evaluation methods so far. Using knowledge processing techniques on a robot has its specific challenges which are often different from those encountered in classical, purely symbolic reasoning problems commonly studied in artificial intelligence research. Parts of these challenges are related to using the methods on a physical system, other challenges are posed by the complex scenarios.

Since the robots act in the physical world, their knowledge bases have to operate in real-time and compute results fast enough not to slow down the other components of the robot. The symbolic knowledge base needs to be grounded in the robot’s sensing and actuation methods to enable the robot to reason about objects it has perceived and about actions it has performed. Abstractly defined actions in the knowledge base need to be linked to parameterizations of executable action components on the robot, and the robot must be able to deal with the uncertain and incomplete information provided by its sensors. The application in complex realistic scenarios is challenging in terms of scalability, expressiveness, comprehensiveness, and usefulness.

The knowledge representation needs to be scalable enough to handle a significant amount of knowledge both on the class level, e.g. for describing a large number of object types, and on the instance level, e.g. to store all observations of objects recorded over an extended period of time.

Capturing all of these aspects in a few aggregated numbers is difficult. We therefore evaluate the quality of the system by a combination of different methods: a quantitative evaluation is performed for the scalability of the system and its response time. Three complex usage scenarios are described to show how the different components of our system contribute to solving realistic tasks. Afterwards, we compare the developed system with other robot knowledge bases in the literature with respect to their features and discuss limitations of our current implementation.

10.1. Scalability and responsiveness

To assess the system’s scalability we performed tests with the internal object representation. The representation of objects is one of the more complex descriptions since it consists of the object instance and, for each detection of the object, a $4 \times 4$ pose matrix and an instance describing the perception event. At the same time, it is one of the most important ones since novel detections of objects are the kind of information that continuously comes in during operation of the robot. We created a test program that creates a large number of object detections in a loop, corresponding to thousands of (virtual) perceptions of this object.

We tested the system with up to 65,000 perception instances that correspond to more than a full week during which the robot continuously detects an object every ten seconds. Figure 9 visualizes the time needed for the creation of new instances (blue square markers). It scales linearly with their number, the 65,000 object detections can be created in about 2.88 seconds. The maximum frame rate for creating object instances is thus about 22,000 detections per second, corresponding to 170,000 triples per second. As a comparison, the paper about the ORO knowledge base (Lemaignan et al., 2010) gives a rate of 7,245 triples per second.

The orange triangles describe the time needed to query for the latest detection of an object. For these tests, we have
used a naïve implementation that attaches a set of object detections at different points in time to an object instance. While this is the straightforward way of describing the relation in a logical form, an optimized implementation that stores the perceptions in a linked list with a pointer to its head can reduce the time to query for the latest detection to a constant time of 10–20 ms. This shows how much potential the optimization for common query patterns, such as the most recent detection of an object, has.

The 65,000 object perceptions correspond to about 490,000 triples in the knowledge base. We have tested up to about 7 million triples on a common laptop computer (Intel Core 2 Duo P8600, 2.4 GHz, 4 GB PC3-8500 RAM, Ubuntu 10.10 32 bit) without noticeably slowing down neither other programs nor the creation of new triples. The 490,000 triples increase the memory consumption of the whole Prolog process from 16.5 to 47.8 MB. According to the authors of the underlying triple store, it scales to several hundred million triples on high-end commodity hardware (Wielemaker et al., 2012).

While these experiments show that reasonable amounts of knowledge, as needed for extended laboratory experiments, can be stored and processed without problems, the scalability of the system is currently limited to what a single computer can handle. There will also be inference tasks that are less scalable than the underlying triple store. Spatial and temporal indexing techniques will help to perform more efficient reasoning about large amounts of knowledge. A hierarchy of memory levels like an in-memory short-term memory and a disk-based long-term memory will increase scalability in terms of storage. When information is moved to the long-term memory, it can also be compressed, for example by only remembering when information has changed. To scale further, cloud-based knowledge bases can be added as another layer of background knowledge, as we started to investigate in the ROBÔEARTH project (Section 9).

### 10.2. Knowledge-enabled task execution

In an experiment presented by Beetz et al. (2011), two robots made pancakes using instructions from wikihow.com for generating a robot plan. The KNOWROB knowledge processing system was used for generating the plan from the Web instructions and for grounding the abstract instructions in the robot’s perception and action system. In order to execute the instructions, the actions have been mapped to plans on the robot, the object specifications have been resolved to physical objects, and the robot decided where to search for these objects.

#### 10.2.1. Generating plans from instructions from the Web

The instructions specified the ingredients, milk and prepared pancake mix, and a sequence of action steps:

1. take the pancake mix from the refrigerator;
2. add 400 ml of milk (up to the marked line) shake the bottle head down for 1 minute; let the pancake-mix sit for 2–3 minutes, shake again;
3. pour the mix into the frying pan;
4. wait for 3 minutes;
5. flip the pancake around;
6. wait for 3 minutes;
7. place the pancake onto a plate.

Using the techniques for translating instructions into robot plans described earlier, a formal task specification was generated, which in turn was transformed into the following robot plan:

```clpl
(def-top-level-plan shewmake-pancakes1 ()
  (with-designators {
    (pancake (an object (type pancake))
             (on (frying-pan)))
    (mixforbakedgoods2 (some stuff
        (type pancake-mix)
        (in (refrigerator2))))
    (refrigerator2 (an object (type refrigerator)))
    (frying-pan (an object (type pan)))
    (dinnerplate2 (an object (type plate)))
    (location0 (a location (on (dinnerplate2))))
    (for (pancake2)))
  )

  (achieve '(object-in-hand ,mixforbakedgoods2))
  (achieve '(container-content-transfilled
    ,mixforbakedgoods2 ,frying-pan))
  (sleep 180)
  (achieve '(object-flipped ,pancake))
  (sleep 180)
  (achieve '(loc ,pancake ,location0)))
)
```

The code above shows the sketchy plan in the CPL plan language (Beetz et al., 2010a) that was generated from the Web instructions. The declaration part creates entity designators for the objects referred to in the instructions. A designator consists of an article (definite or indefinite), an entity type (object, location, stuff, etc.) and a set of attribute–value pairs. Stuff refers to homogeneous things, such as water, milk, etc. The backtick operator indicates that the following block shall not be evaluated as code, while a comma (inside such a block) denotes the opposite. Afterwards, the instruction steps themselves are specified as a sequence of goals to
be achieved which refer to the designators defined earlier. Thus, instead of specifying the action to pick up the pancake mix, the plan specifies the goal of having the pancake mix in the hand as its corresponding plan step.

As can be seen from the evaluation by Tenorth et al. (2010b), the importer correctly translates between 48% and 80% of the instructions. The main issues are caused by complex sentences that the parser does not handle correctly or are related to wrong disambiguation of words with multiple meanings. We therefore see this technique rather as an aid to the programmer that is to be used in a semi-automatic fashion than a completely automated procedure.

10.2.2. Inferring where to search for objects

In household environments, objects are typically stored in containers such as cupboards or drawers, so a robot has to search for them before they can be used. To quickly find the required objects, a robot should search for the objects at their most likely places first. Our robots use a semantic 3D object map of the environment in which structured models of objects, such as cupboards consisting of the container, the door, the handle and hinges, are associated with first-order symbolic descriptions of the objects that mainly come from the robot’s encyclopedic knowledge base (Pangercic et al., 2012).

There are different possibilities to infer likely storage locations of objects. If the robot knows the locations of some objects, it can compute locations for novel object types based on their semantic similarity to the known ones (Schuster et al., 2012). If it does not have such information, it can use abstract rules such as “a refrigerator is the storage place for perishable items”. By combining its knowledge about the properties of objects (to infer whether the object at hand is perishable) with the semantic environment map (to locate an instance of a refrigerator), the knowledge processing system can compute where to search for objects. Figure 10 visualizes these inference steps. The part of the ontology on the right hand side was generated from the germandeli.com online shop as described in Section 8.2.

While these different approaches can co-exist in the knowledge base and independently generate location hypotheses, the integration of the results of different inference mechanisms remains an open question, especially if they produce conflicting results. A straightforward option would be to trust some methods, e.g. the one using information about similar objects, more than others that are, e.g. based on generic class-level rules. Another option could be to combine the results using a utility-based approach as in Kunze et al. (2012) to decide where to search first.

10.2.3. Importance of a knowledge-enabled approach

While a “knowledge-poor” robot system that uses conventional control programs could produce similar behavior on a specific task, the advantage of using a knowledge-enabled approach lies in the gain of flexibility and adaptability: novel tasks can easily be created as a combination of existing primitive actions, their parameters are automatically inferred based on the environment model. If the environment changes, the plans do not need to be adapted since environment-dependent inferences are performed centrally in the knowledge base and are parameterized by the spatial environment model.

10.3. Exchanging information between robots

We have further investigated how the techniques for exchanging task-related knowledge can enable robots to perform mobile manipulation tasks in previously unknown environments (see also Tenorth and Beetz (2012a)). The experiment has been conducted in two locations with two different robot platforms, a PR2 and an Amigo robot. The robots were to serve a drink to a patient in bed in a hospital room, i.e. to first find a bottle in the environment, to open the enclosing cabinet, to pick it up, to move to the patient, and to hand over the bottle. The PR2 performed the task first, thereby estimating the articulation model of the cabinet containing the bottle. This information was then shared with the Amigo robot that could use the model to open the cabinet of the same type in a different location.

Both robots were only given the command “serve a drink” and the address of the environment they operated in. Based only on this information, the robots successfully downloaded all information required for the task. In addition, they were equipped with plans for basic actions such as navigation or grasping, as well as software for, e.g., recognizing objects. However, there was no model for any of the involved objects, nor any environment map, nor any task-specific information available on the robots prior to the experiment. This information needed to be downloaded from the ROBOEARTH knowledge base and had to be grounded in the robot’s perception and action system. The KNOWROB system contributed in the following parts of the experiment:

- formal representation of the task specification, the environment model, and the object models, including information about their parts and articulation properties;
- download of the task specification and the other pieces of information from the central ROBOEARTH knowledge base;
- matching of the requirements of the “serve a drink” specification to the capabilities of the robots, identifying, and downloading missing components;
- run-time knowledge base interfacing the executive to the perception and world modeling components, providing the executive with information about the actions to perform and their parameters, the locations of objects.

The upper part of Figure 11 shows the environment maps that have been downloaded from ROBOEARTH. Based on these maps, the robots (Figure 11 bottom) could navigate to the appropriate positions and locate the objects required for
Fig. 10. Reasoning steps to infer a probable storage location for a bottle of pancake mix. This example is based on a class-level assertion that refrigerators are storagePlaceFor perishable items. The ontology fragment on the right side has been automatically generated from the Web site of an online grocery shop.

Fig. 11. Top: Semantic environment maps of the two hospital rooms, downloaded from ROBOEARTH based on the address and room number. Bottom: PR2 and Amigo robots opening the cabinet and picking up the drink to be served.

So far, the ROBOEARTH system has only been used to exchange symbolic task descriptions that do not provide information about motions, forces or accelerations. In our current research, we work on extending the representations to include lower-level information in terms of constraints that can be interpreted by a motion controller (Kresse and Beetz, 2012). Another direction of work is how to merge ontology fragments created by other robots into the knowledge base, especially if there is some overlap with local modifications. In the current setting, such ontology alignment techniques (Euzenat et al., 2011) are not yet part of the system.

10.3.1. Importance of a knowledge-enabled approach
In particular, in the context of knowledge exchange, explicit representations are highly important. A conventional control program does not “know” which knowledge it has in its internal data structures, and cannot extract this knowledge into a format that other robots can use, unless it has explicitly been modified for this task. A robot without explicit knowledge also cannot detect missing knowledge and “explain” to the Web-based knowledge base what exactly is missing (in terms of a search query) to download these pieces of information. If the control program is compiled code, downloading and executing a plan for a new task becomes impossible. Briefly, without a knowledge-based approach, one could only exchange binary programs among identical robot platforms.

10.4. Inferring which objects are missing on a table
Another integrated experiment showing how the robot can use its knowledge to accomplish complex tasks is the inference which objects are missing in an incomplete table setup. To complete it, the robot must first find out which
meal will take place and which objects are required for it. Therefore, it needs detailed knowledge about the types of food and utensils used in different meals and the preferences of the participating persons. This scenario is also a demonstration how the different components in KNOWROB can jointly be used to solve a complex task. The system is described in more detail by Pangercic et al. (2010). KNOWROB served as the integration platform for perceptual information about the objects that can currently be seen on the table and the learned statistical relational models about which objects are commonly used by different people for different meals. The objects detected by the vision system were added to the knowledge base using the techniques described in Section 7. By combining the detected object poses with the environment information, the system could compute which objects are on a given table based on computables for qualitative spatial relations.

This set of objects served as evidence for the statistical relational inference methods that were integrated as described in Section 6.3. The inference was performed based on models that had been learned on (simulated) observations of human meals and that described the type of the meal, who took part, and who consumed which foods using which utensils. Figure 12 shows the dependency structure of the Bayesian logic network that, after having been trained on this data, effectively represents a joint probability distribution different aspects of human meals. During execution, the robot control program sent queries to KNOWROB asking for the set of missing objects, KNOWROB forwarded this query to the PROBCOG inference engine, which then loaded the respective model, read evidence from KNOWROB (the set of objects that had been perceived on the table) and performed the inference. An example query with its result looks as follows:

\[
P(\text{usesAnyIn}(P, ?u, M) \land \text{consumesAnyIn}(P, ?f, M) \mid \text{meal}(M) = \text{Lunch} \land \text{usesAnyIn}(P, \text{Plate, } M) \land \text{usesAnyIn}(P, \text{Knife, } M) \land \text{usesAnyIn}(P, \text{Fork, } M) \land \text{usesAnyIn}(P, \text{Soup, } M) \land \text{usesAnyIn}(P, \text{Pizza, } M) \land \text{consumesAnyIn}(P, \text{Salad, } M) \land \text{consumesAnyIn}(P, \text{Water, } M))
\]

The result of the inference was a set of object types with assigned probabilities that denoted which objects were supposed to be on the table. After subtracting those objects that were already there, KNOWROB determined which ones were still missing, and inferred where to search for them based on its perceptual memory and its knowledge about storage locations in the environment.

Figure 13 visualizes the results for some exemplary scenes. The upper row shows the input images in which the different objects have been recognized. These objects are shown in the lower images in red on top of the table. The inference results, indicating which objects are concluded to be missing, are visualized behind the table, with the hue value corresponding to the probability from low (blue) to high (red). In the left image, the system inferred a knife and a glass to be certainly missing: which is reasonable, given that there is juice, but no drinking vessel, and no knife for cutting the cake and the sausage. The center image shows a setup where silverware has already been placed on the table, but a cup and a plate are obviously missing. In the right pictures, it is again the glass that is needed for drinking the water and the juice.

**10.4.1. Importance of a knowledge-enabled approach**

Using explicit models of human eating behavior, we could realize this task with a very generic control program that mainly consists of a statement “for all objects that are missing, bring them to the table”. Depending on the model, this control program will work for different meals in different countries, and also for non-meal settings. In a classical program, one would have to hand-code the rules in terms of if-then-else statements, which can be very difficult if the scenario becomes complex: manually encoding the relations between a large number of object types and depending...
11. Related work

There has been a long tradition of using knowledge representation and reasoning techniques on robots: already in the very early days of robotics, robots such as Shakey (Nilsson, 1984) were equipped with world models that described the environment using predicate logic and supported automated inference. However, these models did not scale well beyond the artificial blocks world Shakey operated in, and in the following decades, research in robotics and artificial intelligence (AI) was performed rather independently. In knowledge representation, the field of AI research most related to this article, techniques such as the situation calculus (McCarthy and Hayes, 1969), event calculus (Kowalski and Sergot, 1986), and fluent calculus (Thielscher, 1998) were developed. Knowledge bases such as Cyc (Lenat, 1995) or WordNet (Fellbaum, 1998) collected large amounts of knowledge, and recently the rise of the World Wide Web has spawned research on the automatic extraction of knowledge bases from structured sources such as Wikipedia (Auer et al., 2007; Suchanek et al., 2007) or from unstructured information on the Internet (Banko et al., 2007; Carlson et al., 2010). However, these knowledge bases are disembodied, not grounded into a robot’s control system, and often do not provide the kinds of information a robot needs. If articles on knowledge representation used robotics as application scenario, the approaches were often over-formalized while lacking features needed in real-life scenarios. For example, the paper titled “Representing the knowledge of a robot” (Thielscher, 2000) does not consider aspects such as temporal reasoning, detailed spatial representations, information about object types, processes in the environment, grounding, or integration with the robot’s control program.

On the other hand, work in robotics often focused on problems such as efficient localization and map creation and neglected higher-level semantic aspects. Recent advances in object recognition however lead to more research on semantic maps and thereby to a wider use of higher-level semantic knowledge in robotic applications, although the “level of semantics” ranges from a mere classification into different parts and object types (Rusu et al., 2008) or the co-occurrence of objects (Vasudevan and Siegwart, 2008) to environment representations in DL (Zender et al., 2008), statistical relational environment models (Limketkai et al., 2005) and an embedding of spatial information into an encyclopedic and commonsense knowledge base (Tenorth et al., 2010a). The use of semantic maps as a knowledge source for task planning has been investigated by Galindo et al. (2008). Other AI techniques that have been used in robotics include the grounding of natural language in perception and action Mavridis and Roy (2006); Kollar et al. (2010), planning of actions and motions (Wolfe et al., 2010; Kaelbling and Lozano-Perez, 2011), and the generation of robot controllers from logical constraints (Kress-Gazit et al., 2009).

While these systems are all very successful in their respective domains, they focus on a single dimension of knowledge, e.g. environment maps or robot actions. Robots that autonomously interact with objects, however, are often faced with inference tasks that involve multiple dimensions, for example to reason about the objects involved in a task and their most likely positions in the environment. We therefore think that more comprehensive knowledge bases and more diverse inference techniques are needed to address these challenges. When designing a complete, integrated system, it is often necessary to find a trade-off between techniques that are optimized for a specific problem and a coherent representation that integrates the individual components. Knowledge bases that have been specifically designed for autonomous robots, such as the KNOWROB system described in this article, try to address this challenge. In the following paragraphs, we discuss similar integrated knowledge processing systems for robots that have been developed over the past few years.

The ORO ontology (Lemaignan et al., 2010) is focused on human–robot interaction and on resolving ambiguities in dialog situations. This capability was for example described by Ros et al. (2010), where the robot inferred, based on its knowledge about the objects in the environment and their properties, which queries it should ask to disambiguate a command. ORO uses OWL as a representation format and a standard DL reasoner for inference. An underlying three-dimensional geometrical environment representation serves for computing spatial information and for updating the internal belief state about the positions of objects (Siméon et al., 2001).

The knowledge base presented by Daoutis et al. (2009) is a central component of the PEIS ecology project (physically embedded intelligent systems). PEIS investigates distributed intelligent systems consisting of mobile robots, but also of sensors embedded into the environment which are all integrated into a common framework. The PEIS knowledge base is realized as an extension of the Cyc inference engine. On the one hand, this gives the system full access to the large Cyc ontology, but it comes at the cost of slower inference, of irrelevant knowledge in several branches of the ontology, and of a lack of knowledge in areas such as robotics or mobile manipulation. The authors concentrate on the grounding and anchoring aspects (see also Daoutis et al. (2012)).

The OUR-K system by Lim et al. (2011) is the successor of the OMRKF framework Suh et al. (2007). OUR-K is an extensive system that describes a variety of aspects centered around five main kinds of knowledge: contexts, objects, spaces, actions, and features. Compared with the
Within atomic conjuncts. Thus, in order to do first-order temporal, and spatial reasoning, to name only a few. For example, Prolog lacks special-purpose reasoning capabilities, but only within atomic conjuncts. Thus, in order to do first-order probabilistic reasoning, KNOWROB provides a predicate that transforms a probabilistic query into a Markov logic problem, solves the Markov logic reasoning problem, and extracts the solution to instantiate the query predicate with the answer. Markov logic reasoning is integrated via procedural attachments. This implies that the Markov logic reasoner currently does not do any other kinds of reasoning such as DL inference while reasoning about uncertainty. The advantage of the KNOWROB approach is simplicity, which we consider as essential in the context of robot control. The lack of expressiveness can often be compensated through careful design of the queries, for example by solving the needed inferences before setting up the uncertain reasoning problem.

Another issue is that Prolog’s search strategy, depth-first search with backtracking, is incomplete. This means that Prolog might search forever without returning an answer to a query even if an answer exists. This issue is a more theoretical issue because in practice the Prolog queries in robot plans have to be carefully designed and optimized such that they provide answers within the available resource bounds and time constraints.

In principle, Prolog could be used to implement complete robot control programs. For example, we could state as a Prolog rule that a bottle is in the fridge if the robot can see it when looking into the fridge after opening it. Thus, when asked whether a bottle is in the fridge, the Prolog rule would ask the robot to go to the fridge, open it, and look for the bottle. While this is very elegant, it implies that the robot would perform difficult and complex manipulation without being under control of a plan that is capable of specifying flexible and robust behavior. To avoid this problem, we specify all perception, navigation, and manipulation actions in plans and not in Prolog queries and rules.

In our use cases we rarely ran into situations where Prolog was not expressive enough. This is because the procedural attachment method allows the programmer to specify predicates with arbitrary computational semantics. However, in cases where predicates are defined through procedural attachments it may happen that they do not represent their semantics modularly and transparently.

Lately, software architectures employing ensembles of experts have proven themselves as capable of successfully scaling towards real-world complexity. The potential of such architectures has been very impressively shown by the Watson system (Ferrucci et al., 2010), an expert query answering system that won the US quiz show Jeopardy!. We plan to explore the feasibility of adding an ensemble of expert-based query answering mechanism to KNOWROB. The system will then employ expert query answering mechanisms that detect subqueries they can address and that hypothesize answers to these subqueries. In a second step, the query answering mechanism will then combine the individual answer hypotheses and compute the answer with the highest confidence. Such a system would help to overcome limitations of the Prolog-based inference and to include alternative reasoning mechanisms.

One of the biggest challenges of the upcoming years will be to give robots the ability to understand and competently execute vague, underspecified task instructions. Such instructions are ubiquitous in human communication since humans normally omit information they consider as obvious commonsense knowledge when they explain a task to another human. If robots are to be naturally instructable,
they will need to be able to first identify missing or ambiguous pieces of information in the instructions, will have to reason about where to obtain this information from, how to generate it, and how to integrate it into the task description.

A major part of this challenge is how to generate the comprehensive and diverse knowledge bases that are required to provide robots with the various kinds of information they need. This will require a more thorough integration of learning, which is currently not well-covered by the KNOWROB system. While statistical relational models still face scalability problems, they are a very promising direction of research as they will allow learning relational knowledge bases from data. Learning probabilistic relational models (Getoor et al., 2003) is a very active field of research, and examples of recent accomplishments are methods for structure learning in Markov networks (Davis and Domingos, 2010), learning Markov logic networks (Kiddon and Domingos, 2011), and learning relational dependency networks (Natarajan et al., 2012). While inference in Markov logic networks in general is intractable, there is promising work on finding a subset of Markov logic that is tractable while still supporting useful inferences (Domingos and Webb, 2012) by partitioning the domain of interest into parts and restricting the relations to sub-parts of a part. Developing learning and inference techniques for statistical relational models that can operate on large amounts of data and as part of the robot’s (soft) real-time high-level control loop will be an important research topic and is part of our research agenda.

These learning techniques will likely have to be complemented with other knowledge-acquisition methods depending on which kind of information is to be acquired, represented, and reasoned about. Knowledge bases about objects will need to represent their properties, their appearance, their ingredients, information about fragility or suitable grasping points, their composition of (functional) parts, as well as relations to other objects. Activity descriptions need to better represent the task context and the relation between symbolic or natural-language descriptions of these actions and observations of human demonstrations and robot experiences. Knowledge bases describing single actions should provide information about the actions’ properties and parameters, which includes the exact relations between actions, objects, tools, locations, and grasp types (Nyga and Beetz, 2012). On a lower level, robots need motion models that can be learned from experience, can be parameterized by the robot’s high-level control program if new information imposes additional constraints, can be composed and adapted to changing world states, and abstractly reasoned about. These aspects are not yet handled by KNOWROB or any other integrated knowledge base we are aware of, but realizing appropriate representation, learning and inference methods for these modalities and filling them with content will be crucial for building intelligent robot assistants and co-workers.

13. Conclusions

In this article, we introduced KNOWROB as a practical knowledge processing system for autonomous robots. One of its main concepts are “virtual knowledge bases” that can be defined as a symbolic layer on top of sub-symbolic data. At query time, the required information is loaded and abstracted up to the appropriate level using procedural attachments to the semantic classes and properties. The symbolic layer serves as interlingua for integrating different knowledge sources, such as the robot’s perception system, internal data structures of the control program, or information from the Internet. If the knowledge base needs to answer a query about something that is not yet known, it can forward the query to other components to acquire the desired information.

KNOWROB is based on Prolog and internally stores knowledge in terms of DL, a format that allows to represent information in a very structured way and is a good compromise between sufficient expressiveness and still good support for automated reasoning. The knowledge about classes of objects and actions is represented in form of an ontology. It provides the vocabulary for describing knowledge about actions, events, objects, spatial and temporal information. The overall system architecture and the main components for knowledge acquisition, for automated reasoning, for visualization and for querying for information that are integrated in the KNOWROB system were explained in Section 3, as well as the different sources of knowledge that can be used.

KNOWROB integrates three main inference techniques, each suited for different kinds of knowledge: DL inference forms the backbone of the overall system and is used for all deterministic information, including the large ontologies. For reasoning about probabilistic information, we integrated the PROBCOG toolkit that provides various inference methods for reasoning in statistical relational models which combine the expressiveness of first-order logics with the ability to represent uncertainty. The “computables” described in Section 6.2 allow a procedural computation of semantic information on demand during the inference procedure. They are the main tool for loading information from the perception system and for computing relations between entities based on the information in the knowledge base.

A close integration between the knowledge base and the robot’s perception components allows to generate object instances based on detections of objects. Two kinds of interfaces exist for active, task-directed perception systems as well as for passive listeners that record all perception results. The presented techniques for acquiring knowledge from the Internet and from observations of humans, as well as the methods for exchanging knowledge between robots reduce the time needed for constructing large knowledge bases.

In our ongoing research, we are working towards more knowledgeable and more intelligent autonomous service
robots that are capable of performing realistic tasks in human environments. For these tasks, a comprehensive knowledge base that provides reasoning services to the robot control program is an important resource. Inferring control decisions from the robot’s knowledge leads to more robust, flexible, and re-usable behavior since aspects such as the environment configuration are described separately from the control flow. The large amounts of knowledge to perform complex tasks this way can be acquired using the presented (semi-)automated acquisition methods.

**Funding**

This work is supported in part by the EU FP7 Projects RoboEarth (grant number 248942) and RoboHow (grant number 288533), as well as the DFG excellence initiative research cluster Cognition for Technical Systems (CoTeSys).

**Notes**

1. See http://ros.org/wiki/knowrob
2. See http://ias.in.tum.de/research/probcog
4. See http://ros.org/wiki/comp_knowrob
5. See http://ros.org/wiki/tabletop_object_detector
7. See http://ros.org/wiki/comp_germandeli
8. See http://ros.org/wiki/roboearth
10. See http://ros.org/wiki/comp_missingobj

**References**


Ceri S, Gottlob G and Tanca L (1989) What you always wanted to know about Datalog (and never dared to ask). IEEE Transactions on Knowledge and Data Engineering 1: 146–166.


Frege G (1879) Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens. Louis Nebert.


Nineteenth National Conference on Artificial Intelligence (KI 2010), Karlsruhe, Germany. Berlin: Springer-Verlag, pp. 151–159.


on Active Semantic Perception and Object Search in the Real World, San Francisco, CA.


Chapter 2

Representations for Robot Knowledge

While the previous chapter discussed the system aspects, such as which representation formalisms to use, which inference methods to apply and how to integrate the knowledge base with the rest of the robot’s control system, this chapter focuses on the representations for different kinds of information inside this system and the knowledge content that is required for competent operation in the real world.

In particular, we discuss the representations for events, actions and processes and their effects on objects, object types and their functional and geometric composition, object instances and their positions over time, environment maps and finally the robot’s model of its own components and capabilities. We discuss on the example use case of completing underspecified instructions, which is one of the core research problems in RoboHow, how the individual knowledge aspects and especially their integration into a coherent representational framework contribute to solving the overall task.

This chapter is based on the following publication that is currently under review for the Artificial Intelligence Journal:

Enabling Robots to Interpret Generalized Task Descriptions using the KnowRob Framework

Moritz Tenorth and Michael Beetz

Institute for Artificial Intelligence, Universität Bremen, Am Fallturm 1, 28359 Bremen, Germany

Abstract

In this article, we investigate how representations combining knowledge from different sources into a consistent knowledge base can help robots to make sense of vague task instructions. Incomplete and ambiguous task instructions are ubiquitous in human communication, and being able to translate them into effective specifications that allow robots to perform the tasks is important for robotic agents in factories, co-workers, and home assistant robots. We propose a knowledge representation and reasoning framework that covers multiple knowledge areas and integrates several special- and general-purpose reasoning methods in a coherent representation. We define a common vocabulary for representing knowledge about actions, events, processes, objects, environments, and the robot’s hardware and develop methods for various inference tasks that operate on this common representation. The system is being used by physical robots performing complex object manipulation tasks and has been released as open-source software. We evaluate it through prototypical queries that demonstrate the expressive power and the impact on robotics.

Keywords: Knowledge representation, Autonomous Robots, Knowledge-enabled Robotics

1. Introduction

One of the big challenges that robotic agents, such as robot factory workers, co-workers, and home assistant robots, have to face when providing their services is that the tasks they are to perform will be stated incompletely and ambiguously. When humans give instructions to one another, they often use very brief commands like “please clean up” or “set the table”. Similarly, instruction sheets that are commonly used in factories or for experiments in biological and chemical laboratories just list the main steps to be performed. A major obstacle towards the vision of robots that are able to understand such instructions is the amount of common-sense knowledge that is needed to correctly interpret them and to translate them into robust and flexible robot control programs.

Imagine for instance a typical recipe for a simple meal, such as making pancakes. While very short and vague recipes are sufficient for humans for cooking a meal, they lack much important information required by robots: They do not mention where to fetch the ingredients from. Even if a location like “from the refrigerator” is given, information about the position of the refrigerator and about how to open it is missing. Actions for switching electric devices like a mixer or a stove on or off are often omitted. Recipes rarely explain which tools to use and how to handle them. And finally, all descriptions are symbolic and qualitative, while a robot’s controller requires concrete coordinate frames and geometric information to perform its actions.

The robot therefore has to supply the “delta” between the vague task description and the parameters needed by its action components by adding the missing pieces of information based on its background knowledge. Translating from vague and abstract into concrete and grounded representations is a very knowledge-intensive procedure that requires the identification of knowledge gaps and inference on how to close them. The result of the translation is an effective task specification that contains all information the robot needs to executing the task. In contrast to other machine translation tasks, where words can be manipulated without understanding the meaning of each of them, the robot must...
be able to ground all symbols in perceived objects or action parameterizations, and it needs a knowledge base that is comprehensive enough to provide all missing information. Depending on which kind of information is missing, the robot may need knowledge about objects types, about their parts and properties, about the environment, about the robot’s own hardware and capabilities, or general background knowledge, for example about the properties of actions. Much of this knowledge depends on the spatial and temporal context, e.g. which objects are currently available in the robot’s environment, which actions have already been performed, or which time of the day the action takes place. When tidying up, for example, the robot should consider if a meal has been finished before it starts to clear the table.

In this article, we investigate how representations combining knowledge from different sources into a consistent knowledge base can help robots to make sense of vague task instructions. We propose a knowledge representation and reasoning framework that covers multiple knowledge areas and integrates several special- and general-purpose reasoning methods in a coherent representation. We define a common vocabulary for representing knowledge about actions, events, processes, objects, environments, and the robot’s hardware using the Web Ontology Language (OWL, [45]). Multiple inference methods operate on this common representation to perform temporal reasoning, to compute qualitative spatial relations, or to match robot capabilities against action requirements. These inferences are beyond the expressiveness of the OWL language, but since they read information from the OWL knowledge base and describe the computed relations in terms of OWL statements, they integrate well into the knowledge base. This pragmatic hybrid approach proved to be very efficient, which is of great importance for using reasoning methods on physical robots.

The main contributions of this work are the following: We introduce coherent representations for actions, events, processes, objects and their parts, environments, robot components and capabilities, and their relations, that enable robots to reason about the combination of all these aspects, as required for understanding vague instructions. We present several inference methods operating on the shared representation which perform spatio-temporal reasoning, robot capability matching, envisioning the effects of actions and processes, and reasoning about functional object parts. With our novel representations of change, robots can reason about object locations over time and distinguish between perceived, predicated and desired world states. The integrated representation of actions and processes allows the robot to envision effects of its actions and to plan with the immediate and indirect effects of actions. A novel part-based object representation combines semantic information with geometric models of the object and its functional components, for example to grasp objects at the appropriate parts and to perceptually anchor object-related knowledge. We evaluate the system through prototypical queries that demonstrate the expressive power and the impact on robotics.

With this paper, we would like to give an overview of the representations in our system and explain the benefits of a common representation language for different kinds of knowledge. This overview article is accompanied by open-source software, ontologies and knowledge bases. Several tutorials on simulated and real robot data explain how the system works and what it can be used for. By providing the software and representations, we would like to give the reader the opportunity to try the system and assess its performance for different applications. The remainder of this paper is organized as follows. We start with a brief description of the knowledge processing system, explain the problem of completing vague task instructions and discuss the problem of grounding abstract knowledge for task execution. We then give an overview of the representation of actions and the envisioning methods. Afterwards, we will describe how the task context is described, including objects, their parts, properties, and locations in the environment, before we introduce the robot self-model. We then present several experiments to demonstrate the range of supported information types and inference methods. The paper finishes with a discussion, a comparison with related approaches and our conclusions.

2. The KnowRob Robot Knowledge Processing System

In a companion paper [37] and in [34], we have introduced KnowRob, our knowledge representation and processing framework for autonomous robots, with a focus on systems aspects such as the knowledge storage, techniques for combining different reasoners and for integrating the knowledge base with the robot’s data structures, perception and control system. In this article, we explain the knowledge representations used in the system and how they contribute to its reasoning capabilities. The representational structures we describe in this paper have been implemented in KnowRob which is available as open-source software and is in active use by several research labs worldwide.

\(^1\)Available online at http://www.knowrob.org
Figure 1: Excerpt from the knowledge base, describing information around two actions for picking up a spatula and flipping a pancake. The upper part describes the OWL model, with the different interacting knowledge domains highlighted in different colors. The lower part lists some of the reasoning methods that are hooked to the classes and relations in the OWL model they compute.

The KnowRob system is implemented in SWI Prolog [48] and uses Prolog as interlingua for integrating different knowledge sources and inference methods. Most of the representations described in this article are stored in the Web Ontology Language OWL [45] and loaded into the system using Prolog’s Semantic Web library [47]. At the core is a large ontology that conceptualizes the robotics domain. Its upper levels have been derived from OpenCyc [20], a widely used and comprehensive upper ontology. Staying compatible with OpenCyc allows us to incorporate extensions made by other researchers, for example mappings to other ontologies like WordNet [8]. The general-purpose upper ontology can be extended with micro theories that add domain-specific knowledge or special-purpose inference methods.

Figure 1 visualizes a small excerpt of the knowledge base, representing a sequence of two actions in a plan for making pancakes. The first one, PickingUpAnObject, is performed on an object of type Spatula which is stored in Drawer-31, an instance of a drawer at a specified position and orientation (pose). PickingUpAnObject is a subclass of the class Movement-Translation, which by itself is a specialization of ActionOnObject. The second action, Flipping-Something, is linked to the first one using the nextAction relation, allowing inference about the sequence and temporal aspects. The Spatula serves as toolUsed in this action and is graspedAt the Handle, which is one of its physicalParts. The flipping action is performed on a Pancake that is located on top of a PancakeMaker. Before execution, the system has checked whether the actions are feasibleOnRobot on the PR2Robot to make sure that all required capabilities are available.

The upper part of the figure describes the model in the OWL language. The colors highlight the different knowledge areas that are needed even in this small example. The gray arrows visualize the subClassOf relations, black arrows denote other relations like the objectActedOn of an action. The lower part shows some of the inference methods that are available in the KnowRob system and how they are ’hooked’ into the OWL representation using procedural attachments. Several of these general- and special-purpose inference methods extend the reasoning capabilities beyond logical inference in OWL, for example spatial reasoning (Section 5.1.5), segmentation of geometric object models (Section 5.1.2), or visibility computations (see also [3]). The OWL representation sets the inference results into relation and allows to combine different reasoners to answer a query. For example, the robot can use reasoning about the composition of objects from parts to infer that it should grasp the spatula at the handle, and geometric
3. Overview: Task Completion and Disambiguation

Throughout this article, we will use the task of making pancakes as an example. In prior work [4], we have shown that our robots are capable of performing such a task. While that experiment was still largely based on manually programmed action modules, we have since investigated how robots can acquire such kinds of tasks in a more autonomous fashion using cooking recipes originally written for humans. In addition to repositories of cooking recipes, web pages like wikihow.com also provide step-by-step instructions for thousands of other everyday tasks. Tenorth et al. [39] have proposed a method to parse the natural-language instructions, to map the words to concepts and to represent the task description in the knowledge base.
A problem with these instructions is that they are very vague and superficial. When humans describe a task, they usually omit all information that appears as obvious to them. As example, let us consider the instructions at the top part of Figure 2 that would be sufficient for many humans to make a pancake. These instructions, for example “Pour the dough onto the pancake maker” do not provide any information about the appearance of the objects, that the dough is contained in a bottle of pancake mix, about where to grasp the bottle, about which part to control during the pouring (the opening), or where exactly to pour the dough.

The lower part of the figure illustrates which information is missing and how it can be inferred (indicated by the different colors). This involves some difficult reasoning problems: The tools and ingredients that are needed for a recipe are usually stored in cupboards and drawers in the kitchen. To efficiently fetch them, the robot has to reason about where to find which objects given its (partial) knowledge about the environment. After cracking the egg, the robot ends up with eggshells that are never mentioned in the recipe. Humans would expect that they are thrown away after the task. While the recipe first talks about the “dough”, it later mentions a “pancake”. The robot has to infer that the pancake is supposed to emerge from the dough by a baking process, which needs a heat source to be started, which requires the robot to switch the pancake maker on – none of which is mentioned in the recipe.

Based on the approximate action sequence, which is generated from the instructions and considered as input, the following inferences can be made to supply missing information: Intermediate products can be predicted by envisioning the action effects (Section 4.4), their appearance and models for recognizing them can be obtained from the knowledge base of object types (Section 5.1.1). Actions for fetching the objects from their storage locations are rarely mentioned in cooking recipes and have to be added, requiring information about the properties of objects and the environment setup (Section 5.1.6). Experiences from previous tasks or observations of humans can provide valuable information (Section 4.3). Other plan flaws are more difficult to detect, for example that the instructions do not mention that the pancake maker needs to be switched on for the dough to bake to a pancake. This issue can be detected by comparing the necessary inputs for the actions (e.g. the pancake that is to be flipped) and the envisioned world state at this step of the task. If the inputs are neither the result of previous actions nor available in the environment, the robot can try to plan actions that (directly or indirectly) have the desired effects (Section 4.5).

Another important aspect of task completion is to make the abstract knowledge operational in the robot control system. The symbol grounding problem [14] is particularly relevant for robots that act in the physical world and therefore have to be able to recognize the concepts they reason about and to turn their abstract knowledge into parameterizations of real-world actions. In the pancake scenario, the robot does know that it has to pour the batter onto the pancake maker. Assuming that it has a plan for pouring something somewhere, it still does not know how the bottle containing the batter looks like, where to grasp it, which part of it to control during the pouring motion, and which part of the pancake maker to pour the batter on.

In our system, we use three-dimensional CAD models of objects to bridge the gap between symbolic representations in the knowledge base and geometric descriptions for the robot controllers. In Section 5.1.2, we will explain how we identify functional parts like handles, containers or supporting planes in monolithic object models. Figure 3 shows some examples, namely the main supporting plane of the pancake maker, the handle of the bottle of pancake mix, and its bottle cap. Using these concepts, one can formulate generic rules like “Grasp objects at the handle if one has been identified” or “When pouring something from a bottle, the bottle cap is the part that needs to be controlled”. Having the decomposition of the object model into functional parts, the robot can apply such abstract rules and obtain the concrete sub-meshes and coordinates of the parts with respect to the model that are needed to parameterize its...
actions. By aligning the surface mesh of the model with its sensor data, the robot can also recognize the object in a scene and compute its pose with respect to its camera. This way, it can get the positions of object parts in scene-global or robot-relative coordinates that are needed for motion planning.

4. Task Representation and Envisioning

The different inferences involved in the completion of task instructions require sophisticated hybrid task representations that allow planning as well as envisioning, and reasoning about plan schemata as well as collecting experiences. Besides immediate action effects (e.g. "the pancake batter is in the frying pan"), we also consider indirect effects caused by processes that are started by the actions (e.g. "the batter is baking to a solid pancake"). We start with the foundation of the task representation, the models for events and temporal information.

4.1. Events and temporal information

Robots act in dynamic environments: Actions are performed, objects change their positions and properties over time. Representing and reasoning about temporal information like the start time of an event, the duration of an action, or relations like contemporaneity is therefore highly important. Like OpenCyc, we consider an event as "a dynamic situation in which the state of the world changes". Events can be instantaneous (like the moment when a perception is made) or temporally extended, like the execution of a trajectory for reaching towards an object (Figure 4). Each event has a startTime and, if its duration is finite, also an endTime. Both relations link an event to a TimePoint, a special case of a TimeInterval with zero duration. This event representation allows temporal reasoning on the start- and end times. By combining the after(t2, t1) and temporallySubsumes(t1, t2) relations with the startTime and endTime properties, all of Allen’s temporal relations [1] between time intervals can be computed. Actions are considered as special kinds of events, namely those that are caused by an agent acting in the world. This allows to use the same vocabulary also for describing action properties.

4.2. Action classes

Action descriptions on the class level model action schemata such as robot plans. They describe the composition of tasks from sub-actions, the types of these actions, their relative ordering, as well as action parameters like which object is to be manipulated or at which locations the action is to be performed. The “action” branch of the KnowRob ontology currently contains more than 130 action classes that are commonly observed in everyday activities. Besides the taxonomic structure, the ontology also describes some basic and general

---

2Definition from OpenCyc http://sw.opencyc.org/concept/Mx4rvViADZxpEbGdrcS5Y29ycA
3The modeling of actions is described in more detail in [40]
When it comes to describing a task using these action classes, one usually derives a sub-class from them that is annotated with task-specific information about objects, tools, locations or timing. These derived action classes are then arranged in a (partially) sequential order to form the task description. The code below is an excerpt of a plan for setting a table.

```
Class: SetATable
  Annotations: label "set a table"
SubClassOf: Action
EquivalentTo:
  subAction some PutPlaceMatInFrontOfChair
  subAction some PutPlateInCenterOfPlaceMat
  subAction some PutKnifeRightOfPlate
  subAction some [...] orderingConstraints value [...]  
Class: PutPlaceMatInFrontOfChair
  EquivalentTo:
    PuttingSomethingSomewhere
    objectActedOn value PlaceMat
    toLocation some Place

Class: Place
  EquivalentTo:
    inFrontOf~Generally some Chair~PieceOfFurniture

Individual: PlaceMat
Types: PlaceMat
```

The upper part describes the task `SetATable` as a subclass of `Action` with a set of `subActions`. The lower part consists of definitions of task-specific subclasses of generic action classes. The class `PutPlaceMatInFrontOfChair`, for example, is defined as a subclass of the `PuttingSomethingSomewhere` action, with the additional restriction that the `objectActedOn` needs to be a `PlaceMat`, and the `toLocation` has to fulfill the requirements described for the class `Place`, which by itself is described as some `Place` which is `inFrontOf~Generally` of some `Chair~PieceOfFurniture`. Using this structure, arbitrarily abstract or detailed task descriptions can be composed. The split into task-specific classes in the instructions versus generic classes in the main action ontology allows to extract re-usable knowledge and put it into the main knowledge base so that is inherited by other tasks derived from the same action classes.

### 4.3. Action instances

While the class-level action descriptions describe the abstract structure of actions, `instances` of these classes represent concrete executions. They are for example used to record experiences the robot makes during task execution in a long-term memory, including the hierarchy of actions that have been performed, their `startTime`, `duration`, information on manipulated or perceived objects, as well as the action outcome and possible failures. Such a semantically annotated memory can be used for diagnosing plan failures, learning prediction models or evaluating a plan’s performance in different circumstances. By describing the experiences as instantiations of the abstract plans, it becomes very easy to connect these two complementary information sources.

### 4.4. Effects of actions on objects

Most actions are performed with the intention to produce some change in the world. While pick-and-place tasks mainly change the positions of objects, more complex activities, like cooking meals, include changes of the objects themselves: They can be created or destroyed (e.g. when chopping vegetables) and can substantially change their types, appearance, and aggregate states (e.g. mixing and baking cookie dough). Robots therefore have to be able to represent the world states before and after the action and to track which objects got transformed into which other objects. Figure 5 visualizes the effects of a cracking and two mixing actions. After having cracked an egg, the egg ceases to exist and (at least) two pieces of eggshell, the egg yolk and the egg white appear. In parallel, the milk and flour are mixed to a dough, to which the egg yolk is added.

In KnowRo, the representation of action effects consists of (1) declarative specifications of the inputs and outputs of an action (that can be used for planning actions with the desired effects) and (2) projection rules that can envision the world state after an action has been performed. These representations are explained in more detail in the following sections.
4.4.1. Declarative descriptions of action effects

The exact relation between actions and objects can be described by a detailed set of properties (Figure 6). The sub-properties of preActors, displayed in the upper part of the figure, describe action properties that are supposed to hold before the action takes place. They include the agent (doneBy), the initial locations and states (fromLocation, fromState), or the thingIncorporated that is merged into the objectAddedTo. The postActors relations describe the outcome of an action, for example the outputsCreated, the targetPosture or the toLocation of a transport action.

These properties can be used in class restrictions to describe the inputs, outputs, pre- and postconditions of an action in a way that can easily be queried to find actions with desired properties. For example, the robot can search for an action that turns a PhysicalDevice from DeviceStateOff to DeviceStateOn, and will obtain the action TurningOnPoweredDevice:

Class: TurningOnPoweredDevice
SubClassOf
ControllingAPhysicalDevice
objectOFStateChange some PhysicalDevice
fromState value DeviceStateOff
toState value DeviceStateOn
[...]

Figure 5: Object changes in a simple baking task: An egg is cracked, egg shells and egg yolk appear as single objects, and are mixed to a dough together with some milk and flour. The representation of these changes is coupled with methods for qualitatively predicting the outcome of tasks that cause them.
4.4.2. Temporal projection of action effects

In a concrete situation, the robot can envision the outcome of an action and the resulting world state using projection rules. They are implemented as Prolog rules which create the links between the action instance and the involved objects to describe inputs, outputs, newly created or destroyed objects. In the current implementation, the rules operate on a rather coarse, qualitative level, but can be replaced with more sophisticated techniques such as simulation-based prediction methods [17].

Below is an example of a projection rule for the “mixing baking mix to a dough” action. The predicate first checks its applicability conditions (the action is of the given type, the object is specified, and no projection has already been done). Then it creates the new object instances and asserts the relations between the input objects, the action, and the generated outputs. Before the projection takes place, the only relation set for this action was the generic objectActedOn, which is refined and extended by the projection rule using properties like objectAddedTo and thingIncorporated.

```prolog
% Mixing baking mix to a dough
project_action_effects(Action) :-
  owl_individual_of([Action], kr:'Mixing'),
  \+ owl_has([Action], kr:outputsCreated),
  % at least one objectActedOn is a MixForBakedGoods
  owl_has([Action], kr:objectActedOn, Mix),
  owl_individual_of([Mix], kr:'MixForBakedGoods'),
  findall(Obj, owl_has([Action], kr:objectActedOn, Obj), Objs), !,
  % new objects
  owl_instance_from_class([kr:'Dough'], Dough),
  owl_assert([Action], kr:objectAddedTo, Dough),
  owl_assert([Action], kr:outputsCreated, Dough),
  % new relations
  findall(O, (member(O, Objs),
    owl_assert([Action], kr:thingIncorporated, O)), ).
```

The projection rules are implemented as procedural attachments to the postActors relation, i.e. the user can simply query for the effects of an action, and the projection is automatically performed in the background if it has not already been done.

4.4.3. Reasoning about object transformations

The projection result is an object transformation graph like the one in Figure 5 that describes how the involved objects are transformed by the actions in a task. The projection predicates create the intermediate products and the links to the actions in the task. Due to the taxonomy of input and output relations, queries for preActors or postActors can be formulated in a generic way and return all information asserted using the respective sub-properties. To track the changes made to an object over a sequence of actions, we further define the transformedInto predicate as a transitive relation that covers all modifications of objects, including destruction, creation, and transformation. It links the preActors of an action to its postActors and, due to its transitivity, can read the chain of object transformations from this graph structure.

```prolog
?- owl_triple(transformedInto, egg1, Res).
Res = 'EggYolk=Food1'
Res = 'EggShell0';
Res = 'Dough2';
Res = 'Baked4';
```

4.5. Processes and their effects

In the previous sections, we have discussed how to model the immediate effects of actions, but actions can also have indirect effects: When pouring liquid pancake batter into a hot pan, the direct effect is that the batter is inside the pan, while the liquid batter also transforms into a solid pancake. Determining all side-effects of an action is also known as the ramification problem. We model those indirect effects that are common and relevant in the robot’s domain as processes that become active once their preconditions are fulfilled. With our notion of processes we largely follow the Qualitative Process Theory (QPT) by Forbus [10], the standard work for qualitative reasoning about processes.
The classical QPT formulation only considers processes that take place as a natural consequence of a given situation, like steam production in a boiler. Since robots actively change the world and can also intentionally start processes by their actions, we have extended the QPT representation in two ways: First, we added declarative descriptions of the requirements and outputs of processes that the robot can use in a planning context. Second, we included the process effect axioms into the action envisioning procedure: Each time the robot predicts the effects of an action, it also checks whether processes got started because their preconditions became true. Both aspects are modeled very similar to the descriptions of the inputs and outputs of actions described earlier. This allows to handle both in the same way not only for prediction, but also to plan with both the effects of actions and processes. A more detailed description of the process modeling can be found in [36].

Forbus also described an extension of the QPT to include actions into the predictions [11]: The system computes “action-augmented envisionments” by considering actions as changes in the background assumptions made by the QPT (the prerequisites and quantity conditions). This view on actions is somewhat different from ours: Actions are seen from the viewpoint of the processes as something that affects their preconditions, while we rather consider processes as side-effects of actions.

5. Representation of Task Context

The following sections give an overview of the representation of the context in which a task is performed. We present the modeling of object types, functional object parts and their geometry, and the robot’s environment.

5.1. Objects and spatial information

Robots often have to interact with objects in some way, which requires knowledge about object types and their properties as well as information about concrete object instances. This section explains how knowledge about objects and spatial things in general is represented.

5.1.1. Class-level object knowledge

The ontology of object classes ranges from very general classes like SpatialThing to specific ones like Refrigerator-Freezer. In total, there are about 7,000 classes describing objects in the KnowRon ontology. While parts of the upper levels have been imported from OpenCyc, some of the more specialized classes have automatically been generated from Web data, e.g. by mining an online shop and translating the category structure into a class taxonomy [38].

The classes are described by properties, stating for instance that the primary function of an Oven is HeatingFood, and that it has a Handle as properPhysicalPart. Properties are inherited by sub-classes, and by inheriting from multiple super-classes, different facets of an object can be described: A Refrigerator, for example, is a Box-Container as well as an ElectricalHouseholdAppliance and a RefrigeratedStorageDevice and inherits their respective properties. By separating these aspects, objects can be classified along different dimensions (e.g. shape, need for electricity, temperature), and knowledge about these classes can be stored in a modular fashion.

5.1.2. Object geometry and functional parts

Competently manipulating objects requires knowledge not only about their semantic properties, but also about their geometry – for planning grasps, for selecting places where to put them, or for predicting their appearance. This geometric information can hardly be described in a purely symbolic knowledge base, so we chose to extend the definitions of object classes with links to detailed three-dimensional CAD models, which are supposedly the most detailed geometric description available. Monolithic CAD models are useful for visualization or for computing the approximate object dimensions, but robots often need to interact with specific parts that have some functional meaning for the action at hand: For picking up objects or for using them as tools, it should use the handle; for pouring something from or into a container, it needs information about its opening direction and volume; for putting something down or pouring something onto a surface, it needs a suitable supporting plane, i.e. a horizontal surface.

These inferences require a tight integration of geometric and semantic information, so we define a semantic model of functional object parts as a virtual knowledge base on top of the geometric model. This virtual “view” on the model (Figure 7) can be generated on the fly and allows to answer logical queries about the geometric model and its functional components. The main object and all functional parts are described as instances of the respective classes.
Figure 7: Part-based object model. Object instances in the knowledge base, in this case a spoon, are linked to CAD models describing their geometry and appearance. Relevant functional object parts are modeled symbolically and grounded in sub-parts of the geometric model.

like Cup, Handle or Container, and each of these instances is grounded in a (sub)-mesh in the CAD model that describes its shape and extent in detail. The object surface mesh can be used for recognizing the object in a scene and, after aligning the mesh with the sensor data, the object-relative poses of the identified parts can be translated to world-global positions.

Figure 8: Proposed method for generating the part-based object representation. A three-dimensional CAD model of an object is segmented and its parts are classified into functional categories based on semantic rules in the knowledge base.

In [41], we have described how such a model can be extracted automatically from a CAD model by linking geometric analysis with knowledge-based definitions of these components. Figure 8 describes the main processing steps of the proposed approach: The model is segmented based on the surface curvature and geometric primitives like cones, spheres and planar surfaces are fit to the segments. After these bottom-up processing steps, the system can apply knowledge about functional parts in a top-down fashion. It uses logical rules that define the components in terms of geometric primitives in order to identify these functional object parts. For example, a handle can be defined
as a cylinder of certain dimensions, or a container can be formed by a concave cylinder that is closed by a planar surface at one end. The rules are composable, so definitions of other kinds of handles can easily be included by adding appropriate rules.

5.1.3. Object instances

Object instances in KnowRob are interpreted as designators [23], symbolic descriptions of objects in the real world. An object instance is only the internal representation of the object in the knowledge base, but is not interpreted as the object itself. It is important to draw this distinction because it allows to reason about the object and the representation separately, for example to distinguish the time when an object has been created and when it was first detected, to maintain a belief about objects that do not exist any more, and to unify multiple instances once the robot finds out that they refer to the same object in the real world. The following sections describe how properties of object instances, in particular their poses, are represented.

5.1.4. Positions and orientations

Information about the poses and dimensions of objects is crucial for interacting with them. While humans usually describe locations in qualitative terms (e.g. ‘on the table’), robots require quantitative descriptions for planning their motions. The KnowRob system internally stores geometric poses with respect to some coordinate system and computes qualitative relations if needed (see Section 5.1.5).

In dynamic environments, robots often need to reason about changes in the world and therefore have to store both the current and past world states. This would not be possible by just adding a location property to an object, because OWL only supports binary relations that cannot be annotated with the time at which they were valid. To do this, the relation needs to be reified, i.e. transformed into a first-class object that links the object and the location. This object can then be annotated with information like the time or the recognition probability. In KnowRob, these reified relations are stored as instances of the kind of event that created them, be it a perception of an object, an inference process, or the prediction of future states based on projection or simulation (see left part of Figure 11). These events are subclasses of the MentalEvent class (Figure 9), for instance VisualPerception or Reasoning. Multiple events can be assigned to one object to store different detections over time or to describe the current perceived as well as the intended world state. The object poses are stored as 4x4 pose matrices that can be relative to a reference object or a coordinate frame.

![Figure 9: Ontology of mental events. Each of these events can result in changes in the robot's belief state about object poses.](image)

This representation is similar to the fluent calculus [42], in which fluents are objects that represent the change of values over time. In our case, however, the reified objects are more than just a changing value because (1) they provide a memory of past states, (2) they describe the source of this relation by their type, and (3) allow to reason about multiple “possible worlds”, for example the perceived, desired and simulated world. Since all states are represented in the same system, they can easily be compared or checked for inconsistencies.
5.1.5. Qualitative spatial relations over time

Qualitative spatial relations between objects, like inside, on top of, or underneath are commonly used in human instructions, but they can also be useful to formulate rules in a more generic way. For example, a rule saying that the robot has to open the surrounding container first before it can pick up objects inside can easily be described in qualitative terms. Figure 10 shows the taxonomy of relations that can be described.

Internally, the KnowRoB system stores metric object poses and dimensions, qualitative relations are only computed on demand. This approach avoids the problem of storing an exploding number of pair-wise relations between objects and avoids the frame problem of determining which relations to update when an object has been moved. The procedural attachments for qualitative spatial relations also consider that an object may have been destroyed as result of an action (Section 4.4).

Figure 10: Taxonomy of qualitative spatial relations including directional and topological relations.

The aforementioned representation of object poses using MentalEvents forms the basis for evaluating how qualitative spatial relations between objects change over time. We use the holds(rel(A, B), T) predicate to express that a relation rel between A and B is true at time T, and the holds(jtGoal, [Start, End]) predicate (for “holds throughout”) to express that it is true throughout a time interval. The system makes the persistence assumption that the last perceived pose of an object is still valid at the time the relation is evaluated, which makes sense if the robot is the only agent in the workspace and if no self-moving objects exist.

To evaluate a relation at time T, the latest perception of each object before that time is determined, the poses where objects have been perceived in these events are read using the eventOccursAt relation, the objects’ positions and dimensions are read and compared in order to compute e.g. the inside relation. Figure 11 illustrates this procedure. Since most queries are concerned with the current state of the world, the system offers a simplified query mechanism using procedural attachments that takes the current time as default.

Figure 11: Computables operating on the KnowRoB object representations. The holds predicate compute the on_physical relation for a given point in time. For queries about the current state of the world, a simplified query scheme has been realized that evaluates the relation at the current point in time and maps the holds(on_physical(A, B), T) predicate to the on-Physical OWL property.
5.1.6. Environment maps

Semantic environment maps basically consist of a collection of object instances at different poses in a scene. The representation in KnowRon includes the map itself (including information about its type, properties, and possibly links to external files) and its content, i.e. the objects it contains. The representation format is described in detail in Pangeric et al. [27]. More information about the description of the map data structure and its relation to the described environment can be found in Tenorth and Beetz [35].

6. Representing and Reasoning about Robots Capabilities

Besides information about the outer world, robots also need models of their own structure and their capabilities: They allow a robot to reason about which components it consists of, which properties they have, how they are (kinematically) connected, and which capabilities they enable. This can for instance be useful to determine if the robot is likely able to perform a given action. Commonly used robot description formats like URDF⁴ or COLLADA [2] provide information about the kinematics and dynamics of a robot as well as a collision model and a surface model for visualization purposes. What they are lacking is semantic information, e.g. which of the components are sensors and which group of components form a hand? [18] introduced the Semantic Robot Description Language (SRDL) as an extension of these languages to semantically describe the components of a robot.

The robot’s hardware is, on the lowest level, described in terms of links and joints that can be automatically imported from a URDF description (Figure 12 left). These links and joints can then be manually aggregated to semantic components like arms and hands, thereby assigning meaning to them (Figure 12 right). Those links that have semantic meaning, e.g. those that are sensors, are described as instances of the respective class in the ontology (such as Camera or LaserScanner) and are annotated with their properties like the maximum range or resolution. Capabilities like navigation or object recognition can be provided by (a set of) components and model the robot’s functionality. Actions can define dependencies on components (e.g. a camera with certain properties) and capabilities which can automatically be checked against a robot model to infer if the action will likely be feasible on the given robot.

![Figure 12: Common robot descriptions cover the kinematic and dynamic aspects (left) as well as surface models of a robot (center). A semantic robot description further adds information about the meaning of the different robot parts (right).](image)

7. Experiments

The KnowRon ontology and its sub-ontologies currently include about 8,000 classes describing events and temporal things, actions, objects and spatial things, as well as mathematical concepts and meta-information. There are about 130 action classes, 7000 object types and 150 robot-specific concepts, which can be described by over 300 kinds of

⁴http://www.ros.org/wiki/urdf
properties. The system is being used by several European research projects including RoboEarth\(^5\), RoboHow\(^6\), SRS\(^7\), SMErobotics\(^8\), ACAT\(^9\), and other universities worldwide.

In this section, we will explain how the different representations contribute to the overall goal of understanding vague task instructions. In particular, we emphasize the benefit of having a common representation language that supports a wide range of queries which combine different kinds of information from different sources. Most of the use cases we present here either deal with filling gaps in the instructions or translating between the abstract, symbolic information in the instructions and the continuously-valued grounded information required by the robot.

**Reasoning about robot capabilities.** When processing instructions for a novel task, one cannot assume that all required capabilities are available on the robot. Instead, the system should be able to decide whether important components or capabilities are missing given a novel task description and a model of the robot’s capabilities. Spoken instructions or recipes on the Web do not come with suitable requirement specifications, so usually they need to be derived from the robot’s background knowledge, i.e. the action ontology from which the actions in the task description are inherited. The following example queries ask for the required components and capabilities for a task *MakingPancakes*. The results have been inherited from super-classes of sub-actions of this task, e.g. *Reaching* or *PickingUpAnObject*. The two lower queries determine missing elements by comparing the required ones with those available on a given robot.

```prolog
?- required_comp_for_action (kr: 'MakingPancakes', M).
M = srld2comp: 'ArmMotionController';
M = srld2comp: 'ArmComponent';
M = srld2comp: 'ArmMotionController';
M = srld2comp: 'ArmComponent'

?- required_cap_for_action (kr: 'MakingPancakes', M).
M = srld2cap: 'PickingUpAnObjectCapability';
M = srld2cap: 'ArmMotionCapability';
M = srld2cap: 'PickingUpAnObjectCapability';
M = srld2cap: 'ArmMotionCapability'

?- missing_comp_for_action (kr: 'MakingPancakes', pr2: 'PR2Robot1', M).
false.

?- missing_cap_for_action (kr: 'MakingPancakes', pr2: 'PR2Robot1', M).
M = srld2cap: 'PickingUpAnObjectCapability'
```

**Exchanging information with other robots.** If all hardware components are available, but software components are missing, the robot can try to download information from web-based knowledge bases such as RoboEarth [46], for example more detailed instructions for sub-tasks, environment maps or object models. For formulating the queries and integrating the results with the existing knowledge base, it is important to have all information in a coherent language. The representation language used in RoboEarth [40] is a subset of the representations described in this paper, so information described in that language can simply be loaded into KnowRob. The following query downloads an environment map and an object model; it is formulated using the part-of relation between the building, floor and room. After download, the system automatically downloads models for all object classes whose instances appear in the map if they are not available yet.

```prolog
?- re_request_map_for ([[ 'kr:roomNumber', 3001],
                        ['kr:floorNumber', '3'],
                        ['kr:streetNumber', '45'],
                        ['rdfs:label', 'Karlstrasse']], M).
% Parsed "map.ks.3001" in 0.01 sec; 211 triples
* Missing object models
  * Model for Cup
% Parsed "cup.darkgray_cup.owl" in 0.01 sec; 12 triples
M = roboearth: 'map.ks.3001'.
```

\(^5\)http://www.roboearth.org
\(^6\)http://www.robohow.eu
\(^7\)http://www.srs-project.eu
\(^8\)http://www.smerobotics.org/
\(^9\)http://www.acat-project.eu/
Locating objects in the environment. To ground the abstract object descriptions in an instruction in actual objects in the environment, the robot needs to add actions to search for these objects and to retrieve them from their storage locations. If the objects’ locations are known, they can simply be looked up in the environment map (Section 5.1.6), but often this is not the case. Then, the system has to reason about likely locations given the available knowledge about the locations of other objects and their properties (Section 5.1.1). This can for example be done using generic rules for storage locations [4] or based on a semantic similarity to other objects in the environment [31]. If the inferred location is inside a container, the articulation model stored in the semantic environment map can be used to parameterize actions to open the container. The following query is an example how to obtain the opening trajectory of the container that is inferred to be the most likely storage location for milk; its result is show in Figure 13 (left).

\[ \text{storagePlaceFor}(\text{StPlace}, \text{kr:\ 'CowsMilk\-Product'}). \]
\[ \text{findall}(\text{P}, (\text{owl\_has}(\text{Traj}, \text{kr\: pointOn\trajectory}, \text{P})), \text{Traj}). \]

Integrating experiences and observations of human actions. Observations of human actions and memorized experiences can provide valuable information about objects, motions, locations and other action parameters. In our system, these kinds of memories are stored as instances of the respective action classes. This way, retrieving examples of the execution of some actions reduces to reading all instances of the respective action class. Beetz et al. [5] describe how these kinds of models can be constructed from observations of human actions, Winkler et al. [49] present a system for recording execution logs of robot actions in this format. An example query for the motion of taking a dinner plate out of a cupboard is given below, its results are visualized in Figure 13 (right).

\[ \text{owl\_individual\_of}(\text{A}, \text{kr:\ 'TakingSomething'}). \]
\[ \text{owl\_triple}(\text{A}, \text{kr\: objectActedOn}, \text{Obj}). \]
\[ \text{owl\_individual\_of}(\text{Obj}, \text{kr\: 'DinnerPlate'}). \]
\[ \text{owl\_triple}(\text{A}, \text{kr\: trajectory\=Arm'}, \text{Tr}). \]
\[ \text{owl\_triple}(\text{Tr}, \text{kr\: pointOn\RightTrajectory}, \text{P}). \]

Complementing knowledge about actions and processes. A common problem with task instructions is that humans omit actions that seem obvious to them. In the example in Figure 2, the action to switch on the pancake maker was initially missing – though it is crucial for the success of the overall task, because otherwise the dough would not bake to a pancake. This plan flaw is detected by checking the final result of the plan and by verifying that all action inputs are provided either by objects that can be retrieved from the environment or by the outputs of previous actions. If this is not the case, the effect models of actions and processes are used to plan additional actions that either directly create the required objects indirectly or trigger processes that achieve the desired effects. Figure 2 visualizes the procedure of completing the incomplete instructions. The different colors correspond to the different kinds of information and the mechanisms how they can be acquired and are explained in the legend in the upper right.

16
Selecting functional object parts. Using the part-based object representation, the system can automatically identify and locate relevant functional object parts in CAD models. Once the object is recognized in a scene, these relative locations can be translated into scene-global coordinates that the robot can use for interacting with the object. Below are example queries that are needed for selecting the right parts for pouring pancake batter onto a pancake maker. The query results are visualized in Figure 3. For picking up the bottle of pancake mix, the robot can query for a handle that is part of the bottle instance using the following query:

?- owl:triples(map:mix1, kr:properPhysicalParts, Part),
    annotation_handle(Part, kr:‘Handle’).
Part = kr:‘Cone_wtrOfRtry’.

Being able to identify the functional parts in CAD models, we can define logical rules that describe how action slots are to be filled in terms of these primitives. For example, we could define a rule for selecting the appropriate object part to be used as toLocation for a PouringSomethingOnto action as the largest SupportingPlane. The rule selects all supporting planes that are part of the object, sorts them based on their area and returns the largest one.

pour_onto(Obj, Part) :-
    findall(A-P,
        (owl:triples(kr:properPhysicalParts, Obj, P),
        rdfs:instance_of(P, kr:‘SupportingPlane’),
        rdfs:triples(kr:areaOfObject, P, A), Planes),
        keysort(Planes, PlanesAsc),
        last(PlanesAsc, _-Part).
pour_onto(kr:‘maker1’, Part).
Part = kr:‘FlatPhysicalSurface_UssqOAb’.

For controlling the pouring motion, the part of the bottle that needs to be controlled is the opening. A reasonably general heuristic is to assume that the cone (as generalized cylinder) that is at the top of a bottle is its cap. This heuristic rule can be formulated as follows based on the geometric primitives identified in the CAD model by selecting all cones and sorting them based on their z coordinate.

bottle_cap(Obj, Cap) :-
    findall(Z-P,
        (owl:individual_of(P, kr:‘Cone’),
        rdf:individual_of(P, kr:‘Cone’),
        rdf:individual_of(P, kr:‘Cone’),
        object_part_pos(P, [..,Z]), ConePos),
        keysort(ConePos, ConePosAsc),
        last(ConePosAsc, _-Cap).
bottle_cap(kr:‘pancakemix1’, Cap).
Cap = kr:‘Cone_vcRtyU3K’.

8. Discussion and related work

In this paper, we have discussed how knowledge can help robots to complete vague task instructions by identifying and filling knowledge gaps. Our main insight is that we need a combination of different knowledge areas, different knowledge sources and different inference mechanisms to cover the breadth and depth of required knowledge and inferences. At the same time, all components should be integrated in a coherent knowledge base such that the robot can combine different knowledge sources and inference methods in a single query and integrate their results. We therefore use a shared ontology as common representation and Prolog as interlingua for integrating inference techniques.

Finding appropriate representations for a robot’s knowledge base has been a research topic for decades, dating back to the seminal work on the Shakey [26] robot that already used an internal world model based on predicate logics. Compared to Shakey, modern robots have perception techniques that can handle the complexity of real scenes and manipulation capabilities that allow them to interact with objects. Both aspects lead to significantly more complex scene representations and to a massively increase in knowledge about actions and objects that is required for performing actions.

While various methods have been developed in the AI community for representing and reasoning about temporal relations, action effects and changing situations, most of them focus on individual inference problems: Allen’s interval calculus [1], for example, is mainly used for reasoning about temporal intervals, the Region Connection Calculus [29] extends it to two-dimensional (spatial) problems. Calculi like the Situation Calculus [22, 30] or the related Fluent
calculus [42] focus on the representation and reasoning about changing domains, e.g. caused by robot actions. The Qualitative Process Theory by Forbus [10] allows qualitative inference about physical or chemical processes. Planning languages like STRIPS [9], PDDL [13], or Hierarchical Task Networks [7] specialize in the generation of plans to achieve a given goal. Other rather formalized approaches for representing a robot’s knowledge [43] allow reasoning about what the robot knows and what it does not know, but lack support for e.g. temporal reasoning, detailed spatial representations, information about object types or about processes in the environment.

Similarly to reasoning methods, there are also many knowledge bases that partially cover a robot’s needs: General-purpose knowledge bases like Cyc [20] or SUMO [25] provide a large breadth of encyclopedic knowledge, but often lack the depth in topics like object manipulation that is required by robots. Recent efforts to automate the construction of knowledge bases by extracting encyclopedic knowledge from sources like Wikipedia [50, 33, 15] can provide knowledge in specialized areas, though often not directly relevant for robotics (e.g. countries, people or historic events).

On the other end, representations developed in robotics have largely focused on modeling and reasoning with uncertainty, leading to the development of many sophisticated, though often special-purpose probabilistic models [44]. However, most of these models are specialized for a single modality, for example perception [16], articulation model learning [32], or robot localization [6], and usually lack clear semantics. Just recently, research in semantic environment maps has started to investigate models that combine spatial with grounded semantic representations [12, 52]. Wyatt et al. present a system that integrates geometric and conceptual spatial representations with planning and learning techniques [51]. The system is able to reason about knowledge gaps and tries to resolve them using autonomous learning, though not for object manipulation tasks.

In our research, we try to combine (parts of) these approaches in a common system to provide robots with comprehensive knowledge and inference capabilities. The representation of object poses, for instance, is similar to the Fluent calculus [42], but also stores the provenance of information, i.e. if the robot believes, predicts or desires an object to be at some location. This allows further interpretation of the information as well as describing different (possible) world states without causing conflicts in the knowledge base, which is important if the belief results from noisy perception. The action representation as hierarchical partially-ordered plans with prerequisites, effects, and temporal information is related to Hierarchical Task Networks [7], but extended with qualitative projection methods for processes that are started as side-effects of actions, which are inspired by the Qualitative Process Theory Forbus [10]. Inference about temporal relations between actions or events is implemented according to Allen’s interval calculus [1].

All inferences can be combined in Prolog queries, read the required information from the knowledge stored as OWL statements, and return their results in terms of OWL descriptions. The knowledge base can be populated automatically with perception results and log data of robot actions. Procedural attachments allow symbolic inference about sub-symbolic data by computing semantic information from it on demand at query time. With these capabilities, KnowRon is significantly more expressive and flexible than other robot knowledge bases that purely rely on the rather limited OWL inference, for example ORO [19] or OUR-K [21].

The effective task specifications that are the result of the procedure described in this article are as complete as possible given the robot’s knowledge. However, some decisions are intentionally postponed to execution time, for example where to stand while performing an action or where exactly to put down an object. These action parameters strongly depend on the situation at hand, for example the configuration of obstacles around the object of interest that is very difficult to predict. A parallel research project investigates how these kinds of decisions can be taken using physical reasoning [24].

In general, it needs to be decided how detailed the robot’s models and inferences shall be: As example, take the decision if a robot shall download a task description from the Web given its capabilities and the action’s requirements. The solution we chose, based on the Semantic Robot Description Language, is rather lightweight, but also comparatively shallow, as it does not take the robot’s belief about the current world state into account. There are more sophisticated (recent) approaches [28] that include the current belief to make more detailed predictions if the robot is able to successfully perform the action. The question is, however, how useful the results are in practice: A negative result, for example if knowledge preconditions about object locations are not fulfilled, does not mean the robot cannot perform the task after searching for the items. At the same time, a positive result does not mean that the task will be performed successfully since robot actions tend to fail for a variety of reasons. We therefore chose to limit the complexity of a-priori reasoning to those whatever can reasonably be determined before the execution starts, and to handle other cases during execution using inference on the (then) current world state and using failure handling.
Acknowledgements

This work is supported in part by the EU FP7 Projects RoboEarth (grant number 248942), RoboHow (grant number 288533), and ACat (grant number 600578).

References
