TECHNISCHE UNIVERSITÄT MÜNCHEN
Institut für Informatik
Intelligente Autonome Systeme

Naïve Physics and Commonsense Reasoning for Everyday Robot Manipulation

Lars Kunze

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. Darius Burschka
Prüfer der Dissertation:	1. Univ.-Prof. Michael Beetz, Ph.D., Universität Bremen
	2. Univ.-Prof. Dr. Daniel Cremers
	3. Univ.-Prof. Dr. Joachim Hertzberg, Universität Osnabrück

Die Dissertation wurde am 08.04.2013 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 09.12.2013 angenommen.

For Anke and Max

Abstract

Accomplishing everyday manipulation tasks such as “Making Pancakes” successfully requires autonomous robots to solve many physical reasoning problems. For example, to perform the action of pouring pancake mix into a pan a robot has to infer where to hold the container of the pancake mix, at what angle, and for how long in order to make a proper pancake. Humans reason about such everyday problems seemingly effortless given their experience and common sense. Hence, if we are aiming at robots that master human-scale manipulation, they need to be equipped with similar problem-solving abilities as humans have, namely, the abilities of naive physics and commonsense reasoning.

In this thesis we describe methods that allow robots to reason qualitatively about their own manipulation capabilities, spatio-temporal configurations of objects as well as actions and their effects. To this end, we have developed a formal description language for specifying robot components such as sensors, actuators, and algorithms and matching those specifications against task descriptions to reason about the capabilities of robots. We have also developed interval-based first-order representations called timelines for capturing the state evolution of manipulation tasks performed by humans and/or robots in dynamic environments. These timelines are automatically generated through a novel framework for envisioning the outcome of fully instantiated robot plans on the basis of detailed physical simulations. The developed framework has also been extended to acquire timelines through “Games with a Purpose” in which humans perform manipulation tasks by controlling a robot hand in a virtual environment. By interpreting and learning from timelines robots can understand physical phenomena such as causal effects of parameterized manipulation actions. Finally, we present an approach for crowdsourcing and learning commonsense knowledge about spatial relations using web-based “Games with a Purpose”.

Experiments show how the complementary methods that have been developed in the context of this thesis enable robots to reason about a wide spectrum of commonsense reasoning problems, whereby the overall problem-solving competence of autonomous robots can be improved.

Kurzfassung

Das erfolgreiche Erledigen von alltäglichen Manipulationsaufgaben wie dem Zubereiten von Pfannkuchen erfordert von autonomen Robotern das Lösen einer Reihe physikalischer Inferenzprobleme. Um beispielsweise Pfannkuchenteig in eine Pfanne zu schütten muss ein Roboter inferieren, wo er den Behälter des Pfannkuchenteigs hält, in welchem Winkel und wie lange er schütten muss. Menschen stellen über derartige Alltagsprobleme anscheinend mühelos Inferenzen an, da sie sowohl über Erfahrung als auch einen gesunden Menschenverstand verfügen. Wenn also Roboter auf Menschen zugeschnittene Manipulationsaufgaben lösen sollen, müssen diese auch mit den Menschen ähnlichen Problemlösungsfähigkeiten ausgestattet werden — insbesondere mit naiv-physikalischem und Common-Sense-basiertem Schlussfolgern.

In dieser Arbeit beschreiben wir Methoden die es Robotern ermöglichen, qualitative Schlussfolgerungen über ihre eigenen Manipulationsfähigkeiten, raum-zeitliche Konfigurationen von Objekten, sowie Aktionen und deren Effekten durchzuführen. Zu diesem Zweck haben wir eine formale Beschreibungssprache zur Spezifikation von Roboterkomponenten wie Sensoren, Aktuatoren und Algorithmen entwickelt. Diese Spezifikationen können dann mit Aufgabenbeschreibungen abgeglichen werden, um über die Fähigkeiten eines Roboters zu schlussfolgern. Des Weiteren haben wir sogenannte „Timelines“ entwickelt, eine Intervall-basierte Logiksprache erster Ordnung, mit der sich der Ablauf von durch Mensch und Roboter ausgeführten Manipulationsaufgaben repräsentieren lässt. Derartige „Timelines“ können durch ein neu entwickeltes Framework, welches den Ausgang von vollständig spezifizierten Roboterplänen auf der Basis von detaillierten Physiksimulationen vorhersagt, automatisch generiert werden. Wir haben das Framework um die Möglichkeit erweitert, „Timelines“ auch durch interaktive Computerspiele, sogenannte „Games with a Purpose“, zu akquirieren. Hierbei führen Menschen Manipulationsaufgaben in einer virtuell simulierten Umgebung durch. Durch die Interpretation und das Lernen auf der Basis von „Timelines“ können Roboter physikalische Phänomene wie zum Beispiel kausale Effekte von parametrisierten Manipulationsaktionen verstehen. Zum Schluss stellen wir einen Crowdsourcing-Ansatz zum Wissenserwerb von aufgabenrele-

vantem Common-Sense-Wissen über räumliche Relationen durch web-basierte „Games with a Purpose“ vor.

Experimente zeigen wie es die komplementären Ansätze, welche im Rahmen dieser Arbeit entwickelt wurden, Robotern ermöglichen, über ein weites Spektrum von Common-Sense-Problemen zu inferieren. Hierdurch kann die Problemlösungskompentenz von autonomen Robotern insgesamt verbessert werden.

Acknowledgments

This work would not have been possible without the support of many people. In particular, I would like to thank:

- my supervisor, Prof. Michael Beetz, Ph.D., for his constant support and encouragement throughout my graduate studies. I am very grateful for many opportunities, invaluable discussions and ideas.
- my co-advisor, Prof. Dr. Joachim Hertzberg, for the interesting discussions when he was at TUM, for his feedback to my work and for serving on my thesis committee.
- Prof. Masayuki Inaba, for hosting me three months in the Jouhou System Kougaku (JSK) Laboratory at The University of Tokyo. I am really grateful to Prof. Kei Okada for his assistance and guidance. I would also like to thank all students in the JSK Lab for their help; in particular Manabu Saito and Haseru Azuma. Finally, I would like to thank Rosen Diankov and Asako Kanezaki for all the interesting conversations during my time in Japan.
- my collaborators, friends, and lab-mates at TUM. Thank you for all the useful discussions and feedback to my work. Thanks to Moritz, Lorenz, Dejan, Dominik, Thomas, Karinne, Daniel, Michi, Thibault, Mihai, Zoltan, Nico B., Ingo, Alexis, Federico, Murat, Christoph, Zahid, David G., David W., Alex, Tina, Andreas, Jan, Nico v. H., Bernhard, Séverin, Radu, Suat and Freek.
- the students whom I have supervised and worked with throughout my studies. Thanks to Emitza, Tobias, Reinhard, Johannes, Raphael, and Andrei.
- my family for their love and support. Thanks to my parents Sigrid and Heinz for their love and constant support throughout my life. Further, I deeply thank my great-aunt Helma, my grandma Hilde, and my parents-in-law Beate and Horst. Finally, I would like to thank Anke and Max for being there.

Thank you very much for your support and understanding.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Robot technology has tremendously improved over the past decades. Today, special-purpose robots are entering our daily lives in forms of vacuum cleaners, lawn mowers and more recently telepresence robots. At the same time a new generation of integrated robot systems with general-purpose manipulation capabilities is under research and development. In the future, we will see the latter type of robots as personal assistants, care takers, and co-workers in our households, homes for the elderly, and factories where they will competently perform everyday activities (Bicchi et al., 2007; Hollerbach et al., 2009) (Figure 1.1). Given the latest developments in robotics hardware and software robots have made a considerable progress towards human-scale everyday manipulation. The wide availability of compliant robot platforms like the PR2[1] and open-source software frameworks like the Robot Operation System (ROS)[2] greatly facilitated this progress. However, the task spectrum of robots is still limited, mainly because current control mechanisms are over-specialized and not flexible enough. That is, scaling the task repertoire of autonomous mobile robots towards an open-ended set of human-scale tasks like doing laundry and preparing meals requires novel ways of programming and controlling robots. We strongly believe that robots that are to accomplish everyday manipulation tasks successfully in human environments require reasoning capabilities similar to those of humans deeply embedded within their control programs. In this work we will call such control methods and robots *cognition-enabled*.

[1]`http://www.willowgarage.com/pages/pr2/overview`
[2]`http://ros.org`

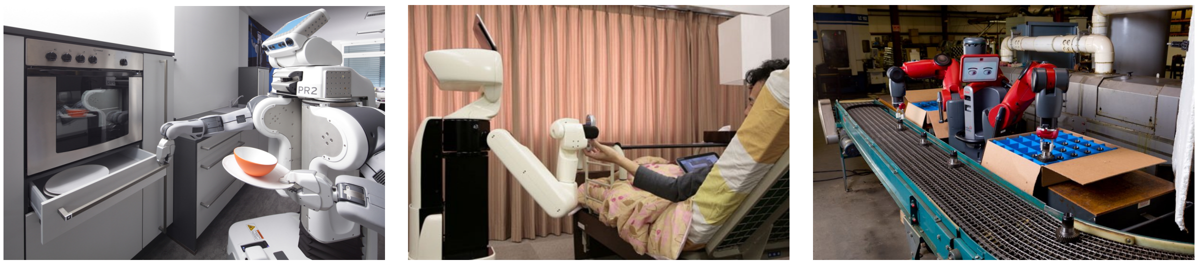


FIGURE 1.1 Robot platforms with general-purpose manipulation capabilities. Left to right: PR2 as household assistant, Human Support Robot as care taker, and Baxter as co-worker. Courtesy: Intelligent Autonomous Systems (IAS), Toyota, Rethink Robotics.

1.1 Challenges for Everyday Robot Manipulation

Human environments exhibit properties that are quite challenging for everyday robot manipulation. These properties are diametrically opposed to those of a traditional industrial setting. Figure 1.2 illustrates the different working environments of industrial and personal robots. Industrial robots are made to perform repetitive tasks with high precision at a very high speed. Often they are not equipped with any kind of sensor and therefore they are only operated using open-loop control, that is, without any feedback from the environment. Within a typical industrial setting the work and task space of a robot is completely specified. The environment of the robot is static and behaves deterministic. The robot performs only episodic tasks. These properties stand in stark contrast to those in an everyday household setting where the environment is only vaguely specified and partially observable by the robot. Additionally, the robot has to account for dynamic and uncertain aspects of the environment. Previously made decisions can also have long-term effects within the robot's course of action. Hence, robots need flexible and robust control and reasoning mechanisms in order to cope with the inherent properties of human environments.

In contrast to the traditional industrial setting, everyday manipulation requires robots to take quite a number of decisions within their course of action autonomously. Even a seemingly simple action like picking up an object requires a robot to decide on where to stand, which hand to use, how to reach for the object, where to grasp it, how to grasp it, how much force to apply to it, how to lift it, and how to hold the object. Such decisions should not be made once and for all, but rather based on the context. That is, task-related objects, their states, the environment and the overall task determine how these decisions have to be made in an appropriate way. Employing action-specific models based on the context and the consequences of actions within their control programs robots are enabled to make informed decisions and thereby can improve their overall performance.

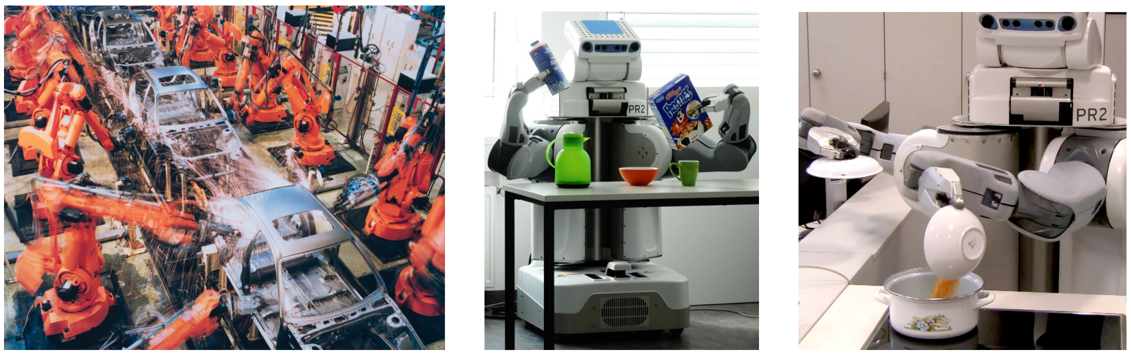


FIGURE 1.2 Robots in an industrial and household setting. Left: Industrial robots perform repetitive tasks with high precision at a car assembly line under real-time conditions. Middle: PR2 performs pick-and-place actions in the context of setting a table for breakfast. Right: PR2 performs a complex meal preparation task in which it pours a granular fluid from a bowl into a preheated cooking pot. Courtesy: ABB (left), IAS (middle, right).

In the long run, we envision cognition-enabled robots to perform everyday manipulation tasks at a similar scale as humans. Those robots will perform and learn these tasks based on naturalistic action specifications like, for example, *clean the table*. Reasoning components that make sense of such ambiguous and under-specified instructions play a major role for realizing systems with these capabilities. In particular, robots have to infer *what* they have to do and also *how* they have to do it. The latter is especially important because the outcome of a manipulation action highly depends on the way it is executed. Therefore it is crucial to infer the missing information about the right parameterization of an action in order to bring about the desired physical effects. In this thesis we refer to this kind of inference as *naive physics reasoning*.

Overall we have seen that everyday manipulation in human environments poses many challenges to robots. A good overview of these challenges is given in (Kemp et al., 2007). One of them is *Common Sense for Manipulation*. Enabling robots to do commonsense reasoning about everyday manipulation can be achieved by integrating methods from Artificial Intelligence (AI) into robot control mechanisms. In order to develop control programs that exhibit robust and flexible robot behavior the reasoning components have to incorporate knowledge about the robot platform, the manipulation task, the environment and the situated context. To this end, this work provides means to equip robots with naive physics and commonsense knowledge that allow for reasoning about everyday robot manipulation. But before we lay out what reasoning components robots need in order to perform everyday manipulation tasks successfully, we will first look at the example of making pancakes to illustrate what makes everyday robot manipulation so complicated.

1.2 Beyond Pick-and-Place: "Making Pancakes"

In this thesis we consider "Making Pancakes" as our running example by which we illustrate what knowledge robots need to competently accomplish everyday manipulation tasks. Understanding everyday physical phenomena, that is representing and reasoning about them, is an endeavor in the field of Artificial Intelligence which dates at least back to the work of Hayes (1979). More recently, there has been work on realistic physical reasoning problems like, for example, "Cracking an Egg" (Morgenstern, 2001). This and similar problems as well as solutions to some of them can be found on the Common Sense Problem Page[3]. Basically, the web page can be considered as a collection of benchmark problems for commonsense reasoning. Often these problems are accompanied by a number of variants. Solutions to one of these problems should, first, apply to other problems (at least partially), and second, account for a wide range of variants.

In analogy to this collection of benchmark problems, we have formulated the problem of "Making Pancakes" as follows:

A robot pours a ready-made pancake mix onto a preheated pancake maker. Properly performed, the mix is poured into the center of the pancake maker without spilling where it forms a round shape. The robot lets it cook until the underside of the pancake is golden brown and its edges are dry. Then, the robot carefully pushes a spatula under the pancake, lifts the spatula with the pancake on top, and quickly turns its wrist to put the pancake upside down back onto the pancake maker. The robot waits for the other side of the pancake to cook fully. Finally, it places the pancake using the spatula onto an upturned dinner plate.

whereby a solution to the problem should take the following variants into account:

What happens if: the robot pours too much pancake mix onto the pancake maker? too little? the robot pours the mix close to the edge of the pancake maker? the robot flips the pancake too soon? too late? the robot pushes only half of the spatula's blade under the pancake? the robot turns its wrist too slow? the robot uses a knife/fork/spoon to flip the pancake? the pancake mix is too thick? too thin? the ingredients of the mix are not homogeneously mixed?

[3] `http://www.commonsensereasoning.org/problem_page.html`

Humans can understand the challenge problem of “Making Pancakes” easily and can immediately answer questions about it. However, for robots this problem formulation is highly under-specified and therefore they need means to acquire the relevant knowledge which enables them to perform and reason about the task.

Furthermore, this challenge problem illustrate the difference of *carrying out* and *mastering* an everyday manipulation task like making pancakes. This difference becomes evident when a robot is presented with slight variations of the problem. A robot is only *mastering* a task if it is able adapt its behavior to handle also variations of a problem. That is, it understands the nature of the task and knows about related problems. Furthermore, the robot knows what it is doing, how it is doing it, and why it is doing it. Given this knowledge about the task, the robot is able to answer questions about its own decisions and its overall performance.

Reasoning about this challenge problem involves different kinds of knowledge. To illustrate what information is needed to competently answer questions about this problem let us consider the two principle actions of the task in greater depth, namely *pouring the pancake mix* and *flipping the pancake*. Within the problem description these two actions are described as follows:

POURING *“A robot pours a ready-made pancake mix onto a preheated pancake maker. Properly performed, the mix is poured into the center of the pancake maker without spilling where it forms a round shape.”*

FLIPPING *“Then, the robot carefully pushes a spatula under the pancake, lifts the spatula with the pancake on top, and quickly turns its wrist to put the pancake upside down back onto the pancake maker.”*

A robot about to make a pancake has to reason about various aspects with respect to the pouring and flipping actions mentioned above. At first, it has to asses whether it is at all capable to perform the actions given its own specification. For example, can it lift the container of pancake mix which has a certain weight? Further, the robot has to reason how it could perform the action in order to meet its respective constraints. For example, how should it pour the mix in order to make a round shaped pancake? Additionally, the robot should be aware of potential problems that might occur during the course of action. For example, how could it avoid spilling some mix onto the table? Table 1.1 gives further examples of different types of knowledge that a robot should know about in order to master both pouring and flipping.

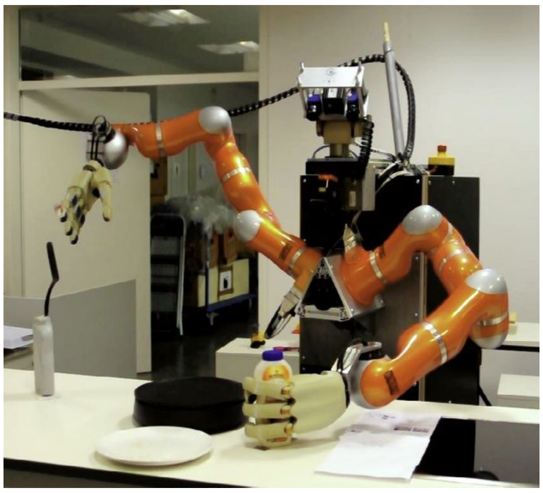
TABLE 1.1 Examples of different kinds of naive physics and commonsense knowledge.

Knowledge	Pouring	Flipping
Capabilities	◇ able to grasp bottle? ◇ able to lift bottle? (*weight*)	◇ able to grasp spatula? ◇ able to turn wrist *quickly*?
Spatial Concepts	◇ where is the *center*? ◇ where is *close to* the edge?	◇ where is *under the pancake*? ◇ what means *upside down*?
Actions	◇ where to hold the bottle? ◇ for how long?	◇ at what angle to push? ◇ with how much force?
Effects	◇ spilling of pancake mix ◇ too less/much pancake mix	◇ push pancake off the oven ◇ pancake stuck on spatula

Based on the challenge problem of “Making Pancakes”, we have identified three major categories of naive physics and commonsense knowledge which are subject of this work, namely robot capabilities, spatial concepts, and actions and effects.

Robot Capabilities Robots need mechanisms to assess their own capabilities when confronted with a task. Do they have the appropriate resources in terms of sensors, actuators, algorithms and models to accomplish a task? For example, can a robot hand exert a certain force onto another object and can it be controlled at a certain speed? Can the robot apply enough force to lift the container holding the pancake mix? And can it turn its wrist *quick* enough to flip the pancake? If robots know about their own capabilities and limitations, they could asses which everyday manipulation tasks they are able to perform and/or what components are missing in order to do so. This knowledge is also helpful for the distribution of tasks and the exchange of information and skills between multiple robot platforms.

Spatial Concepts Everyday manipulation requires robots to have a sound understanding of, sometimes vaguely specified, spatial concepts. For example, when is a fluid like the pancake mix *on* the pancake maker? And when is a spatula *under* the pancake? In the latter case, it is actually only the spatula’s blade that has to be under the pancake. Many of these spatial configurations can be assessed by looking at a particular situation. However, some spatial concepts like *pancake upside down* involve at least two situations, that is, reasoning over time. In general, knowledge about spatial relationships between objects is an important aspect for understanding everyday manipulation and a prerequisite for the next category, namely actions and their effects.



How to pour the pancake mix?
◇ where to hold the bottle of pancake mix?
◇ at what height?
◇ at what angle?
◇ for how long? . . .

How to flip the pancake?
◇ how to push a spatula under the pancake?
◇ at what angle?
◇ with how much force?
◇ how to lift the pancake? . . .

FIGURE 1.3 Left: Robot Rosie about to make a pancake. Right: Questions Rosie has to answer to accomplish the task. Courtesy: IAS.

Actions and Effects We consider knowledge about actions and their effects as the most important type of knowledge in the context of everyday manipulation. By following the description of the problem of “Making Pancakes”, a robot can acquire some basic knowledge about the task. It does however not know *what could happen* when the robot itself performs the task in a certain way. The knowledge how to perform a task is obtained by what we call here *envisioning* (de Kleer, 1977). Figure 1.3 highlights some questions a robot has to answer in order to accomplish the task successfully.

The main actions of this task are pouring the mix and flipping the pancake where the latter action can naturally be split into the following sub-actions: pushing the spatula under the pancake, lifting the pancake, and turning the spatula.

Pouring the mix onto the pancake maker successfully requires that the container holding the mix is positioned at a certain height over the pancake maker. At this position the container has to be tilted for some time at a certain angle, so that the mix flows out onto the pancake maker. Figure 1.4 shows a robot pouring a pancake mix onto a pancake maker.

As mentioned above, flipping the pancake can be considered as several sub-actions. For pushing the spatula under the pancake the spatula has to be hold at an appropriate angle to get under the pancake. When lifting and turning the pancake the spatula has to be tilted at a certain height that the pancake falls off and lands upside-down on the pancake maker. Figure 1.5 shows some images of Rosie performing the flipping action.

The pouring and the flipping actions performed by the robot could have various effects ranging from undesired to desired. Some of the undesired effects are depicted in Fig-

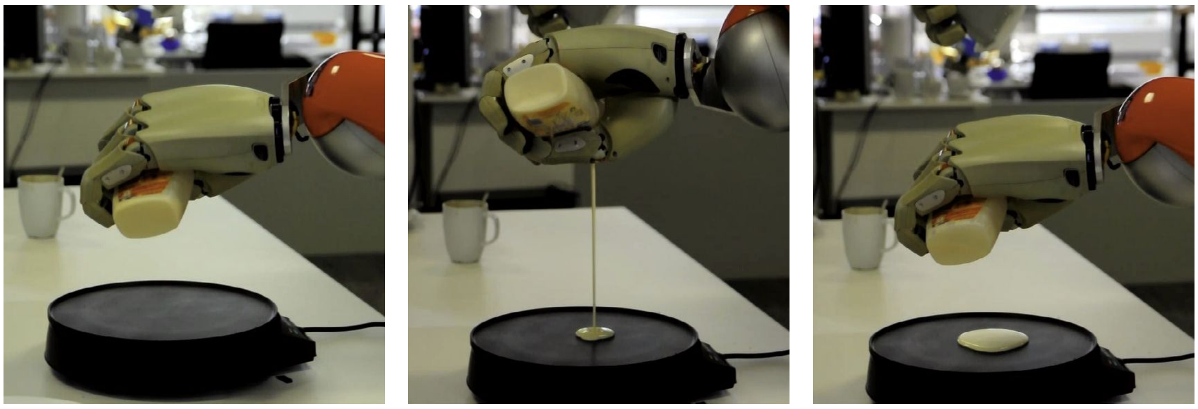


FIGURE 1.4 Rosie pouring mix onto the pancake maker. Courtesy: IAS.

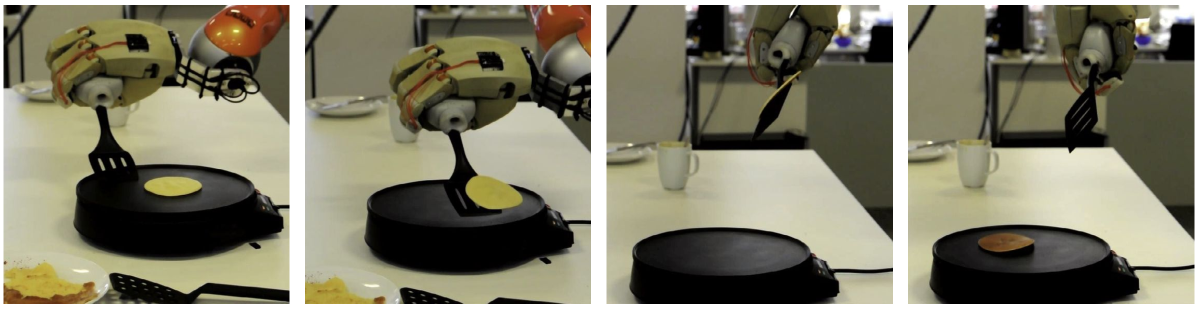


FIGURE 1.5 Rosie flipping a pancake. Courtesy: IAS.

ure 1.6. During the pouring action, the robot could spill some pancake mix onto the table or it could only pour the mix onto the pancake maker with lots of splashes. During the flipping action, robot could damage the pancake by touching it from the top or the pancake could get stuck to the spatula.

Since knowledge about actions, their effects and related problems plays a key role in naive physics and commonsense reasoning for everyday robot manipulation, it is the primary focus of this thesis.

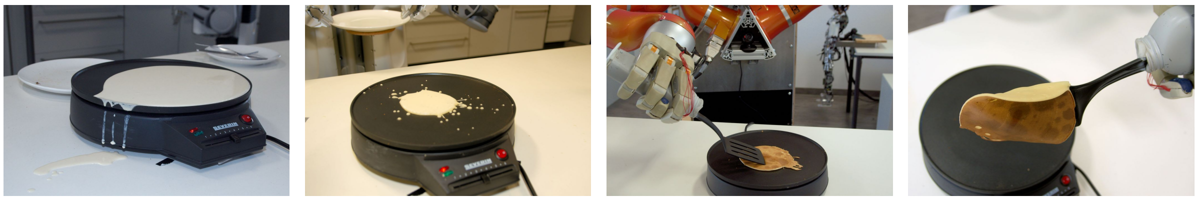


FIGURE 1.6 Physical Behavior Flaws: pancake mixed spilled, spatula placed over the pancake, pancake stuck on spatula. Courtesy: IAS.

1.3 Need for Naive Physics and Common Sense

How can humans accomplish and reason about a task like making pancakes given fairly abstract natural language descriptions as presented in the previous section? Two important factors that allow humans to do this are their everyday experience and their common sense. But what are the computational principles that enable robots to master such tasks? Before we present our approach in Section 1.4, let us first consider different kinds of questions a robot has to answer to demonstrate that it understands a task, and second, how some of these questions are seen from the fields of Artificial Intelligence and Cognitive Science.

1.3.1 Reasoning Problems

The example of making pancakes described in Section 1.2 illustrates why everyday manipulation is a hard problem and why robots should be equipped with the human-like capabilities of naive physics and commonsense reasoning. In particular, robots should be able to answer some of the following questions:

MONITORING What is the expected outcome of an action?

PLANNING Which action will lead to an intended goal?

DIAGNOSIS What has caused something to happen?

QUESTION ANSWERING Why has an action being performed?

REINFORCEMENT LEARNING When does the outcome of an action change qualitatively?

IMITATION LEARNING What is the intended outcome of an action?

These questions illustrate exemplarily to what types of problems this kind of reasoning can be employed in order to adjust the behavior and/or to improve the overall performance of robots.

Within this line of research we are interested in how robots can understand the underlying principles of everyday manipulation. That is, they should not only be able to perform a task, but rather master it. We have designed and implemented methods to envision the outcome of single actions like reaching for an object and complex tasks through *mental* simulations. We also consider examples of human performances to inform robots about important parameters of their actions. Furthermore, we investigate how robots themselves can assess whether they are physically capable of performing a task.

Given that the underlying ideas and the developed methods are applicable to a wide range of problems, we believe that this work can have a broad impact in field of Robotics.

1.3.2 An Artificial Intelligence Perspective

Autonomous robots assistants that can operate in human-scale environments has always been a great motivation to researchers in the fields of Artificial Intelligence (AI) and Robotics. Maybe the earliest example of an integrated robot system was *Shakey* realized by SRI's Artificial Intelligence Center[4] in the late 1960s (Nilsson, 1984). However, since then, research in AI and Robotics has mainly been focused on isolated problems such as knowledge representation and reasoning, planning, uncertainty, learning, and perception. Therefore, another big challenge remains the integration of the developed methods from AI and Robotics. This demand has already been recognized by researchers of both fields and is documented, for example, by a series of symposia titled *"Designing Intelligent Robots: (Re)-integration of AI"*[5]. Also, several European projects target at the integration of methods of both fields, namely IntellAct[6], RACE[7], Xperience[8], RoboEarth[9], First-MM[10], and RoboHow[11]. In a similar light, this work integrates symbolic approaches from the area of knowledge representation and (probabilistic) reasoning and learning with methods from the area of Robotics.

Within Artificial Intelligence (AI), the problem of reasoning about actions was often considered in the area of classical planning. But in the context of Robotics the problem has to be approached in a different way, because, first, the open-endedness of tasks makes classical planning intractable, and second, robot control programs are not represented adequately by sequences of actions as noted by McDermott (1992). Logical axiomatizations for representing and reasoning about actions and their effects have also been developed for problems like, for example, "Cracking an Egg"[12] (Morgenstern, 2001). However, when temporal projections are made on the basis of logical formalizations physical details of

[4]http://www.sri.com/about/organization/information-computing-sciences/aic
[5]http://people.csail.mit.edu/gdk/dir2
[6]http://www.intellact.eu
[7]http://project-race.eu
[8]http://www.xperience.org
[9]http://www.roboearth.org
[10]http://www.first-mm.eu
[11]http://www.robohow.eu
[12]http://www.commonsensereasoning.org/problem_page.html

the manipulation actions are abstracted away and variants of the problem could only be handled by extending the underlying theory. To enable robots to reason about the future, they have to predict the effects of their actions before committing to them which requires the handling of an enormous amount of interdependent temporal data (Dean, 1989).

1.3.3 A Cognitive Science Perspective

Evidences from cognitive psychology and the neurosciences show that humans perform their actions based on the expected consequences of their actions (Haazebroek and Hommel, 2009; Ingram et al., 2010). One of the important factors that determines how humans perform an action is, for example, the end-state comfort (Weigelt et al., 2006). Mirror neurons allow humans to perform mental simulations and thereby enable us to recognize and understand the outcome of actions (Oztop et al., 2006). The simulation theory of cognition is mainly based on three components: firstly, behavior can be simulated, secondly, perception can be simulated, and thirdly, real and simulated actions can provoke perceptual simulations of their most likely consequences (Hesslow, 2012). Thus, the question *"What would happen if I perform this action?"* can be answered by simulating the action and looking at the simulated perceptual outcome. Subsequently, the perceptual outcome can serve as stimulus for new simulated behavior. Evidences show that even high-level cognitive processes are grounded in bodily-based simulations (Svensson and Ziemke, 1999).

1.4 Approach

The basic idea of our approach is to equip robots with reasoning (or cognitive) capabilities similar to those of humans. We embed reasoning components deeply within robot control programs. Thereby, programmers can write very general control programs in a concise way. Task and context related decisions are made and action parameters are determined based on the underlying models. The following excerpt of Lisp pseudo-code illustrates the basic idea of constraint-based action specifications where an action has to fulfill a set of constraints:

```
(perform (an action
             (attribute/constraint$_1$ value$_1$)
             ...
             (attribute/constraint$_n$ value$_n$)))
```

A more complex example of a pouring action is shown in the following where the type of the action is *pour*, the object *?obj* is a part of an object of type *pancake-mix* contained in a *mug*, and the destination *?loc* is a location on the *pancake-maker*. The desired effect of the action is a conjunction of several constraints. The object bound to the variable *?obj* should be of small size and have a round shape. Furthermore, the location bound to *?loc* should be centered on the pancake-maker. An undesired effect of the pouring action is the spilling of *?obj* of type *pancake-mix*.

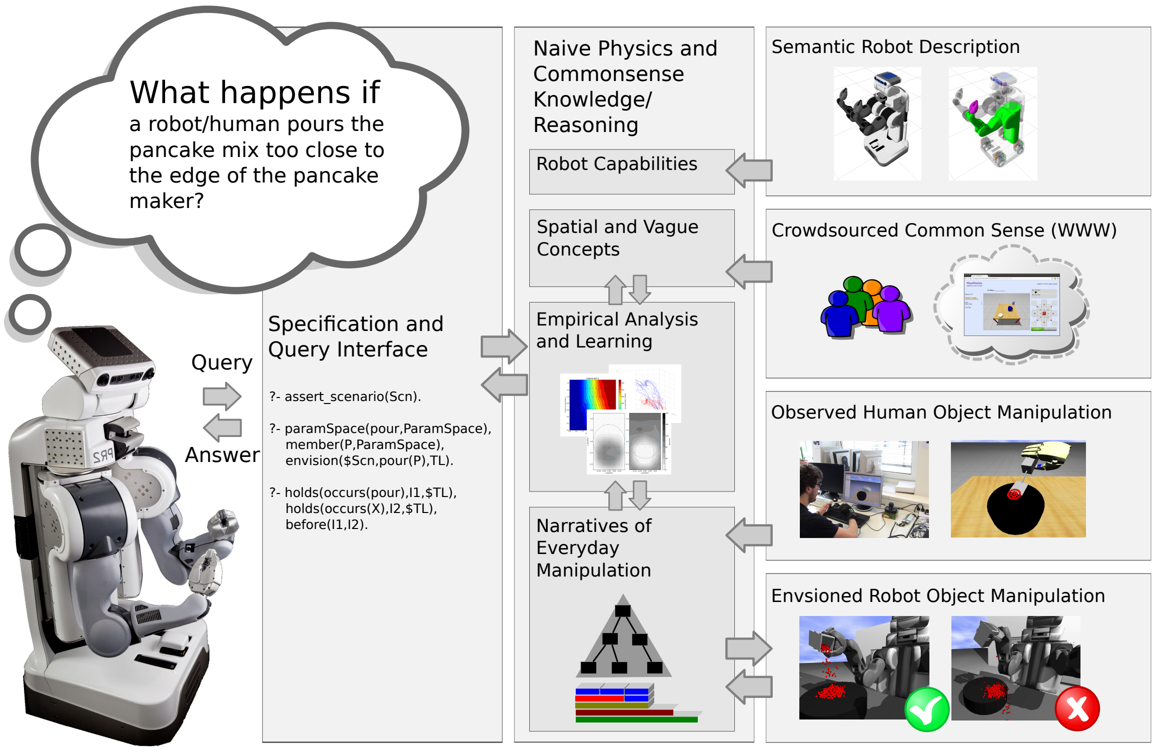
```
(perform (an action
             (type pour)
             (object ?obj = (an object-part
                                 (contained-in mug)
                                 (type pancake-mix)))
             (destination ?loc = (a location
                                     (on pancake-maker)))
             (desired-effect (and (size ?obj small)
                                  (shape ?obj round)
                                  (centered ?loc pancake-maker)))
             (undesired-effect (spilled ?obj counter))))
```

The above example should give the reader an idea of how we envision the use of naive physics and commonsense reasoning within cognition-enabled robot control. However, the aim of this thesis is the acquisition of knowledge and the generation of models that can be employed for this kind of reasoning.

Figure 1.7 gives a functional overview of the different components at a very high level. The main goal of this work is to equip robots with naive physics and commonsense reasoning capabilities that allow them to answer questions in the context of everyday robot manipulation tasks. To this end, we obtain knowledge from various resources and transform it into powerful models that can be queried by the robot and employed during execution.

The first information resource is robot itself. More precisely, a formal specification of the robot platform. Given a description of the robot's sensors, actuators and software components we are able to match it with a particular task description. The individual task steps require certain capabilities which might be provided by the set of components a robot has. Thereby, the robot is able to asses whether it is capable to perform a task.

A second resource is the World Wide Web. We crowdsource information about directional spatial relations and the proximity between objects using "Games with a Purpose".


What happens if
a robot/human pours the pancake mix too close to the edge of the pancake maker?
Query
Answer
Specification and Query Interface
Naive Physics and Commonsense Knowledge/ Reasoning
Robot Capabilities
Spatial and Vague Concepts
Empirical Analysis and Learning
Narratives of Everyday Manipulation
Semantic Robot Description
Crowdsourced Common Sense (WWW)
Observed Human Object Manipulation
Envsioned Robot Object Manipulation

```
?- assert_scenario(Scn).

?- paramSpace(pour,ParamSpace),
   member(P,ParamSpace),
   envision($Scn,pour(P),TL).

?- holds(occurs(pour),I1,$TL),
   holds(occurs(X),I2,$TL),
   before(I1,I2).
```

FIGURE 1.7 Functional view of naive physics and commonsense reasoning.

These are computer games with the purpose to solve problems in a certain domain. That is, by playing such games users provide valuable information about a problem. In our case, we employ Games with a Purpose to acquire information about spatial relations from Internet users. Based on this information we learn models that can determine the spatial relations of objects within a manipulation scenario, for example, a breakfast table. Furthermore these models can also generate geometric poses given a qualitative spatial relation.

A third information resource are humans. We observe humans performing manipulation tasks at a table with a fixed set of objects. We have built a virtual manipulation environment in which a user can manipulate virtual objects using a data glove and/or a game controller. The acquired data sequences, also called narratives, are analyzed with respect to behaviors of interest that can inform robots how to select meaningful values for their parameters within their robot control programs.

A fourth and very essential part for the acquisition of naive physics and commonsense knowledge about manipulation tasks is the envisioning component. We have designed and implemented a framework by which a robot can envision the qualitative outcome of its parameterized manipulation actions. The framework takes a formal description of a

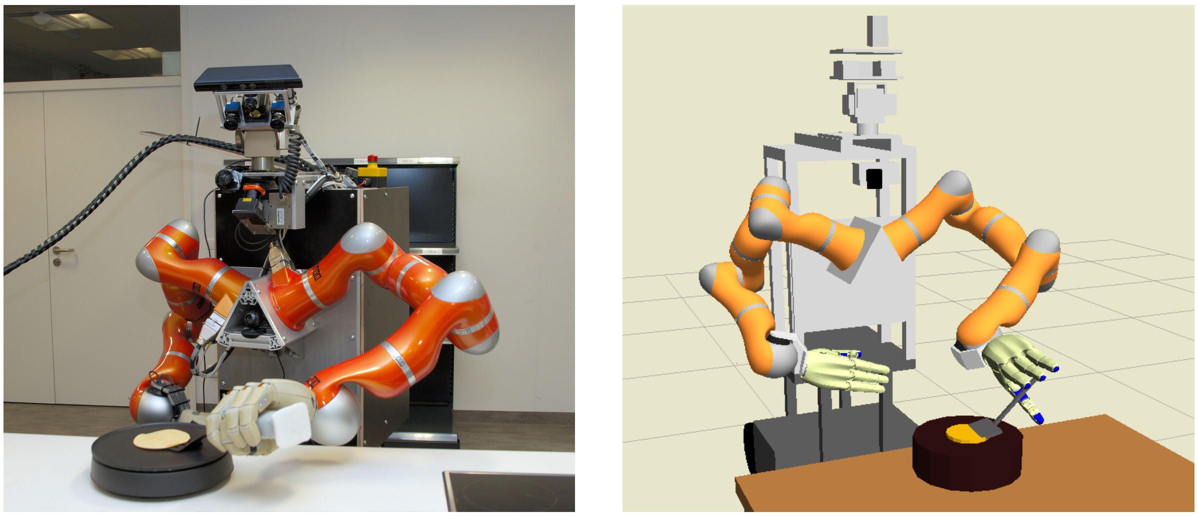
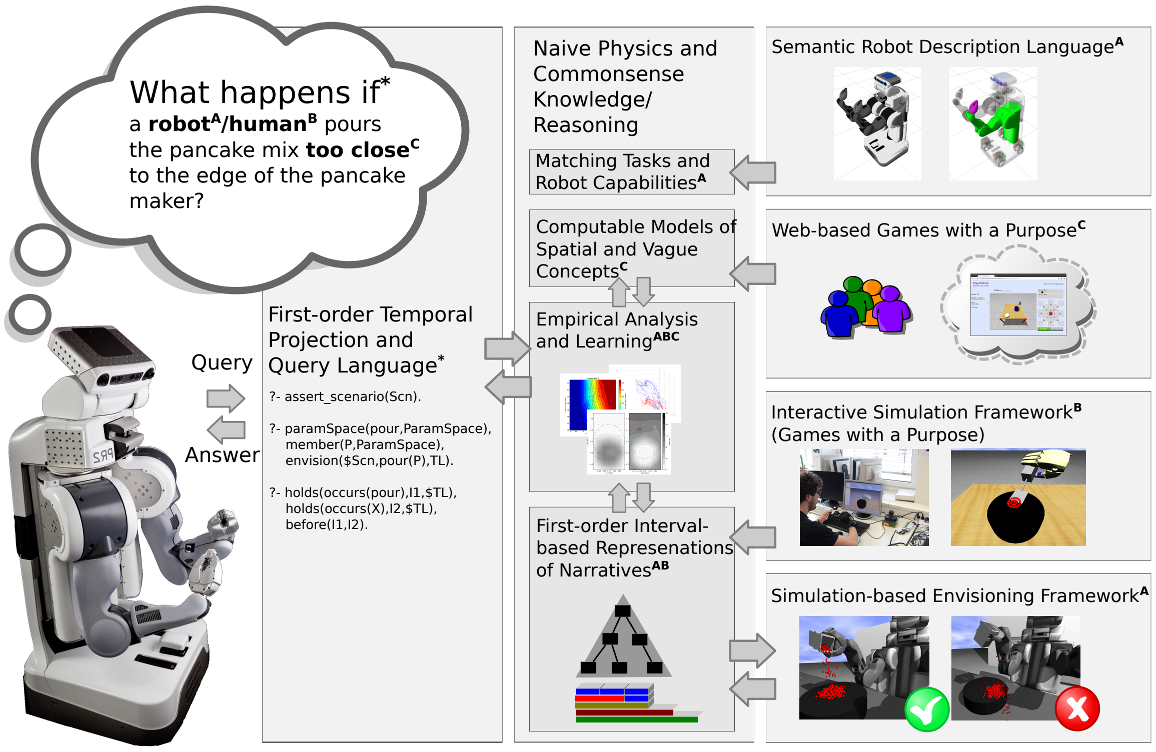


FIGURE 1.8 Left: Rosie pushing the spatula under the pancake. Right: Envisioning the *pushing* action using a physics-based simulation. Courtesy: IAS.

manipulation scenario and a parameterized instance of a robot control program as its input and generates a symbolic timeline (or narrative) grounded in the logged data structures of physics-based simulations. Based on the timelines, robots can answer questions related to the outcome of their actions.

By considering the manipulation problem of making pancakes, we aim at finding appropriate representations and inference mechanisms that enable robots to predict the effects of their own actions which depend very much on the way the actions are executed, i.e., their parameterizations. Therefore we have designed, implemented, and analyzed a framework that allows us to envision the outcome of parameterized robot actions based on physical simulations (Kunze et al., 2011b,a). Figure 1.8 shows the robot Rosie pushing a spatula under a pancake and envisioning the action through mental simulation. However, within our research we do not aim to interpret the physical effects at a very detailed level, but rather at a coarse one. That is, we strive for an abstraction that gets the qualitative behavior right by which we can understand how physical effects depend on the parameters of the respective manipulation actions. Furthermore, the developed representations and inference mechanisms should allow to diagnose and to revise the executed actions. In the context of learning they could also make it possible to direct the exploration in the search space based of the information of causal models.

Given the strong dependence of physical effects and concrete action parameterizations it is hard to predict the outcome of an action solely on abstractions. In particular within Robotics it is crucial that robots perform actions which maximize the expected utilities. Let us illustrate this problem by a robot navigation task where a robot has to find a position


What happens if*
a robotA/humanB pours the pancake mix too closeC to the edge of the pancake maker?
Query
Answer
First-order Temporal Projection and Query Language*
Naive Physics and Commonsense Knowledge/ Reasoning
Matching Tasks and Robot CapabilitiesA
Computable Models of Spatial and Vague ConceptsC
Empirical Analysis and LearningABC
First-order Interval-based Represenations of NarrativesAB
Semantic Robot Description LanguageA
Web-based Games with a PurposeC
Interactive Simulation FrameworkB (Games with a Purpose)
Simulation-based Envisioning FrameworkA

```
?- assert_scenario(Scn).

?- paramSpace(pour,ParamSpace),
   member(P,ParamSpace),
   envision($Scn,pour(P),TL).

?- holds(occurs(pour),I1,$TL),
   holds(occurs(X),I2,$TL),
   before(I1,I2).
```

FIGURE 1.9 Contributions to naive physics and commonsense reasoning.

from where it can reach a cup. If the robot only infers qualitatively that the cup is in the kitchen and stands on a table, and then, uses this information to move to a place in front of the table, the robot might not be able to reach the cup, e.g., because of its kinematic structure and/or kinematic singularities of the robot's manipulators. The concept of ARPLACE (Stulp et al., 2012) solves such problems by updating the decision about the target place as soon as new task information is available. Thereby, for example, information about obstacles and the robot's kinematic is effectively incorporated within the decision.

1.5 Contributions

In this work we have integrated methods of the fields of Artificial Intelligence and Robotics in order to reason about robot manipulation tasks qualitatively. To this end, we have realized several open source components which integrate information of different resources, namely, robot specifications, the World Wide Web, observations of human object manipulations, and simulated robot manipulations, in a coherent framework. The main contributions of this work are as follows:

- as the main contribution of this thesis we have designed, implemented and analyzed a framework for envisioning robot manipulation tasks based on parameterized physics-based simulations (Chapter 4 and 5). To this end:
  - we have established an interface for parameterizing and controlling physics-based simulations using the logic programming language PROLOG[13];
  - we have linked first-order representations to physical object models that can be instantiated in simulation;
  - we have started a library of physical object models and controllers realizing specialized physical behaviors which are not covered by rigid-body simulations, for example, the breaking of an egg and the mixing of liquids;
  - we have developed a monitoring and logging mechanism that observe data structures of interest within the simulator; and
  - we have introduced interval-based first-order representations (called timelines) that tightly integrate subsymbolic and symbolic information from logged simulations as a powerful means for reasoning about narratives.
- we have extended the framework mentioned above with an interface that can be used to process data of object manipulation tasks performed by humans through Games with a Purpose played in a virtual environment. The captured data has been used to analyze the parameter space of particular manipulation actions and to learn compact models of their effects (Chapter 6 and 7).
- we have crowdsourced information from the World Wide Web using Games with a Purpose in order to generate models for spatial relations. We have also investigated spatial relations in the context of particular tasks, for example, setting a table for breakfast (Chapter 8).
- we have specified a robot description language that allows to describe sensors and actuators as well as the software components of a robot semantically. Additionally, we have implemented reasoning components that match a particular robot description with a formalized task description in order to assess the capabilities of a robot with respect to a given task (Chapter 3).

[13]`http://www.swi-prolog.org`

1.6 Thesis Outline

The thesis can be read in a linear fashion. However, the interested reader might skip chapters or go though them in a different order. To provide some guidance while reading this thesis, we briefly summarize the content of the chapters and comment on their dependencies to others.

CHAPTER 2 introduces the general concepts of the approach and thereby sets the scope and limits of this thesis. As this chapter defines the conceptual apparatus of this work it should be read by all readers to gain a better understanding of the author's mindset.

CHAPTER 3 describes the basic ideas and implementations of the underlying representations used within this work. In particular, the semantic robot description language is introduced as well as the first-order representations for manipulation episodes. The reader might browse through this chapter in the first place and only return to it whenever she needs more detailed explanations about the formal representation methods.

CHAPTER 4 explains the design principles and the realization of the envisioning framework for robot manipulation tasks. First, it describes how manipulation tasks are formalized using first-order representations, second, how these tasks are transformed into parameterized simulations, and third, how the logged simulations are transformed into time-interval-based first-order representation. Experimental results for various everyday tasks are presented and analyzed. Finally, the overall approach of simulation-based envisioning is thoroughly discussed.

CHAPTER 5 presents a detailed account on the representations and algorithms for simulating and reasoning about fluids within the envisioning framework described in Chapter 4.

CHAPTER 6 shows how the representations and methods introduced in previous chapters enable a knowledge-based and physics-aware analysis of virtual manipulation tasks performed by humans. Readers with a primary focus on robots only might skip this chapter. However, in Chapter 7 we will see how these techniques are used for teaching physical aspects of manipulation tasks to robots.

CHAPTER 7 describes a learning framework for teaching robots aspects of manipulation tasks in a structured way. For learning especially physical aspects of tasks robot learners employ the envisioning framework explained in Chapter 4 to deeply understand manipulation actions. Techniques presented in Chapter 6 are used to teach examples to robots. This chapter basically builds on concepts described in all previous chapters.

CHAPTER 8 explains how general and task-depended knowledge about spatial relationships of objects can be acquired through web-based Games with a Purpose.

CHAPTER 9 concludes this thesis by summarizing the major contributions and by giving an outlook into the future of physics-aware robot manipulation. The reader looking for a condensed description of this work's findings might want to look at this chapter.

1.7 Publication Note

Part of the research presented in this work appeared in publications of international workshops, conferences and journals. In particular, research on a semantic robot description language has been published in (Kunze et al., 2011c). Work on the envisioning of robot manipulation tasks in (Kunze et al., 2011b,a; Beetz et al., 2012; Klapfer et al., 2012). A system for the acquisition of commonsense task knowledge through Games with a Purpose is described in (Kunze et al., 2012). Work on commonsense knowledge for robots and semantic environment models is described in (Kunze et al., 2010; Tenorth et al., 2010a).

CHAPTER 2

Everyday Robot Manipulation

Today's robot systems are physically capable to perform complex everyday manipulation tasks like, for example, cleaning a room[1], folding clothes (Lakshmanan et al., 2012) and preparing a meal (Beetz et al., 2011). However, current robot control programs are tailored to particular tasks, specific environments, and the robot platforms themselves. That is, these programs are far from being very flexible and robust. Furthermore, most of these programs will fail when presented with slight variations of a problem they should solve. But if robots are to *master* everyday manipulation tasks instead of only *performing* them, they would need more general reasoning capabilities embedded within their control programs. The aim of this work is to equip robots with naive physics and commonsense reasoning that allows them to understand everyday phenomena of manipulation tasks. But before we present our account on the problem we lay out the general concepts and terminology we are using throughout this thesis.

This chapter defines the basic concepts of everyday robot manipulation, namely everyday manipulation *scenarios*, *tasks*, *environments*, *robots* and *plans*. We introduce the concept of *narratives* in order to represent and reason about episodes of manipulation tasks and experiences of a robot. Furthermore, we explain which of these concepts are involved within the different types of naive physics and commonsense reasoning. In particular, we address reasoning methods by which robots can assess their own manipulation capabilities, understand vaguely specified spatial concepts, and envision the physical effects of actions performed by humans and by themselves. Finally, we summarize and discuss the presented concepts and reasoning methods.

[1]`http://personalrobotics.stanford.edu`

2.1 Scenario

Robots performing everyday manipulation in the real world face many challenges (Kemp et al., 2007). These challenges include, for example, the dynamics of the world, varying object poses and/or object types, the need for specialized tools, and handling of nonrigid objects and substances. Although we address all of the above mentioned challenging characteristics in this work, we have to leave out many others. In order to be clear by what we mean by "everyday robot manipulation" we define an everyday robot manipulation scenario as follows:

An everyday robot manipulation ***scenario*** is defined by four components:

- a ***task***,
- an ***environment***,
- a ***robot***, and
- a ***plan***.

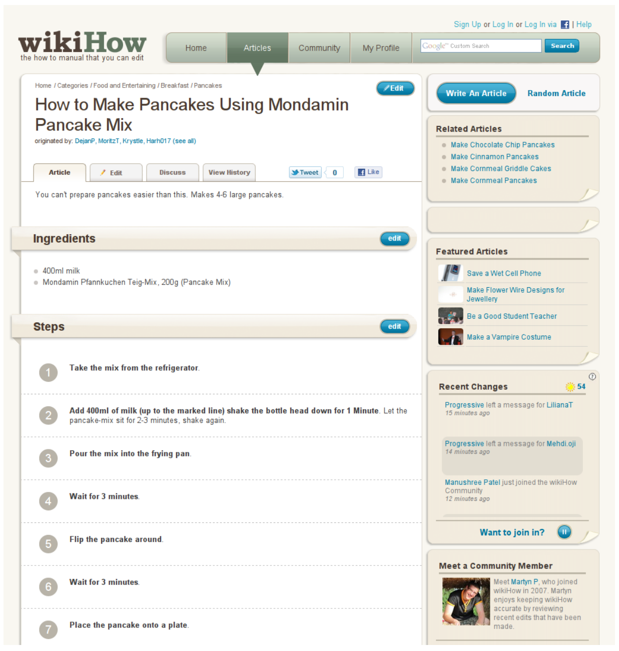
None of these concepts is new. On the contrary, they are all very common and have been used extensively throughout the AI and Robotics literature. The problem is that there are already quite a number of definitions for them and that everybody has a slightly different understanding of these concepts. In order to lay out a common terminology in the context of this thesis we specify what we mean by task, environment, robot, and plan in the subsequent sections.

2.1.1 Task

Everyday manipulation tasks are often specified by natural language instructions like, for example, "Clean the table". It is difficult to interpret such instructions because they are ambiguous and/or incomplete specifications of a task. The instruction "Clean the table" might be interpreted as:

> *"Take all items from the table, put them where they belong to, and finally, use a wet washcloth and wipe down the table."*,

whereby *"put them where they belong to"* could mean to put dirty dishes into the dishwasher, a box of cereals back into the cupboard and a half-eaten banana into the bin. However, the interpretation of instructions is always dependent on the context.



Home / Categories / Food and Entertaining / Breakfast / Pancakes

How to Make Pancakes Using Mondamin Pancake Mix

originated by: DejanP, MoritzT, Krystle, Harh017 (see all)

You can't prepare pancakes easier than this. Makes 4-6 large pancakes.

Ingredients

- 400ml milk
- Mondamin Pfannkuchen Teig-Mix, 200g (Pancake Mix)

Steps

1. **Take the mix from the refrigerator.**
2. **Add 400ml 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.**

Related Articles

- Make Chocolate Chip Pancakes
- Make Cinnamon Pancakes
- Make Cornmeal Griddle Cakes
- Make Cornmeal Pancakes

Featured Articles

- Save a Wet Cell Phone
- Make Flower Wire Designs for Jewellery
- Be a Good Student Teacher
- Make a Vampire Costume

Recent Changes 54

Progressive left a message for LilianaT
15 minutes ago

Progressive left a message for Mehdi.oji
14 minutes ago

Manushree Patel just joined the wikiHow Community
12 minutes ago

Meet a Community Member

Meet Martyn P, who joined wikiHow in 2007. Martyn enjoys keeping wikiHow accurate by reviewing recent edits that have been made.

How to Make Pancakes (Steps)

1. Take the mix from the refrigerator.
2. Add 400ml 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.

FIGURE 2.1 Left: Recipe for making pancakes (`wikihow.com`). Right: Step-by-step instructions for making pancakes according to the website on the left.

A more complex example of task instructions is shown in Figure 2.1. It shows a website of `wikihow.com` that explains step-by-step how to make pancakes using a particular type of pancake mix. Work by Tenorth et al. (2010b) has demonstrated how natural language instructions of such recipes can be extracted from the Web, parsed, analyzed, and transformed into formal task specifications. Given that the disambiguation and completion of naturalistic task instructions are by themselves difficult and challenging research problems they are beyond the scope of this thesis. Hence, this work assumes that a task is always specified completely using unambiguous and well-defined concepts:

A ***task*** is a complete and formal specification of a manipulation problem using well-defined concepts that can be interpreted by a ***robot***. The specification has a hierarchical structure and is composed of complex sub-tasks and/or primitive actions. Additionally, task-related problems and capabilities needed to perform the task (or the action) are specified in a common terminology.

Following the instructions (3-7) of Figure 2.1, the task of making pancakes could be formalized as shown graphically in Figure 2.2. The main task *MakePancake* comprises five steps of four different types, namely *Pour*, *Flip*, *Place*, and *Wait*. According to the challenge problem in Section 1.2, the *Flip* task is further split into three sub-tasks, namely *Push*, *Lift*, and *Turn*. Using well-defined concepts of an ontology the task specification can be unam-

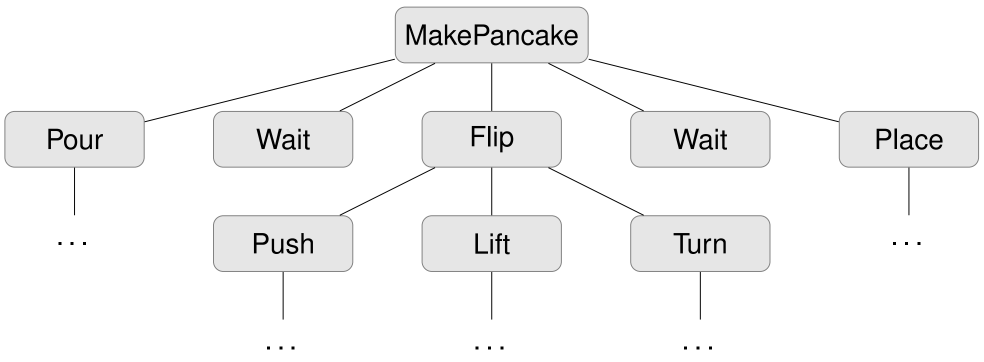


FIGURE 2.2 Conceptual representation of the task of making pancakes.

biguously interpreted by a robot. Furthermore, the hierarchical composition of tasks using complex sub-tasks and primitive actions allows designers of tasks to heavily reuse previously defined concepts (tasks). Hence, given a set of very basic and elementary tasks the overall task library can be kept quite compact. The creation of a comprehensive and reusable task library for robots that is based on a common terminology makes performances and reasoning tasks about everyday manipulation comparable across different robots. Furthermore, the capabilities that are needed to perform a task are associated with the individual task and action types. For example, the *Lift* task requires a robot to exert a certain force when picking up an object.

Please note, that Chapter 6 deviates from the above definition of a task. In Chapter 6 we use the colloquial meaning of the word *task* since humans are the main actors within the everyday manipulation scenarios instead of robots.

2.1.2 Environment

We envision robots to perform everyday manipulation tasks in real world human environments such as households, homes for the elderly, and factories. In the following, we characterize the environment of an everyday robot manipulation scenario according to the criteria listed in (Russell and Norvig, 2009):

FULLY/PARTIALLY OBSERVABLE The environment of an everyday robot manipulation scenario is only *partially observable* by the robot given its restricted field of view and its noisy and inaccurate sensors.

DETERMINISTIC/STOCHASTIC Everyday robot manipulation is clearly *stochastic* as the robot cannot foresee the exact outcome of its manipulation actions, for example, when pouring the pancake mix onto the pancake maker or cracking an egg.

EPISODIC/SEQUENTIAL The environment is *sequential*. That is, manipulation actions of the robot can have long term effects. For example, if a robot pours too much pancake mix onto the pancake maker, the robot might not be able to flip the pancake afterwards.

STATIC/DYNAMIC The environment is *dynamic*. That is, the environment will change although the robot is not manipulating it. For example, if the robot decides to wait too long until it flips the pancake, the pancake will burn.

DISCRETE/CONTINUOUS The state of the environment is *continuous*. For example, the pose of the robot's hand is characterized through a range of continuous values in space and time.

SINGLE/MULTI-AGENT Everyday robot manipulation is a *multi-agent* environment. If robots perform manipulation tasks in households, homes for the elderly and factories they have to cooperate with robotic co-workers and/or humans.

In summary, this leaves us with an environment that is *partially observable*, *stochastic*, *sequential*, *dynamic*, *continuous* and *multi-agent*. Basically, the most difficult environment that one could characterize. However, in the context of this work we make assumptions that simplify some of the above criteria.

Our first assumption, or simplification, is about the presence of robotic and/or human co-workers. Challenging problems for everyday manipulation in *multi-agent* environments include the distribution of tasks and the interaction between agents. Both settings multi-robot and human-robot span their own research fields which are not addressed in this work. Therefore, we consider everyday manipulation tasks only in scenarios where a single agent, robot or human, is present.

A second assumption is about the change of the environment. Since we assume a single agent environment, the change through other agents is already out of question. However, physical processes like, for example, cooking might still change the environment without the interference of the robot. But since the number of such physical processes that could change the world is restricted within our manipulation problems we consider the environment as *semi-dynamic*.

The third and last assumption is about the observability of the environment. Here we assume that the robot can fully observe all the details of its environment. We achieve this by using a physical robot simulator. The use of a physical simulation might be controversial. However, we belief that the advantages of using physical simulations for reasoning about manipulation problems outweigh their disadvantages as is elaborated in the following.

Physical Simulations In this work we employ physical simulations as a means to acquire knowledge about everyday manipulation. With regards to the simulation we use the Gazebo[2] simulator which will be introduced in Section 4.3. Obviously, there is a risk that drawn inferences that are based on simulations do not reflect the circumstances in the real world. To minimize this risk, all design decisions related to the simulation have to be made very carefully. For example, in the next section we explain how the controllers of a simulated and a real robot are related. On the other hand, the use of simulations gives us the opportunity to focus on those aspects that are most relevant to naive physics and commonsense reasoning about everyday manipulation. For example, we can fully ignore the problem of robot navigation.

Other advantages of physical simulations include their repeatability, variability, extendability, and their inexpensiveness. First, given that simulations can be repeated over and over again make them very attractive for systematic testing and benchmarking. Methods from statistics can effectively be applied to evaluate the results from hundreds of simulations. Second, it is important to show the feasibility and generality of an approach by applying the developed methods to various problems. Simulations can cover a wide range of manipulation problems and their respective variants by altering their initial configurations and/or adding noise to certain processes. Third, physical models of objects and robots can relatively easy be substituted and/or enhanced by improved versions. Finally, all of the above can be achieved in a simulation at very low costs compared to the efforts and expenses that would be necessary in the real world. A thorough discussions about physical simulations and pointers to related work are given in Section 4.6 and 4.7.

Figure 2.3 shows an image of a kitchen environment in the Gazebo simulator. The scene consists of static objects like cupboards, a refrigerator, and a kitchen counter and dynamic objects like a container of pancake mix and a spatula. Whereas the static objects function as supporting planes, the dynamic objects can be manipulated by the robot.

In general, we distinguish between three different types of objects within the simulated environment, namely rigid, deformable, and fluid objects:

RIGID OBJECTS or solid objects are composed of links and joints whereby joints between links are always fixed. That is, we do not consider articulated objects composed of rigid parts like, for example, scissors. Examples of rigid objects include a spatula and a container. Most rigid objects are decomposed of sub-parts. For example, a spatula is composed of a handle, a blade, and an elongated part which connects both former parts.

[2]`http://gazebosim.org`

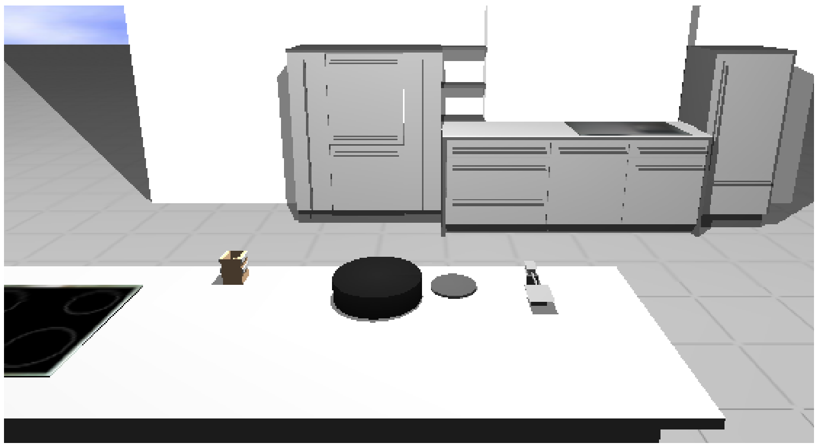


FIGURE 2.3 Simulated kitchen environment.

DEFORMABLE OBJECTS are composed of links and joints whereby joints allow for deformation between the links. That is, the overall structure of an object can change. An example of a deformable object is a pancake. A special case are fragile objects because a joint can also cease to exist whereby the object breaks apart. An example for the latter type of object is an eggshell.

FLUID OBJECTS are either liquids or granular fluids. Fluids are basically particles (links) without connecting joints. Therefore, particles can move freely within the environment. A example is the pancake mix. When in contact with a heat source, particles of the pancake mix can form a pancake, and thereby become a deformable object.

All of the above characteristics and properties yield to the following definition of an environment:

A simulated ***environment*** of a manipulation scenario is *fully observable*, *stochastic*, *sequential*, *semi-dynamic*, *continuous* and *single-agent*. It comprises objects with different properties:

- *static objects* are *rigid* and serve as supporting planes in the environment (e.g., furniture)
- *dynamic objects* are either *rigid/solid* (spatula), *deformable* (pancake), or *fluid* (pancake mix) and can be manipulated by a ***robot***

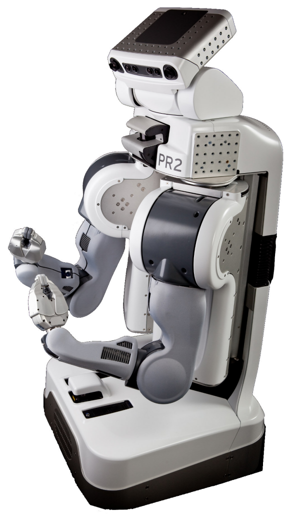
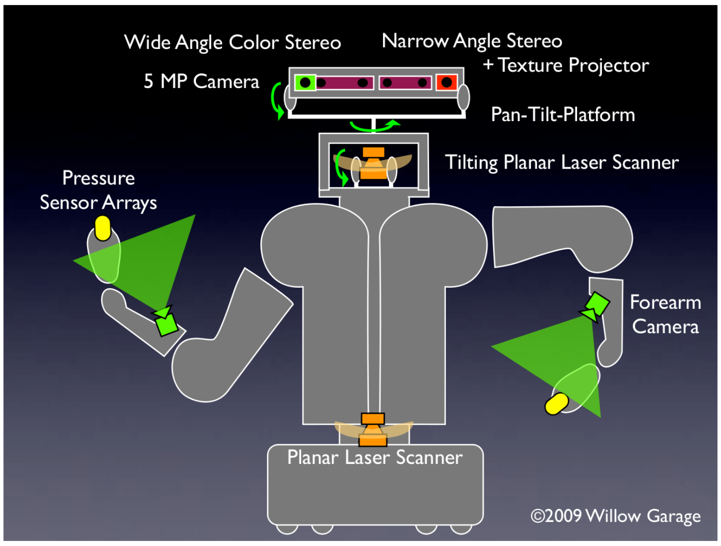



FIGURE 2.4 Left: Personal Robot 2 (PR2). Right: Sensors of PR2 in a schematic illustration. Courtesy: Willow Garage.

2.1.3 Robot

Robots performing everyday manipulation tasks need to fulfill certain hardware and software requirements. Robots need appropriate sensors and actuators to perceive and manipulate objects within their environment. In addition to their hardware components, they need software components that operate and control them.

The robot model mainly used in our work is the PR2 robot platform developed by Willow Garage[3]. The PR2 has an omnidirectional base, a telescoping spine and a pan-tilt head. Each of the two compliant arms of the platform have four degrees of freedom (DOF) with an additional three DOF in the wrist and one DOF gripper. The sensor setup is comprised of a laser sensor on the base, a tilting laser sensor for acquiring 3D point clouds, two stereo camera setups and a high resolution camera in the head. The hands also have cameras in the forearms, while the grippers have three-axis accelerometers and fingertip pressure sensor arrays.

Major challenges in everyday robot manipulation are the perception and manipulation of objects. Robots have to perceive, recognize and localize objects in an environment which is only partially observable. Perception of everyday objects is already a complex and complicated task and therefore it is beyond the scope of this work. The same is true for dexterous manipulation of arbitrary objects. Since this work focuses on reasoning about everyday manipulation we suppose that robust perception and action routines are avail-

[3] `http://www.willowgarage.com/pages/pr2/overview`

able for a fixed set of everyday objects within respective libraries. Throughout this work we make use of off-the-shelf software components whenever available as ROS library. Whenever necessary we extend these libraries, for example, we implemented a faked perception. That is, whenever an object is in the field of view of the robot, the object and properties like its extensions, pose, and bounding box can be perceived. Furthermore, we assume that the robot knows about the static parts of its environment given a semantic environment map.

As mentioned in the previous section, it is critical that results obtained through simulations resemble those of the real world. Therefore we use standard software components that can be applied to control both the real and the simulated PR2 robot. This is possible, because the simulated robot is realistically modeled according to the mechanical specification of the real robot. Table 2.1 and Table 2.2 show excerpts of the mechanical specification of the PR2 Manual[4]. For example, the gripper of the simulated robot has the same joint limits and can apply the same force as the real robot. We will use such information in order to reason whether robot are capable to perform a task. Another important aspect in the kinematic structure of the robot. Figure 2.5 shows the arm kinematics of the PR2. Given that information, we can use the algorithms from the ROS arm navigation package to control the arm of the simulated robot.

TABLE 2.1 Forces and torques.

Joint	Velocity (rad/s or m/s)	Torque (Nm or N)
∗_wrist_flex_joint	3.10	10.00
∗_wrist_roll_joint	3.60	10.00
∗_gripper_joint	0.20	1000

TABLE 2.2 Joint limits and types.

Joint	Type	Limit (+)	Limit (-)
∗_wrist_flex_joint	revolute	130°	0°
∗_wrist_roll_joint	continuous	-	-
∗_gripper_joint	prismatic	86 mm	0 mm

[4] `http://pr2support.willowgarage.com/wiki/PR2%20Manual/Chapter8#Mechanical_Specification`

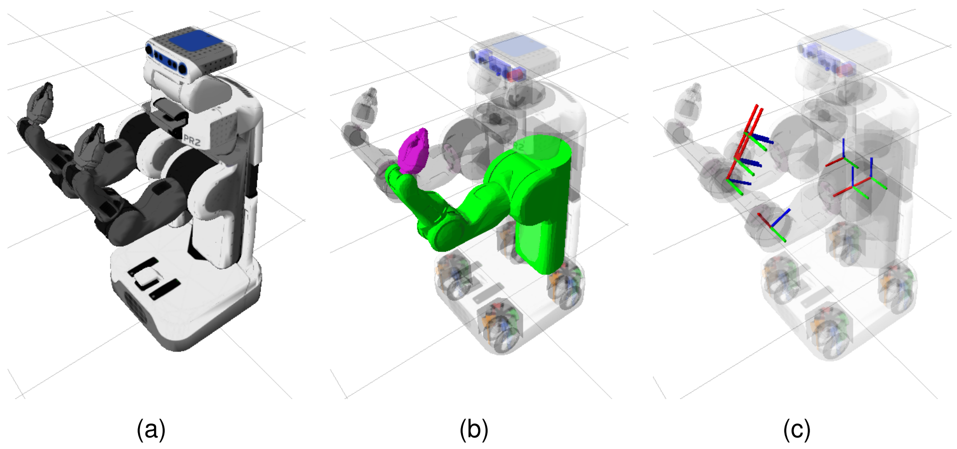


FIGURE 2.5 Kinematics of PR2's left arm. Left: PR2 in its initial pose. Middle: Hardware parts of its arm are highlighted. Right: Coordinate frames of all joints involved in the arm kinematics are visualized.

Considering the above aspects we define the concept of a robot as follows:

A ***robot*** is a physical agent that has sensors and actuators. It perceives and manipulates objects in its ***environment*** by performing actions of a ***plan***. A robot is characterized by:

- a kinematic model,
- a semantic description of its components,
- an action and perception library.

Please note, that Chapter 6 slightly deviates from the above definition of a robot. In Chapter 6 we use simply a robotic hand that is teleoperated by a human.

2.1.4 Plan

A robot plan is basically a procedural description of how to solve a task. It is composed of complex and primitive actions. A robot can execute a plan if its action and perception libraries contain all actions of the plan. A plan usually resembles the structure of the task. That is, in analogy to tasks, plans are organized in a hierarchical fashion. However, to what extend they match a specification of a task depend on the granularity of the robot's action library. For example, a detailed task specification can describe how flip a pancake in several steps, namely push the spatula under the pancake, lift the spatula, and turn the spatula. On the other hand, an action library might only have a flipping action as its most primitive action. That is, the processes of pushing, lifting, and turning are not represented explicitly within the plan.

A plan can be represented using a plan language like, for example, CRAM (Beetz et al., 2010). The CRAM plan language is a reactive plan language and is based on Common Lisp. Control structures in CRAM allow programmers to embed lightweight reasoning mechanisms that infer control decisions rather than requiring the decisions to be preprogrammed. These cognition-enabled programs are more flexible, reliable, and general than robot control programs that lack these capabilities. The following CRAM pseudocode shows an example of a consequence-based action specification of a push action.

```
(perform (an action
            (type push)
            (object (an object-part
                        (part-of spatula)
                        (type blade)))
            (destination ?loc = (a location
                                   (under pancake)))
            (desired-effect (and (pose spatula ?loc)
                                 (succeeds (an action
                                              (type lift)
                                              (object pancake)
                                              (starting-pose ?loc)))))
            (undesired-effects (and (damaged pancake)
                                    (on pancake counter)))))
```

The parameters of the push action are determined by grounding the symbolic expressions into the control and perception routines of the robot. For example, the reasoning components determine the set of parameters that allows to, first, push the spatula under the pancake, and second, lift it afterwards.

Plan languages like CRAM allow robots to reason about their own programs explicitly and even modify them at execution time. However, in the context of this thesis we only assume that robots know about the structure of their plan/program. That is, during execution robots know when an action starts and when it finishes. Hence, plans/programs in this work can be implemented in any kind of programming language as long as they report the internal state of their actions.

Following the above considerations we define a plan as follows:

A ***plan*** (or robot control program) is a procedure (or sequence of actions) that can be executed by a ***robot*** within a particular ***environment*** to accomplish a ***task***. It is implemented by a plan language or any other programming language and reports the state and progress of the executed actions.

2.2 Narratives

In the previous section we formalized an everyday robot manipulation scenario as consisting of four components, namely a task, an environment, a robot, and a plan. However, everyday manipulation and reasoning about it is most fascinating and challenging when a robot actually executes a plan within a particular environment in order to accomplish a task. To capture the dynamics of the world during the execution of the robot plan we introduce the concept of a narrative:

A ***narrative*** is a record of states and events that results from the actions performed by a ***robot*** that executes a fully instantiated ***plan*** (or which is teleoperated) in a particular ***environment*** in order to accomplish a ***task***.

In this work we use narratives to describe "manipulation stories" performed by both robots and humans, where the latter control a teleoperated robot hand. Thereby, narratives are an excellent way to capture experiences of the robot's own performances (Chapter 4) and examples from observations of human object manipulations (Chapter 6). Hence, the detailed study of a manipulation problem like "Making Pancakes" and its variants result in a comprehensive set of such narratives (Figure 2.6).

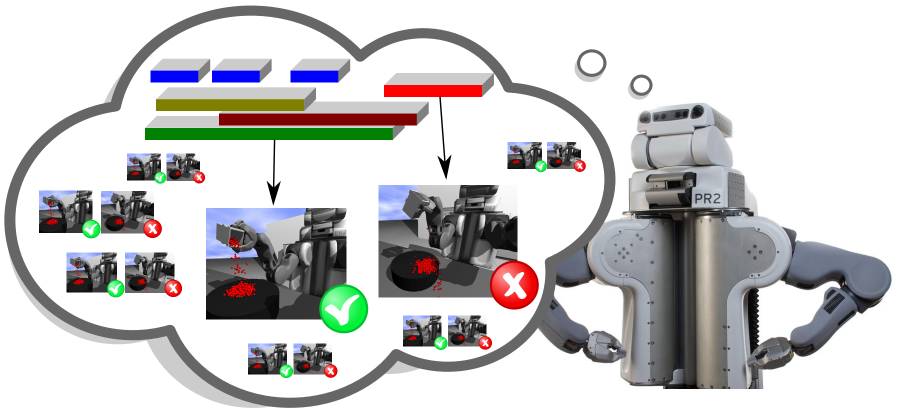


FIGURE 2.6 PR2 reasons about a set of narratives.

2.3 Naive Physics and Commonsense Reasoning

We have seen that everyday robot manipulation poses many challenges. A robot that is to competently reason about such problems has first to understand how a specification of a task is related to its own sensors, actuators and software components. Using routines of its perception and action libraries the robot can follow the individual task steps and perform the manipulation task. In order to interpret the outcome of the task, the robot need reasoning capabilities that capture the dynamic aspects of the environment. Therefore, the robot needs conceptual knowledge about space and time. It also has to understand physical phenomena. Hence, the robot's main reasoning tasks include the reasoning about its own manipulation capabilities, about spatial and physical configurations of objects in the environment, and the effects of its actions.

2.3.1 Capability Reasoning

There is a gap between simple but high-level task instructions like "Make a pancake" and low-level robot descriptions that model, for example, the structure and kinematics of a robot's manipulator. Currently, programmers bridge this gap by mapping abstract instructions to parametrized algorithms and rigid body parts of a robot within their control programs. By linking descriptions of robot components, i.e., sensors, actuators and control programs, via capabilities to actions in an ontology we equip robots with knowledge about themselves that allows them to infer the required components for performing a given action. Thereby a robot that is instructed by an end-user, a programmer, or even another robot to perform a certain action, can assess itself whether it is able to perform the requested action. For example, it can answer queries like:

> *"Can you make a pancake? And if not, why not? What capabilities are missing?"*

This self-knowledge for robots could considerably change the way of robot control, robot interaction, robot programming, and multi-robot communication. In Chapter 3 we present the underlying representations of tasks and robots and explain how they can be matched in a similar way as semantic web services (Martin et al., 2007). Thereby, we define capability reasoning as follows:

Capability reasoning involves the reasoning about a ***task*** and a ***robot***. A task requires certain capabilities of its *doer*. These capabilities are specified by a designer of a task. On the other hand, a robot is composed of a set of different components: sensors, actuators and software modules. These components provide certain capabilities. Inference algorithms basically match the specifications tasks and robots via the concept of capabilities in order to asses the general ability of robots to perform a task. This kind of reasoning is not about concrete performances and/or execution traces of robots.

2.3.2 Spatial Reasoning

An important factor to understand everyday robot manipulation is the ability to reason about spatial relations between objects and the robot. That is, robots have to know about topological relations like *in* and *on*, directional relations like *left of* and *in front of*, and distance relations like *close* and *distant*. The meaning of these concepts is often very vague and/or context-dependent what makes it even more difficult for a robot to interpret a scene and answer a question like:

"Is the pancake close to the edge of the pancake maker?"

However, humans have a sound and intuitive understanding of such concepts given their wealth of experience. For example, they know that the instruction *"Place the knife right to the plate"* means to put the knife to right side of the plate at a certain distance and orientation. Such implicit and context-dependent knowledge cannot be extracted directly from task instructions but rather has to be acquired by other means. After we present the basic formalisms for spatial concepts in Chapter 3, we explain how knowledge about spatial relations can be bootstrapped from the WWW using Games with a Purpose in Chapter 8. In the context of this thesis we define spatial reasoning as follows:

Spatial Reasoning is the ability to make inferences about geometric configurations between different objects of the ***environment*** and the ***robot***. It includes the reasoning about topological, directional and distance relations and sometimes depend on the context of a ***task***.

2.3.3 Reasoning about Narratives

Narratives, as introduced in Section 2.2, capture the dynamics of the world. In contrast to reasoning about spatial relations described in the previous section, reasoning about narratives is reasoning over time. That is, most often inferences are based on a sequence of situations. For example, the query

“Does the pancake eventually lie upside down on the pancake maker?”

can only be answered by looking at the orientation of the pancake in at least two (subsequent) situations. As spatial reasoning is used to evaluate the situations individually it is an essential building block of reasoning about narratives.

In general, reasoning about narratives can be considered as spatial, temporal, and physical reasoning as well as reasoning about actions. Basically, it considers all aspects of everyday robot manipulation captured in a narrative. As a single narrative represents a “manipulation story” of a robot or a human queries can extract information about individual experiences. However, a knowledge base of thousands of narratives can provide finally answers to questions listed in Section 1.3.1 by using methods from statistics and probability theory. For example, a question like *“What is the expected outcome of an action?”* is answered based on a set of narratives stored in a knowledge base. Having seen quite a number of examples of that action, whereby only a few had an undesired outcome, it can be inferred that the expected outcome will have a desired effect with a high probability.

Reasoning about Narratives makes qualitative inferences about spatial and physical configurations of objects of the ***environment*** and the ***robot*** over time. It also considers the actions of a robot ***plan***, relates them to other events and interprets their outcome as desired and/or undesired effects of a ***task***.

Envisioning Envisioning is a special form of reasoning about narratives. Basically, a robot hallucinates what would happen if it executes a fully instantiated plan in an environment. For example, it can answer a question such as:

“What happens if the pancake is flipped too late?”

In this work, the process of hallucination is based on physical robot simulations. The result of a hallucination is a narrative. By repeating this process for different parameterizations of an action and/or different variants a problem the robot can actively explore the physical effects of its manipulation actions and thereby get an understanding of the problem.

Envisioning is a “mental” process of projecting a state of the ***environment*** into all possible future states by “executing” fully instantiated ***plans*** with a ***robot***. The result of this process is a set of ***narratives***.

Reasoning about narratives and envision is explained in Chapter 4, 5, 6 and 7.

2.4 Summary

In this chapter we have laid out the basic terminology as it is used and understood in this thesis. We have defined an everyday robot manipulation ***scenario*** by four components: a ***task***, an ***environment***, a ***robot***, and a ***plan***. Furthermore, we have presented different types of reasoning relevant for everyday manipulation, namely ***capability reasoning***, ***spatial reasoning***, and ***reasoning about narratives*** including the special case of ***envisioning***. For the different types of reasoning we have explained how these are realized based on the previously defined concepts of a ***scenario***. The following chapter (Chapter 3) introduces the underlying representations of the most important concepts and describes our approach on capability reasoning. The reasoning methods about spatial concepts and narratives are presented in the subsequent chapters (Chapter 4, 5, 6, 7 and 8).

CHAPTER 3

Representations

In this chapter we present the formal representations for naive physics and commonsense reasoning for everyday robot manipulation. In particular, we describe the underlying representations needed for capability reasoning, spatial reasoning and reasoning about narratives. First, we revisit the example of “Making Pancakes” and examine the different entities involved in reasoning about everyday manipulation. Afterwards, we discuss general aspects regarding logical representations. And finally, we describe the representations that have been developed in greater depth, relate them to the concepts introduced in Chapter 2, and conclude with a discussion in Section 3.7.

3.1 “Making Pancakes” Revisited

Let us reconsider the example problem of “Making Pancakes” from Section 1.2. To understand what representations are needed to reason about the task we first examine the task-related objects, second, the actions a robot has to carry out to perform the task, and finally, the physical effects that could result from the actions.

The task-related objects mentioned in the natural language description are a pancake mix, a pancake maker, a pancake, a spatula and a plate. Not mentioned in the description are additional tools, for example, a container holding the pancake mix. Inferring these missing objects is straight forward for humans given their common sense. However, robots have to figure out these objects by other means and then locate and retrieve them from the environment. Overall, the task scenario comprises objects with different physical properties, namely rigid, liquid, and deformable objects. Given that the task involves a spectrum of object types it covers a wide range of manipulation problems. Furthermore, objects are composed of multiple parts. For example, the spatula has a handle, a blade and an elongated part connecting the former two. Hence, different representations are needed to handle the various types of objects and their respective parts effectively.

A robot performing a task generally follows a plan, i.e., it executes a robot control program. A plan is basically composed of a sequence of actions. In the case of the "Making Pancakes" problem the main actions of the task are pouring the pancake mix and flipping the pancake. Both actions have a set of parameters that determine the way how an action is performed. For example, the parameters of the pouring action include the position of the container over the pancake maker, its tilting angle, and the duration how long the container is tilted. That is, representations of actions should cover sequences of actions, primitive actions and their respective parameters.

Eventually, the robot should reason about the effects of its actions. That is, the robot reasons about spatial and physical configurations of objects in the environment and how these configurations evolve over time when objects are manipulated by the robot. Thus, representations for spatial and temporal configurations of objects are needed. Moreover, the formalization should allow to represent and distinguish between multiple performances of a robot. By storing a large collection of these "manipulation stories" in a knowledge base a robot can give answers using methods from statistics and probability theory.

Table 3.1 lists some of the relevant variables, their sorts and meanings that need to be represented within this work. We mainly distinguish between objects, actions, space and time. Whereas most concepts are very general, some of the object and action variables are mainly related to the "Making Pancakes" problem.

3.2 Logical Representation

In this thesis we chose logic for representing the robot's knowledge about everyday manipulation. However, there is not a single thing called "logic". Rather there are several different types of logic. For example, it can be distinguished between propositional, first-order, temporal, probabilistic and fuzzy logic. These types of logic can represent different aspects of the world and a robot can belief these aspects in different ways. That is, committing to a certain logic determines what a robot can know about the world and what its beliefs might be. There is always a trade off between the expressiveness of a certain logic versus its efficiency for making inferences. For example, propositional logic allows a knowledge engineer to represent facts only. However, there exist efficient inference algorithms that can solve large problems quickly. On the contrary, first-order logic allows a knowledge engineer to model the world using facts, objects and relationships. But generally, reasoning about large problems in first-order logic is slow. However, there exist logic programming

TABLE 3.1 Variables, their Sorts and Meanings.

Sort	Variable	Meaning
Object	*o*	objects, object parts
	fo, lo	fluid and liquid objects
	do	deformable objects
	mix	pancake mix (fluid)
	pancake	pancakes (deformable)
	spatula	spatulas
	pan	pans (pancake makers)
	container	containers
	table	tables
Action	*a*	actions
	pour	pouring actions
	flip	flipping actions
	par	parameters of actions
	plan	plans (or robot control programs), a sequence of actions
Space	*pos*	positions in 3D space
	x,y,z	coordinates in 3D space
	orient	orientations in 3D space
	ro, pi, ya	roll, pitch, yaw (angles in 3D space)
	d,w,h	depth, width, height (dimensions in 3D space)
	bbox	bounding boxes
Time	*e*	events
	f	fluents
	t	time points
	[t1,t2], i	time intervals
	tl	timelines

systems like PROLOG[1] that are very fast for particular types of inferences. Temporal logics allow a robot to represent aspects about the world over time, probabilistic logics allow it to believe facts only to a certain degree, and fuzzy logics to represent facts with a degree of truth. In order to represent everyday manipulation scenarios, that is objects, actions and their relations over time, we chose a temporal, many-sorted first-order logic. Whenever necessary we used probabilities to express the degree of belief of the robot.

The knowledge of the robot is basically represented by logical *sentences* in a knowledge base. The sentences formalized in first-order language contain different symbols, namely constants to represent objects, predicates to represent relations and functions to represent

[1]`http://www.swi-prolog.org`

functions. These basic elements must obey the syntax of the particular logic. Furthermore, sentences should have a semantics, that is, the symbols should refer to entities in the world. Later we will see how the symbols are grounded in the data structures of physical simulations. The interface to a knowledge base comprises mainly two methods: One for asserting new sentences, and a second for querying the knowledge base. In the literature these two operations are often named *tell* and *ask* (Russell and Norvig, 2009).

In this work we use the Web Ontology Language (OWL)[2] and the logical programming language PROLOG to represent sentences about everyday manipulation scenarios in a knowledge base. The interactive PROLOG interpreter also functions as an interface to assert knowledge and query the knowledge base.

Web Ontology Language (OWL) There are three sub-languages of the Web Ontology Language: OWL-Lite, OWL-DL, and OWL-Full. They differ with respect to their expressiveness and computational completeness. In our work we use OWL-DL as it is a good compromise between expressiveness and computability. The OWL-DL language basically corresponds to Description Logics. Description Logics have two appealing features: Firstly, information about objects can be structured hierarchically in a sub-concept/super-concept relationship whereby sub-concepts inherit all constraints of their super-concepts. Secondly, it is possible to reason over the structured information. In Description Logic it is distinguished between a terminological and an assertional box, also called TBox and ABox respectively. Whereas the TBox describes relations between concepts, the ABox describes relations between individuals and concepts. Individuals are concrete instances of general types that are described by concepts. Overall, the language elements of OWL-DL are ideally suited to describe individuals of objects, their types, and relationships among each other. For example, different types of objects can be modeled as sub-concept of a general *object* concept as we will see in Section 3.3.

PROLOG The logical programming language PROLOG is used as an interface to the robot's knowledge base. We defined a logical language to query the knowledge base and retrieve answers by utilizing PROLOG's reasoning engine based on backtracking. Additional libraries like the Semantic Web Library[3] are used for accessing knowledge described in OWL. Besides the function of an interface, PROLOG is used as logical language for modeling spatial and temporal relationships that would be difficult to represent in OWL.

[2]`http://www.w3.org/TR/owl-features/`
[3]`http://www.swi-prolog.org/pldoc/package/semweb.html`

3.3 Objects (Environment)

As laid out in the previous section, we use OWL to represent concepts in an ontology. All physical objects are derived from a generic *Object* concept. The three major subclasses are *Solid*, *Fluid*, and *Deformable*. *Fluid* has further specializations, namely *Liquid* and *GranularFluid*. Given the principle of inheritance, properties of a concept are derived from their super-concepts. For example, *Object* has a property named *HasModel* that relates *Object* to *PhysicalModel*. Thereby all sub-concepts of *Object* as well as their respective sub-concepts have this property. As we have defined ***environment*** in Section 2.1.2 as a simulated environment it makes sense to relate all objects to physical models. These models can basically be described in all formats that can be loaded into the physical simulator Gazebo[4]. In this work we mainly make use of physical object models described in the Unified Robot Description Format (URDF)[5]. But other formats such as COLLADA[6] are also feasible. Eventually, a concrete instance of an object is linked to physical model that can be instantiated within the Gazebo simulator.

Figure 3.1 shows an excerpt of the ontology. At the top, it visualizes the relation between *Object* and *PhysicalModel*, on the left, the hierarchical object taxonomy including the concepts *Liquid*, *Solid*, *PancakeMix* and *Container*, on the right, the sub-concepts of *PhysicalModel*, and at the bottom, it shows concrete instances and their relations among each other. The *in* relation between *PancakeMix* and *Container* is explained in Section 3.4.

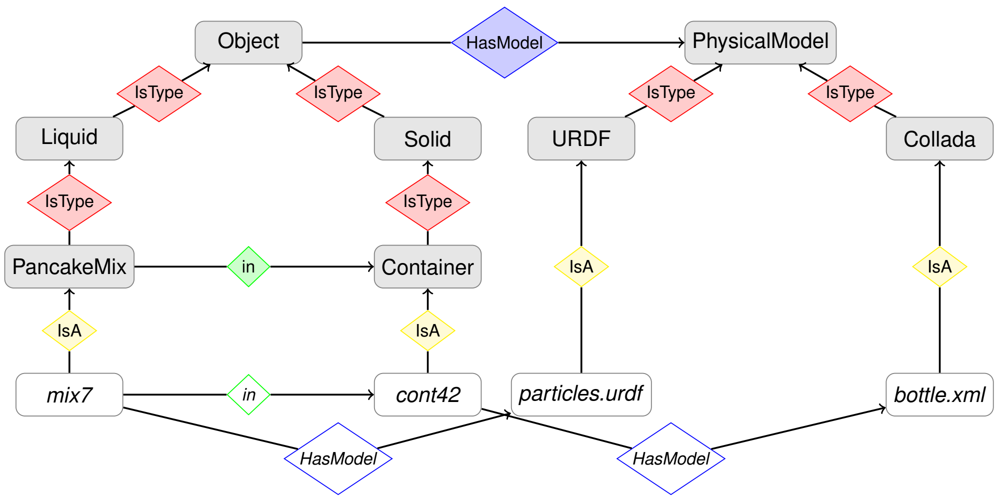


FIGURE 3.1 Simplified Ontology about "Making Pancakes".

[4]http://gazebosim.org
[5]http://www.ros.org/wiki/urdf
[6]http://www.collada.org

Another important aspect for the representation of objects is their physical structure. Let us consider, for example, a spatula. It is a solid object and it is composed of two parts, namely a handle and a blade. Using the predicate *HasPart* we relate an object to its respective parts. The following logical sentences show some knowledge about an instance of a *Spatula*:

$$\begin{aligned}&\mathit{IsA}(\mathit{spatula}, \mathit{Spatula}) \wedge \\ &\quad \mathit{HasPart}(\mathit{spatula}, \mathit{handle}) \wedge \\ &\quad \mathit{HasPart}(\mathit{spatula}, \mathit{blade}) \wedge \\ &\quad \mathit{HasModel}(\mathit{spatula}, \mathit{spatula.urdf})\end{aligned}$$

As we have seen above, an individual of a spatula is linked to a concrete physical model. This model is described in a language like URDF. In such a specification, parts of the spatula are described by their extension, mass, position, and orientation. The individual parts are connected by different types of joints. However, in order to reason about the individual parts of the spatula like, for example, the blade, it is necessary that the logical instance of a blade is also linked to its respective part within the physical model. We achieve this by a naming convention.

3.4 Space and Time

Reasoning about everyday object manipulation requires robots to understand the spatial and physical configurations of objects and their parts over time. A very good and general overview on spatial, physical and temporal reasoning is given in the book by Davis (1990). Robots should be able to extract information about an object's position, its contacts, and its spatial relations to other objects from its environment in order to reason about a task. Since we employ physical simulations all these information can be abstracted from the data structures of the simulator. Conceptually, the robot can access this information using the predicate *SimulatorValue* as follows:

$$\mathit{SimulatorValue}(\overbrace{\mathit{position}(o, pos)}^{\mathit{Function}}, \overbrace{\;t\;}^{\mathit{Time\ point}})$$

where *position* is an exemplary function for retrieving information about an object *o* at a certain point in time *t*. Eventually, the information about the object's position is bound to the variable *pos*. Table 3.2 lists further functions that can be used to access information about an object's world state. All functions provide information for a given object, e.g. its position, orientation, velocity, dimension, and its bounding box. As we will see later, many spatial relations are computed based on the object's bounding box. Thereby, the *bbox* function plays an important role for reasoning about the object's state.

TABLE 3.2 Functions for accessing the object's world state.

Function	Description
position(o, pos)	3D position of object o, pos is a vector $\langle x, y, z\rangle$
orientation(o, $quat$)	3D orientation of object o, $quat$ is a quaternion $\langle q_1, q_2, q_3, q_4\rangle$
linear_velocity(o, lv)	linear velocity of object o, lv is a vector $\langle lv_x, lv_y, lv_z\rangle$
angular_velocity(o, av)	angular velocity of object o, av is a vector $\langle av_x, av_y, av_z\rangle$
dim(o, dim)	dimensions of object o, dim is a vector $\langle d_x, w_y, h_z\rangle$
bbox(o, $bbox$)	bounding box of object o, $bbox$ is a vector $\langle x_{min}, y_{min}, z_{min}, x_{max}, y_{max}, z_{max}\rangle$

Another set of functions that can be accessed via the *SimulatorValue* predicate provides information about an object's contacts. Contact information are crucial for the interpretation and analysis of the physical effects of actions. As information about contacts are always reported between two objects all functions take also two objects as arguments. For example, the *contacts* function is true when there is a contact between two objects at a certain point in time. It is a symmetric function, that is, whenever object o_1 is in contact with o_2, object o_2 is in contact with o_1. However, please note that not all functions about contacts are symmetric. For example, the *force* function provides information about the force one object exerts onto another. Thus, the reported information about the force has a direction. Table 3.3 shows functions that extract information about contacts between objects including contact positions, normals, penetration depths and forces. Figure 3.2 depicts physical configurations of blocks within a simulated environment, and visualizes the internal belief state of the robot including contact information in the 3D visualization tool RVIZ[7].

Fluents Based on the low-level information about an object's world state and its contacts we define fluents, i.e. conditions over time, about an object's spatial relationships to other objects. Here we distinguish between three different types of spatial relations, namely topological, directional, and distance relations. Whereas topological relations are independent from the point of view, directional relations always depend on the current position and view of the robot. In our work we denote this position as *origin*. Figure 3.4 gives an overview of the implemented fluents.

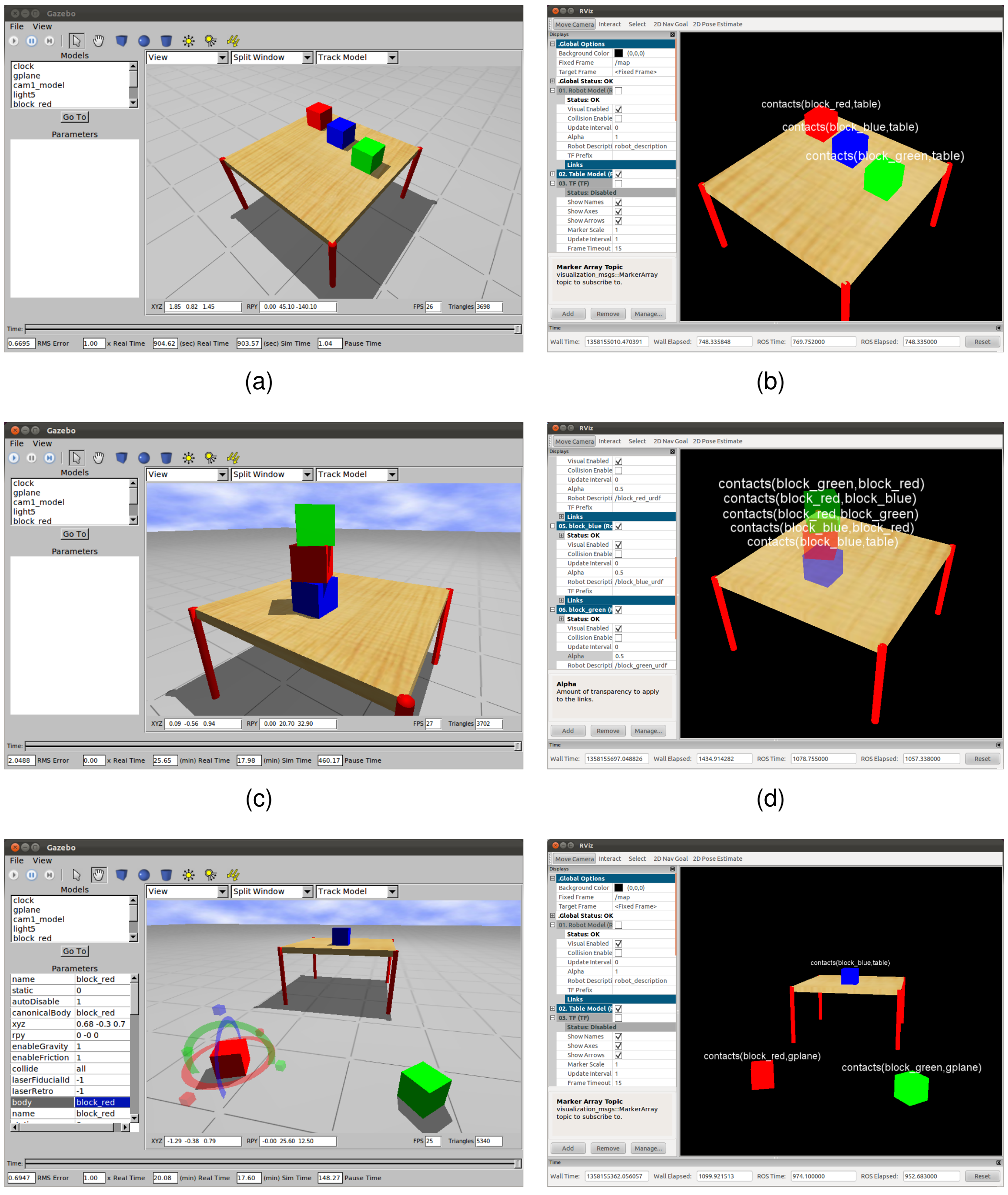
[7]`http://www.ros.org/wiki/rviz`

TABLE 3.3 Functions for accessing contact information between objects.

Function	Description
contacts(o_1, o_2)	true if object o_1 is in contact with object o_2
positions(o_1, o_2, $positions$)	list of contact positions between o_1 and o_2
normals(o_1, o_2, $normals$)	list of contact normals at contact positions
depths(o_1, o_2, $depths$)	list of penetration depths at contact positions
force(o_1, o_2, $forces$)	forces between o_1 and o_2, $forces$ is a vector $\langle f_x, f_y, f_z\rangle$
torque(o_1, o_2, $torques$)	torques between o_1 and o_2, $torques$ is a vector $\langle t_x, t_y, t_z\rangle$

TABLE 3.4 Fluents of Topological, Directional and Distance Relationships.

Type	Fluent	Description
Topological	*on*(o_1, o_2)	either *on*$_{rigid}(o_1, o_2)$ or *on*$_{fluid}(fo_1, o_2)$ holds
	on$_{rigid}$(o_1, o_2)	rigid object o_1 is on object o_2
	on$_{fluid}$(fo_1, o_2)	fluid object fo_1 is on object o_2
	in(o_1, o_2)	either *in*$_{rigid}(o_1, o_2)$ or *in*$_{fluid}(fo_1, o_2)$ holds
	in$_{rigid}$(o_1, o_2)	rigid object o_1 is in object o_2
	in$_{fluid}$(fo_1, o_2)	fluid object fo_1 is in object o_2
Directional	*in_front*($o, ref, orig$)	o is in front of reference ref given origin $orig$
	behind($o, ref, orig$)	o is behind of reference ref given origin $orig$
	left($o, ref, orig$)	o is left of reference ref given origin $orig$
	right($o, ref, orig$)	o is right of reference ref given origin $orig$
	below($o, ref, orig$)	o is generally below of ref given $orig$
	below$_{strict}$($o, ref, orig$)	o is strictly below of ref given $orig$ whereby o's x-y-bounding-box is inside ref's x-y-bounding-box
	below$_{relaxed}$($o, ref, orig$)	o is below of ref given $orig$ whereby o's center is inside ref's x-y-bounding-box
	above($o, ref, orig$)	o is generally above of ref given $orig$
	above$_{strict}$($o, ref, orig$)	o is strictly above of ref given $orig$ whereby o's x-y-bounding-box is inside ref's x-y-bounding-box
	above$_{relaxed}$($o, ref, orig$)	o is above of ref given $orig$ whereby o's center is inside ref's x-y-bounding-box
Distance	*close*(o_1, o_2)	object o_1 is close to object o_2
	distant(o_1, o_2)	object o_1 is distant from object o_2


contacts(block_red,table)
contacts(block_blue,table)
contacts(block_green,table)
contacts(block_green,block_red)
contacts(block_red,block_blue)
contacts(block_red,block_green)
contacts(block_blue,block_red)
contacts(block_blue,table)
contacts(block_blue,table)
contacts(block_red,gplane)
contacts(block_green,gplane)

(a) (b)

(c) (d)

(e) (f)

FIGURE 3.2 Blocks world scenario. Left: Various physical configurations of blocks in the Gazebo simulator. Right: The robot's belief state about the scenario including annotated contact information visualized in RVIZ.

Fluents describe conditions over time. For example, the condition that object *A* is on object *B* changes during the course of action. Therefore, fluents have to be related to time. In this work, we represent fluents as functions and use similar notations as in the Event Calculus (Kowalski and Sergot, 1986a). The *Holds* predicate is used to test whether a fluent is true at a certain point in time or not. Fluents that are interpreted as functions are called *reified*. Generally, the *Holds* predicate looks as follows:

$$\mathit{Holds}(\overbrace{f}^{\text{Fluent}}, \overbrace{t}^{\text{Time point}})$$

where *f* is an arbitrary fluent from Table 3.4 and *t* a point in time.

The truth values of fluents are defined by logical sentences. These sentences are formed on the basis of other fluents or/and predicates that are grounded in the data structures of the simulator. For example, the *on* fluent is defined as follows:

$$\begin{aligned}&\mathit{Holds}(\mathit{on}(o_1, o_2), t_i) \Leftrightarrow\\&\quad \mathit{Holds}(\mathit{contacts}(o_1, o_2), t_i,)\wedge\\&\quad \mathit{Holds}(\mathit{above}(o_1, o_2), t_i,)\end{aligned}$$

Basically, an object o_1 is on another object o_2 whenever the objects are in contact with each other and the first object is above the second. The *contacts* fluent is simply defined by the value reported by the simulator:

$$\begin{aligned}&\mathit{Holds}(\mathit{contacts}(o_1, o_2), t_i) \Leftrightarrow\\&\quad \mathit{SimulatorValue}(\mathit{contacts}(o_1, o_2), t_i)\end{aligned}$$

The *above* fluent retrieves the bounding boxes of both objects and compares them in order to compute its truth value:

$$\begin{aligned}&\mathit{Holds}(\mathit{above}(o_1, o_2), t_i) \Leftrightarrow\\&\quad \mathit{SimulatorValue}(\mathit{bbox}(o_1, \mathit{bbox}_1), t_i)\wedge\\&\quad \mathit{SimulatorValue}(\mathit{bbox}(o_2, \mathit{bbox}_2), t_i)\wedge\\&\quad \mathit{Above}(\mathit{bbox}_1, \mathit{bbox}_2)\end{aligned}$$

The *Holds* predicate, introduced above, can be used to assess a condition at a certain point in time. However, many conditions hold not only a single point in time, but rather during a certain time span. To express that a fluent holds during whole interval we define another predicate as follows:

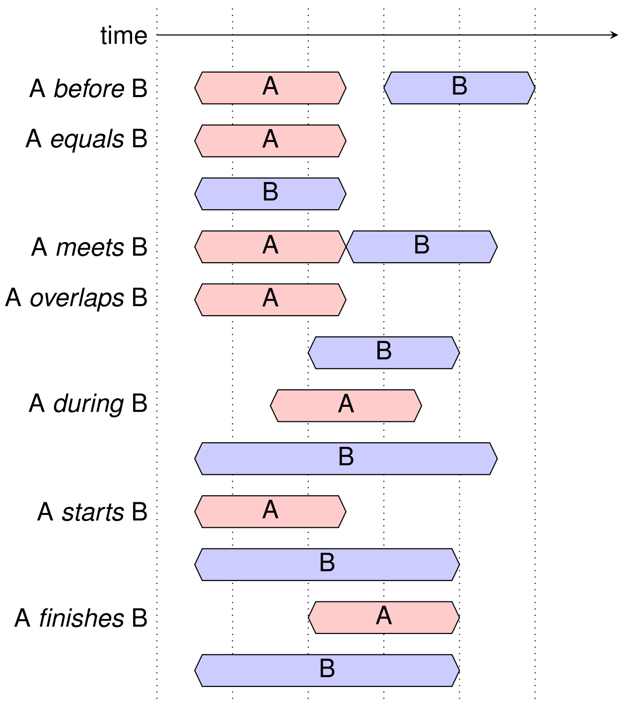


FIGURE 3.3 Possible relationships between pairs of intervals (Allen, 1983).

$$\mathit{Holds}_{\mathit{Throughout}}(\overbrace{\ f\ }^{\text{Fluent}},\ \overbrace{[t_1,t_2]}^{\text{Time interval}}\ )$$

whereby *f* denotes a fluent and $[t_1, t_2]$ is a time interval. Sometimes we also use *i* to denote a time interval. The $\mathit{Holds}_{\mathit{Throughout}}$ predicate allows us to talk about enduring conditions over time. In order to relate fluents that hold at different intervals we implemented predicates realizing the thirteen temporal relationships according to Allen (1983). Figure 3.3 depicts the possible ways two intervals can be related. For example, to describe that the pancake mix was in the container before it was on the pancake maker the following logical sentence can be used:

$$\begin{aligned}&\mathit{Holds}_{\mathit{Throughout}}(\mathit{in}(\mathit{mix},\mathit{container}), i_1)\wedge\\&\quad \mathit{Holds}_{\mathit{Throughout}}(\mathit{on}(\mathit{mix},\mathit{pan}), i_2)\wedge\\&\quad \mathit{Before}(i_1, i_2)\end{aligned}$$

Events Besides fluents, a temporal representation must be able to describe the occurrence of events and actions. In analogy to fluents, we assess the truth value of events and actions using the *Holds* and the $\mathit{Holds}_{\mathit{Throughout}}$ predicates, i.e.:

$$\mathit{Holds}(\mathit{occurs}\overbrace{(e)}^{\text{Event}}, \overbrace{t}^{\text{Time point}}) \text{ or } \mathit{Holds}_{\mathit{Throughout}}(\mathit{occurs}\overbrace{(e)}^{\text{Event}}, \overbrace{[t_1, t_2]}^{\text{Time interval}})$$

For example, the event of cooking the pancake mix is formalized as

$$\mathit{Holds}_{\mathit{Throughout}}(\mathit{occurs}\overbrace{(\mathit{cook}(\mathit{mix}))}^{\text{Event}}, \overbrace{[t_1, t_2]}^{\text{Time interval}}).$$

Actions of a robot can be represented similarly. Pouring the pancake mix can be represented as

$$\mathit{Holds}_{\mathit{Throughout}}(\mathit{occurs}\overbrace{(\mathit{pour}(\mathit{mix}))}^{\text{Event}}, \overbrace{[t_1, t_2]}^{\text{Time interval}}).$$

The next section explains how “manipulation stories” or narratives can be represented effectively using timelines.

3.5 Timelines (Narratives)

In this thesis we have developed timelines as a data structure to represent all information about narratives. Similar to chronicles (Ghallab, 1996), timelines represent reified fluents in temporally qualified predicates. Figure 3.4 visualizes fluents and events of the “Making Pancakes” problem that are captured by a timeline. The actions of the robot correspond to the formal representation shown in Figure 2.2 (yellow). The cooking event (blue) is relatively short, since it only transforms the *mix* into a *pancake*. This behavior can be recognized by a change from fluent *on(mix,pan)* to *on(pancake,pan*) (green). The undesired effect of spilling some pancake mix onto a table is represented by the fluent *spilled* (red).

Timelines are comprehensive data structures that are consulted for answering queries about a particular narrative or a set of multiple narratives. Therefore, it is important that queries can be formulated in a way that they relate either to a single or to multiple timelines. We achieve this, by extending all previously introduced predicates by a third argument, namely a *timeline*. Hence the *Holds* predicate is finally formalized as follows:

$$\mathit{Holds}(\overbrace{f}^{\text{Fluent}}, \overbrace{t}^{\text{Time point}}, \overbrace{tl}^{\text{Timeline}})$$

and the $\mathit{Holds}_{\mathit{Throughout}}$ predicate as:

$$\mathit{Holds}_{\mathit{Throughout}}(\overbrace{f}^{\text{Fluent}}, \overbrace{[t_1, t_2]}^{\text{Time interval}}, \overbrace{tl}^{\text{Timeline}})$$

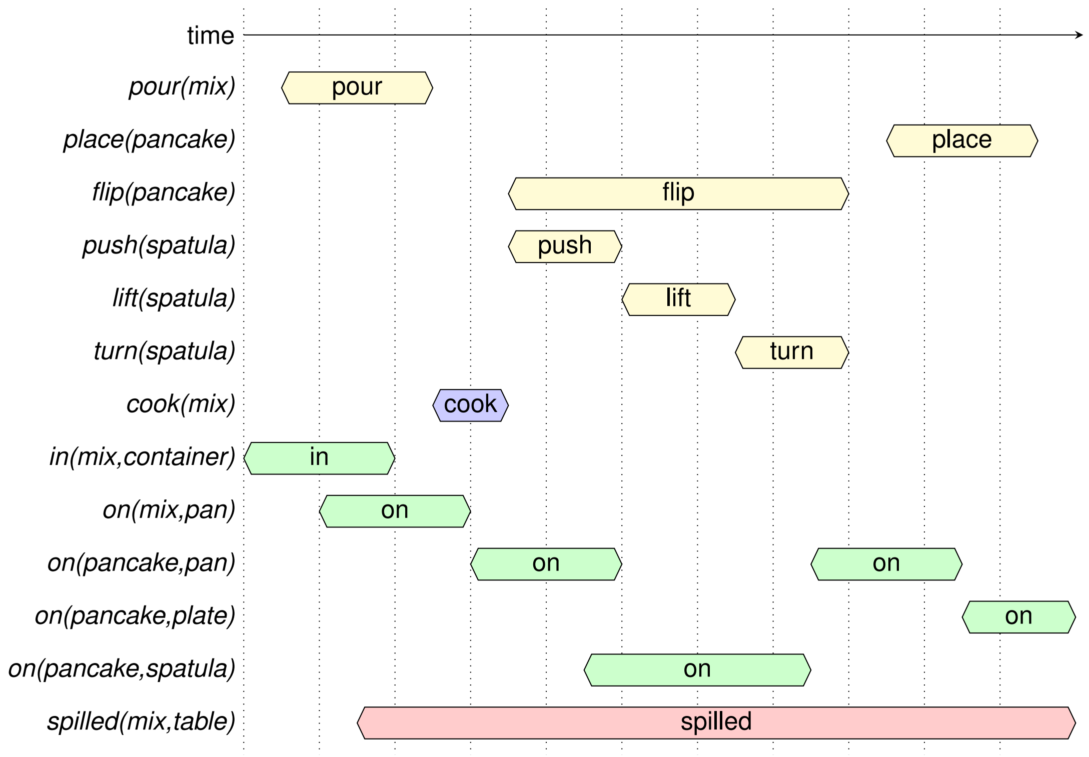


FIGURE 3.4 Timeline representation of making pancakes.

whereby *tl* is a unique *ID* for accessing a timeline. Similarly, the *SimulatorValue* predicate is extended with an additional argument.

We use PROLOG's search mechanism to retrieve answers from a set of timelines. In general, the linear search for particular objects, fluents, and/or events over a set timelines is quite slow. Therefore, we have designed and implemented several internal data structures that allow for an efficient search. For example, we use object-dependent skip lists (Munro et al., 1992) of temporal events to find fluents and events related to an object rather quickly. Furthermore, instead of linear search we use binary search methods to retrieve information from a timeline in logarithmic time.

3.6 Robots and Actions

Increasing the re-usability of robot control programs and software libraries has spawned several efforts to decouple the control programs as much as possible from the robots they are applied to. One line of abstraction is the use of robot description languages which provide models of a robot and then design and implement software components that work

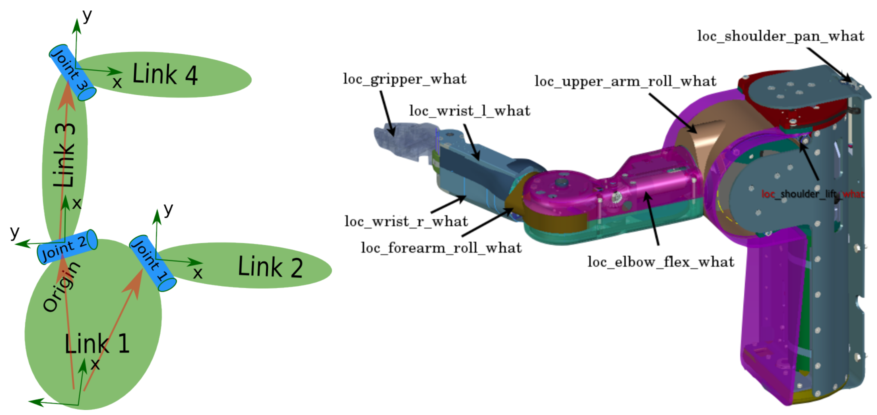


FIGURE 3.5 Left: Visualization of URDF's basic language elements: links and joints. Right: Naming convention of labeling the parts of PR2 robot. Courtesy: Willow Garage (`http://www.ros.org/wiki`), available under a Creative Commons Attribution 3.0 license.

on the model components rather then the particular robot instance. The specialization to particular robots is then performed through the parametrization of the control programs.

A recent and successful example of such a robot description is the Unified Robot Description Format (URDF)[8], which can be used to specify the kinematics and dynamics, the visual representation and the collision model of a robot. By using the information of a robot description which is specified in URDF robot systems can calculate the 3D pose of a robot and detect potential collisions between a robot and its environment. Programmers can use the URDF description to visualize the respective robot and to simulate it in a physics-based simulator.

URDF specifies robot models using primarily two different language elements, namely *links* and *joints* (Figure 3.5, left). A *link* element describes a rigid body part of a robot by specifying its origin, mass, inertia, geometry, visual appearance, and a collision model, whereas a *joint* element describes the connection of two *links*. The *joint* specification includes for example the joint type, e.g. revolute, continuous, or prismatic, and the joint limits. Figure 3.6 shows the arm and hand of our robot Rosie and visualizes the corresponding URDF specification.

While URDF robot descriptions go a long way they are also still limited. In this work we investigate one of these limitations: the lack of semantics of robot parts in the URDF model. In URDF what a link corresponds to is only present in the link name. Thus, links have the name left-gripper-motor or left-gripper-encoder but the robot system does not know what this name means (Figure 3.5, right).

In this thesis, we explicitly represent the components of URDF descriptions in a sym-

[8]`http://www.ros.org/wiki/urdf`

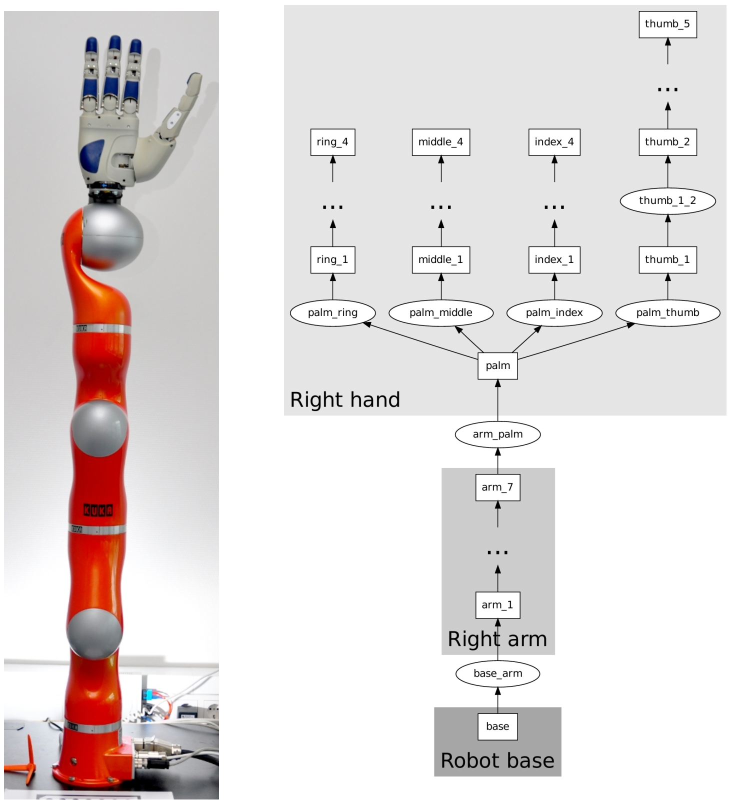


FIGURE 3.6 Left: Rosie's KUKA-lightweight LWR-4 arm with DLR-HIT hand. Right: Visualization of the corresponding URDF specification. URDF's *links* and *joints* are visualized by boxes and ellipses respectively.

bolic knowledge base containing encyclopedic knowledge about robots and their components. We show how the robot specification format URDF can be embedded into a more powerful semantic robot description language, which we call SRDL. SRDL addresses not only robot descriptions but also the descriptions of actions and capabilities. It also provides inference mechanisms that allow reasoning about the ability of robots to perform certain actions.

SRDL enables robots to map an instruction like "Turn the wrist quickly to put the pancake upside down onto the pancake maker" onto its own robot model and thereby generate it into an executable robot plan, as described in (Tenorth et al., 2010b). The action in this generated plan refers to the concepts *wrist*, but not to the individual *links* or *joints*. Although *wrist* is implicitly described by the URDF specification, the concepts cannot be located within the robot body.

The following fictitious dialog between a human and a robot illustrates the kind of questions that could be answered using SRDL:

HUMAN: Can you make a pancake using a pancake maker?

ROBOT: Yes, I can.

HUMAN: How good are you in making pancakes?

ROBOT: I've a success rate of 78%.

HUMAN: Can you also make pancakes using a frying pan?

ROBOT: No, I cannot.

HUMAN: Why not?

ROBOT: I am missing an object model for perceiving a frying pan.

For answering these questions the robot basically has to match the action specifications and requirements with its own robot description and its collected experience.

To this end, this work contributes the following:

- we have developed a semantic description language for robots (including sensors, actuators, control algorithms and information objects);
- we have extended the action representation established in (Tenorth et al., 2010b) and link it to the developed robot descriptions; and
- we have designed and implemented inference mechanisms for matching robots and actions and computing a success probability for actions based on experience.

3.6.1 Semantic Robot Description Language (SRDL)

SRDL is used to specify robots, capabilities and everyday actions. Inference algorithms are used to match action descriptions to components of robots via the concept of capabilities. In the following, we first give an overview of the overall system and clarify the meaning of important concepts. Secondly, we present how robots, actions and capabilities are represented by the means of ontologies. And finally, we explain how the developed SRDL inference algorithms work.

3.6.1.1 System Overview and Terminology

SRDL is tightly integrated within the knowledge processing system KNOWROB (Tenorth and Beetz, 2009). In terms of knowledge representation and reasoning this means that SRDL uses the Web Ontology Language OWL[9] for modeling knowledge about robots,

[9]http://www.w3.org/TR/owl-features/

TABLE 3.5 Terminology.

Concept	Definition	Examples
Robot	We define *Robot*, according to (Russell and Norvig, 2009), as physical agent that performs actions by manipulating the physical world and that have effectors and sensors. (Section 2.1.3)	*Rosie*, *PR2*
Component	A *Component* of a robot is defined as hardware, e.g. sensors and effectors, software control program or information object, e.g. maps and object models.	*DLR-HIT-Hand*, *GraspPlanner*, *TeaCupModel*
Action	Following OpenCyc's definition of *PurposefulAction*, an *Action* is consciously, volitionally, and purposefully done by at least one actor.	*MakingPancakes*, *PickingUpAnObject*
Capability	A *Capability* is the ability to perform a certain action. A robot has a *Capability* due to its components.	*PickAndPlace-Capability*

capabilities and actions, and that it uses the logical programming language PROLOG for implementing inference algorithms.

Using an OWL representation allows SRDL to build on the inference mechanisms already available for OWL reasoning. For example, an object can be classified as being a *LaserScanner* based on the necessary and sufficient conditions for a *LaserScanner* defined in the OWL ontology. This classification can be used in SRDL inference if a *LaserScanner* is necessary for a certain action.

The integration within KNOWROB allows SRDL to build on an established knowledge base for robots. Whereas it mostly re-uses KNOWROB's knowledge about actions, SRDL extends the knowledge with information about robots, components and capabilities and relates it to existing knowledge. Furthermore, we established a separate module that automatically imports URDF specifications into SRDL's ontology.

But before we delve into the details of SRDL, it is important to understand the most relevant concepts within our approach, namely *Robot*, *Component*, *Action* and *Capability*. Definitions and examples for these concepts are given in Table 3.5, whereas their relationship is visualized in Figure 3.7.

3.6.1.2 Ontological Representation

In this section we present how robot components, actions, and capabilities are modeled within SRDL using OWL.

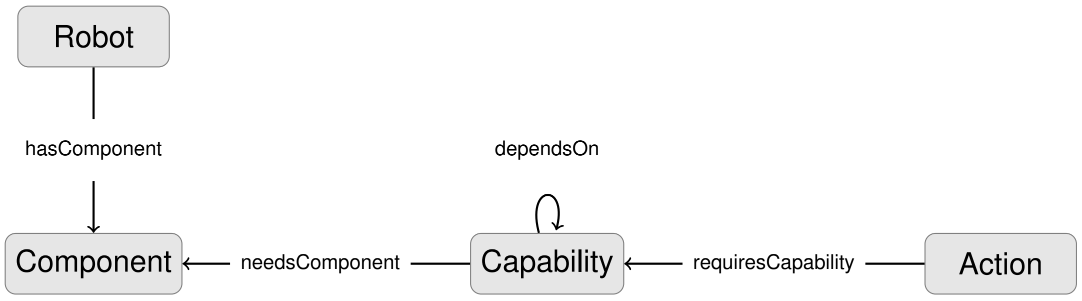


FIGURE 3.7 Relationships between the concepts *Robot*, *Component*, *Capability* and *Action*

Robot Components A component is everything a robot consists of including but not limited to hardware components, software control programs and information objects.

Each component is represented in the SRDL ontology as being an instance of a component class to capture its type and its specific properties. The component taxonomy is shown in Figure 3.8 and defines component types, their relationships to other components and specific properties on component level. The component taxonomy covers hardware components (sensor and actuators), software components (control programs) and information objects (object models, data sets, and maps). The inheritance relationship of a taxonomy can be used to query for certain types of components or components with specific properties, e.g. all sensors or all information objects.

In order to capture the kinematic chain between hardware components of a robot the transitive object property *hasSuccessorInKinematicChain* can be used. Examples for such relationships in kinematic chain are a hand being a successor of an arm or a joint being the successor of a link. When using the URDF import module, an individual for each component is created and the relationships are asserted in this way. For example, in Figure 3.6 each arrow corresponds to a *hasSuccessorInKinematicChain* assertion. Due to the transitivity of property *hasSuccessorInKinematicChain* it is easy to obtain all successors of an arbitrary component just by traversing the kinematic chain spanned by *hasSuccessorInK-inematicChain* assertions.

The robot description at link and joint level is a rather low-level description. It can be used in order to calculate forward and inverse kinematics but it does not contain knowledge about the purpose of a certain link or joint. Often it is interesting to know what high-level component such as hand or arm is formed by a set of links and joints. In order to be able to model such high-level components that consist of low-level components, a high-level component can be defined as a component composition, e.g. a *DLR-HIT-Hand*, and the set of low-level components that are part of the high-level component can be

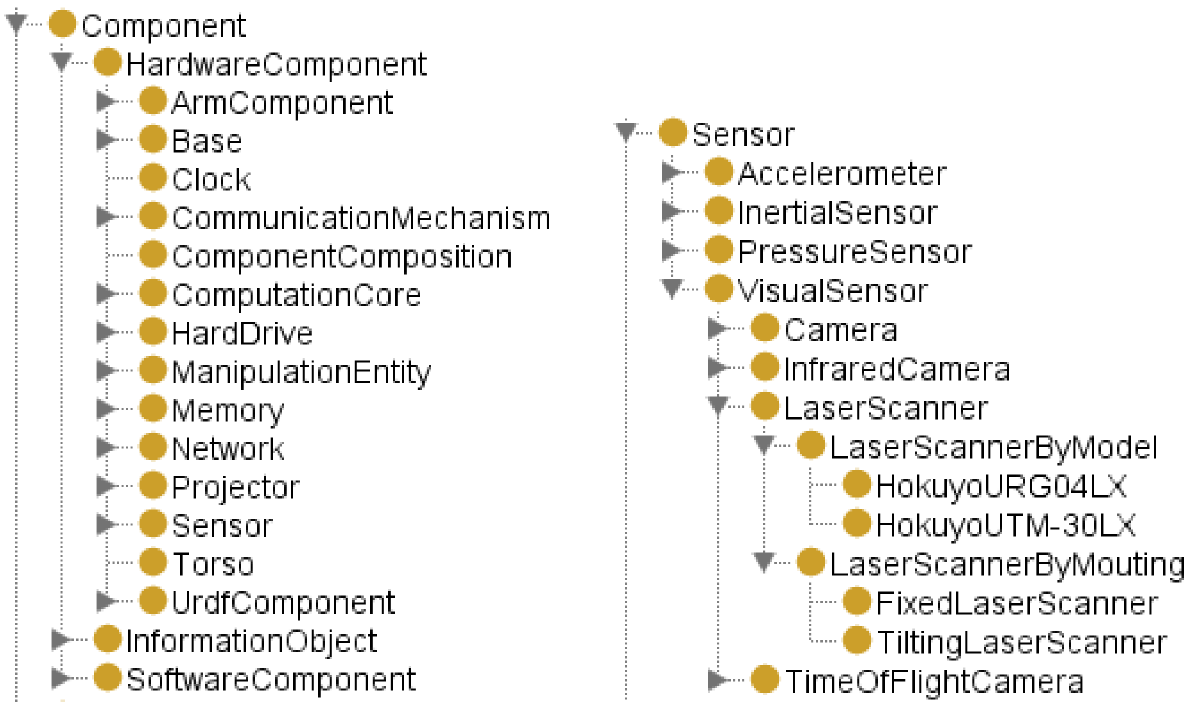


FIGURE 3.8 (Part of) Component and sensor taxonomy

specified. This can be done using the object properties *hasBaseLinkInComposition* and *hasEndLinkInComposition*. As the kinematic chain is a tree structure there is always one base link of a component composition but there may be more than one end links (as is the case for a hand). For example, if we want to define the right hand pictured in Figure 3.6 as a composite component, we would define *palm* as base link and *ring_4*, *middle_4*, *index_4*, and *thumb_5* as end links.

So far we have discussed how single components and component compositions can be defined. In the following we present how specific components can be asserted to be part of a specific robot. There are three ways how this can be achieved: first by the kinematic chain mechanism as explained above; second by directly asserting a component to a robot using the property *hasComponent*, and third by the property chain mechanism for software components.

Actions Actions are events that are performed by a robot purposefully to change its environment and accomplish a goal.

An action consists of steps or sub-actions and those themselves are actions and can consist of sub-actions. This structure results in a hierarchical tree, called *action tree*. The action tree of an action is specified in SRDL using the object property *hasSubAction*. For each sub-action an assertion of property *hasSubAction* between the action and the sub-action is made. Figure 3.10 shows an example of an action tree.

An action depends on a set of capabilities. The robot must possess all of these capabilities, otherwise it is judged to be incapable of performing the action. These capability dependencies of an action are specified using the object property *requiresCapability*. For example, action *PickingUpAnObject* in Figure 3.11 needs capability *MovingArmCapability* among others in order to perform the picking action.

A goal of SRDL is to enable a robot to use experience about narratives of an action in order to predict how likely it will be successful when performing an action. To accomplish this, experience information about an action has to be modeled. Datatype properties *hasNumSuccesses* and *hasNumNarratives* are used to store the number of successful narratives and total number of narratives for an action.

Capabilities Capabilities represent the ability of robots to perform certain actions.

It is difficult to compare a robot and an action directly because they have very different characteristics: a robot has physical, spatial characteristics whereas an action abstract, hierarchical ones. In order to be able to match between robots and actions the concept of “capabilities” is introduced. An action depends on a set of capabilities and a set of components equips the robot with a certain capability. A capability is defined as a class in the capability taxonomy and its properties are specified as explained below.

In order to exhibit a certain capability, a robot has to possess a set of components that cooperate and jointly enable the robot to exhibit the capability. For example, to exhibit the capability of navigating around in its environment, the robot needs components such as mobile base, a motion controller, a sensor, a object detection component and a map. Component dependencies of a capability can be modeled using object property *needsComponent*.

Additional to depending on components, a capability can also depend on other capabilities, so called "sub-capabilities". Sub-capabilities denote prerequisites of capabilities. This design allows modularization and reuse of definitions of capabilities and their component dependencies. The inference algorithm has to check sub-capabilities recursively when matching between robots and actions. Dependency of a capability on another capability can be modeled using object property *dependsOn*.

3.6.1.3 Inference Algorithms

The developed inference algorithms are realized by utilizing PROLOG's reasoning engine and additional libraries like the Semantic Web Library[10], but not employing of-the-shelf Description Logic reasoners. This allows us to easily extend the inference mechanism with other types of reasoning which are available in KNOWROB, e.g. commonsense reasoning. For instance, if a robot could not find cups for setting a table, it can propose to use glasses which are also of type container (cf. (Galindo et al., 2008)).

In this section we will discuss inference algorithms that match between robots and actions and calculate a success probability for performing an action. It is helpful to keep the knowledge model that is discussed above and summarized in Figure 3.7 in mind when studying the inference algorithms.

Matching between Robots and Actions The main question of SRDL is the matching between robots and actions: to judge if a robot can perform an action, totally independent of a real execution.

The matching algorithm depends on the action tree and capability dependencies of actions. Robot *R* must fulfill two conditions in order to be able to perform an action *A*: First, all dependencies on capabilities of *A* must be fulfilled, meaning that all capabilities needed by *A* are available. Second, *R* must be able to perform all sub-actions of *A*.

In order to check those conditions, the inference algorithm iterates over all capability dependencies of *A* and checks for each single capability if it is available on *R* as described below. Further, the inference algorithm checks all sub-actions of *A* recursively. If a single missing capability or unfeasible sub-action of *A* is found, *A* is being judged to be unfeasible on *R*. If all capability dependencies are met and all sub-actions are feasible, *A* is judged to be feasible on *R*.

There are two conditions a robot *R* must fulfill in order to exhibit capability *C*: First, all component dependencies of *C* have to be fulfilled. A single component dependency is fulfilled if *R* possesses an instance of the component specified by the component dependency. Second, all sub-capabilities of *C* must be available on *R* also.

The inference algorithm checks these two conditions for capability *C* by iterating over all component dependencies and sub-capabilities of *C* and checking each single one (in case of sub-capabilities recursively). If only one missing component dependency or unavailable sub-capability is found, *C* is judged to be unavailable on *R*.

[10]`http://www.swi-prolog.org/pldoc/package/semweb.html`

In case of a negative matching outcome between a robot *R* and an action *A* it is desirable to know the reason for the failure. Thus, SRDL must be able to explain its inference result and enumerate all capabilities and components that are required for performing *A* but are missing on *R*.

The inference algorithm achieving this is an extension of the matching algorithm described above. It collects all capabilities that are required by *A* or one of its sub-actions, checks which of these capabilities are unavailable on *R* and produces a list of missing capabilities *MissingCapabilities*. Iteration over *MissingCapabilities* and collecting all their component dependencies results in the list of missing components. Note that sub-capabilities have to be considered recursively and a component dependency has to be checked if it is really nonexistent on *R* as missing capabilities usually depend on existing and nonexistent components.

Success Probability of an Action A second inference task of SRDL is to predict a success probability $P_{Success}(A)$ for robot *R* performing action *A*. We implemented two strategies for computing the success probability; both of them are based on the asserted numbers of narratives and successes in the past.

If numbers of successes and narratives are asserted for action *A* the success probability $P_{Success}(A)$ can be calculated by dividing the number of successes by the number of narratives: $P_{Success}(A) = \frac{NumSuccesses}{NumNarratives}$.

If none or only one number for narratives and successes is asserted for action *A* on robot *R*, direct calculation is not possible. But the success probability of *A* can be calculated based on the success probabilities of the sub-actions of *A*. This is done by computing the success probabilities $P_{Success}(SubA)$ of all sub-actions of *A* and setting the success probability $P_{Success}(A)$ of *A* to the product of success probabilities of all sub-actions. The product of success probabilities of sub-actions is used assuming that all sub-actions are independent and due to the fact that the joint probability of independent events is the product of the probabilities of each single event. Sub-actions for which no success probability can be calculated are ignored and the calculation fails if there is no sub-action for which a success probability can be calculated.

3.6.1.4 Example Problems

In this section we demonstrate the applicability of our approach by providing examples for robot and action descriptions and by explaining how the inference algorithms compute their consequences. We take a situation where our robot Rosie (Figure 3.9) is supposed to make pancakes as sample problem. We first elaborate on the robot specification of Rosie, then present the action description for making pancakes, and finally explain how the inference algorithms compute their results for various queries.

Rosie the Robot The URDF description of Rosie can be automatically imported into the SRDL ontology. Thereby Rosie's low-level description including its kinematics is available in SRDL. Based on this low-level description, high-level components, i.e. sensors and actuators, are defined as instances of their respective classes. For example, the hand of Rosie is of type *DLR-HIT-Hand* and it is constituted of 18 instances of type *URDFLink* and 17 instances of type *URDFJoint* (Figure 3.6). The information about Rosie's components can be retrieved from the ontology via a PROLOG interface. For example, the following query asks for all components of Rosie that are of type *Sensor*:

```
?- hasComponent(srdl:'Rosie', Comp),
   isType(Comp, srdl:'Sensor').

 Comp = 'camera1-left' ;
 Comp = 'camera2-right' ;
 Comp = 'infraredCamera' ;
 Comp = 'swissRangerTOF' ;
 Comp = 'videreStereoOnChip' ;
 Comp = 'hokuyo-shoulder' ;
 Comp = 'hokuyo-rear' ;
 Comp = 'hokuyo-front'.
```

Note, that *hasComponent* collects all components of Rosie by following the underlying kinematic chain imported from URDF and that *isType* filters all instances whose type is derived from the concept *Sensor*.

Exemplarily, let us now look more closely at the last three instances, i.e. *hokuyo-<pos>*. All of them have are asserted to be of type *HokuyoURG4LX* and their membership of being of type *Sensor* is derived because of the subclass specifications of *Sensor*, *2D-VisualSensor* and *2D-LaserScanner* in the sensor taxonomy. Although all three instances have the same type, they differ in the way they are mounted on Rosie. But despite the different mounting locations, a more critical distinction is that the shoulder scanner can be tilted whereas the others are fixed. This distinction is reflected within the ontology where *hokuyo-front* and *hokuyo-rear* are additionally modeled as sub-concepts of *2D-FixedLaserScanner* whereas *hokuyo-shoulder* is additionally modeled as sub-concept of *3D-TiltingLaserScanner*. This differentiation within the taxonomy is highly relevant for

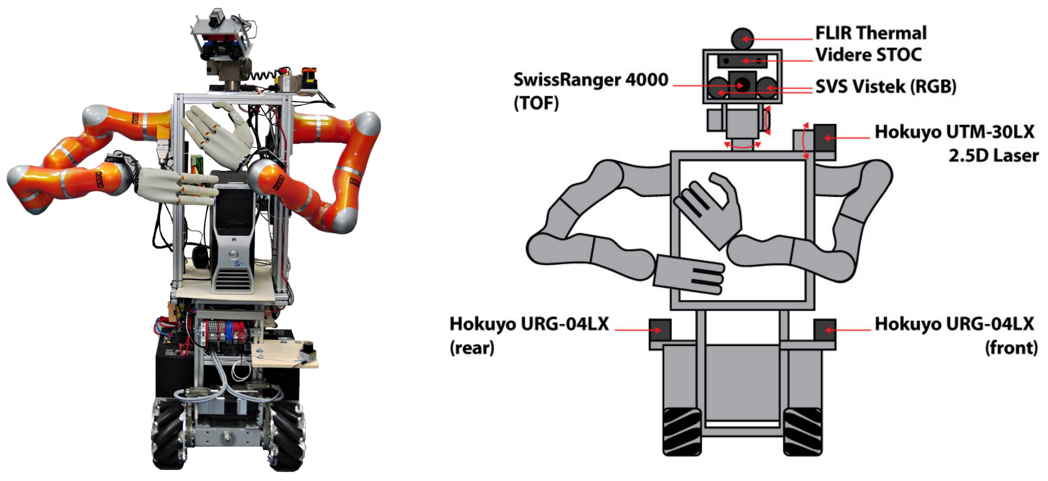


FIGURE 3.9 Rosie and its high-level components. Courtesy: IAS.

Rosie's software components that rely on data gathered by these sensors. For example, perception routines for 3D object recognition require as input 3D point cloud data which can only be provided by an instance of type *3D-TiltingLaserScanner* but not *2D-FixedLaserScanner*.

After we highlighted some details about our robot Rosie, we now shortly consider the action description for making pancakes and explain how these concepts are related to robot components via capabilities.

Making Pancakes For modeling actions we adopted the formalism established within KNOWROB. The action description for making pancakes is represented by an instance of type *MakingPancakes* which consists of several sub-actions. For example, the action *PouringDoughOntoPancakeMaker* refers to an object on which is acted on, e.g. an instance of type *Dough* and a location to which the dough is poured to, namely the *Pancake-Maker*. *FlippingAPancake* has itself an ordered sequence of sub-actions: *PushingAnObject*, *LiftingAnObject* and *TurningAnObject*. Again, these actions denote further sub-actions, for example, *PickingUpAnObject*. A simplified excerpt of the hierarchical action representation of making pancakes is depicted in Figure 3.10.

We now explain how the action representations are related to robot components by examining the action *PickingUpAnObject* in detail. Picking up an object is basically an ordered sequence of actions, namely recognizing the object, moving the arm/hand towards it, grasping it, and finally lifting it. For doing so, the robot needs the respective

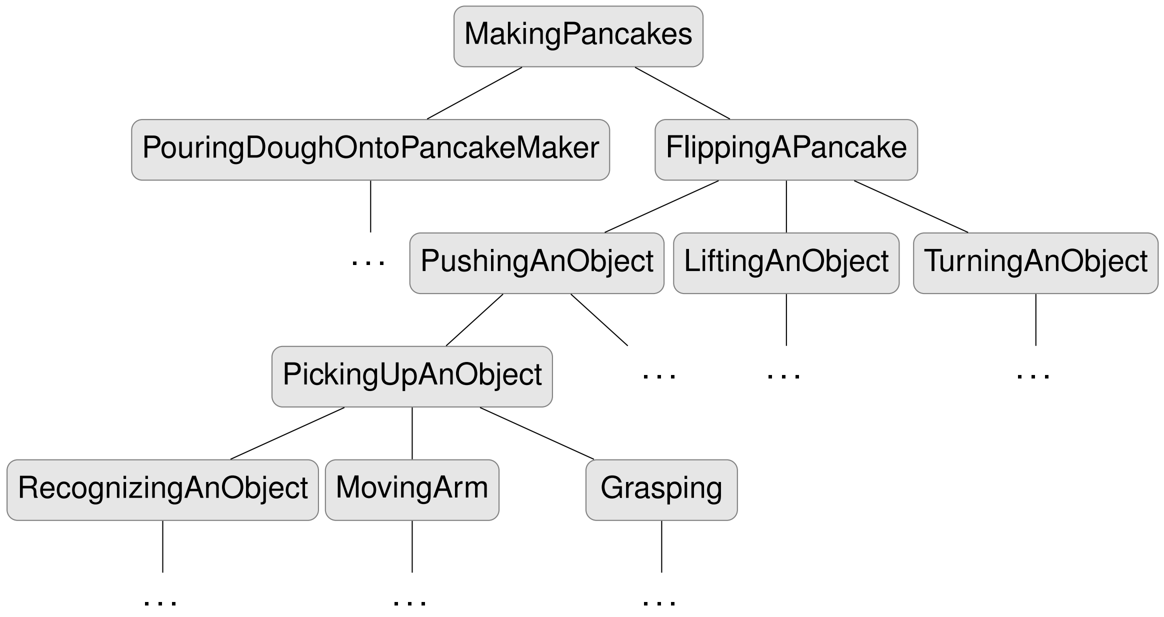


FIGURE 3.10 Hierarchical action tree for making pancakes.

capabilities. These capabilities are associated with *PickingUpAnObject* within the ontology. More specifically, the logical conjunction of role *requiresCapability* with range restrictions *3D-ObjectRecognitionCapability*, *MovingArmCapability*, and *GraspingCapability* form a super-class of *PickingUpAnObject*. The capabilities themselves refer to component classes they require, e.g. *3D-ObjectRecognitionCapability* depend on components of class *3D-VisualSensor* (e.g. *3D-TiltingLaserScanner*) and *3D-ObjectRecognitionAlgorithm*. The latter component has further dependencies, for recognizing a certain object the algorithm needs the respective *3D-ObjectModel*. Figure 3.11 visualizes the relations between Rosie's components, its capabilities and the action of making pancakes.

Inferences After we have seen how knowledge about robots and actions can be represented by SRDL, we demonstrate the inference mechanisms by examining the questions of the dialog presented in Section 3.6.

Q1 *Can you make a pancake using a pancake maker?*
For determining whether Rosie is able to make a pancake the inference algorithm matches the instance of *MakingPancakesUsingPancakeMaker* with Rosie's robot description.

In the first step the algorithm collects all sub-actions of *MakingPancakesUsingPancake-Maker*. The result is a list including the following actions: *PouringDoughOntoPancake-Maker*, *FlippingAPancake*, *PushingAnObject*, *LiftingAnObject*, *TurningAnObject*, *PickingU-*

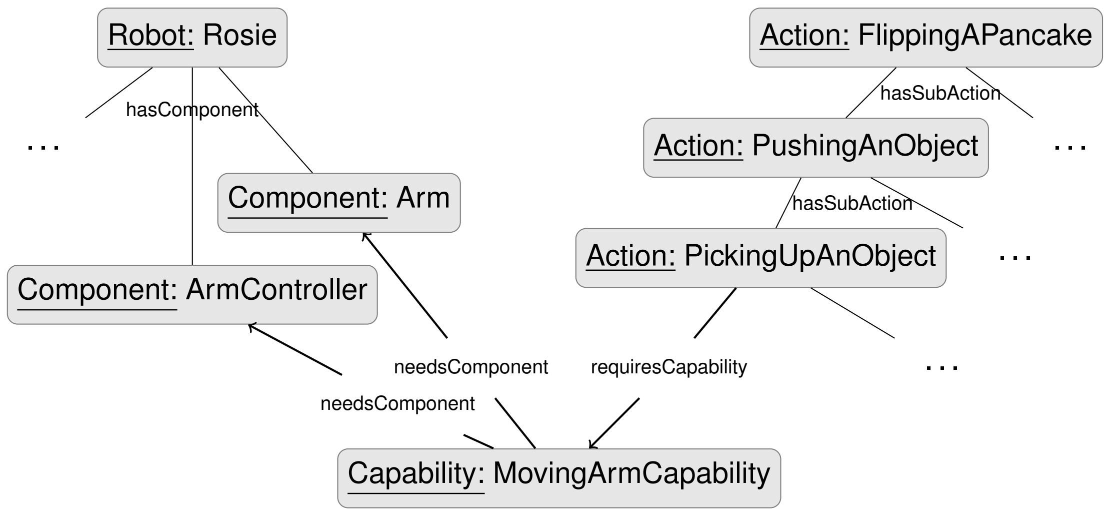


FIGURE 3.11 Relations between Rosie's components and a flipping action through the concept of capabilities.

pAnObject, *RecognizingAnObject*, *MovingArm*, *Grasping* etc.

In the second step the algorithm proves for each action found in step one that Rosie is capable of performing it. For example, for proving that Rosie is capable of *FlippingAPancake* all capability dependencies of this action sub-tree are collected and verified. As we saw earlier, *PickingUpAnObject* is one of the sub-actions in this action-tree. And since we cannot provide details for all sub-actions we give some explanations for *PickingUpAnObject*: *MovingArmCapability* and *GraspingCapability* are provided by Rosie's hardware components of type *KUKA-LWR4-Arm* and *DLR-HIT-Hand* and their respective control programs. Rosie's component instances of type *3D-TiltingLaserScanner*, *3D-PerceptionPipeline*, and *3D-SpatulaModel* provide the *3D-ObjectRecognitionCapability*. Similarly, the other actions retrieved in step one can be verified. Hence, the following PROLOG predicate evaluates to true:

matchRobotAndAction(srdl:'Rosie', srdl:'MakingPancakesUsingPancakeMaker').

Q2 *How good are you in making pancakes?*
The success probability for setting the table is calculated on the basis of Rosie's experience made in the past, that is, narratives. If the number of successful narratives and the total number of narratives are explicitly available for an action the success probability is directly given by the ratio. Otherwise the success probability of an action is the product of the success probabilities of all its sub-actions. In the case of *MakingPancakesUsingPancakeMaker* the success probability, computed on made-up data, is 78%.

Q3 *Can you also make pancakes using a frying pan? Why not?*
This prove is exactly the same as for *Q1*, despite that it fails. The reason why Rosie's description cannot be matched with *MakingPancakesWithFryingPan* is, that Rosie does not have an adequate object model. SRDL is designed that it can report why a prove fails by providing a list of missing capabilities and components. Here it says that the requirements for *3D-ObjectRecognitionCapability* are unmet because a component of type *3D-FryingPanModel* is missing.

3.6.2 Related Work on Robots and Actions

Related work oon robots description languages and action specifications comprises three directions: robot descriptions, capability matchmaking and a combination of both.

3.6.2.1 Robot Descriptions

URDF was already discussed in some detail in the beginning of Section 3.6. Its application to kinematics, collision detection and visualization make it a powerful and indispensable tool for robot development. However, URDF is not designed for specifying robot components such as sensors, actuators, and control programs and matching those to action descriptions. In contrast, SRDL exactly addresses these issues and thereby aims at filling the gap between low- and high-level descriptions.

COLLADA[11] is an XML Schema designed for describing 3D objects including their kinematics. It mainly focuses on modeling information about scenes, geometry, physics, animations, and effects. But similar to URDF, it lacks elements for describing sensors, actuators and software.

Both (Schlenoff and Messina, 2005) and (Chatterjee and Matsuno, 2005) developed an OWL ontology to describe robots including their capabilities that operate in the domain of urban search and rescue. In contrast to SRDL, components and attributes are directly asserted to robots without modeling a kinematic chain. Also capabilities are directly asserted and not inferred from robot components.

The state-of-the-art of semantic sensor specifications is reviewed by (Compton et al., 2009a). An OWL ontology that focuses on the composition of sensors is developed in (Compton et al., 2009b). Efforts for providing a standard for sensor specifications are undertaken by the W3C Semantic Sensor Network Incubator Group[12]. As sensors are an

[11]http://www.collada.org
[12]http://www.w3.org/2005/Incubator/ssn

important aspect within robotics, the work by (Compton et al., 2009a,b) and the Semantic Sensor Network Incubator Group is highly relevant for SRDL.

3.6.2.2 Capability Matchmaking

In general, the term *capability matchmaking* refers to the process of matching an advertisement of a capability with a request. In context of SRDL, a robot with its components corresponds to the advertisement whereas an action description resembles the request. The matching algorithm has to determine whether a robot is capable to perform an action.

In (Gil and Ramachandran, 2001), agent and task descriptions are modeled in an ontology and matched by subsumption-based reasoning.

The LARKS system developed by Sycara et al. (2002) is built for matching web-based software agents. It employs several matching mechanisms based on semantic similarity and service specification structure to determine different degrees of matching.

OWL-S (Martin et al., 2007) allows to model the functionality and the process of web services semantically by using domain ontologies and predefined modeling constructs. Information about the functionality is used by the matchmaking algorithm to semantically compare advertisement and request.

As the approaches above, SRDL uses an ontology for representing knowledge. But a major difference lies in the way the capability advertisements and requests are matched. Whereas the described systems match structurally equivalent specifications, SRDL matches action specifications to robots by verifying required capabilities and robot components.

3.6.2.3 Matching Sensor Descriptions to Tasks

In (Preece et al., 2008), the directions of robot descriptions and capability matchmaking are combined. The approach is able to compute compositions of assets that are jointly able to perform specific tasks in a military setting. Matching between tasks and assets is done by the means of capabilities.

When comparing SRDL to work of Preece et al. (2008) it is noticeable that similarities exist. This is because we adopted and transferred some of their ideas to the domain of household robots. But there exist also considerable differences: the support of kinematics, sets of components providing a capability and modeling of diverse robot components like actuators, control programs and information objects. Furthermore, SRDL includes mechanisms to handle experiences about actions.

3.7 Discussion

Spatio-temporal Representations In the beginning of this chapter we revisited the example of "Making Pancakes" and identified the relevant aspects that need to be represented for reasoning about such problems, namely, objects, actions, space, and time. We provided arguments for using logical representations and explained why we have chosen OWL and PROLOG to represent everyday manipulations scenarios. We further described how objects are represented within an ontology and how they are linked to physical models of a simulator. The logical representations of space and time which are based on the notations of the Event Calculus have been explained in detail. Further, we showed how these representations are grounded in the data structures of a physics-based simulator. As a major contribution of this thesis timelines has been introduced as a data structure for representing and reasoning about narratives of manipulation actions. In this thesis we demonstrate that timelines are suitable to capture robot and human object manipulation tasks. However, the applicability of timelines for representing everyday activities in general would be an important direction of future research.

Robots and Actions In this work we have investigated semantic robot description languages and inference mechanisms for matching robot descriptions with hierarchical action specifications. We presented SRDL for describing robot components, capabilities and actions, and showed that linking those descriptions allow us to make inferences about the ability of robots to perform certain actions. The proposed system has been implemented and integrated within the knowledge processing system KNOWROB. We demonstrated the applicability of our approach by closely looking into the descriptions of our robot Rosie and the action specifications of making pancakes, and by carefully following some sample inferences.

In future work SRDL can be extended in several ways. For example, mechanisms that keep track of the current state of robot components, including sensors, actuators, control programs and information objects could be integrated. Furthermore, constraints of components or sets of components could be considered in more detail. Another line of future work could investigate the monitoring of future tasks and the current state of robot components. For example, consider a robot that is supposed to use a rolling pin. Since it needs both hands for accomplishing this task, a problem can be detected when it is monitored that the robot already holds another object in one of its hands or one of its hand is broken. Such a monitor for tasks and components would help to observe potential problems, to recognize them in advance, and to deal with them proactively. Finally, a further direction that one could investigate is a tighter integration of SRDL with robot planning and robot learning.

CHAPTER 4

Envisioning Robot Object Manipulation

Autonomous robots that are to perform complex everyday tasks like making pancakes have to understand how the effects of an action depend on the way the action is executed. In case that the executed action do not achieve the desired goals or even lead to some undesired effects, a robot has to recover from the caused circumstances (if possible) and has to redo the action, but this time in the right way. By envisioning the outcome of its actions before committing to them, a robot becomes aware of everyday physical phenomena and thereby it can prevent itself of ending up in unwanted situations. Hence, robots can perform manipulation tasks more efficient, robust and flexible and they can even accomplish previously unknown variations of tasks successfully.

Within Artificial Intelligence, classical planning reasons about whether actions are executable but make the assumption that the performed actions will succeed (with some probability). In this work we have designed, implemented and analyzed a framework that allow us to envision the physical effects of robot manipulation actions. The envisioning is achieved by translating a qualitative physics problem formalization into a parameterized simulation problem, performing a detailed physics-based simulation of a robot plan, logging the state evolution into appropriate data structures and then translating these subsymbolic data structures into interval-based first-order symbolic/qualitative representations, called timelines. The result of the envisioning is a set of detailed narratives represented by timelines which are then used to infer answers to qualitative reasoning problems.

4.1 Physical Reasoning (in AI)

The basic idea of how human reasoning about problems like making pancakes (Section 1.2) can be realized using formal logic-based methods is depicted in Figure 4.1.

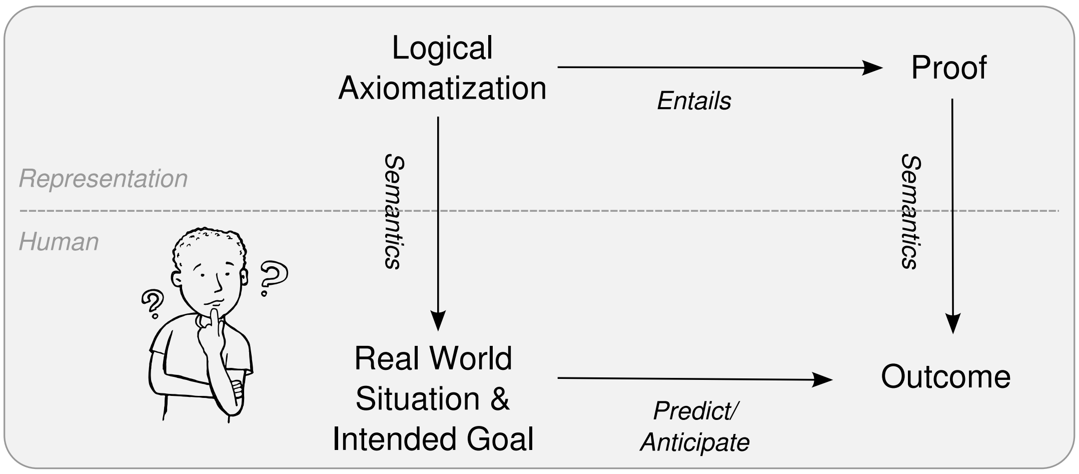


FIGURE 4.1 Commonsense reasoning and its formalization using first-order logic.

Given an initial situation and an intended goal, humans try to anticipate the appropriate sequence and configuration of actions that will lead to a desired outcome. Formally, the initial situation and a script of actions are described using a logical axiomatization. Then, a specialized calculus, for example, the situation or the event calculus (Kowalski and Sergot, 1986a), are applied in order to transform the axiomatization from the initial situation into a proof that resembles the intended goal.

However, if robots are supposed to use a similar mechanism several problems arise. Figure 4.2 shows a robot that has to infer how its behavior and physical effects influence the world state after executing a parameterized plan. Basically we see three major problems with this approach:

LEVEL OF ABSTRACTION Physical effects strongly depend on the concrete parameterization of actions. For example, the position of the hands and the tilting angle of the container when pouring the pancake mix onto the pancake maker clearly affect the outcome of the action. Abstracting away from these relevant details yields to over-simplified and inadequate conclusions.

INTERFERING EFFECTS/CONCURRENT ACTIONS Effects of single and/or concurrent actions can interfere with each other. For example, a robot that simultaneously moves its hand holding the pancake mix to a position over the pancake maker and tilts it might spill some mix onto the table before reaching its target position. Generally, such interfering effects are difficult to model using rules in a logical calculus.

HANDLING VARIANTS Robots should to be able to handle variants of the original problem. An intrinsic feature of everyday manipulation tasks is that they will never be per-

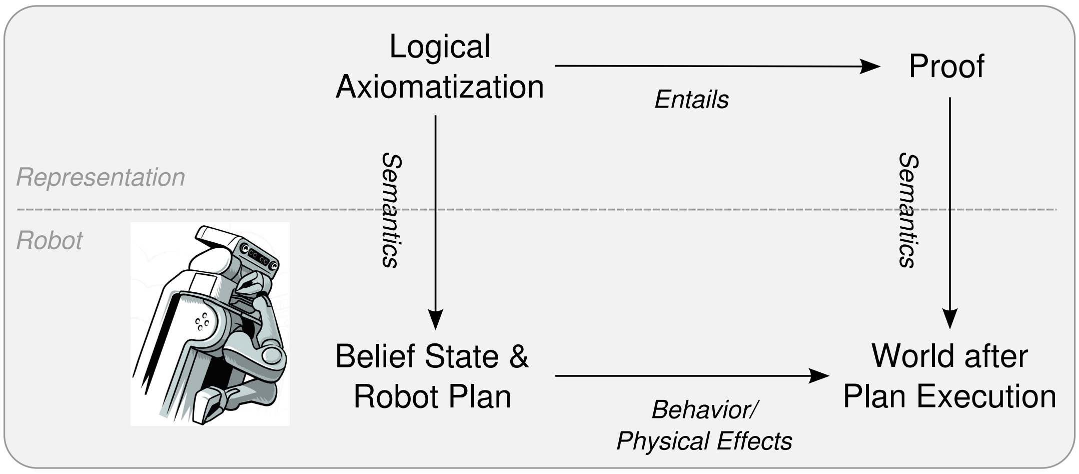

Logical Axiomatization
Entails
Proof
Representation
Robot
Semantics
Semantics
Belief State & Robot Plan
Behavior/ Physical Effects
World after Plan Execution

FIGURE 4.2 Reasoning about everyday robot manipulation. Given a belief state and a robot plan a robot reasons about its own behavior and the resulting effects.

formed under the same conditions. For example, ingredients or tools a robot has to use might differ. If the pancake mix has a higher viscosity the robot has to pour for a longer duration than usual. Of course, not all variants of a problem can be foreseen. However, to some extend a robot should be able to cope with variants without the need to extend the underlying theory.

In conclusion, reasoning components for robots have to be realized by other means than pure Logic. In the next section we will outline the principle by which we enable robots to envision the outcome of their own actions adequately.

4.2 What is Envisioning?

As described in the work of de Kleer (1977), *envisioning* denotes the kind of reasoning that predicts what might or could happen in a given situation. Thereby the reasoning considers multiple or all possible situations that could be generated from an initial situation.

Figure 4.3 shows how the underlying idea of commonsense reasoning explained in the previous section is extended. Based on the logical axiomatization, that is, a description of a manipulation scenario and a fully instantiated robot plan, a physics-based simulation is parameterized. The states of task-relevant objects and actions are monitored and their data structures are logged. These log files are interpreted and translated into interval-based first-order representations, called timelines. Eventually, logical queries of the robot can be answered based on timelines which are grounded in the logged data structures of physical simulations.

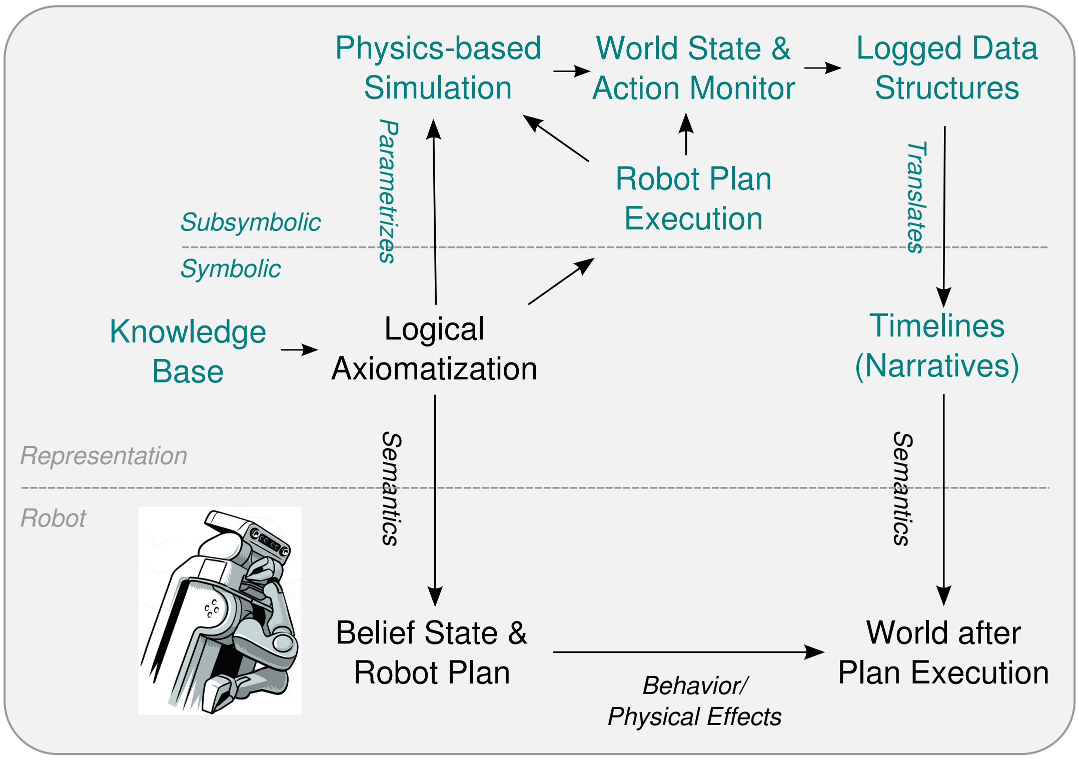
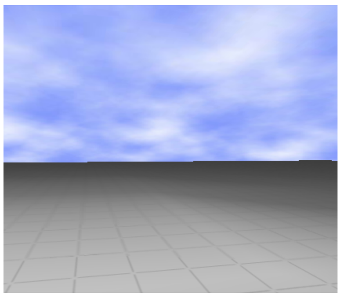
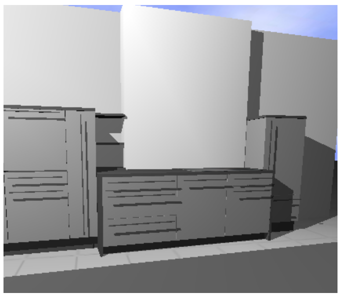
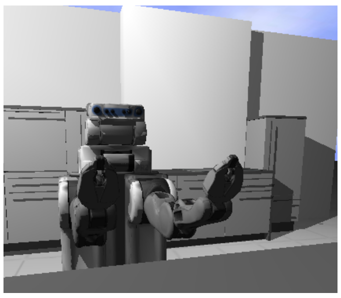
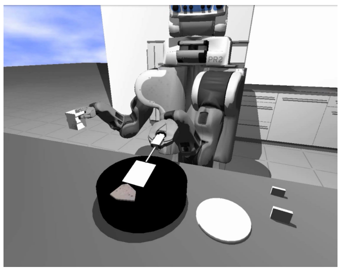
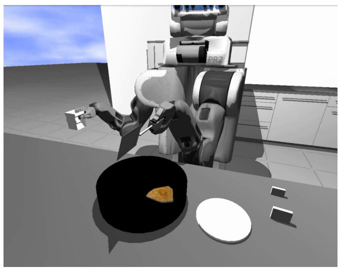


FIGURE 4.3 Envisioning of robot manipulation tasks based on physics-based simulations.

The general principle of the envisioning framework is depicted in Figure 4.4. The framework is accessed via a logic-based interface, meaning that both the framework's input and output are formalized using first-order representations. The input is a description of a situated *scenario* σ of a manipulation problem along with an parameterized *action plan* ϕ that potentially solves the problem. The output of the envisioning framework is a *timeline* τ which hold information about object states, their relationships to other objects and the performed actions of the robot. For example, a robot formalizes the problem of making pancakes by providing a minimalist description of the environment including the kitchen work space, manipulable objects like a container holding the pancake mix and a spatula and a specification of the robot itself. In addition, the robot provides an instantiated action plan based on the plan's corresponding parameter space for pouring the ready-to-use pancake mix onto the pancake maker, flipping the half-baked pancake and placing the full-baked pancake onto a plate. Finally, based on the envisioned timeline the robot is able to evaluate its parameterized action plan with respect to various performance measures, for example, whether the pancake mix was poured onto the pancake maker without spilling. In order to evaluate a set of parameterized action plans we sample parameter values from the parameter space associated with a plan.

TABLE 4.1 Envisioning process for flipping a pancake. The process steps comprise the assertion of the scenario, envisioning over a set of parameterized plans, and question answering based on timelines grounded in logged simulations.

Logic Programming Environment	**Physical Simulation**
`?- assert_scenario(Pancakes).`	
`?- assert_scenario($Pancakes,kitchen).`	
`?- assert_scenario($Pancakes,pr2).`	
`?- param_space(flip,ParamSpace), setof(TL,(member(P,ParamSpace), envision($Pancakes,flip(P),TL)), TLs).`	
`?- member(TL, $TLs), holds_tt(on(pancake,pancake_maker),I1,TL), holds_tt(on(pancake,spatula),I2,TL), holds_tt(on(pancake,pancake_maker),I3,TL), before(I1,I2),before(I2,I3), holds_tt(occurs(flip(P)),I4,TL), during(I2,I4).`	

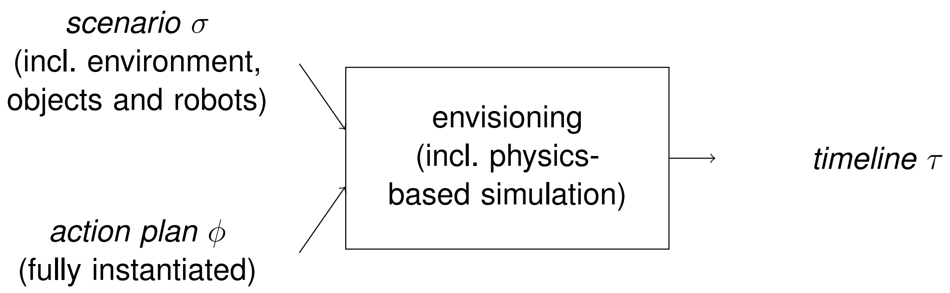

scenario σ
(incl. environment, objects and robots)
action plan ϕ
(fully instantiated)
envisioning (incl. physics-based simulation)
timeline τ

FIGURE 4.4 Input and output of the envisioning framework.

Table 4.1 illustrates the individual steps of the envisioning process by the example of flipping a pancake. The logical language allows to assert a certain scenario and initiate the envisioning process. Eventually, logical predicates such as *Holds* and *Holds*$_{\textit{Throughout}}$ ($holds\_tt$) are used to determine whether certain conditions hold within a time interval. For example, the query in Table 4.1 asks whether the pancake was first on the pancake maker, then on the spatula during the flipping action, and finally back on the pancake maker.

Having shown the envisioning process step-by-step in Table 4.1, Figure 4.5 visualizes how the process is embedded within PROLOG's backtracking mechanism. Given the depth-first search strategy of PROLOG a proof tree is generated whereby the branches correspond to different parameterizations of robot plans. Eventually, the timelines are evaluated with respect to desired and undesired effects.

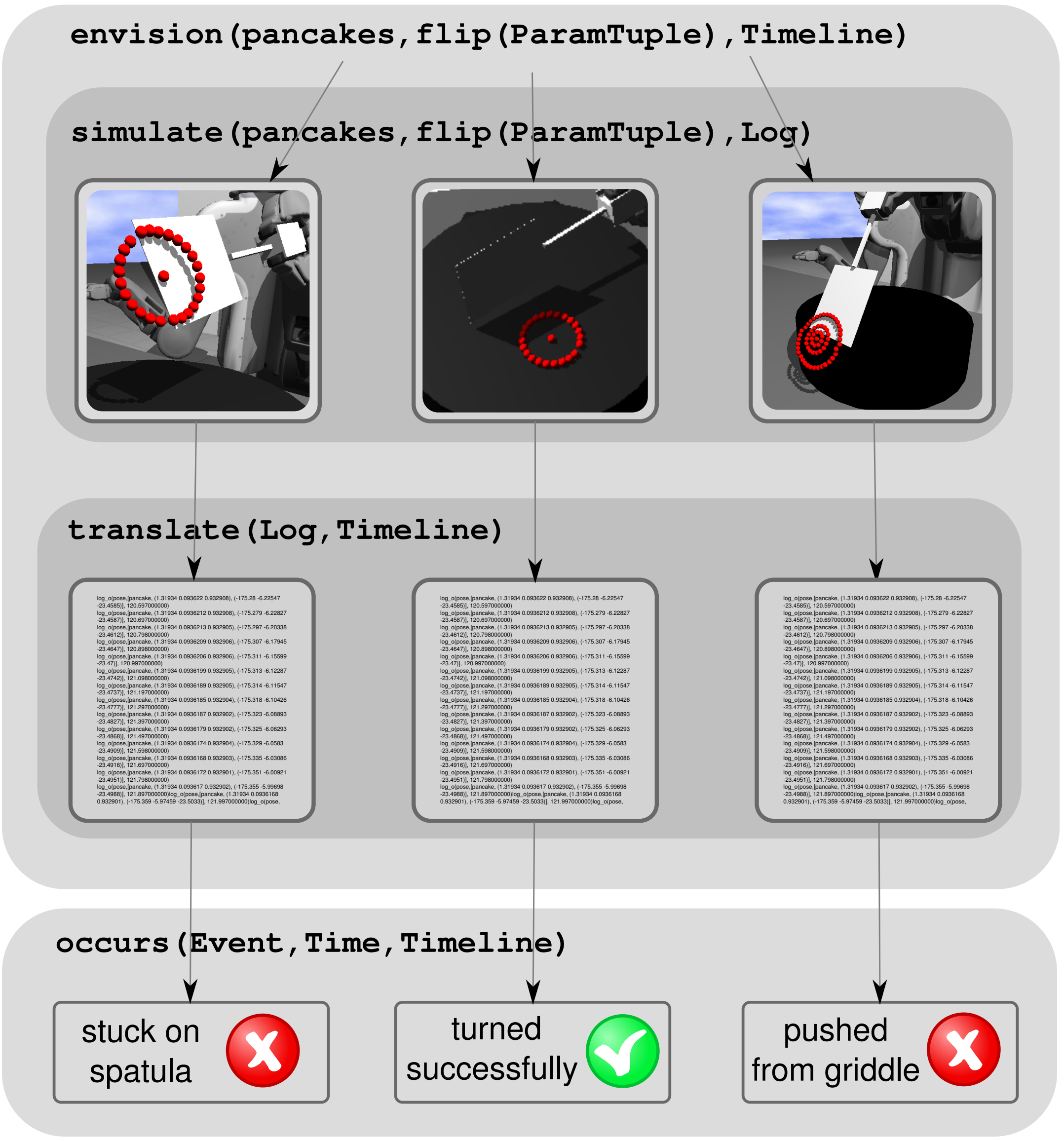
4.3 Physics-based Robot Simulator: Gazebo

As the physics-based simulator can be considered as the "workhorse" within our approach, we would like to introduce it separately, before we present the overall framework in detail.

Gazebo[1] is a multi-robot simulator which is currently under active development at the Open Source Robotics Foundation[2] and has a large user community. Moreover, the Gazebo software is the basis of the simulator used in the current DARPA robotics challenge[3].

Gazebo provides feedback from sensors and physical interactions between objects in a three-dimensional space. One of the reasons why we have chosen Gazebo is because it gives us direct control over the parameters of the underlying physics engine, namely ODE[4]. Other advantages include the direct control over world components such as ob-

[1]`http://gazebosim.org`
[2]`http://osrfoundation.org`
[3]`http://www.theroboticschallenge.org/aboutsimulator.aspx`
[4]`http://www.ode.org/`


envision(pancakes,flip(ParamTuple),Timeline)
simulate(pancakes,flip(ParamTuple),Log)
translate(Log,Timeline)
occurs(Event,Time,Timeline)
stuck on spatula
turned successfully
pushed from griddle

```
log_o(pose,[pancake, (1.31934 0.093622 0.932908), (-175.28 -6.22547
-23.4585)], 120.597000000)
log_o(pose,[pancake, (1.31934 0.0936212 0.932908), (-175.279 -6.22827
-23.4587)], 120.697000000)
log_o(pose,[pancake, (1.31934 0.0936213 0.932905), (-175.297 -6.20338
-23.4612)], 120.798000000)
log_o(pose,[pancake, (1.31934 0.0936209 0.932906), (-175.307 -6.17945
-23.4647)], 120.898000000)
log_o(pose,[pancake, (1.31934 0.0936206 0.932906), (-175.311 -6.15599
-23.47)], 120.997000000)
log_o(pose,[pancake, (1.31934 0.0936199 0.932905), (-175.313 -6.12287
-23.4742)], 121.098000000)
log_o(pose,[pancake, (1.31934 0.0936189 0.932905), (-175.314 -6.11547
-23.4737)], 121.197000000)
log_o(pose,[pancake, (1.31934 0.0936185 0.932904), (-175.318 -6.10426
-23.4777)], 121.297000000)
log_o(pose,[pancake, (1.31934 0.0936187 0.932902), (-175.323 -6.08893
-23.4827)], 121.397000000)
log_o(pose,[pancake, (1.31934 0.0936179 0.932902), (-175.325 -6.06293
-23.4868)], 121.497000000)
log_o(pose,[pancake, (1.31934 0.0936174 0.932904), (-175.329 -6.0583
-23.4909)], 121.598000000)
log_o(pose,[pancake, (1.31934 0.0936168 0.932903), (-175.335 -6.03086
-23.4916)], 121.697000000)
log_o(pose,[pancake, (1.31934 0.0936172 0.932901), (-175.351 -6.00921
-23.4951)], 121.798000000)
log_o(pose,[pancake, (1.31934 0.093617 0.932902), (-175.355 -5.99698
-23.4988)], 121.897000000)log_o(pose,[pancake, (1.31934 0.0936168
0.932901), (-175.359 -5.97459 -23.5033)], 121.997000000)log_o(pose,
```

FIGURE 4.5 PROLOG's proof tree for three different flipping plans.

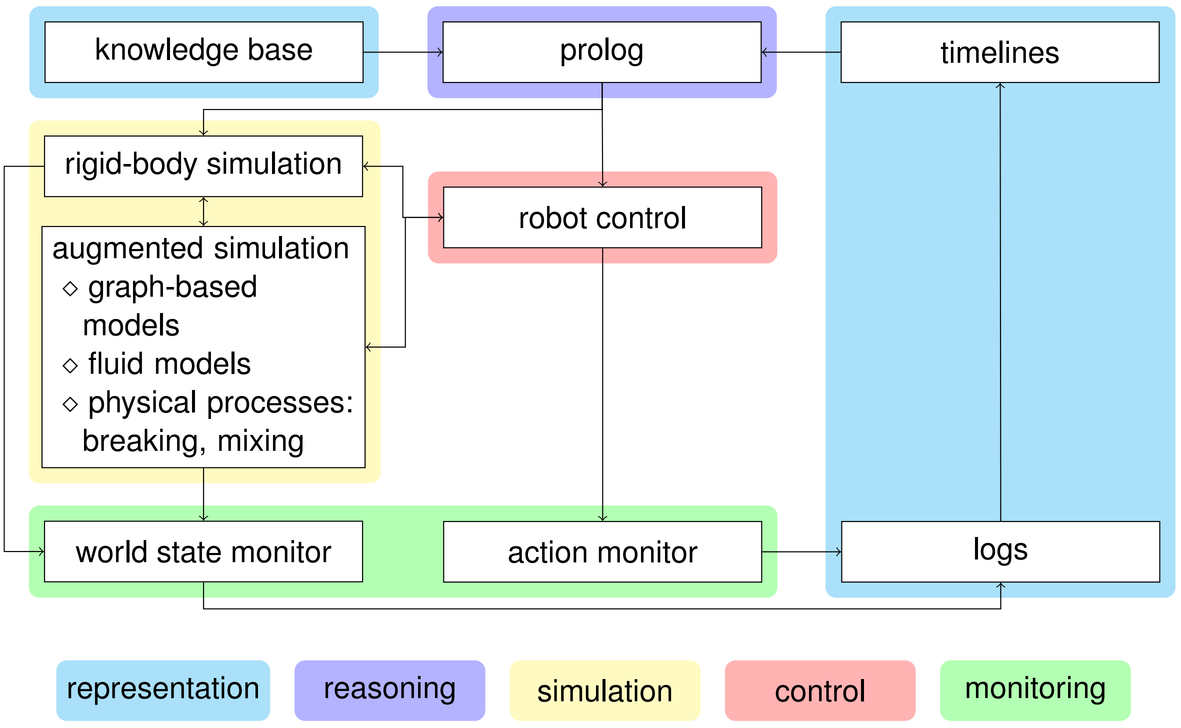


FIGURE 4.6 Envisioning framework overview.

jects and robots, access to the world and contact state information at each time step, the availability of ready-to-use object and robot models and a tight integration with the ROS[5] software framework.

4.4 Framework Design

After we have looked at the general principle of the envisioning framework and its input and output specifications in Section 4.2 let us now investigate how the envisioning functionality is achieved through the interplay of various components. Figure 4.6 shows the various components of the framework as well as their interactions among each other.

As stated earlier, the framework's interface is based on first-order representations. In our work we employ PROLOG[6] to realize the interface of the envisioning framework. Given domain knowledge (knowledge base), a scenario description and an action plan, PROLOG initializes and orchestrates the overall envisioning process and eventually evaluates the resulting timelines. One of the main constituents of the envisioning process is a physics-based simulation in which a specified robot performs manipulation actions according to its

[5]`http://www.ros.org`
[6]SWI-PROLOG: `http://www.swi-prolog.org/`

parameterized action plan. During simulation dedicated monitoring routines observe the world state, including object poses, velocities, contacts between objects, etc. Similarly, the actions of the robot are monitored and logged. After the execution of the robot control program the logs are read by PROLOG and translated into interval-based first-order representations, called timelines. Eventually, the timelines are evaluated with respect to predefined goal conditions and other performance measures. Now that we have shortly put all components of the framework into context we will explain how the individual components are realized.

4.4.1 Knowledge Base

In this work the robot runs physics-based simulations and evaluates their outcome in order to reason about a manipulation problem such as flipping a pancake. To pursue this task most effectively the robot is equipped with knowledge about situated manipulation problems, action plans that generally describe how to solve such problems, and parameter spaces of primitive actions occurring in the plans. We use first-order representations to formalize these information within a knowledge base. For representing the knowledge we mainly use Description Logic (DL), in particular the semantic web ontology language OWL[7]. We build our representations on OpenCyc's[8] upper-ontology and extend type and property descriptions whenever necessary.

Following our previous work, we represent the environment of the robot with semantic maps (Tenorth et al., 2010a). These maps describe not only the geometric properties of the environment but also the semantic categories like *cooking top* within an ontology. Similarly, everyday objects that are to be manipulated by the robot are described within the ontology. For example, a spatula is composed of an elongated handle and a blade for turning or serving food. The robot itself is specified by the Semantic Robot Description Language (SRDL) (Kunze et al., 2011c) which we introduced in Section 3.6. The description includes the kinematic structure of the robot as well as a semantic description of its sensors and actuators. All of the above mentioned descriptions have links to physical models that can be instantiated within a simulator. Action plans are represented hierarchically within the ontology as depicted in Figure 4.7.

To give the reader an idea about the physically relevant knowledge of everyday objects, their type, properties and relations to other objects let us have a closer look at an object which is subject in one of our grasping experiments: an *egg*. We consider an egg as con-

[7]`http://www.w3.org/2004/OWL`
[8]`http://www.opencyc.org`

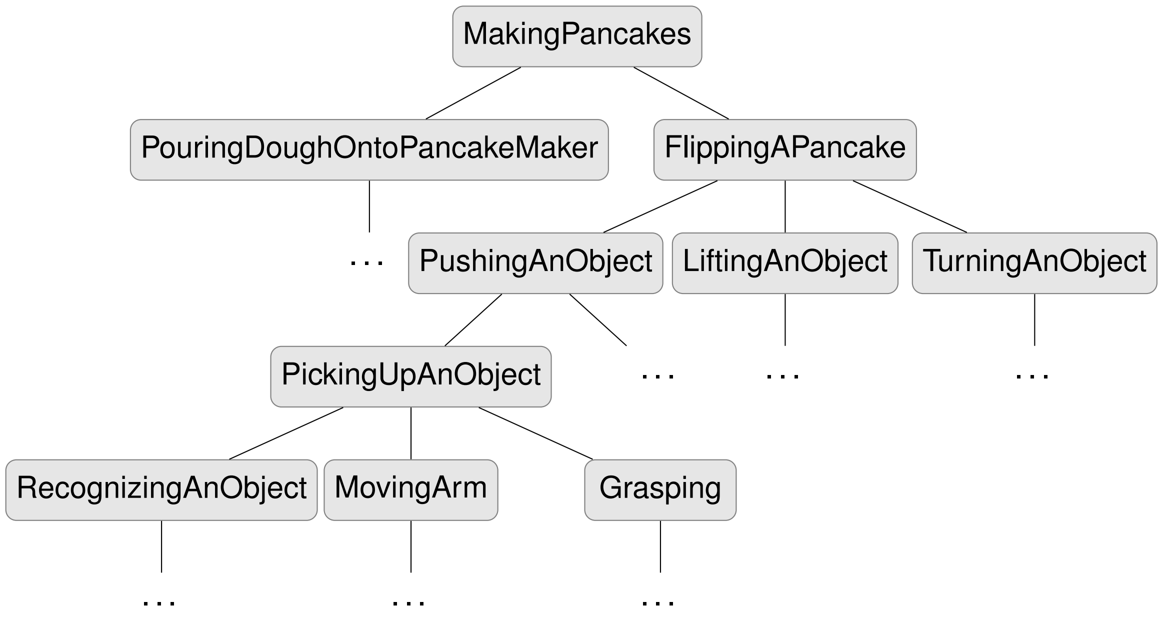


FIGURE 4.7 Ontological representation of an action plan for making pancakes.

sisting of an eggshell and its content, i.e. egg white and egg yolk. An eggshell is a solid rigid (but fragile) container that has a shape, a mass, and extensions in space. Since the eggshell is fragile it can break. The egg's content is a liquid which has a viscosity and a mass. Figure 4.8 depicts a simplified excerpt of the ontology that shows type, relation and property information about eggs. For describing a specific situation individuals of relevant objects and their properties are explicitly asserted, e.g., an individual of type *Egg*, *egg3*, has *eggshell3* and *yolk3* as its parts, where *eggshell3* and *yolk3* have a mass of 0.01 kg and 0.04 kg respectively. Other properties and relations are specified similarly. For a specific task like grasping an egg information about all relevant objects is asserted in the knowledge base. These assertions build the basis for parameterizing the physics-based simulation which is explained in Section 4.4.3.

4.4.2 PROLOG — A Logic Programming Environment

PROLOG is the heart of the envisioning framework. It serves as an interface to the robot, and coordinates all other components of the framework using a simple language for making temporal projections.

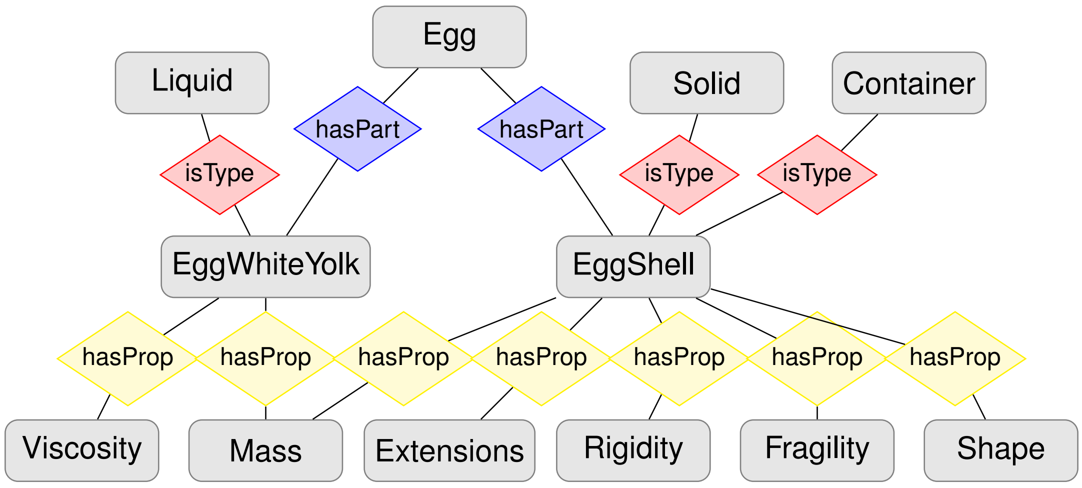


FIGURE 4.8 Ontology showing physical aspects of eggs.

4.4.2.1 Overview

In order to answer a query from the robot, PROLOG retrieves the descriptions of all task-related objects and the robot's environment from the knowledge base and sets up the simulation using these descriptions and launches a parameterized robot control program that is executed within the simulator. During the execution object states and associated data structures are monitored and logged. After the execution of the program, PROLOG stops the simulation and the object logs are translated back into first-order representations and eventually they are evaluated in regard to specified performance criteria. If the evaluation is successful, PROLOG presents a solution, otherwise it backtracks over different parameterizations. To realize this functionality we have developed a temporal projection language which is explained in the following section.

4.4.2.2 Temporal Projection Language

The basic idea of a logic programming language for making simulation-based temporal projections is as follows. First, a new scenario is asserted and task-relevant descriptions are added to it successively, for example, an environment description, a robot description, and a number of object descriptions. After initializing the simulator with the asserted scenario descriptions, a robot control program is executed whereby formal control parameters are selected from a specified range of possible values. States of the robot and objects that are traversed during simulation are monitored, logged, and translated into interval-based first-order representations, namely timelines. Eventually, the generated timelines are subject to further evaluations of specialized predicates. For example, a timeline is evaluated with respect to desired (or undesired) outcomes, qualitative spatial relations, or other performance criteria like the speed of execution.

The following PROLOG query shows exemplarily how the simulation-based temporal projection can be used where terms starting with an upper-case letter like *Scenario* denote variables, terms starting with a lower-case like *kitchen_env* denote concrete instances in the knowledge base, and the predicate *holds(occurs(event),Time,Timeline)* stands exemplarily for a specialized predicate that evaluates a given timeline:

```
?- assert_scenario(Scenario),
   assert_scenario(Scenario,kitchen_env),
   assert_scenario(Scenario,pr2_robot),
   assert_scenario(Scenario,obj1),
   param_space(actionplan,ParamSpace),
   setof(TL, (member(P,ParamSpace),
               envision(Scenario,actionplan(P),TL)),TLs),
   member(Timeline,TLs), holds(occurs(event), Time, Timeline).
```

Values for the formal parameters, e.g. *param1*, are selected from their respective ranges and are bound to the parameter variables (e.g. *P1*) in order to make them accessible for further evaluations. How to generate and/or select the values of the parameters more effectively is another interesting problem. Intuitively, the parameters could be chosen depending on the qualitative outcomes of the simulation, however, this is beyond the scope of this work.

The language elements for making temporal projections, i.e., the PROLOG programs (or predicates) that have been implemented in order to assert a scenario, to perform a simulation-based temporal projection, and to logically evaluate the resulting timelines, are explained in the following.

ASSERT_SCENARIO(SCENARIO) Asserts a new scenario and generates a unique identifier (*Scenario*) to access the scenario within other predicates.

ASSERT_SCENARIO(SCENARIO, ENTITY) Asserts an entity or set of entities to a given scenario. There are three ways of asserting an entity: either by naming the entity, if it is already known by the knowledge base including its physical specifications that are needed by the simulator, by providing the physical specifications of a previously unknown entity explicitly, or by providing an object type that can be generated by the object model factory.

ENVISION(SCENARIO, PLAN(PARAMS), TIMELINE) Performs a simulation-based temporal projection for an asserted scenario, a fully instantiated robot control program/plan, and returns an ID of the projected timeline. This program is realized by two subprograms, namely *simulate* and *translate*.

SIMULATE(SCENARIO, PLAN(PARAMS), LOG) Sets up and runs the simulation. First, Gazebo is launched and all entities that were asserted by the *assert_scenario* command are loaded successively. If necessary, entity specifications are generated on-the-fly by the object model factory and spawned into the simulator. Second, the robot control program is executed, where formal parameters are selected from their respective ranges. By utilizing PROLOG's backtracking mechanism the cross product of all valid parameter instantiations is automatically generated. After a certain time, the simulation is stopped and all processes are shut down. The output variable *Log* points to the log files of the robot control program and the task-related objects.

TRANSLATE(LOG, TIMELINE) Translates the logged simulations into a interval-based representation, that is, a timeline, by using the first-order predicates *Holds(f,t)* and *Holds(Occurs(e),t)*. To differentiate between the individual timelines a unique ID (*Timeline*) is generated and attached to the individual fluents and events.

HOLDS(OCCURS(EVENT), TIME, TIMELINE) Retrieves the given *Timeline* and evaluates it with respect to an event (*Event*) that might have occurred at a point in time (*Time*) during the simulation. If the specified event is found in the timeline the predicate evaluates to true.

HOLDS(FLUENT, TIME, TIMELINE) Retrieves the given *Timeline* and evaluates it with respect to a fluent (*Fluent*) that might have hold at a point in time (*Time*) during the simulation. If the specified fluent is found in the timeline the predicate evaluates to true.

4.4.3 Physics-based Simulation

Within our approach, we utilize a physics-based simulator, namely Gazebo[9], for computing the effects of robot actions, object interactions and other physical events. We augmented the rigid-body physics to simulate specialized behaviors.

[9]`http://gazebosim.org`

4.4.3.1 Rigid-Body Simulation

We parameterize the simulator on the basis of the logical axiomatization, i.e. the domain knowledge, run simulations and log data of features like position, velocity, forces, and contact points between objects over time. After explaining shortly how a physics-based simulator computes physical effects generally, we present how the Gazebo simulator can be configured and how we derive a configuration based on the assertions in the knowledge base.

Generally a physics-based simulator works as follows: the simulator starts its computation of physical effects based on an initial configuration. Then it periodically receives motor control commands which are translated into forces and updates the state of the simulated world according to physical laws. Within each tiny update step, forces are applied to affected objects by considering both the object's current dynamic state and its properties like mass and friction. Later we explain how we augment the simulation in order to account for physical phenomena like breaking or absorbing.

The initial configuration of the Gazebo simulator is based on an XML file, called *world file*. The world file describes properties of the simulation, specifies parameters for the physics engine (ODE) and describes all things occurring in the world, including robots, sensors and everyday objects. Within a world file each object has its own model description. Such model descriptions comprise mainly the object's shape and a set of physical properties like size, mass, and rigidity. When properties are not explicitly specified within the knowledge base, we simply assume default values.

To simulate physical phenomena like breaking objects we augment the model descriptions, how this is realized is presented in the next section.

4.4.3.2 Augmented Simulation

The Gazebo simulator is designed for simulating robots, sensors and objects, whereby physical aspects of objects and their interactions are more or less limited to rigid body dynamics. Since we want to simulate naive physics problems with phenomena like breaking, mixing, and cooking we augment object model descriptions with detailed shape models, controllers for simulating physical phenomena, and monitors for logging states of objects. The extended model descriptions are collected in a library for simulating phenomena of everyday physics.

Instead of modeling objects as rigid bodies, we describe the shape of objects similar to work by Johnston and Williams (2008) with graph-based structures which allow us to

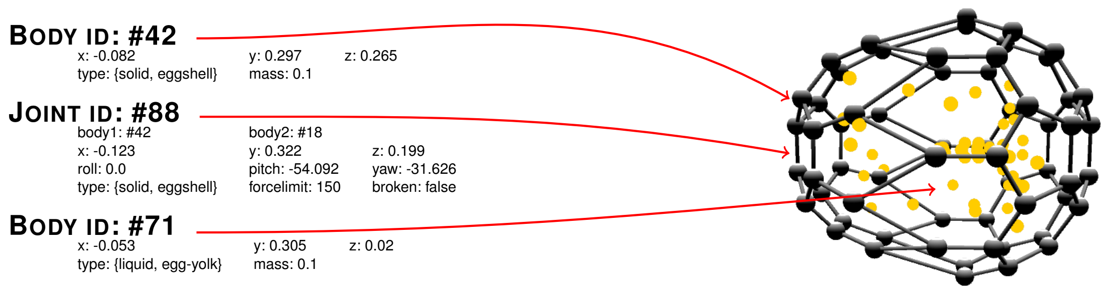

BODY ID: #42
x: -0.082 y: 0.297 z: 0.265
type: {solid, eggshell} mass: 0.1
JOINT ID: #88
body1: #42 body2: #18
x: -0.123 y: 0.322 z: 0.199
roll: 0.0 pitch: -54.092 yaw: -31.626
type: {solid, eggshell} forcelimit: 150 broken: false
BODY ID: #71
x: -0.053 y: 0.305 z: 0.02
type: {liquid, egg-yolk} mass: 0.1

FIGURE 4.9 Modeling shape and physical properties of an egg. The shape of the egg is modeled with a graph-based structure of bodies which are linked by joints. The physical properties of these individual bodies and joints, which are shown exemplarily on the left side, determine the physical properties of the whole egg, e.g. its mass and fragility.

inspect physical aspects at a more detailed level. Figure 4.9 visualizes the shape of an eggshell with egg yolk inside. These models configurations are derived from the information stored in the knowledge base. The basic entities for modeling the shape of an object are *bodies*[10] and *joints*, which are mutually connected. Properties of an object like type, mass, spatial extensions, and rigidity determine the attributes of these basic entities.

In order to simulate new classes of objects, for example, objects that are breakable and objects that change their state from liquid to a deformable structure we add controllers to the object model descriptions. These controllers are called within each simulation step and perform some specialized computation. The computation can be based on physical properties calculated by the simulator or on results computed by other controllers. Thereby object attributes like being broken and being cooked can be computed. This allows us to simulate a new range of processes like breaking and cooking.

Given that there is only a limited number of processes that have to be implemented makes this approach scalable. Furthermore, the implementation of the various continuous processes with procedural programs is easier, than their realization by the means of logical axioms. In Chapter 5, we explain in detail how fluids are represented and simulated within the framework.

4.4.4 Monitoring of Simulations and Actions

In addition to controllers realizing physical behaviors, we add monitoring routines to observe and log the state of objects at each simulation step. Additionally we monitor the actions the robot is performing.

[10] *Bodies* are called *links* in URDF.

TABLE 4.2 Excerpt of the action log. Example: Pick-up action.

Time	ID	Status	Action	Parameter
53.917	**12**	**begin**	**pick_up**	(spatula_handle, left_arm)
53.917	13	begin	open_l_gripper	(width, 0.09)
56.844	13	end	open_l_gripper	–
56.959	14	begin	move_l_arm	(l_wrist_flex_link, (position, (0.310885, 0.490161, -0.195159)), (orientation, (-2.73593e-05, 0.013794, 0.00194869, 0.999903)))
59.175	14	end	move_l_arm	–
59.200	15	begin	move_l_arm	(l_wrist_flex_link, (position, (0.380885, 0.490161, -0.195159)), (orientation, (-2.73593e-05, 0.013794, 0.00194869, 0.999903)))
60.573	15	end	move_l_arm	–
60.590	16	begin	close_l_gripper	(width, (0.0))
72.020	16	end	close_l_gripper	–
72.063	17	begin	move_l_arm	(l_wrist_flex_link, (position, (0.380885, 0.490161, 0.00484052)), (orientation, (-2.73593e-05, 0.013794, 0.00194869, 0.999903)))
74.307	17	end	move_l_arm	–
74.315	**12**	**end**	**pick_up**	–

Actions of the robot are monitored as follows. Ideally, robot control programs would be written as plans. For example, using a plan language like CRAM (Beetz et al., 2010) allows robots to interpret and reason about their own programs. However, in this work we treat a robot control program as black box meaning that it can be implemented in any kind of programming language. In order to reason about the actions of a robot we assume that at least the actions of interest are logged using a simple interface. Basically, the begin and the end of an action as well as its parameters should be logged by the control program. This allows robots to related their actions to the physical events of the simulation. An example of an action log for picking up a spatula using the left robot arm is shown in Table 4.2. In the excerpt of the log the hierarchical decomposition of actions and sub-actions is visible. The *pick_up* action is decomposed into several action primitives including opening and closing the gripper, and moving the arm's end-effector into certain pose.

The data structures of the world state we are monitoring are basically the position, orientation, linear and angular velocities, and the bounding boxes of objects and their

TABLE 4.3 Excerpt of the world state log.

Time	Object	Position	Orientation	Linear velocity	Angular velocity
72.70	blade	(1.32,-0.04,0.90)	(-0.00,0.02,-0.00,0.99)	(0.00,0.01,0.09)	(-0.18,0.03,0.01)
72.80	blade	(1.32,-0.03,0.91)	(-0.00,0.02,-0.00,0.99)	(0.01,0.00,0.09)	(0.16,0.09,-0.10)
72.90	blade	(1.32,-0.03,0.92)	(-0.00,0.02,-0.00,0.99)	(0.00,0.00,0.12)	(0.41,0.06,-0.09)
...	...	...	...	...	...
74.30	blade	(1.32,-0.04,1.08)	(0.00,0.03,0.00,0.99)	(0.00,-0.00,-0.00)	(-0.04,0.00,0.01)

TABLE 4.4 Excerpt of the contact log.

Time	1st Object	2nd Object	Total force	Total torque	Num. of Contacts
67.00	spatula_handle	l_gripper_r_finger	(-0.17, 0.75, -0.21)	(0.00, -0.01, -0.06)	9
68.00	spatula_handle	l_gripper_r_finger	(0.32, -2.37, -0.46)	(0.00, -0.02, 0.18)	12
68.00	spatula_handle	l_gripper_l_finger	(-0.44, -1.55, -2.02)	(-0.00, -0.17, 0.11)	14

respective parts. Furthermore, we observe the physical contacts between objects and log information such as contact points, contacts normals, and forces. All these information are constantly monitored and only changes are logged. Table 4.3 and Table 4.4 show excerpts of the world state and the contact log respectively. Both excerpts are taken from the interval during which the pick-up action (see above) took place. Whereas Table 4.3 report the world state information for the blade of the spatula, Table 4.4 reports contact information between the handle of the spatula and the robot gripper.

4.4.5 Timelines

In this section we explain how we ground first-order representations in logged data structures of the simulator. Before we explain how log files are translated into logic, we will present the representation formalism for temporal knowledge and shortly discuss its relation to domain knowledge.

For representing temporal knowledge, i.e. object configurations and events at given time points, we make use of notations common in the *event calculus* (Kowalski and Sergot, 1986a) and its extensions. In the following we present predicates relevant for temporal reasoning.

The notation is based on two concepts, namely fluents and events. Fluents are conditions that change over time, e.g., a cup contains coffee: *contains(cup,coffee)*. Events (or actions) are temporal entities that have effects and occur at specific points in time, e.g., consider the action of pouring coffee from a pot into a cup: *pourTo(coffee,pot,cup)*. Logical statements about both fluents and events are expressed mainly by two predicates:

TABLE 4.5 Fluents for static physical configurations.

fluent	intuitive description
contacts(o_1, o_2)	object o_1 and object o_2 contact each other
attached(o_1, o_2)	object o_2 is attached to object o_1
supports(o_1, o_2)	object o_1 supports object o_2
contains(o_1, o_2)	container o_1 contains object (or stuff) o_2
broken(o_1)	object o_1 is broken
spilled(o_1)	object o_1 is spilled

- *Holds(f,t)* and
- *Occurs(e,t)*,

where *f* denotes a fluent, *e* denotes an event and *t* simply denotes a point in time. The statement *Holds(f,t)* represents that fluent *f* holds at time *t*, whereas *Occurs(e,t)*[11] represents an occurrence of event *ev* at time *t*. Although fluents and events look as if they were predicates themselves, they are not: both fluents and events are reified as functions returning respective instances. Thus, by treating them as 'first-class citizens' in a first-order representation allows us to state at what points in time they hold or occur.

The relation of domain and temporal knowledge is straight forward. Domain knowledge, in particular the assertions about individual objects, characterize the initial conditions for the temporal reasoning. From the temporal reasoning point of view, the assertional knowledge holds at time point *0.0*, i.e. *Holds(f,0.0)*. That the assertional knowledge describes the initial conditions for the temporal reasoning perfectly makes sense since it is also used for parameterizing (or initializing) the simulation as we explained earlier.

Logged simulations are translated into interval-based timeline representations by using extended versions of the predicates *Holds* and *Occurs*. We have added an additional variable for distinguishing between the individual timelines. That is, the predicates become *Holds(f,t,tl)* and *Holds(occurs(e),t,tl)* respectively where *tl* denotes a timeline. Whenever a fluent or event is recognized an instance of its corresponding type is generated and either the *Holds* or the *Occurs* predicate is asserted for the observed timepoint. We reuse a generated instance only if the fluent or event is also valid in successive timesteps. Thereby we get an interval-based representation of timelines. Table 4.5 and Table 4.6 list examples of implemented fluents and events for which we assert predicates from the logged simulations.

[11]Please note, in the context of this thesis we use *Holds(occurs(e),t)* instead of *Occurs(e,t)*

TABLE 4.6 Fluents for physical events.

fluent	intuitive description
moving(o_1)	object o_1 is moving
openingGripper(o_1)	robot is opening gripper o_1
closingGripper(o_1)	robot is closing gripper o_1
breaking(o_1)	object o_1 is breaking

How fluents and events are grounded in the data structures of the simulator is exemplarily explained for the fluents *contacts*(o_1, o_2) and *supports*(o_1, o_2) and the events *moving*(o_1) and *breaking*(o_1).

A contact between objects is directly reported by the simulator:

$$\begin{aligned}&\mathit{Holds}(\mathit{contacts}(o_1, o_2), t_i, tl) \Leftrightarrow\\ &\quad \mathit{Collisions} = \mathit{SimulatorValue}(\mathit{Collisions}, t_i, tl) \wedge\\ &\quad \mathit{Member}(\langle o_1, o_2\rangle, \mathit{Collisions})\end{aligned}$$

Object o_1 supports an object o_2 when there exists a contact between both objects and the maximum value of o_1's bounding box within z-dimension is slightly less or equal than the minimum value of o_2's bounding box and o_2's center of mass lies within the spatial extensions of object o_1 regarding the x-y-dimensions. The later condition is captured by the *isDirectlyBelow* predicate. Furthermore the gravity force of o_2 has to be canceled out:

$$\begin{aligned}&\mathit{Holds}(\mathit{supports}(o_1, o_2), t_i, tl) \Leftrightarrow\\ &\quad \mathit{Holds}(\mathit{contacts}(o_1, o_2), t_i, tl) \wedge\\ &\quad p_1 = \mathit{SimulatorValue}(\mathit{Pose}(o_1), t_i, tl) \wedge\\ &\quad p_2 = \mathit{SimulatorValue}(\mathit{Pose}(o_2), t_i, tl) \wedge\\ &\quad \mathit{isDirectlyBelow}(p_1, p_2) \wedge\\ &\quad \mathit{gravityForceIsCanceledOut}(o_2)\end{aligned}$$

An object o_1 is moving when its pose has changed between two successive timesteps:

$$\begin{aligned}&\mathit{Holds}(\mathit{occurs}(\mathit{moving}(o_1)), t_i, tl) \Leftrightarrow\\ &\quad p_1 = \mathit{SimulatorValue}(\mathit{Pose}(o_1), t_i, tl) \wedge\\ &\quad p_2 = \mathit{SimulatorValue}(\mathit{Pose}(o_1), t_j, tl) \wedge\\ &\quad \mathit{previousTimestep}(t_j, t_i) \wedge\\ &\quad p_1 \neq p_2\end{aligned}$$

An object o_1 is breaking in timestep t_1 when one of its joints is detached within that timestep. The controller that realizes the breaking phenomenon of objects directly reports which joints are detached in a timestep:

$$\begin{aligned}&\mathit{Holds}(\mathit{occurs}(\mathit{breaking}(o_1)), t_i, tl) \Leftrightarrow\\ &\quad j_1 = \mathit{SimulatorValue}(\mathit{Detached}(\mathit{joint}_1), t_i, tl) \land\\ &\quad \mathit{Member}(j_1, \mathit{GetJoints}(o_1))\end{aligned}$$

The next section illustrates in various experiments how robots can interpret and evaluate the grounded timelines with respect to desired and undesired outcomes.

4.5 Experiments

For showing the feasibility of our approach we have conducted several robot manipulation experiments including the problem of making pancakes as described in Section 1.2. In these experiments we addressed the requirements posed in the problem formulation.

4.5.1 Cracking an Egg

Besides *making pancakes*, *cracking an egg* is another interesting problem with respect to both commonsense reasoning and robot manipulation and has been proposed by (Davis, 1997) as a challenge problem for logical formalization and reads as follows:

> *"A cook is cracking a raw egg against a glass bowl. Properly performed, the impact of the egg against the edge of the bowl will crack the eggshell in half. Holding the egg over the bowl, the cook will then separate the two halves of the shell with his fingers, enlarging the crack, and the contents of the egg will fall gently into the bowl. The end result is that the entire contents of the egg will be in the bowl, with the yolk unbroken, and that the two halves of the shell are held in the cook's fingers."*

Solutions to this problem should not only characterize aspects mentioned above but also account for variants of the problem:

> *"What happens if: The cook brings the egg to impact very quickly? Very slowly? The cook lays the egg in the bowl and exerts steady pressure with his hand? The cook, having cracked the egg, attempts to peel it off its contents like a hard-boiled egg? The bowl is made of looseleaf paper? of soft clay? The bowl is smaller than the egg? The bowl is upside down? The cook tries this procedure with a hard-boiled egg? With a coconut? With an M & M?"*

The cracking an egg problem poses many challenges, especially in the context of everyday robot manipulation. In order to solve it we regard the following aspects to be substantial: First, the abstraction level of a formalization should reflect the sensing and acting capabilities of the manipulating robot. Second, variants should be handled without the need of explicit modeling. And third, concurrent actions and events should be taken into account.

Cracking an egg against another object and then separating (splitting) it requires a robot to be able to grasp an egg at all. Therefore we start our experiments with a scenario where a robot is supposed to simply grasp an egg lying on a table.

The first experiment consists of several trials in which a robot, in this case the PR2, is using different values of gripper force to grasp an egg. The experiment underlines the importance of a physical simulation since it allows to determine an appropriate force for grasping an egg which would not be possible by pure symbolic reasoning.

A robot would not know that an egg will break if it is grasped with a force that is too high, that it will fall down if it is grasped with a force that is too low, and that it will slip away if it is grasped at an inappropriate position.

We setup a grasping experiment where the parameters force and position were varied. A scenario (*grasping*) was asserted and entities like kitchen environment, PR2 robot, and an egg were added. The following code was used for running the experiments and evaluating their respective results:

```
?- assert_scenario(grasping),
   assert_scenario(grasping,kitchen),
   assert_scenario(grasping,pr2),
   assert_scenario(grasping,egg).
?- envision($grasping,
            grasp_object([(P1, force, [1, 3, 5, 7, 9, 11]),
                          (P2, position, [-0.02, 0.00, 0.02])]),
            Timeline),
   not(occurs(break,_,Timeline)),
   not(occurs(slip,_,Timeline)).
```

The qualitative results of this experiment are shown in Table 4.7, where the grasping position is measured as the center of the gripper pad relative to the center of the egg (in meters) and the different grasping forces are indicated by the identifiers *f1–f11*. During the experiments we found three possible outcomes (see Figure 4.10): the egg slips out of the robot's gripper, the egg is held by the robot successfully, and the egg is crashed by the

TABLE 4.7 Qualitative results: Grasping an egg.

position	f1	f3	f5	f7	f9	f11
-0.02	slip	ok	slip	ok	break	break
0.00	slip	ok	ok	ok	ok	break
0.02	slip	ok	ok	ok	break	break

robot (Figure 4.11). The different outcomes are denoted as *ok* if the robot could grasp the egg successfully, as *slip* if the egg slipped away, and as *break* if the the egg was broken within the trial.

In the second experiment we used a valid grasping force to pick up an egg and test its behavior when hitting it against obstacles and tables. The egg is picked up and then is hit or pressed against an obstacle. The results here are of course dependent on the forces that affect the egg model: while hitting the egg against another object very gently would not break it, hitting it stronger or pressing it firmly against the table would produce breaking. Figure 4.12 shows the egg model being cracked after being hit against an obstacle. This experiment can be used to gather information on how to safely manipulate such a fragile object and how the robot's actions influence the forces applied to the object.

The last experiment focused on egg splitting. The robot is grasping the egg from the table that is lying on the table and, after hitting it against an obstacle and cracking it, it is trying to split it using his other gripper (Figure 4.13). This experiment was not entirely successful as the egg to be too fragile for the PR2 grippers. The result of most of the trials involved in this experiment was the cracked egg being completely crashed by the two grippers.

In contrast to the logical formalization that has been proposed by (Morgenstern, 2001) our approach is able to make temporal projections about almost all aspects of the problem specification and its variants. Their theory (Morgenstern, 2001) is based on roughly 70 axioms, but variations as follows cannot be handled without further extensions.

- the cook brings the egg to impact very quickly or very slowly
- the bowl is upside down
- the cook tries the procedure with a hard-boiled egg, coconut, or an M&M
- the cook puts the egg in the bowl and exerts steady pressure with his hand
- the cook, having cracked the egg, attempts to peel it off its contents like a hard-boiled egg

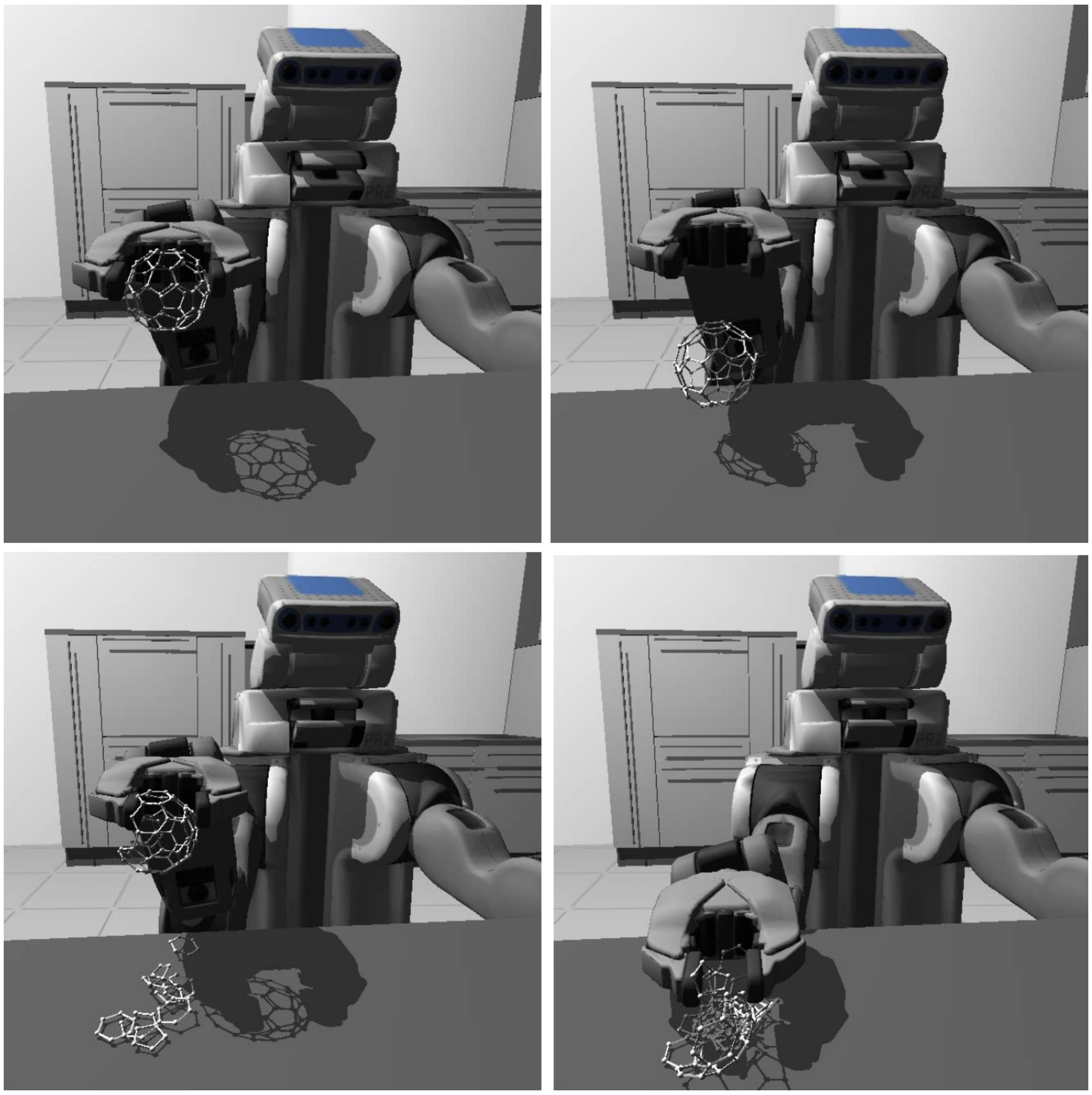


FIGURE 4.10 PR2 robot picking up an egg with different force levels (upper left: successful; upper right: egg slipping; bottom left: egg crashed, parts of the eggshell fell onto the table; bottom right: egg crashed).

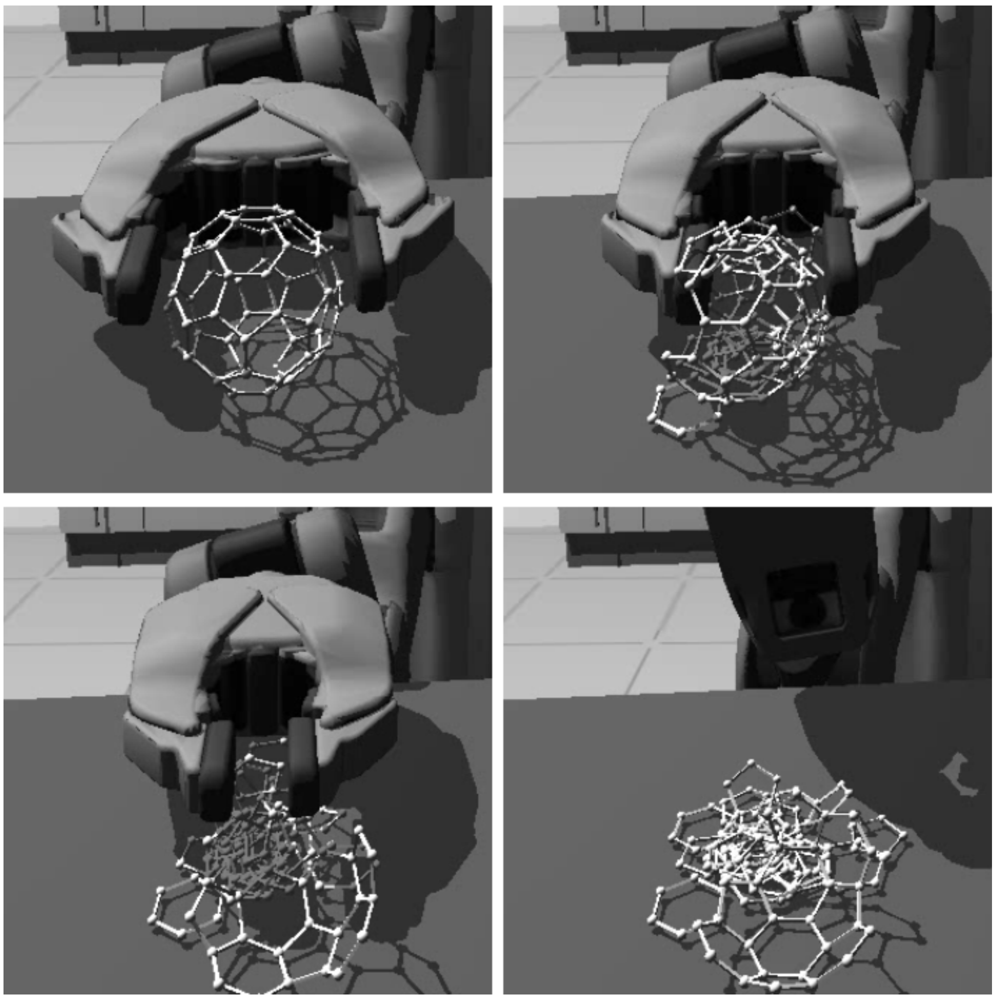


FIGURE 4.11 The egg model crashing in a grasp trial because of too much gripper force.

All these variations, except the last, seem to be feasible with our simulation approach. We simply have to adapt the robot control program to induce a different manipulation behavior, to change the configuration of the environment, or to adjust the physical parameters of the object models, e.g. size, structure and/or fragility of objects. Although the adjustment of the physical parameters is not trivial, it seems to be much easier than the extension of a logical theory since machine learning techniques can be applied for finding the appropriate physical models.

Figure 4.14 shows the robot moving its arm away from the egg after breaking it. In the beginning, parts of the eggshell stuck to the gripper and fell off at a later point in time. It is impossible to model such phenomena within logical abstractions, whereas in detailed simulations they simply emerge from the laws of physics. This example strongly emphasizes the benefit of combining first-order representations with physical simulations.

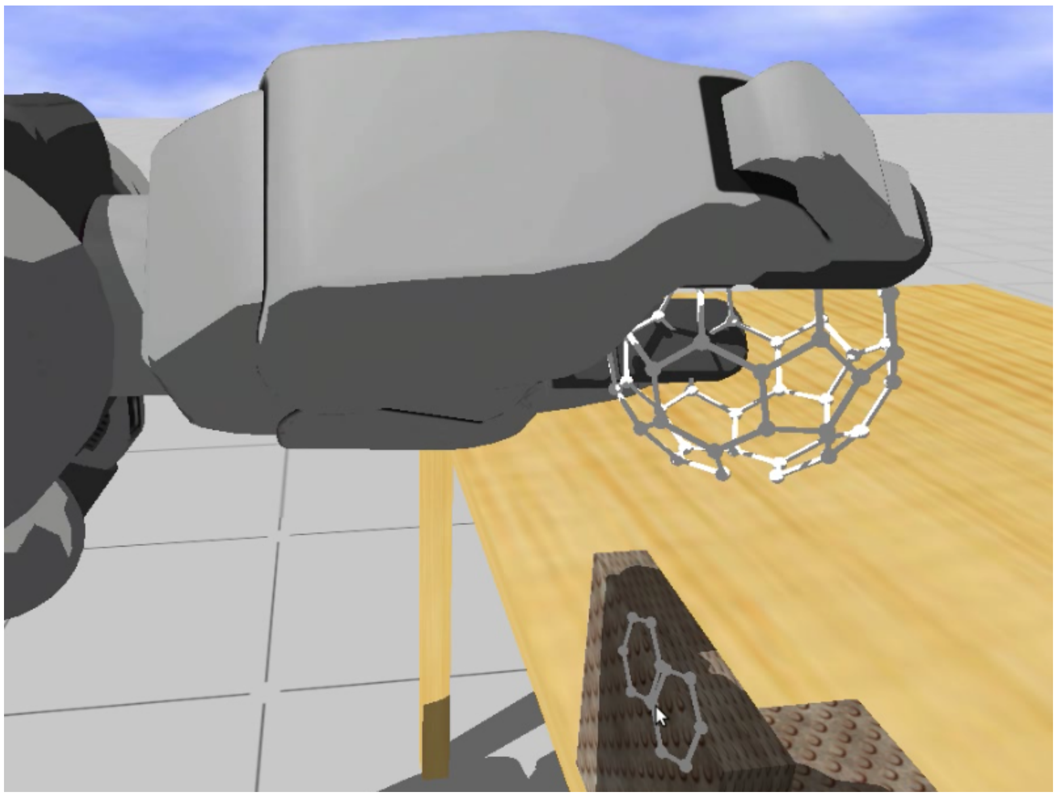


FIGURE 4.12 An egg model cracked by hitting an obstacle.

4.5.2 Making Pancakes

4.5.2.1 Pouring a Pancake Mix

In a recent experiment[12] two robots jointly performed the task of making a pancake (Beetz et al., 2011). Following this experimental setup we conducted an experiment where a robot is supposed to pour a certain amount of pancake mix onto a pancake maker in order to determine the right parameter values for the pouring position, angle and duration. Depending on these values the mix is either poured onto the pancake maker or spilled onto the table. Furthermore, if the pouring duration is too short, it will not be enough pancake mix, but if the duration is too long, it will be too much mix on the pancake maker. So, finding the right parameters is essential for making a proper pancake.

For simulating a liquid in a rigid body simulator as Gazebo, a model constructed out of small particles with an attached controller can be used. The controller has to regulate the particles' behavior, so that the model's behavior is close to the liquid's. In our experiments we used models containing a high number of small spheres, with an attached controller

[12]Making a Pancake: `http://bit.ly/fNB6I5`

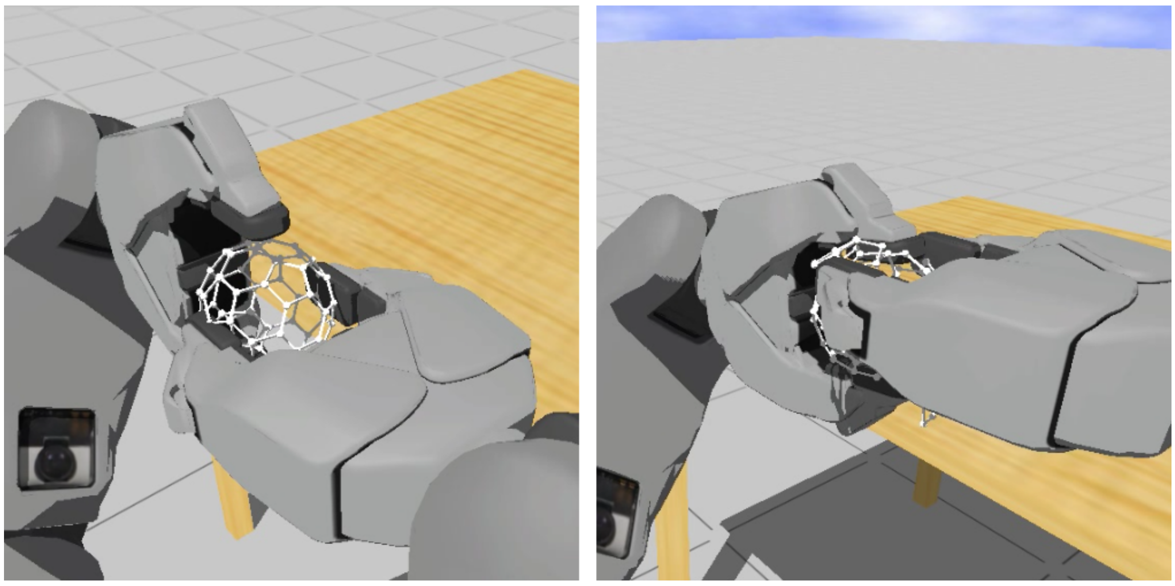


FIGURE 4.13 An egg splitting trial using the PR2 robot model.

that applies forces to these spheres to simulate the viscosity of the pancake mix. The particles tend to stay together and have high friction coefficients. Another task that the controller needs to address is logging at each timestep the state of each particle of the liquid: with which models is it in contact, and is it free falling or supported by an external body. Figure 4.15 shows different moments from a pouring action performed in the simulator by the PR2 robot.

After setting up the scenario the following code was used to run the experiments which results are shown in Table 4.8.

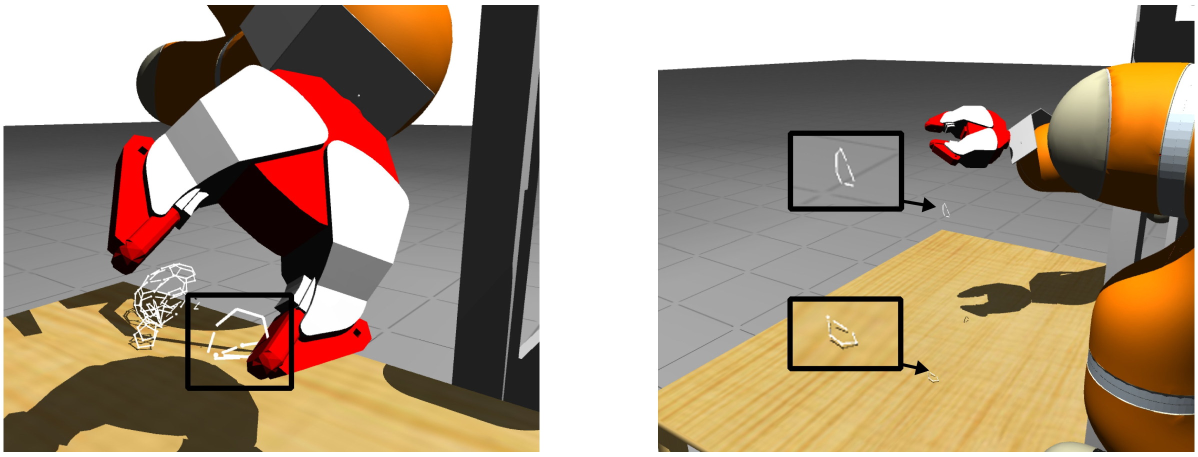


FIGURE 4.14 Grasping an egg. (1) eggshell stuck to gripper (2) eggshell fell off the gripper.

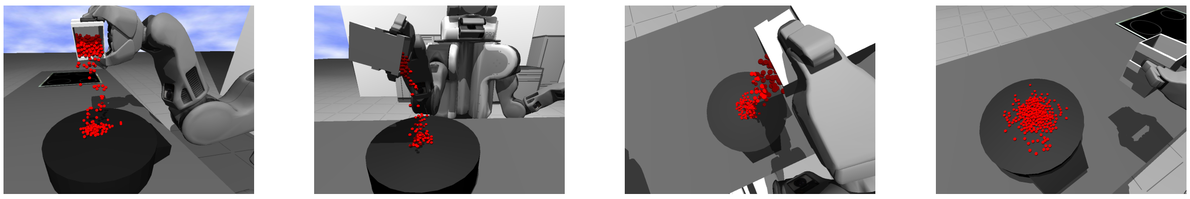


FIGURE 4.15 The simulated pancake mix, made out of small spherical particles is poured onto the pancake maker.

```
?- envision(pouring,
         pour_mix([(P1, position, [-0.09,0.0,0.07]),
                   (P2, time,     [0.0,0.5,1.0]),
                   (P3, angle,    [1.84,1.99,2.14])]),
         Timeline),
   not(holds(amount20,_,Timeline)),
   not(occurs(spill,_,Timeline)).
```

In addition to the evaluation of timelines with respect to a certain amount of pancake mix and to an occurrence of a spilling event further criteria can be evaluated on the data structures of the logged simulations. For example, qualitative spatial predicates can evaluate whether the mix was poured to the center of the pancake maker, or close its edge. Furthermore, predicates can assess shape of the poured pancake mix, e.g. its roundness. The evaluation of qualitative spatial relations is possible because logical predicates can directly be grounded in the simulator's data structures.

In the following we explain the calculations that are performed in order to evaluate the predicate *round* based on the data structures of the logged simulations.

For this, at the end of each pouring experiment, the positions of the particles that are found on the pancake maker are used for determining the centroid of the particles dataset. The centroid is considered to be the position of the poured pancake mix and is then used to determine the offset of the pancake relative to the pancake maker's center. The shape of the poured mix is evaluated using Principal Component Analysis (PCA). This is done by computing the covariance matrix of the particles dataset and afterwards by performing the eigen decomposition of this matrix. Naturally, this returns three eigen value – eigen vector pairs. The smallest eigen value is going to be a measure of the thickness of the poured pancake mix and its corresponding vector is perpendicular to the pancake maker's surface, while the other two values express the variance of the particles dataset in the other two dimensions, along their corresponding vectors. The ratio of these values, with the biggest one as denominator, gives us a good measure of the dataset's *roundness*: a

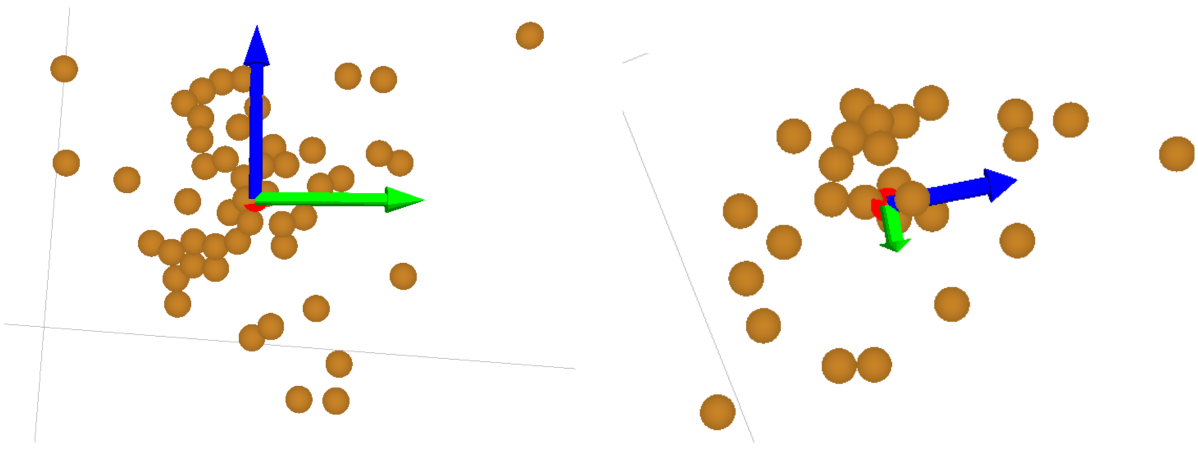


FIGURE 4.16 PCA of particle configurations on the pancake maker. *Roundness* of the area left: 0.96 and right: 0.36.

value close to zero of this ratio describes a highly elliptical shape, and for values close to 1, the shape approaches a circle. Results of the PCA for two datasets are shown in Figure 4.16.

4.5.2.2 Flipping a Pancake

The third experiment also follows the "Making Pancakes" scenario. We investigated the problem of flipping a half-baked deformable pancake using a spatula at different angles.

The simulation model used in this scenario is built out of small spherical particles connected by flexible joints. This enables the model to have the behavior of a soft deformable body (Figure 4.18). During this scenario, the PR2 robot uses a spatula to flip a pancake on the pancake maker. The parameter of interest in this case is the angle of the spatula with the horizontal and the outcome can either be failed if a pancake flip is not achieved or successful if it is. The qualitative results of the experiments performed in this scenario are shown in Table 4.9. The experiments were launched using the following query:

```
?- envision(flipping,
            flip_pancake([(P1,angle,[0.1,0.3,0.4,0.5,0.7,0.9])]),
            Timeline).
```

Figure 4.17 visualizes the temporal relations between the *on* fluent and the flipping action. Generally, all 13 possible temporal relations between time intervals can be used to constrain the query (Allen, 1983).

The following PROLOG query was used to evaluate whether the flipping action was successful or not:

TABLE 4.8 Results: Pouring a pancake mix.

pos[1]	time[2]	angle[3]	pan[4]	spill[5]	off$_x$[6]	off$_y$[7]	round[8]
-0.09	0.0	1.84	2	0	-0.004	-0.098	0.00
		1.99	25	2	-0.028	-0.096	0.19
		2.14	106	6	-0.011	-0.095	0.55
	0.5	1.84	14	1	-0.021	-0.094	0.14
		1.99	95	6	-0.005	-0.087	0.89
		2.14	205	16	0.005	-0.086	0.51
	1.0	1.84	57	1	-0.007	-0.091	0.37
		1.99	163	6	0.000	-0.083	0.58
		2.14	247	28	0.000	-0.085	0.59
0.00	0.0	1.84	1	0	0.008	-0.110	—
		1.99	41	0	-0.025	-0.004	0.51
		2.14	100	1	-0.015	-0.003	0.77
	0.5	1.84	19	0	-0.010	-0.005	0.19
		1.99	81	0	-0.022	-0.004	0.57
		2.14	212	0	-0.009	-0.003	0.89
	1.0	1.84	36	1	-0.016	0.000	0.57
		1.99	132	0	-0.015	0.001	0.57
		2.14	255	2	-0.019	0.000	0.88
0.07	0.0	1.84	0	0	—	—	—
		1.99	44	3	-0.017	0.067	0.59
		2.14	117	1	-0.002	0.062	0.76
	0.5	1.84	24	1	-0.007	0.054	0.39
		1.99	109	2	-0.012	0.074	0.62
		2.14	222	3	-0.011	0.062	0.90
	1.0	1.84	40	0	-0.012	0.071	0.85
		1.99	156	4	-0.002	0.068	0.79
		2.14	287	5	0.001	0.070	0.76

Abbreviations

[1] Offset of cup position relative to the center of the pancake maker.
[2] Duration that the cup is maintained at the pouring angle.
[3] Angle of the cup's inclination in radians (e.g.: 0 = cup upright; 1.57 = cup tilted 90 degrees counterclockwise).
[4] Number of particles found on the pancake maker
[5] Number of particles spilled onto the table.
[6] X-offset in meters of particles' centroid relative to pancake maker.
[7] Y-offset in meters of particles' centroid relative to pancake maker.
[8] Degree of roundness. Ratio of first two eigenvalues of particles on pancake maker (range: 0.0–1.0, values closer to 1 mean more round).

TABLE 4.9 Qualitative results: Flipping a pancake.

angle	0.1	0.3	0.4	0.5	0.7	0.9
	ok	ok	ok	fail	fail	fail

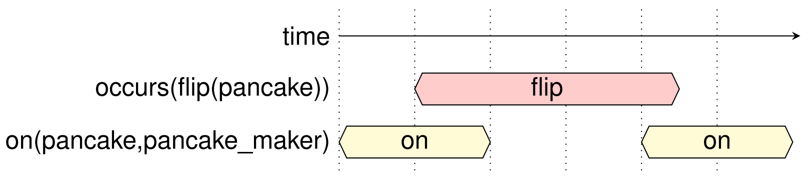


FIGURE 4.17 Timeline representation.

```
?- holds_tt(on(pancake,pancake_maker),I1,TL),
   occurs_tt(flip(pancake),I2,TL),
   holds_tt(on(pancake,pancake_maker),I3,TL),
   overlaps(I1,I2),
   overlaps(I2,I3).
```

Pouring and flipping experiments can be combined by using the poured particles that end up on the pancake maker for generating a more complex pancake model. We start at the particle closest to the center of the cluster and create a graph like flexible joint structure. Joints are created between the seed particle and the particles found within a certain radius from it, and afterward these new particles become seeds themselves. To make the pancake model look more realistic, a flexible textured mesh created from the convex hull is attached to the structure (Figure 4.18).

4.6 Related Work

The present work can be considered as interdisciplinary research of two fields: robotics and AI. With this research, we want to enable robots to reason about the consequences of action parameterizations and thereby allowing them to make appropriate decisions in their course of action by using well-established methods of AI and detailed physical simulations.

Only recently, Smith and Morgan (2010) stressed the importance of using simulations in AI research. They developed the open source simulator *IsisWorld* for investigating problems in commonsense reasoning. Although they also employ a physics engine for their simulations, they consider actions like *picking up an object* only at a very abstract level,

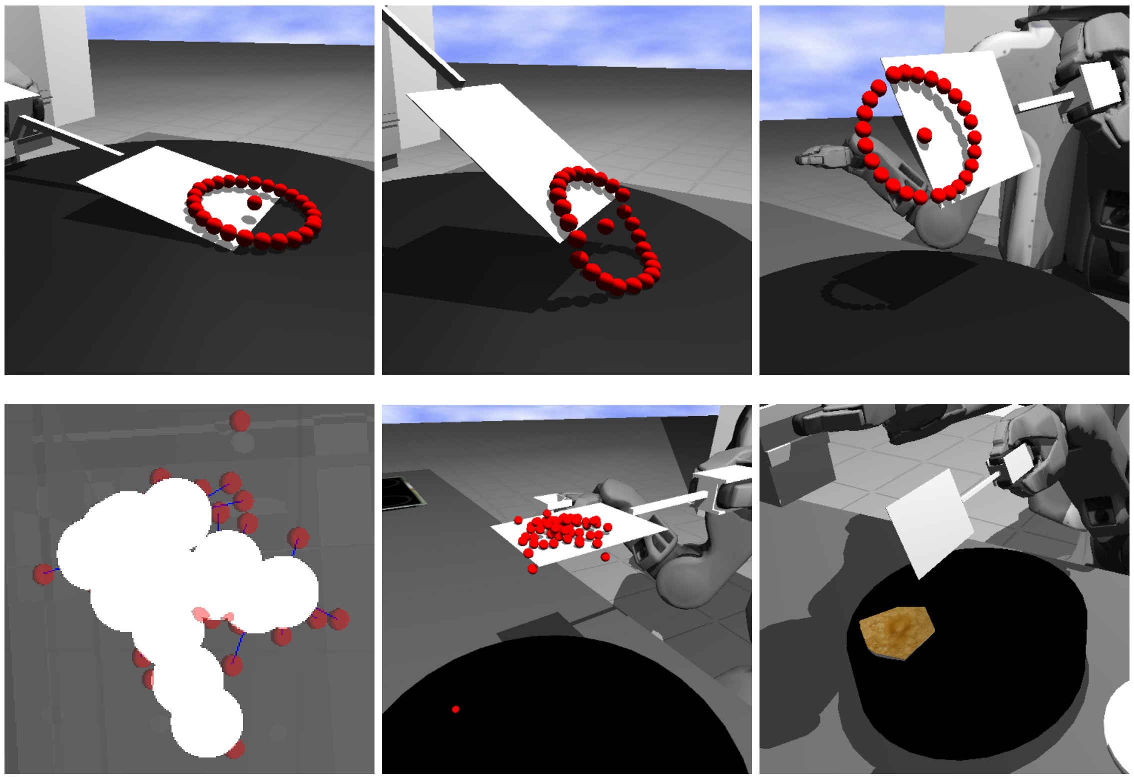


FIGURE 4.18 The pancake model in different views: a generic circular model (up), a generated model showing the joint structure and flexible mesh (down).

whereas we focus on the physical details of such actions in order to recognize qualitative phenomena occurring during their execution.

Our simulation-based approach is in a similar line of work by (Johnston and Williams, 2008) who integrated logic and simulation for a practical approach to commonsense reasoning. Whereas they use a general purpose simulation, we utilize a physics-based simulator augmented with phenomena of everyday physics since we are particularly interested in naive physics reasoning for robot manipulation. Instead of looking at isolated problems, we aim for a tight integration between the our proposed reasoning system and other processes like planning, e.g., to predict whether a pancake can be flipped when executing a specific plan for making a pancake.

Work by Ueda et al. (2008) describes the design and implementation of a programming system based on EusLisp that make use of a simulation for deformable objects. Thereby, robot control programs can easily exploit the specialized computations made by the simulation. Similarly, we use the logic programming environment PROLOG and utilize a physics-based robot simulator. In addition, we integrated methods for making simulation-based temporal projections into PROLOG's backtracking mechanism in order to perform reasoning about action parameterizations for robot manipulation tasks.

Research on the interactive cooking simulator (Kato et al., 2009) is relevant for our work, since it aims at a deep understanding of cooking operations, which could bring new insights with respect to representations and reasoning mechanisms for manipulation actions in everyday meal-preparation tasks.

Exploiting physical simulators for effectively solving sub-problems in the context of robotics has become more attractive as shown by a number of recent investigations, where simulations are employed for planning in robocup soccer (Zickler and Veloso, 2009), for navigating in environments with deformable objects (Frank et al., 2009), and for reasoning about the consequences of everyday manipulation tasks (Kunze et al., 2011b). A detailed evaluation for using physics engines for improving the physical reasoning capabilities of robots is given in (Weitnauer et al., 2010). But other fields also recognize simulators as valuable tools and utilize them, e.g., for character animation (Faloutsos et al., 2001) and motion tracking (Vondrak et al., 2008).

In the context of Naive Physics (Hayes, 1979, 1985), solutions to the problem of egg cracking (Davis, 1997), were formulated by (Lifschitz, 1998; Morgenstern, 2001) based on logical axiomatizations. Limitations of these approaches are mainly that physical details are abstracted away and that variants cannot be handled very flexibly. To overcome such limitations this work proposes a simulation-based approach: we take a logical axiomatization and translate it into a parametrized simulation problem, simulate and log simulation data, translate logged simulation data into an interval-based first-order representation which is used for answering queries about a qualitative reasoning problem.

The integration of numerical simulation and qualitative methods has been investigated before (Weld and Kleer, 1990), for example, work on qualitative-numeric simulation (Berleant and Kuipers, 1992) and self-explanatory simulations (Forbus and Falkenhainer, 1990). Work by Lugrin and Cavazza (2007) has shown an integration of numerical simulation and qualitative modeling based on the Qualitative Process Theory (Forbus, 1984) for virtual interactive environments. But none of the approaches, we are aware of, have investigated a simulation-based approach for making predictions in the context of every robot object manipulation.

The grounding of logical predicates like *contacts*(o_1, o_2) in data of logged simulations is done similar to work by Siskind (2001) who grounded semantics in visual perception. Similarly, we ground only primitive predicates in logged simulations. Complex predicates are formulated in PROLOG and are based on primitive or other complex predicates similar to definitions of symbolic chronicles (Ghallab, 1996).

A simulation-based approach for temporal projection in reactive planning is proposed in (Mösenlechner and Beetz, 2009), that is, predicting the interfering effects of continuous and concurrent actions. Similarly, this work proposes a simulation-based approach for naive physics reasoning for robot manipulation tasks.

4.7 Discussion

In this section we discuss why autonomous robots should be endowed with methods allowing to make temporal projections about naive physics problems. We provide arguments to base these methods on detailed physical simulations and elaborate on the right fidelity of these simulations. Furthermore, we outline how this approach of logic programming using a simulation-based temporal projection can be used to adjust the behavior of robots. Finally, we examine how far the proposed approach can be taken, and also name some possible application scenarios.

One might argue that most robotic applications are developed for specialized tasks and thereby robots do not need robust commonsense reasoning capabilities, or as we propose, capabilities for naive physics reasoning. But in the context of autonomous personal robots the set of everyday manipulation tasks is not fixed, and furthermore, task and environment conditions change all the time and therefore robots need flexible mechanisms to reason about the appropriate parameters of their control programs.

In the literature there exist some approaches using symbolic reasoning methods for making inferences about simple physical problems (Morgenstern, 2001). The main limitations of these approaches especially in the context of robotics are threefold: (a) important details like positions of manipulators and objects are abstracted away; (b) variants of problems like manipulating an object with different physical properties cannot be handled without extending the logical theory; and (c) consequences of concurrent actions and events are very difficult to foresee with pure symbolic reasoning, e.g., what does a robot see when turning its camera while navigating through its environment? All of these limitations do not occur in physics-based simulators. Even if the simulation do not reflect the physical world, parameters can be learned by applying machine learning technologies as in (Johnston and Williams, 2009).

One important issue when using high-fidelity physics models in simulations is performance. Currently, the system cannot make predictions in a reasonable time that would allow us to use it for planning during execution. Nevertheless, it is a powerful tool for robots to *mentally* simulate (offline) the consequences of their own actions either to prepare themselves for new tasks or to reconsider task failures.

Related to the issue of performance is the issue of the *right* fidelity, i.e. how to make the physical models robust enough to enable effective behavior and yet small enough to be usable during execution. When creating physical models we are concerned about getting the qualitative behavior of objects right, i.e., we are not aiming at models that reflect every sheer detail. Very detailed models do not readily provide the information needed to choose the appropriate action parametrization, therefore we abstract the reality into a smaller qualitative state space.

Although it would be desirable to use the presented approach for planning during execution by using more realistic physical models, we are currently not aiming at both, high performance and very realistic models. Rather the developed system represents a proof-of-concept how to use simulation technologies for symbolic reasoning. In the long run, we assume that issues regarding performance and the appropriate fidelity of physical models will be addressed by the game and character animation industry (Cho et al., 2007), which will provide powerful technologies that could be employed.

The realized logic programming framework allows robots and programmers to automatically determine the appropriate action parameters by setting up a manipulation scenario, by executing differently parametrized control programs in simulation, and finally, by evaluating queries based on the resulting timelines. An interface to the logic programming framework is provided by both PROLOG's command-line and a ROS[13] service, which takes arbitrary PROLOG queries as request and provides the respective variable bindings as response. Thereby, naive physics reasoning for manipulation tasks can be flexibly integrated into control programs and planners in order to effectively change the robot's behavior.

Finally, we want to approach the question of how far this approach can or should be taken, before we point to some potential application scenarios. It is clear that one would not want to do this kind of full-fidelity physics simulation for all kinds of problems, e.g. problems in motion planning can be solved by employing more specific planners as primitives. However, some kind of limited simulation seems to be very plausible, at least for some very hard problems. We believe that lifting physics-based simulations to a symbolic level is beneficial for deriving solutions for robot manipulation, and also other domains, where current methods are not effective.

Planning is increasingly considering physical platforms in complex, real world environments. The presented framework could provide a more precise guidance in the planning

[13]`http://www.ros.org`

process since the simulation-based methods for making temporal projections are tightly linked to the technical details of platforms under question. Naive physics reasoning could also be used as a tool for developing robot control programs. Programmers could recognize and prevent problems occurring during execution more easily. Additionally, the presented framework could be used for benchmarking purposes. For example, data generated by the simulations could serve as basis for inference tasks. Thereby, different approaches to physical reasoning could be compared in a straight forward way. In general, the usage of the open source software like Gazebo and ROS allow to employ the naive physics reasoning to new problems including other robot platforms and different objects quite easily. Therefore we believe that logic programming using detailed physical simulations is a well-suited tool for making predictions about every manipulation tasks and also for other potential applications.

CHAPTER 5

Naive Physics Reasoning for Handling Fluids

Personal robot assistants that are to accomplish an open-ended set of everyday manipulation tasks like making pancakes are required to understand the physical effects of their own actions. In particular challenging are tasks that involve the handling of fluids.

In this work we investigate how robots can infer the consequences of their parameterized manipulation actions (here particularly pouring and mixing) in order to make competent and failure-aware decisions during their course of action. The proposed system allows robots to determine the action parameters that lead to the desired effects by asking queries using a first-order language. The queries are answered based on interval-based first-order representations, called timelines (Section 3.5), which are grounded in detailed physics-based simulations of parameterized robot control programs as explained in Chapter 4. Preliminary work on handling fluids has been published in (Klapfer et al., 2012).

5.1 Fluids in Everyday Manipulation

Modern robotics attempts to go beyond simple pick-and-place scenarios and equip robots with the capabilities for executing more complex tasks like, for example, making pancakes (Beetz et al., 2011) or baking cookies (Bollini et al., 2011). When preparing pancakes, ingredients for the dough like milk, eggs, and granular fluids like sugar and flour have to be mixed, stirred and eventually poured into a pan.

Accomplishing such everyday manipulation tasks successfully requires robots to understand the physical effects of manipulation actions. That is, robots have to predict the consequences of their parameterized actions before committing to them. In the domain of meal preparation tasks there is only a limited set of core actions and physical processes as shown by Nyga and Beetz (2012). Therefore in seems a feasible approach to analyze and model the physical principles of each of these core actions. In this work we place our emphasis on robots performing manipulation tasks which involve the handling of fluids. For example, a robot about to pour pancake mix onto a pancake maker has to decide where and how to hold the container without spilling the contents. Inferring answers to such questions provide valuable information for robots when actually performing the task.

Humans are able to reason about these physical processes and to estimate the right *parameterization* intuitively based on both experience and common sense. Davis (2008) presents a formal solution to the problem of pouring liquids and in his work on the representation of matter (Davis, 2010), he investigated the advantages and disadvantages of various representations including those for liquids. Further Davis (2012) claims that it is tempting to use simulations for spatial and physical reasoning problems. But he also argues that simulations are not suitable for the interpretation of natural language texts because many entities in texts are highly underspecified. However, in the context of robotics entities in the environment can often be sufficiently recognized by sensors and represented using internal models. Therefore we believe that if we equip robots with Naive Physics, that is knowledge about the characteristics of physical processes, objects, and substances, we will enable them to make informed decisions in context of planning, diagnosis and learning.

In this work we build on the concept of simulation-based temporal projections as proposed in Chapter 4 and (Kunze et al., 2011b,a). Everyday robot manipulation tasks are simulated with varying parameterizations, world states of the simulation and states of the robot control programs are monitored and logged. The resulting logs are translated into a first-order representations, called timelines. These timelines are then used to answer logical queries on the resulting data structures in order to understand the physical effects of the robot's actions. The main contribution of this work is the design and implementation of data structures and algorithms for representing and reasoning about fluids within this framework.

The remaining chapter is structured as follows: In Section 5.2 we give a short overview of the envisioning framework and briefly explain its components. Section 5.3 describes how fluids are represented and simulated. The experimental results are presented in Section 5.4. Finally, we discuss the present work in Section 5.5.

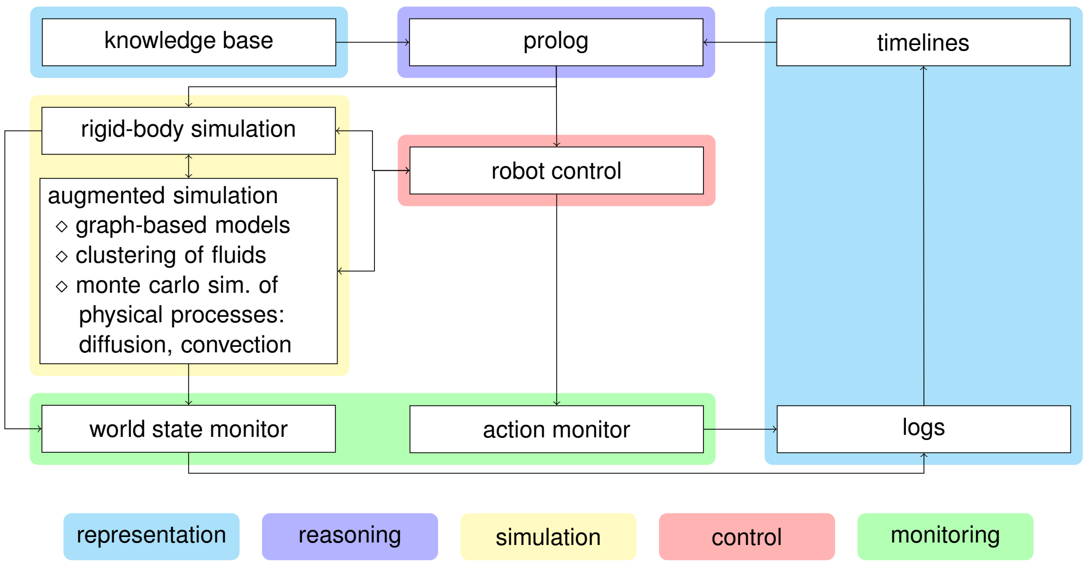


FIGURE 5.1 The simulation-based temporal projection framework.

5.2 Envisioning Revisited

This section briefly revisits the envisioning framework introduced in Chapter 4 and explains its extensions for handling fluids.

The overall framework is depicted in Figure 5.1. It is based on state-of-the-art technologies such as ROS[1], the Gazebo simulator[2] and the point cloud library PCL[3]. A manipulation scenario can be specified using predicates of a first-order language within PROLOG. Based on this description a physical simulation is instantiated. Within the simulation a robot can freely navigate and interact with objects. The behavior of the robot is specified by a robot control program. In the simulator we represent, e.g., a pancake mix as particles using the data structures of Gazebo. Given that we are particularly interested in analyzing the behavior of liquids we group the simulated particles by an Euclidean clustering technique. Having obtained information of clusters makes it possible to reason about the fusion or division of volumes or chunks of liquids. The clustering is realized as ROS node located at an augmented simulation layer. As Gazebo uses ODE[4] which is only capable of dealing with rigid bodies the simulation of liquids is only an approximation. Therefore we use the information about the clusters to initialize a more accurate simulation of liquids by

[1]http://www.ros.org
[2]http://www.gazebosim.org
[3]http://www.pointclouds.org
[4]http://www.ode.org

considering physical aspects such as molecular motion due to diffusion and convection. The robot's actions, its interactions with the objects, the state of the liquid, the clusters and the state of the environment (world) are crucial information for the reasoning framework. How these data structures are represented and accessed is explained in detail in Chapter 3 and 4. For example, the world state comprises information about the position, orientation, linear and angular velocities, and the bounding box of an object at a point in time and is denoted as follows:

World state : $\langle$*time, obj, pos, orient, lin_vel, ang_vel, bbox*$\rangle$.

Additionally, we also monitor contact events:

Contact state : $\langle$*time, o1, o2, num, force, torque, normal*$\rangle$,

whereby we observe the number of contact points as well as the forces, torques and normals between them. We have implemented monitoring routines as Gazebo controllers that keep track of the dynamics of objects and write this information to log files.

These logs are then translated into interval-based first-order representations. We access and evaluate the data structures from PROLOG using the following predicate

$$\mathit{SimulatorValue}(\overbrace{\mathit{position}(o, pos)}^{\text{Function}}, \overbrace{t}^{\text{Time point}}, \overbrace{tl}^{\text{Timeline}}),$$

whereby different functions are available for accessing the time-stamped information of the world and contact states.

In order to define more high-level predicates we use concepts similar to those in the Event Calculus (Kowalski and Sergot, 1986b) (Section 3.4). The notation is based on two concepts, namely fluents and events. Fluents are conditions that change over time, e.g., a mug contains a pancake mix: *contains(mug,mix)*. Events (or actions) are temporal entities that have effects and occur at specific points in time, e.g., consider the action of pouring the mix from the mug onto the pancake maker: *occurs(pour(mug,pan))*. Logical statements about both fluents and events are expressed by using the predicate *Holds(f,t,tl)* where *f* denotes a fluent or event, *t* simply denotes a point in time, and *tl* a timeline. Using the similar predicate $\mathit{Holds}_{\mathit{Throughout}}$ we can query for a time interval throughout the fluent holds. Logical queries to the framework are basically answered through Prolog's backtracking mechanism over a set of timelines.

Having given an overview of the overall system, the next section will lead us to the representation and simulation of fluids within this framework.

5.3 Representation and Simulation of Fluids

Simulating liquids is of great interest in physics and chemistry (Allen and Tildesley, 1989). As some processes occur very fast, events might not be observable in all its details in reality. The purpose of simulating liquids in our work is to observe the impact of the robot's action with respect to the liquid's behavior, which is of great importance when, e.g., pouring and mixing liquids. Different approaches have been incorporated to simulate liquids depending on the required level of accuracy needed (Griebel et al., 2007). In this work we propose two complementary approaches for simulating liquids, (1) a graph-based model similar to (Johnston and Williams, 2008) and (2) a Monte-Carlo simulation for modeling diffusion and convection (Frenkel and Smit, 2001). Both do not simulate liquids in all their aspects but provide enough information for making logical inferences about qualitative phenomena.

5.3.1 Representing Fluids using Graph-based Models

The model for representing fluids was adapted from the work of Johnston and Williams (2008). Originally, it was designed to simulate a wide range of physical phenomena including diverse domains such as physical solids or liquids as hyper-graphs where each vertex and edge is annotated with a frame that is bound to a clock and linked to update rules that respond to discrete-time variants of Newton's laws of mechanics.

Our pancake mix model can be in two states: first, the mix is liquid, and second, the mix becomes a deformable pancake after cooking. In the simulation we use a graph-based model for representing the mix and the pancake. The vertices of the graph are particles where each particle is defined by a round shape with an associated diameter, a mass and a visual appearance model. The benefit of this model is that it is realized as graph with no connection between the vertices whenever the state is liquid. This means that the individual particles could move freely to some extent. This was useful for performing the pouring task. Due to the fact of the particles not being connected with joints, the simulated liquid can be poured over the pancake maker where it dispenses due to the round shape of the particles. A controller was attached to the particles that applies small forces to them in order to simulate the viscosity of the pancake mix. Currently, we do not consider heat as the trigger of transforming the liquid to a solid pancake but simply assume the event to occur after constant time. We identified all particles on the pancake maker and created the pancake based on a graph traversal algorithm starting at the cluster center (Figure 5.2).

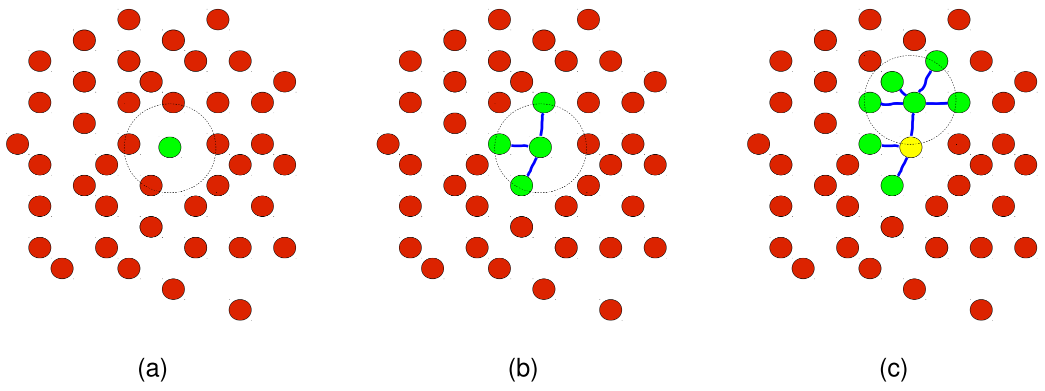


FIGURE 5.2 Generating a deformable pancake model from liquid particles. Illustration of the algorithm's procedure: (a) Radial search from the seed Point. (b) Creation of hinge joints to the neighbors. (c) Radial search and creation of joints in a recursive step. Courtesy: Reinhard Klapfer.

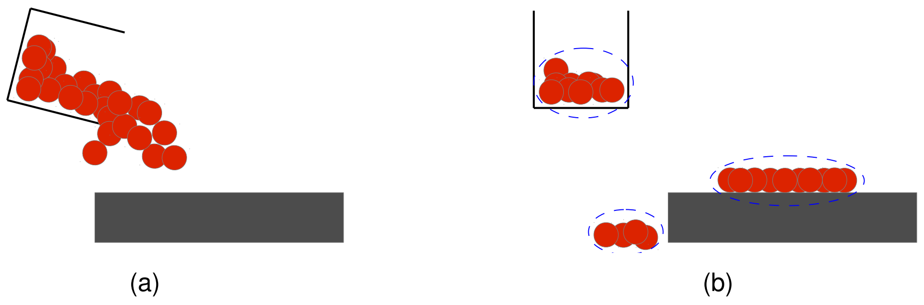


FIGURE 5.3 Basic idea of the clustering approach: during simulation we identify clusters of particles. For example, after pouring, one cluster resides still in the mug, a second is on the pancake maker and a third is spilled onto t he table. We are able to extract information including contacts, position, extension, and size of the individual clusters. Courtesy: Reinhard Klapfer.

5.3.2 Clustering of Fluid Particles

The basic idea of applying clustering methods is as follows. Let us, for example, assume that someone pours some pancake mix onto a pancake maker as illustrated in Figure 5.3. After the pouring action some particles reside in the container, some are spilled onto the table, and some others are on the pancake maker which will eventually form the pancake. If we want to address the particles in these three locations it perfectly makes sense to group them in chunks (clusters). This reflects also how humans address fluids like milk or sugar in natural language, for example, there is some milk spilled onto the table. Therefore the behavior and the contact information of clusters of particles in everyday manipulation tasks are of particular interest. We decided to use a Euclidean clustering strategy for

Algorithm 1 Euclidean clustering of particles.

```
1) Set up an empty list of clusters $Clst$

2) For every particle $p_i \in P$ do
   □ Add $p_i$ to the current cluster $C$
   □ For every point $p_j \in C$ do
      □ Find particle $p_k$ using a radial search around particle $p_j$
      □ For each particle $p_k$ add it to $C$ if not processed, yet
      □ Terminate if all $p_j \in C$ have been processed
   □ Add $C$ to the list of clusters $Clst$ and reset $C$ to an empty list

3) The algorithm terminates if all particles have been assigned to a cluster $c_i \in Clst$
```

computing the groups of particles as shown in Algorithm 1.

Instead of looking at the individual particles when interpreting the outcome of a manipulation scenario we look at clusters of particles. For every cluster we compute information such as mean, covariance, size (number of particles), and its bounding box. Since we have full knowledge about every particle and its belonging to a cluster, we can keep track of it. Thereby we can detect if the pose or extension of a cluster change over time. However, whenever new particles become part of or are separated from a cluster we assign a new ID to it. That is, clusters of particles have only a limited time during which they exist. Hence, we can recognize which actions cause changes to clusters.

5.3.3 Monte Carlo Simulation of Fluids

Deformable bodies are seen as a big challenge in simulation and usually require a lot of computational power (Brown et al., 2001). The physical simulation approach (Frenkel and Smit, 2001) uses a Monte-Carlo process to simulate diffusion of liquids. Molecular movement is either provoked from heat or from a difference in potential. The rate of change depends on the diffusion coefficient and its respective change. This is a well known concept in physics described by equation 5.1 and denoted as the *macroscopic diffusion* equation or *Fick's second law* of diffusion. This differential equation takes into consideration a change of concentration over time.

$$\frac{\partial C}{\partial t} = D \cdot \frac{\partial^2 C}{\partial^2 t} \tag{5.1}$$

Frenkel and Smit (2001) showed that *Random Walk* gives one particular solution for the above partial differential equation. Motivated by this idea we applied Algorithm 2 proposed by Frenkel and Smit (2001) to simulate this physical effect. The algorithm follows the Metropolis scheme and uses a probability function to decide if a particle is going to be displaced or not. The Leonnard-Jones Potential Function (Equation 5.2) was used to

Algorithm 2 Metropolis scheme.

```
1) Select a particle $r$ at random and calculate its energy potential $U(r^N)$
2) Give the particle a random displacement, $r' = r + \Delta$
3) Calculate the new energy potential $U(r'^N)$
3) Accept the move from state $r^N$ to $r'^N$ with probability

        acc( $r^N \mapsto r'^N$) = $min\left(1, \exp\left(-\beta\left[U(r'^N) - U(r^N)\right]\right)\right)$
```

model the interaction among the particles in the liquid, that is, to model the particles' behavior according to the concentration of particles in their neighborhood. The parameters σ and ϵ are used to shape the function and r is the distance to neighboring particles.

$$U(r) = 4\epsilon\left[\left(\frac{\sigma}{r}^{12}\right) - \left(\frac{\sigma}{r}^{6}\right)\right] \tag{5.2}$$

Stirring a material is another type of mass transfer called convection. Convection is the movement of mass due to forced fluid movement. Convective mass transfer is a faster mass transfer than diffusion and happens when stirring is involved. The faster the fluid moves, the more mass transfer and therefore the less time it takes to mix the ingredients together (Gould et al., 2005). We simulated this physical property by simply introducing an impulse in stirring direction to the particles in the point cloud that are in reach of the cooking spoon. In this way, we could achieve the behavior of molecular motion due to forced fluid movement.

5.3.4 Measuring the Homogeneity of Mixed Fluids

The homogeneity of the liquid was of particular interest when stirring was involved in the conducted experiments. It was decided to use the local density of the particles represented as point cloud as a measure of divergence, while using the assumption that the inverse of this is a measure of homogeneity. This distance measure (Majtey et al., 2005) is known as the Jensen-Shannon divergence and used widely in information theory. The Jensen-Shannon divergence is defined as:

$$JS(P,Q) = \tfrac{1}{2}S\left(P, \tfrac{P+Q}{2}\right) + \tfrac{1}{2}S\left(Q, \tfrac{P+Q}{2}\right) \tag{5.3}$$

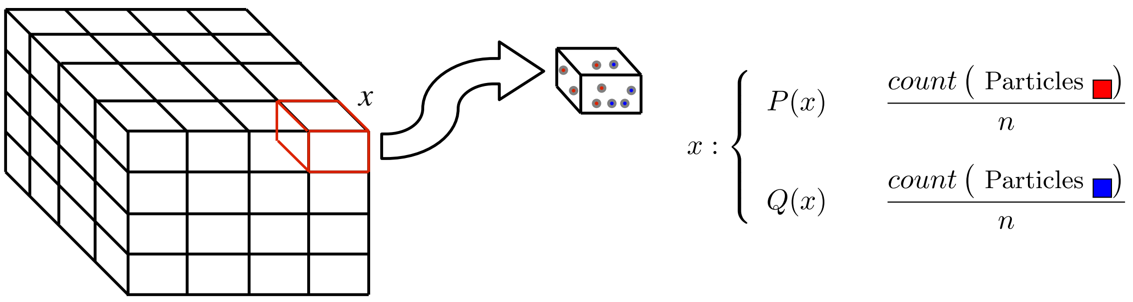

x
$x :$
$P(x)$ $\frac{count\ (\text{Particles } 🟥)}{n}$
$Q(x)$ $\frac{count\ (\text{Particles } 🟦)}{n}$

FIGURE 5.4 Density grid used for discretization and local density estimation. Courtesy: Reinhard Klapfer.

where $S(P, Q)$ is the Kullback divergence shown in equation 5.4, and P and Q two probability distributions defined over a discrete random variable x.

$$S(P, Q) = \sum_x P(x) \log \left(\frac{P(x)}{Q(x)} \right) \tag{5.4}$$

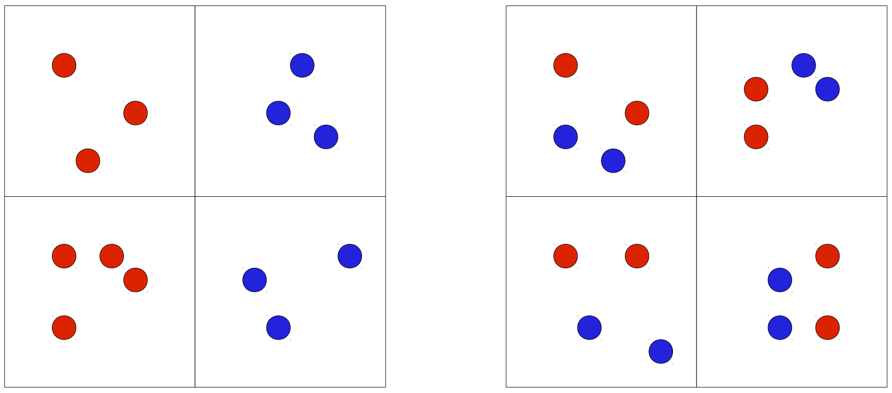
We propose the division of the point cloud in a three-dimensional grid (Figure 5.4). Each cell of the grid represents a discrete probability distribution x defined on the mixed probabilities of the two classes P and Q, that could be computed as the relative frequency.

The following example emphasizes the usage of this distance function related to the homogeneity of a liquid which consists of two classes of particles. If we assume a perfect separation of the two classes as shown in Figure 5.5(a), we would expect a high divergence and a low homogeneity as we define the homogeneity as its inverse.

5.4 Experimental Results

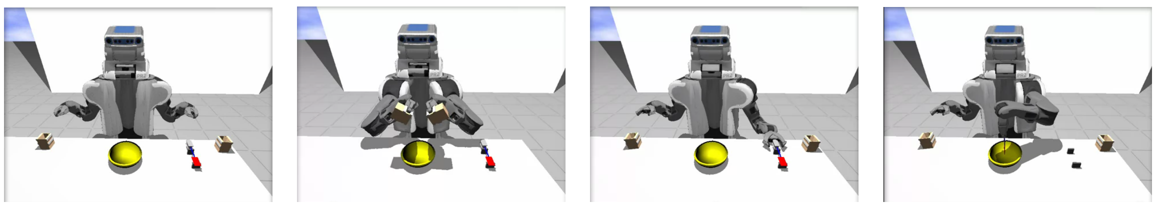
In this section we are going to highlight the results of both experiments, namely mixing ingredients in a bowl while measuring the level of homogeneity[5], and second, pouring the mix onto a pancake maker and reasoning about whether some mix was spilled[6].

[5]Video (mixing): `http://www.youtube.com/watch?v=ccHXmkKT8CE#!`
[6]Video (pouring): `http://www.youtube.com/watch?v=tzQk7S5PRaY`



(a) Maximum divergence, minimal homogeneity. (b) Minimal divergence, maximal homogeneity.

FIGURE 5.5 A simple 2D density grid for a two class problem. Courtesy: Reinhard Klapfer.



(a) Initial Position (b) Pouring (c) Grasping Spoon (d) Stirring

FIGURE 5.6 PR2 pours two ingredients in a bowl and stirs them.

5.4.1 Mixing Liquids — Analysis of Homogeneity

We used the Monte Carlo method previously described in Section 5.3.3 to simulate the physical effects when mixing liquids with different trajectories. Figure 5.6 shows the PR2 performing the task.

We selected the coefficients to represent two viscous liquids. Figure 5.7 and Figure 5.8 show the course of homogeneity when the robot stirs the liquids using (1) an elliptic trajectory, (2) a spiral trajectory, (3) a lineal trajectory, and (4) no trajectory (without stirring). As we expect, the ingredients do not mix very well when the robot does not stir the liquids. Hence, the result of the experiment confirms our hypothesis: Stirring increases the homogeneity of mixed liquids.

Furthermore, the result shows that with an elliptic trajectory the best result could be achieved. Although Figure 5.7 visualizes continuous data of the homogeneity, we are mainly interested in qualitative effect models. These qualitative models of *homogeneous*, *semi-homogeneous* and *inhomogeneous* regions are simply defined by thresholding the

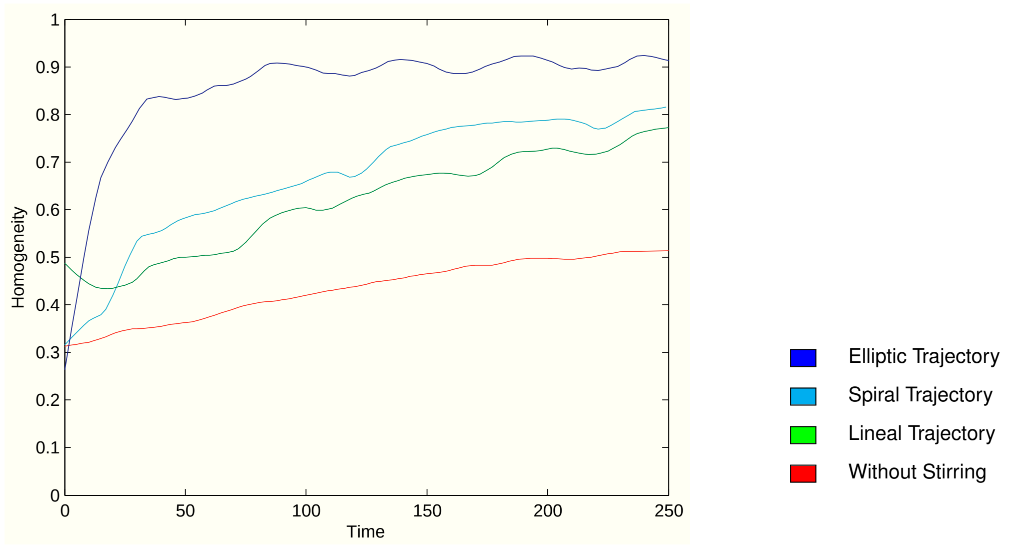


FIGURE 5.7 Homogeneity over time of different stirring trajectories. The graph shows the change of homogeneity on the vertical axis for different trajectories in direct comparison with the result of scenario of not stirring over time. Courtesy: Reinhard Klapfer.

quantitative data. Given the knowledge of homogeneous, semi-homogeneous and inhomogeneous regions a robot could adapt the trajectory dynamically by applying techniques known from Reinforcement Learning.

5.4.2 Pouring Fluids — Reasoning about Clusters

In this experiment we address the scenario of pouring some pancake mix located in a container onto a pancake maker: the robot grasps a mug containing pancake mix from the table, lifts it and pours the content onto a pancake maker (Figure 5.9). In this experiment we used the resulting timelines to analyze the qualitative outcome of the executed action. The parameterization of the task included the gripper position, the pouring angle and the pouring time. The task was considered to be successful if no pancake mix has been spilled, that is, the liquid resides on the pancake maker or in the container and not on other objects such the kitchen table after the pouring action ends. We used the resulting clusters and their corresponding contact and spatial information to examine the outcome. Figure 5.10 shows exemplarily how clusters of pancake mix are spatially related to other objects before, during and after the pouring action.

More results are summarized in Table 5.1 where the numbers for *Mug*, *Pan*, and *Table* denote the number of particles of clusters in contact with the respective models. The following PROLOG expression shows exemplarily how such information about clusters

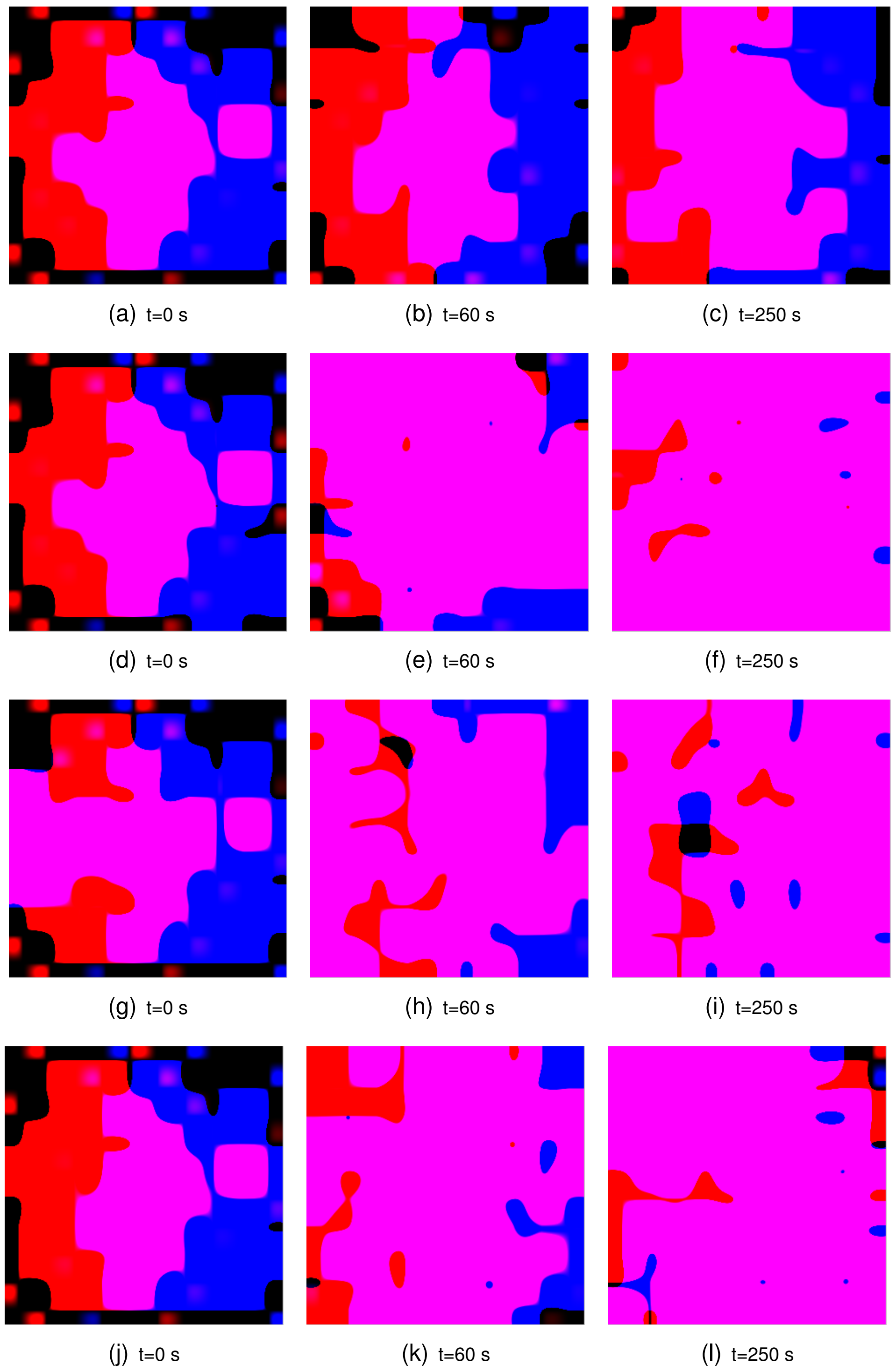


FIGURE 5.8 The color coded images show the spatial distribution of homogeneity of two liquids. Black stands for uncovered regions, red and blue for inhomogeneous liquids of corresponding classes and purple homogeneous regions. Stirring trajectories: (a-c) without stirring, (d-f) elliptic, (g-i) lineal, (j-l) spiral. Courtesy: Reinhard Klapfer.

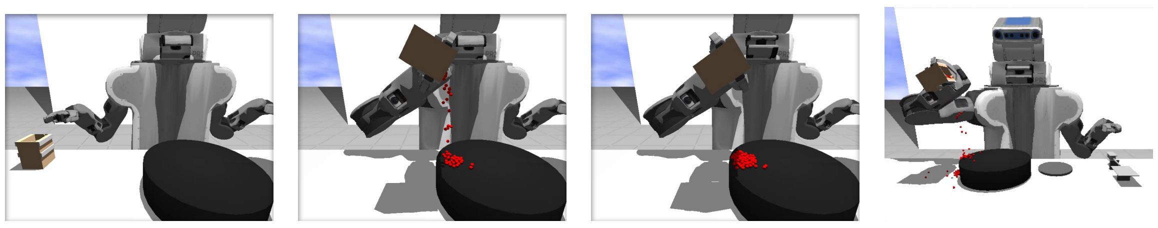

(a) Grasping mug (b) Pouring (c) Success (d) Failure

FIGURE 5.9 PR2 pours mix onto pancake maker.

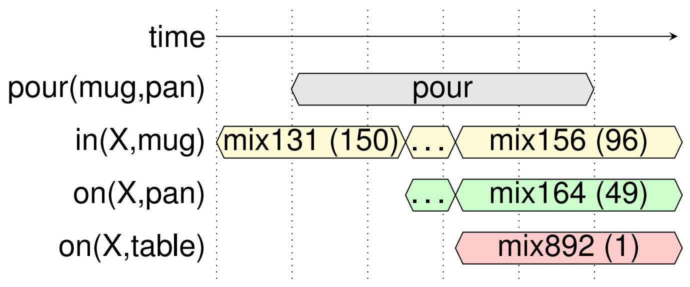


FIGURE 5.10 Visualization of the main clusters of particles before, during and after the pouring action. The pancake mix was represented by 150 particles. The number of particles of a cluster is shown in parenthesis. During the simulation there were more than 20 clusters generated on this timeline.

can be retrieved from timelines *TL*

```
?- holds_tt(occurs(pour(Params)),I,TL), [_,End] = I,
     partOf(X,pancake_mix), holds(on(X,table),Time,TL),
     after(Time,End),
     simulator_value(size(X,Size),Time,TL),
     simulator_value(mean(X,Mean),Time,TL),
     simulator_value(var(X,Var),Time,TL).
```

where *X* denotes a cluster of pancake mix in contact with the table after the pouring action has been carried out.

We used logical queries such as above to extract data for learning decision trees in order to classify pancake sizes and pouring angles (Chapter 7). Although decision trees can also be learned on continuous data, we mapped the numerical data to qualitative concepts such as, for example, *Small*, *Medium* and *Large* pancakes. Thereby, the resulting qualitative models are intuitively interpretable by humans.

TABLE 5.1 Distribution of particles in clusters and their contacts.

Control parameters			**Desired contacts**		**Undesired**	**Clusters**
pos.	angle	time	mug	pan	table	
0.00	2.09	1.0	102	98	-	2
		1.5	1,2,100	96	-	4
		2.0	123,1,1	75	-	4
	2.44	1.0	22	178	-	2
		1.5	1,25	174	-	3
		2.0	27,1,1	1,170	-	5
	2.61	1.0	1,7	1,1,1,1,187	1	6
		1.5	13	187	-	2
		2.0	1,7	192	-	3
0.05	2.09	1.0	97	103	-	2
		1.5	138	62	-	2
		2.0	1,1,64,132	-	-	4
	2.44	1.0	3,23	1,1,172	-	5
		1.5	1,1,34	1,163	-	5
		2.0	1,1,29	1,168	-	5
	2.61	1.0	10	190	-	2
		1.5	11	189	-	2
		2.0	1,9	190	-	3
0.10	2.09	1.0	1	93,105	-	3
		1.5	1,119	1,79	-	4
		2.0	125	74	1	3
	2.44	1.0	18,1	1,1,2,177	-	6
		1.5	24	1,1,174	-	4
		2.0	19	1,179	-	3
	2.61	1.0	6	193	1	3
		1.5	1,11	1,186	1	5
		2.0	19	181	-	2
0.20	2.09	1.0	1,133	10	1,53	5
		1.5	1,1,113	15	1,1,1,1,64	9
		2.0	1,1,127	11,57	1	6
	2.44	1.0	25	1,50	124	4
		1.5	2,24	1,1,23	1,1,1,146	9
		2.0	1,26	1,28	1,1,142	7
	2.61	1.0	11	-	189	2
		1.5	11	42	1,146	4
		2.0	1,11	1,19	1,167	6

5.5 Discussion

The present work can be considered as interdisciplinary research of two fields: Robotics and Artificial Intelligence.

With our approach we enable robots to reason about the consequences of their own actions. We equip them with the capability of making appropriate decisions about their parameterizations throughout their activity using well-established methods of AI and detailed physical simulations.

To this end, we have developed a system that simulates robot manipulation tasks, monitors relevant states and actions, and translates this information into first-order representations, called timelines. Then, we use the logic programming language PROLOG to answer queries based on the data structures of the temporal projections.

The main contribution of this work is the extension of the framework with respect to data structures and algorithms for representing and simulating fluids. We conducted two experiments within the framework: mixing and pouring liquids. The resulting timelines of the experiments were qualitatively evaluated with different performance criteria, e.g., the homogeneity of the mix and the number of spilled particles.

CHAPTER 6

Acquiring Task Knowledge through Games with a Purpose

Teaching robots everyday tasks like making pancakes by instructions requires interfaces that can intuitively be operated by non-experts. By performing novel manipulation tasks in a virtual environment using a data glove task-related information of the demonstrated actions can directly be accessed and extracted from the simulator. We translate low-level data structures of these simulations into meaningful first-order representations whereby we are able to select data segments and analyze them at an abstract level. The proposed system uses Games with a Purpose for acquiring examples of manipulation actions. By analyzing such examples robots can be understand how humans would perform a task.

6.1 Human-scale Manipulation Tasks

Scaling the task repertoire of autonomous manipulation robots towards open ended sets of human-scale manipulation tasks requires novel ways of efficient programming. A promising way to do so is the transformation of task instructions made for human use into executable robot plans (Tenorth et al., 2010b). Key problems are that the instructions are typically very incomplete and ambiguous and omit to state what everybody knows anyway. Even worse, many aspects that are key factors for the success of everyday manipulation actions are so obvious that people have problems stating them. People can be asked to cut a piece of bread but almost everybody would have serious problems in describing the action detailed enough such that a successful control program could be written from this specification. This is a huge problem for autonomous robots that are to perform everyday manipulation tasks because they lack the commonsense knowledge that humans have.

The most prominent approach to enable robots to acquire the skills for such actions is imitation learning or learning by demonstration (Khansari-Zadeh and Billard, 2010; Schaal et al., 2004). While these approaches have proven successful they are also limited because they only copy observed behavior without understanding the object interactions that are desired, intentions behind the behavior, and which variations of behavior are caused by which context conditions.

For example, in imitation learning it is difficult to acquire a general and flexible routine for pouring pancake mix into the pan. It is hard to learn how fast and how far to turn the bottle and how high to hold it because the robot does not know that the demonstrator tries to avoid spilling the pancake mix. In addition, how the bottle is held depends on the viscosity of the pancake mix in the bottle and how big the hole in the bottle is, state aspects that the learning observer cannot see.

In this work we propose the use of computer games (GWAP) that are coupled with a physics simulator as a powerful means to extend the information of task instructions written for humans. The computer game puts all ingredients and tools specified in the instructions of a task on a table. It then prints the individual instruction steps on the screen and asks the player to perform the step. The physics simulator provides the information about object interactions (contact, object modifications), cause-effect relationships for action effects, and forces. The simulator even lets the learner actively acquire example action executions for variations in the physical situation, such as pouring pancake mix with varying viscosity of the pancake mix.

The solution that we propose are Games with a Purpose in which players have to perform descriptions of manipulation actions. The players are controlling a robot hand to perform the action. The simulator is extended with specific physical processes such as mixing, cooking and so on as presented in Chapter 4 and 5. The data structures of the simulator are logged throughout the game. The log data structures are then turned into a virtual knowledge base in which the learning robot can query abstract information about the game episode in a PROLOG-based query language. Using the query language introduced in Chapter 3 the robot can answer queries such the following:

- What is the effect of an action?
- Which action lead to the desired outcome?
- Which action caused the current circumstances?
- Why should an action being performed?
- How should an action be parameterized to yield a different effect?

To this end, we have designed and implemented a framework for the acquisition of task knowledge. In a virtual manipulation environment users can play games using a data glove. The gaming data is stored virtual knowledge base. Through the realized framework data in the knowledge base can be retrieved by semantic queries in a first-order language.

The remainder of the chapter is structured as follows. We first reconsider the problem of making pancakes in Section 6.2. In Section 6.3.1, we give an overview of the overall framework for the acquisition of task knowledge through Games with a Purpose before we describe the virtual manipulation environment in more detail in Section 6.3.2. We explain how virtual manipulation tasks are represented using interval-based first-order abstractions called timelines, and how these are grounded within the data structures of physics-based simulations in Section 6.3.3. We describe two realized games in which users have to perform a pouring task and evaluate the thereby acquired data in Section 6.4. Finally, we present and discuss related work in Section 6.5 before we conclude in Section 6.6.

6.2 Revisiting the Example of Making Pancakes

In their daily routines personal robot assistants are supposed to accomplish novel tasks for which they have not been pre-programmed in advance. In this work we take the task of making a pancake as our running example. Tenorth et al. (2010b) demonstrated how robots can extend their task repertoire by extracting natural language step-by-step descriptions from the Web and translating them into well-defined executable plans. The Web instructions for making a pancake read as follows:

1. pour the pancake mix into a pan
2. flip the pancake using a spatula
3. place the pancake onto a plate

Figure 6.1 shows a hierarchical plan using primitive actions from the robot's action library. In (Beetz et al., 2011), we conducted an experiment where our robot Rosie actually performed the task of making pancakes on the basis of such a plan (Figure 6.2).

The natural language instructions are descriptive enough for humans to understand the task. However, for robots these instructions are highly underspecified. That is, within the experiment many parameters of the plan were determined and specified by programmers. But in principle, a robot has to infer the appropriate parameters of these actions by other means. By observing humans performing the task the robot can estimate some

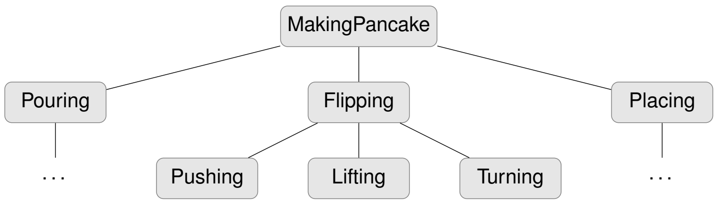


FIGURE 6.1 Ontological representation of an action plan for making pancakes.

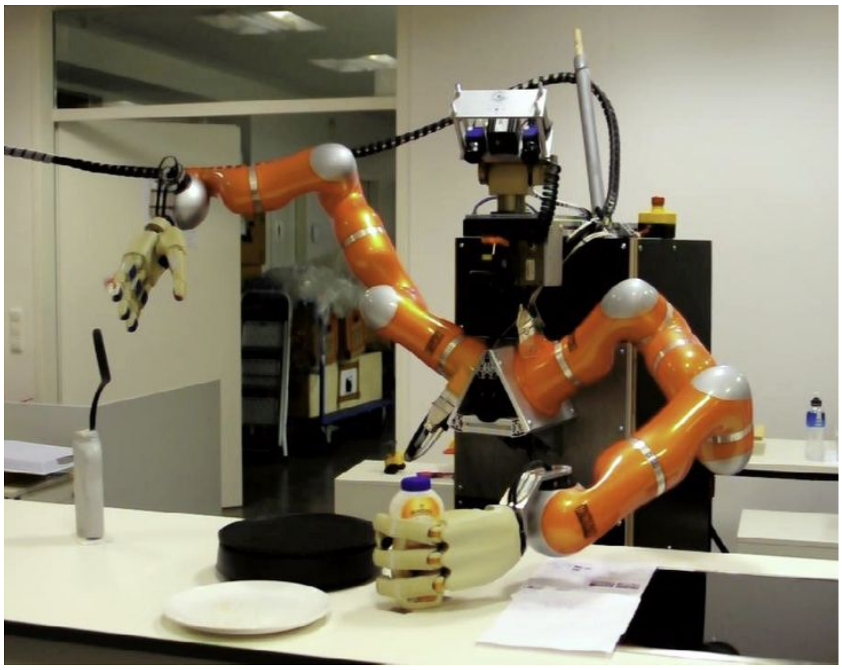


FIGURE 6.2 The robot Rosie preparing a pancake.

of the missing parameters. For example, the robot could estimate parameters like height and angle of the container while the pouring action is performed. Also the duration of this action could be estimated. Such information could be extracted from instruction videos retrieved from the Web or from a human tracking system (Beetz et al., 2009). Since our goal is to acquire a deep understanding of the physical effects of such manipulation actions, we propose to acquire such task-related knowledge based on games played in a physics-based simulator. When considering, for example, the pouring action we would like to answer questions such as

- how much mix was spilled during the game?
- how much was poured onto the pancake maker?
- did it form a proper pancake?
- how much mix was left in the original container?

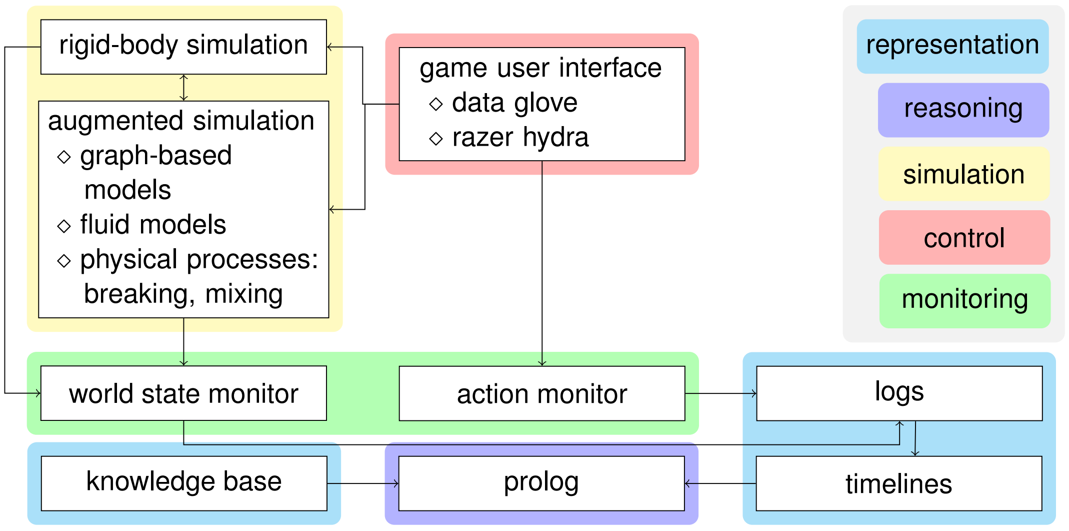


FIGURE 6.3 Framework of the virtual manipulation environment.

- how long did the user hold the container over the target?
- at what height?
- at what angle?

6.3 Framework Design

First, the section gives an overview of the overall framework for the acquisition of task-knowledge through Games with a Purpose. Second, we explain the virtual manipulation environment and its user interface. Third, we describe how the monitored and logged data structures of the simulator serve as a virtual knowledge base for answering queries formulated in a first-order language. Finally, we illustrate the processing steps of the framework by an example.

6.3.1 Overview

Figure 6.3 gives an overview of the overall framework for the acquisition of task knowledge. The framework consists of two parts: a virtual manipulation environment and a knowledge processing module for the extraction and analysis of the acquired data.

In the virtual environment objects can be manipulated using a data glove and a 3D position sensor where the sensor information is directly translated into a pose and articulation

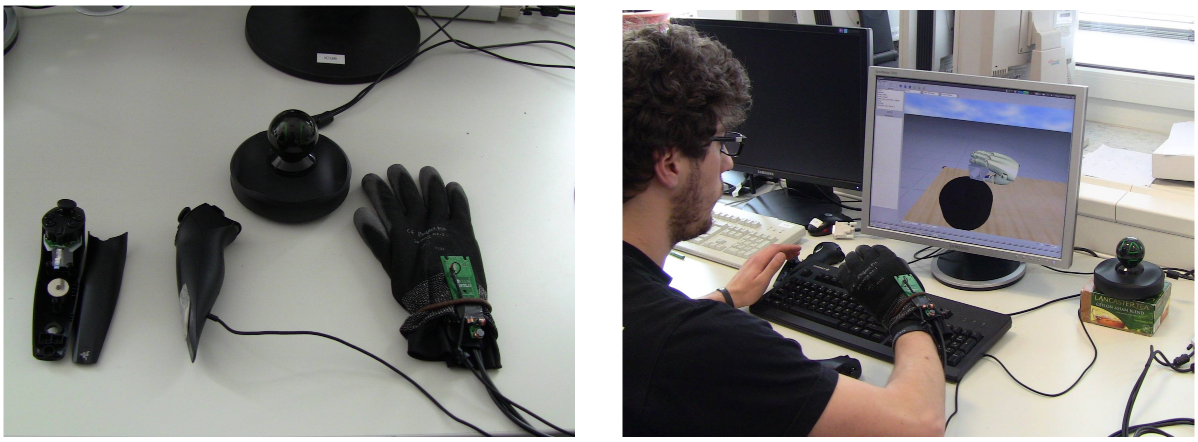


FIGURE 6.4 Virtual Manipulation Environment.

of the simulated hand model. Since we have complete knowledge about the simulated world state we are able to extract different kinds of information of the task-related objects. These information include, for example, an object's position, orientation, linear and angular velocities as well as its bounding box. Also contacts between objects are reported in each time step. In contrast to vision-based systems we do not have to deal with occlusions and other typical problems like the recognition of transparent objects.

The framework, that we have designed and implemented, can be used as a tool for the acquisition of task-related information by logging the internal states of the simulator. The logged simulations are then translated into interval-based first-order representations, called timelines, as described in (Kunze et al., 2011b). By formulating logical queries we can extract task-related information from these timelines semantically. For example, we can query the pose of the container while it was hold by the hand. Then, further methods can be applied on the selected data to analyze the manipulation actions with respect to various aspects.

6.3.2 Virtual Manipulation Environment

The virtual environment is based on Gazebo[1] a 3D multi-robot simulator with rigid-body physics. In the environment a user wearing a data glove controls a robot hand which allows him/her to interact with various objects. Figure 6.4 shows the hardware equipment and a user controlling the robot hand in the virtual environment.

The virtual hand is a simulated version of the DLR-HIT robot hand, described using the Unified Robot Description Format (URDF), which is an XML format for representing a

[1]`http://gazebosim.org`

robot model, and then loaded in the simulator. The hand consists of four identical fingers, each having four joints except the thumb which has an additional degree of freedom for mimicking the human opposable thumb.

The data glove we use for detecting the positions of the finger joints is the X-IST Dataglove. It is equipped with 15 bend sensors (three per finger, one for each joint).

For detecting the absolute position and orientation of the hand, we use the Razer Hydra gaming controller. It has a base station that emits a weak magnetic field, and with the help of the sensors, that were initially integrated in the controllers and afterwards attached to the data glove, we can have a true six degree-of-freedom motion tracking.

The position of the virtual hand is controlled by calculating at each simulation time step (1000 hz) the required linear/rotational velocities that need to be applied on it in order to move to the desired position. Having the difference between the simulated position and the real one a proportional-integral-derivative (PID) controller returns the required velocities in order to have smooth hand movements. For simplifying the controller, the gravity force acting on the hand was disabled, which does not influences its behavior as the inertial forces are still present.

The fingers are controlled in a similar manner. Each joint having a PID controller that takes as input the difference between the virtual and real fingers position and returns the amount of force needed to apply on the joint in order to reach the desired angle.

6.3.3 Representation and Reasoning about Manipulation Tasks

Within the framework, monitors track the state evolution of the simulator and log information whenever there is a difference between two succeeding states. Hence, logged simulations are a sequence of states over time. We basically distinguish between two kinds of states, namely *World states* and *Contact states*. World states comprise the position, orientation, linear and angular velocities, as well as the bounding box of an object at a certain point in time:

World state : $\langle$*time, obj, pos, orient, lin_vel, ang_vel, bbox*$\rangle$.

A contact state holds information about the number of contacts (*num*) between two objects at a point in time. Additionally, it includes the respective forces, torques, and normals for each of these contact points:

Contact state : $\langle$*time, o1, o2, num, force, torque, normal*$\rangle$.

Data structures of the logged simulations are accessed using a predicate, called *SimulatorValue*, as follows:

$$\textit{SimulatorValue}(\overbrace{\textit{position}(o, pos)}^{\textit{Function}}, \overbrace{\ t\ }^{\textit{Time point}}, \overbrace{\ tl\ }^{\textit{Timeline}}),$$

whereby different functions are available for accessing the time-stamped information of the world and contact states.

By using the above predicate, logged simulations are translated into interval-based first-order representation, called timelines. We access and evaluate the timelines from PROLOG by using predicates similar to those in the Event Calculus (Kowalski and Sergot, 1986b). The notation is based on two concepts, namely fluents and events. Fluents are conditions that change over time, e.g., a mug contains a pancake mix: *contains(mug,mix)*. Events (or actions) are temporal entities that have effects and occur at specific points in time, e.g., consider the action of pouring the mix from the mug onto the pancake maker: *occurs(pour(mug,pan))*. Logical statements about both fluents and events are expressed by using the predicate *Holds(f,t,tl)* where *f* denotes a fluent or event, *t* simply denotes a point in time, and *tl* a timeline. The following logical formulas show how the fluent *on* is based on two other fluents, namely *contacts* and *above*, which in turn are grounded in the data structures of the simulator:

$$\begin{aligned}
&\textit{Holds}(\textit{on}(o_1, o_2), t_i, tl) \Leftrightarrow \\
&\quad \textit{Holds}(\textit{contacts}(o_1, o_2), t_i, tl) \wedge \\
&\quad \textit{Holds}(\textit{above}(o_1, o_2), t_i, tl)
\end{aligned}$$

$$\begin{aligned}
&\textit{Holds}(\textit{contacts}(o_1, o_2), t_i, tl) \Leftrightarrow \\
&\quad \textit{SimulatorValue}(\textit{contacts}(o_1, o_2), t_i, tl)
\end{aligned}$$

$$\begin{aligned}
&\textit{Holds}(\textit{above}(o_1, o_2), t_i, tl) \Leftrightarrow \\
&\quad \textit{SimulatorValue}(\textit{bbox}(o_1, \textit{bbox}_1), t_i, tl) \wedge \\
&\quad \textit{SimulatorValue}(\textit{bbox}(o_2, \textit{bbox}_2), t_i, tl) \wedge \\
&\quad \textit{Above}(\textit{bbox}_1, \textit{bbox}_2).
\end{aligned}$$

Using the predicate *Holds_tt* and *SimulatorValue_tt*, instead of *Holds* and *SimulatorValue*, we can even query for a complete time interval throughout a fluent holds.

The following simplified excerpt shows how a pouring action can be defined using the above mentioned language elements. The *pour* predicate is true if there is some mix X inside the *Mug* at the beginning of the timeline and if there is a subset of X, namely Y,

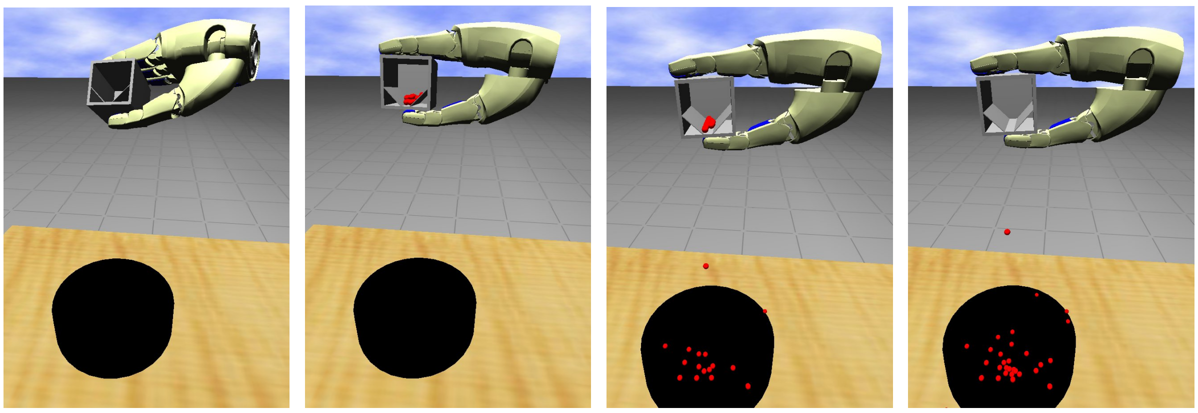


FIGURE 6.5 Virtual Manipulation Task: Pouring pancake mix onto a pancake maker.

inside the *Pan* at the end of the timeline. Note that this predicate does not determine when the action has happened and whether there has been mix spilled onto the table.

```
occurs(pour(Mug,Pan)) :- hasType(Mix,liquid),
                         partOf(X,Mix), subsetOf(Y,X),
                         holds_tt(in(X,Mug),I1,TL),
                         holds_tt(in(Y,Pan),I2,TL),
                         begin(TL,Begin), end(TL,End),
                         starts(I1,[Begin,End]),
                         finishes(I2,[Begin,End]).
```

Similarly, we can formulate first-order queries to retrieve the answers to questions as listed at the end of Section 6.2.

6.3.4 An Example: Acquiring Knowledge of a Pouring Task

In this section we provide an example, how task knowledge can be acquired from execution traces of the virtual manipulation environment.

A user performed a task related to the *making pancakes* scenario, namely pouring pancake mix onto a pancake maker. Figure 6.5 illustrates how the task was performed in the virtual manipulation environment.

By translating the data structures of the simulator into timelines we can use first-order logic to query task-related data semantically. For example, we can ask for pose, velocities, and bounding box of the mug in a time interval when there was a contact between mug and the robot hand as follows:

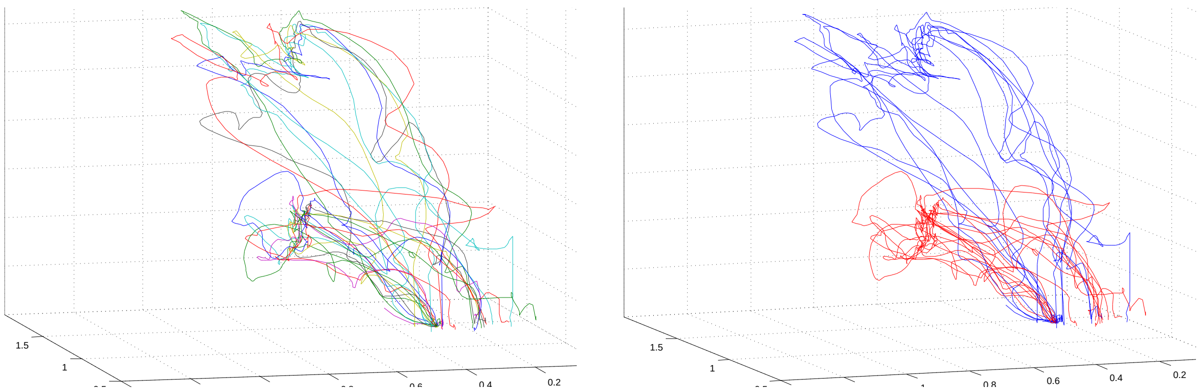


FIGURE 6.6 Trajectories of the mug when it was in contact with the hand. Raw (left) and clustered (right) trajectories after aligning them using dynamic time warping.

```
?- holds_tt(contacts(mug,hand),I,TL),
   simulator_values(position(mug,Ps),I,TL),
   simulator_values(orientation(mug,Os),I,TL),
   simulator_values(linear_velocities(mug,LVs),I,TL),
   simulator_values(angular_velocities(mug,AVs),I,TL),
   simulator_values(bboxes(mug,BBs),I,TL).
```

where I denotes a time interval and the other variables denote lists of their respective data types. Similarly, we can get the last position of the mug in that interval for analyzing where the user has placed the mug after the pouring.

In the experiments liquid was poured from different heights which can be seen by clustering the trajectories (Figure 6.6). We applied dynamic time warping to align the trajectories and then we clustered the trajectories as in (Albrecht et al., 2011).

Logical queries allow us to select data segments of the logged simulations on an abstract level. For example, we can select only the data segments when the mug is over the pancake maker or when it is tilted at an angle in a certain range.

6.4 Experimental Results

We set up two games within the virtual environment for acquiring task knowledge. In both games the task is to pour pancake mix onto a pancake maker. However, the conditions and contexts in the games were varied. For each game we first explain its initial conditions and the task the user has to achieve, and second, we describe how we analyze and evaluate the extracted data.

6.4.1 Pouring without Spilling

6.4.1.1 Overview

Within the first game, the task of the user is to pour pancake mix onto a pancake maker without spilling[2]. The simulation is initialized with a pancake maker and a mug on a table. The virtual robot hand of the user is floating around in the environment. The user has to grasp the mug, move it to a position over the pancake maker, tilt the mug that the mix flows out of the container onto the pancake maker, and eventually put the mug back onto the table. Since the user should not spill anything, he has to be careful while performing the task.

To analyze the behavior of users with respect to the viscosity of liquids we changed the fluidity of the pancake mix within the game.

The model of the liquid has a controller attached that sets a given damping factor to each of its particles. The damping is realized by multiplying the current angular velocity of each particle with one minus the damping factor value (1-8) multiplied by 0.05 at every time step (1000 hz). Hence we have a controllable level of viscosity for the liquid.

We used eight different levels of fluidity. For each level the user has to perform ten trials, that is, 80 trials in total. However, the user does not know the level of fluidity in advance. So, she has to experience it in the first round(s) of each game.

Our hypothesis is that a user would lower the position of the mug while pouring if the fluidity of the mix is increased in order to prevent the liquid to be spilled onto the table. On average we would expect also an increased angle while tilting the mug if the mix has a higher viscosity to increase the flow velocity.

6.4.1.2 Results

In total, we analyzed 80 trials of eight different levels of fluidity with respect to various aspects. The logged data of the individual trials is represented using the timeline data structures which allows us to make queries using a first-order language. For the analysis we basically selected the data from the interval where the user hold the mug over the pancake maker and tilted it by more than 30 degrees. The following query shows how the data was selected where $I1$ and $I2$ denote time intervals on a timeline *TL*, and *Data* includes all information about the mug within a given interval, for example, its position, orientation, linear and angular velocities, and contacts.

[2]Video: `https://ias.cs.tum.edu/~kunzel/videos/exp1-viscosity.mp4`

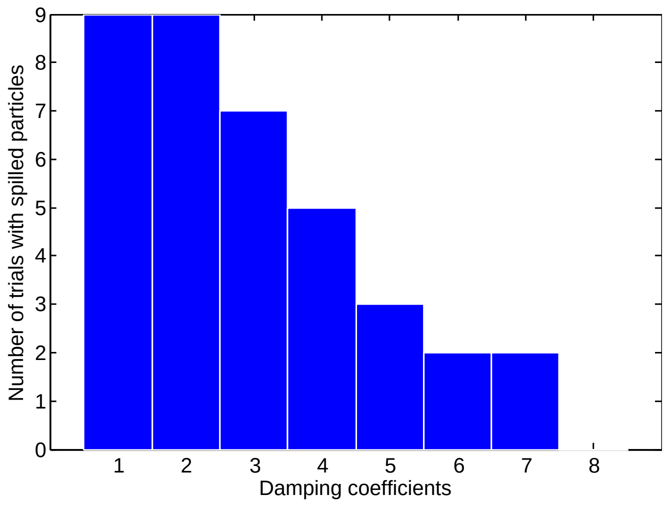


FIGURE 6.7 Number of games with spilled particles for different damping coefficients.

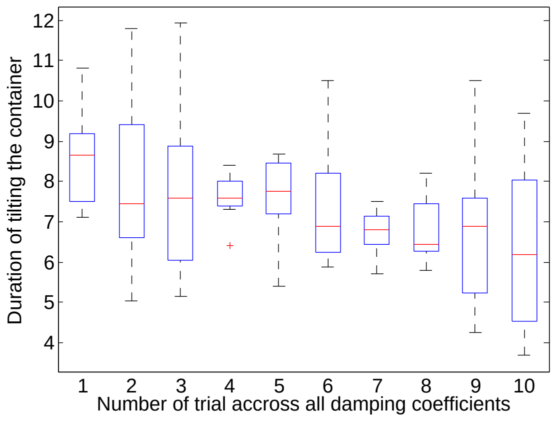
```
?- holds_tt(tilted-X(mug,pi/6),I1,TL),
     holds_tt(over(mug,pancake_maker),I2,TL),
     cooccur(I1,I2),
     simulator_values(data(mug,Data),I1,TL).
```

First of all, we evaluated whether the user spilled liquid onto the table for the different levels of fluidity. Figure 6.7 show that liquid was spilled in almost all trials when the fluidity was high. The number of trials in which the user spilled something decreased when the viscosity increased.

Figure 6.8 indicates clearly how the user became acquainted with the task and optimized his behavior during the ten trials across all damping factors. Both duration and height decrease visibly when the game advances to the next round (1-10).

However, more interesting is how the observed behavior changes for the different levels of viscosity. We analyzed the pose of the mug in the interval when it was tilted over the pancake maker. For each of the ten trials we calculated the arithmetic mean of height and angle during that interval. Figure 6.9 shows that both height and angle generally increase if the viscosity increases. An exception can be seen for the angle with the lowest damping value. Maybe this is due to the fact that this was the first game played by the user.

We also calculated the Pearson product-moment correlation coefficient for the height of the container and the level of viscosity. The *r* and *p* values of 0.968 and 0.0001 indicate that the correlation between the variables is significant.



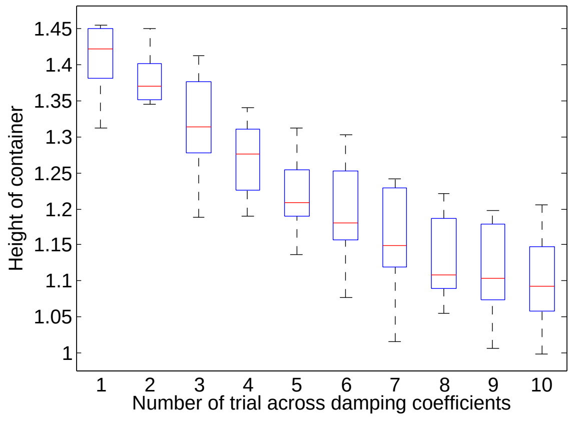


FIGURE 6.8 Left: Duration of tilting the container for the individual trial across all damping coefficients. Right: Height of container with respect to the individual trials.

6.4.2 Pouring the Right Amount

6.4.2.1 Overview

The task of the user in this game is to pour a certain amount of pancake mix onto the pancake maker[3]. Similar to the other game, the simulation is set up with a pancake maker and a container containing some pancake mix. The user should grasp the container with the robot hand, pour a certain amount of its contents onto the pancake maker and put it back onto the table.

Within the game we varied two conditions. First, the user is presented either a mug or a bottle of pancake mix. The opening of the latter is smaller, that is, less liquid flows out of the container while tilting it. Second, we varied the filling level of the container. In total, we looked at four filling levels of the respective containers. The amount of pancake mix the user is asked to pour corresponds to the lowest filling level. Figure 6.10 show the different types of containers and different levels of filling.

By performing the task in various contexts, that is, with different types of containers and filling levels, we want to analyze whether the behavior of users is context dependent. If our hypothesis is true, we should be able to extract parameters like pouring angle, height and time that depend on the task context. Overall the user performed 80 trials, 40 for each container and the different filling levels.

[3]Video: `https://ias.cs.tum.edu/~kunzel/videos/exp2-container-level.mp4`

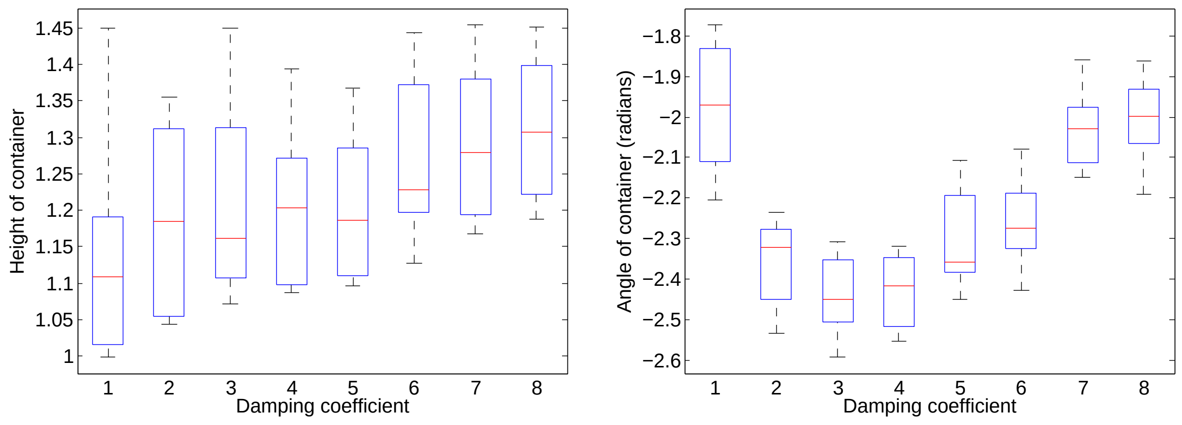


FIGURE 6.9 Left: Height of container for different damping coefficients. Right: Angle of container for different damping coefficients.

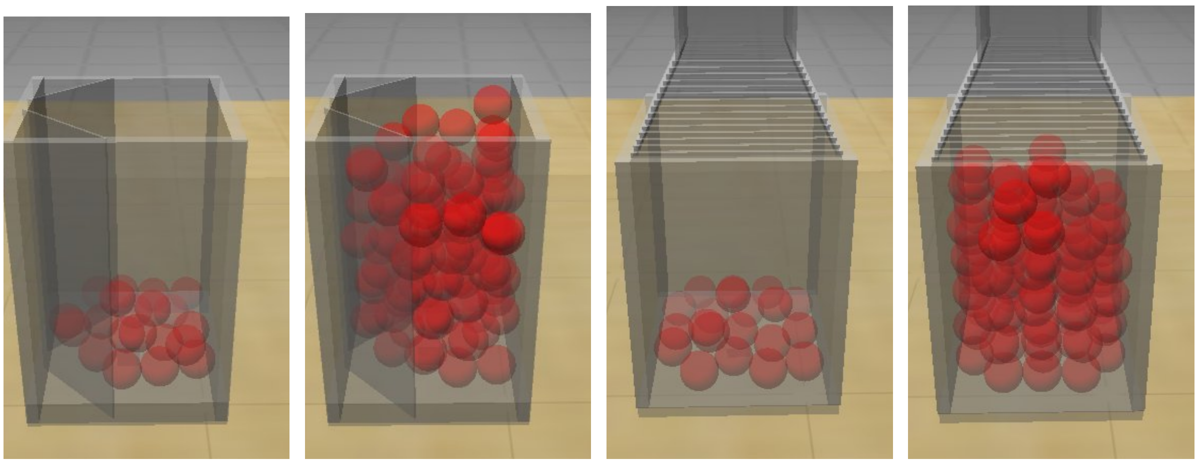


FIGURE 6.10 Different types of containers and different filling levels.

6.4.2.2 Results

In this section we briefly present the results of the second game.

Firstly, Figure 6.11 (left) shows that the user was able pour an amount reasonable close to the target amount of 20 particles onto the pancake maker with both containers at all filling levels. Figure 6.11 (right) indicates that the user generally poured longer when the container was a bottle. This makes sense since the bottle has an opening that is smaller than that of the mug.

Figure 6.12 (left) illustrates that the height for both containers decrease on average when the filling level increases. The Figure 6.12 (right) show an interesting result with respect to the tilting angle of the container. The greater the angle, the steeper the inclination

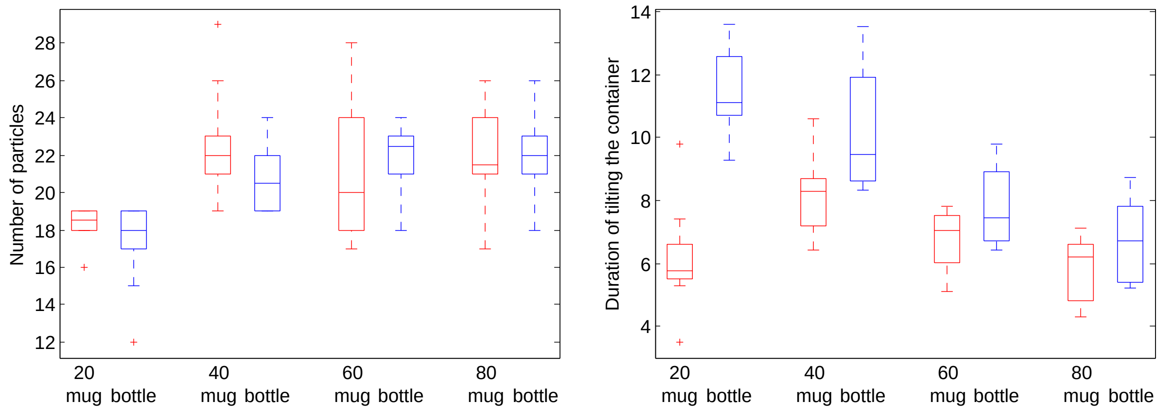


FIGURE 6.11 Left: Number of particles on pancake maker for different types of containers and filling levels. Right: Duration of tilting the container during the pouring action.

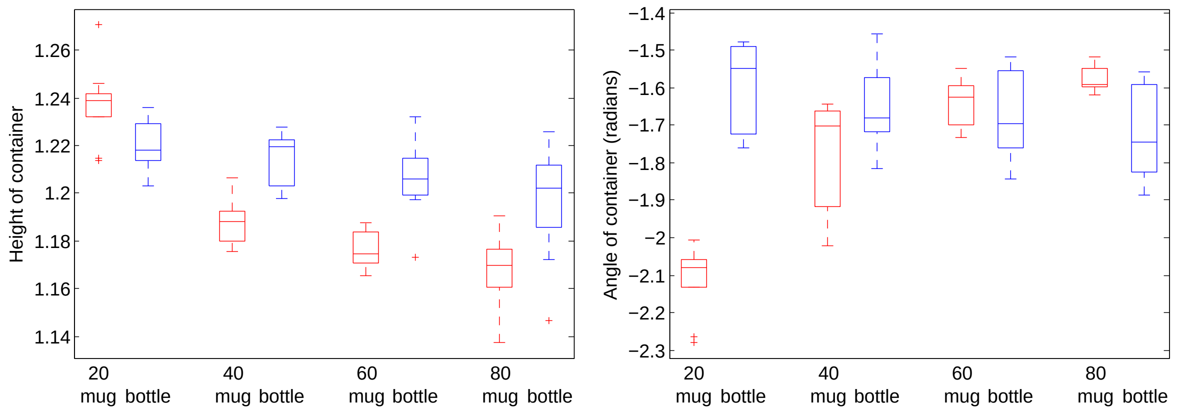


FIGURE 6.12 Left: Height of container. Right: Angle of the tilted container in the main pouring direction.

of the container. We will investigate this behavior by more performing more trials and an in-depth analysis of the data.

6.5 Related Work

Research related to this work stems from a range of areas including physics-based simulation, data acquisition through games, tracking and recognition of human activities, and imitation learning. Therefore a thorough review of the related work is beyond the scope of this thesis.

Within robotics, physics-based simulators are have been successfully used in the context of planning (Zickler and Veloso, 2009), and navigation (Frank et al., 2009). In (Kato et al., 2009), a simulator is developed to gain a deep understanding of cooking tasks.

Games with a Purpose have been used to acquire commonsense knowledge from Internet users (Ahn, 2006). However, most of these games focus on image and natural language tasks.

Beetz et al. (2009) describe an approach of markerless tracking of human motions. Tempting are also technologies like Kinect and its tracking software to capture human motions for games. However, we used a data glove and the Razer Hydra sensor to reliably control the virtual robot hand. Data sets like (De la Torre et al., 2009) provide a wide range of sensor information for everyday cooking tasks.

Work on imitation learning focuses often on the imitation and optimization of observed trajectories (Albrecht et al., 2011). That is, information about the context and the user's intentions are neglected. In line with the present work is (Chella et al., 2006) it also incorporates high-level information about tasks.

6.6 Discussion

In this chapter we have presented an approach for acquiring knowledge about manipulation tasks through Games with a Purpose. Within a game a user is instructed to perform a manipulation action in a virtual environment. In the simulated environment the user manipulates objects by controlling a robot hand with a data glove. The data structures of the simulator are monitored, logged, and translated into timelines. The timelines are queried using a first-order language and the query results are analyzed and interpreted using methods from statistics and machine learning in order to generate informative models for manipulation tasks.

In future work it would be interesting to explore how robots can learn actively missing task knowledge. To this end, one could investigate how the initial configurations of games can be generated automatically based on the perceptual information of perceived situations.

In general, common sense of everyday manipulation tasks has never been described because its information always seemed so obvious. In addition, it would be very difficult to express most of this kind of knowledge by the means of natural language. However, people are able to appropriately apply it when asked to actually perform a task. Therefore we believe, that the acquired task knowledge from GWAP ideally complements the information extracted from Web instructions written in natural language (Tenorth et al., 2010b).

By linking both information resources we are able to ground high-level task instructions in the execution traces of performed actions. For example, based on these grounded data structures robots are able to make decisions about the appropriate parameters of their actions which can impossibly derived from natural language instructions.

The acquired task knowledge from games goes beyond the information that is usually extracted in imitation learning. Whereby imitation learning often learns and optimizes stereotypical trajectories we also consider information about the task context as well as the relationship between objects with respect to spatial and physical aspects during the overall task execution. Thereby we can build models that reflect, for example, the situated context, physical phenomena, and intended goals of a user.

Acquiring data from games has also the advantage that large amounts of information can be crowdsourced from users over the Internet. However, currently our system is limited to a workplace given the equipment requirements.

As a downside of our approach, we would see the problem that the current state-of-the-art robot simulators are not as robust, flexible and easy to use as one would desire. However, we believe that issues regarding performance and usability will be solved by the game industry which heavily employs physics engines in the game development. Another strong indicator that the development of simulation software will get facilitated in the near future is the commitment of DARPA to support the Open Source Robotics Foundation (OSRF)[4] in the development of an open-source robot simulator to be used in the DARPA Robotics Challenge.

Overall we believe that the proposed framework for extracting information from Games with a Purpose can be a useful tool for the acquisition of task-related knowledge that can be applied in many areas of robotics research including execution monitoring, planning, diagnosis, question answering, and learning.

[4] http://www.osrfoundation.org

CHAPTER 7

Learning Action Models from Narratives

In Chapter 2 we have introduced *narratives* as a way to capture and describe the dynamics of manipulation actions over time. We have explained in Chapter 3 how narratives can be formalized using interval-based first-order representations called timelines. In Chapter 4 and Chapter 6 we have presented a framework for envisioning and interpreting everyday object manipulation actions performed by both robots and humans and represent them using timelines. In this chapter we describe how robots can learn compact action models based on information extracted from these timelines which are obtained from robot and human performances.

Learned action models are compact representations of physical behaviors that are generalized from individual cases. Thereby they allow robots to reason about a whole range of problems and their variations. For example, a robot about to make a small pancake has to decide on the appropriate parameters of its pouring action. If it has learned a model that predicts the size of a pancake given a parameterized action, the robot can also apply this model to novel situations in which, for example, it pours the contents of some eggs into a pan for making an omelet. Such models would not only allow a robot to plan its actions, but also to monitor their effects and thereby to detect flaws in the physical behavior.

Another reasoning problem were action models can be applied is question answering in the context of everyday manipulation, as stated in Section 1.3.1. When answering questions, it is often desirable to provide an explanation of why an answer has been given. For example, a robot could explain that it chose a low angle to pour the contents of a container onto the pancake maker, *because* the container had a wide opening and it was rather full. Likewise the robot could explain that it chose a high angle, *because* the bottle holding the pancake mix had a small opening and it was almost empty. In order to provide such explanations, it is necessary that models are represented in an interpretable and meaningful way. Hence, this work focuses on models whose internal structure is represented explicitly by propositions, predicates, and rules using a logical language.

But how can robots actually learn action models of physical behaviors for accomplishing and reasoning about everyday manipulation tasks? A promising way to do so is to look at examples performed by skilled human subjects. Imitation learning analyzes the performances of humans and estimates critical parameters (Schaal et al., 2004; Billard et al., 2004; Azad et al., 2007). The learned behavior thereby copies and imitates the movements performed by humans. Albrecht et al. (2011) use methods from optimization theory to improve learned models in a post-processing step. However, such systems only capture aspects that were present within the example data. That is, examples of exceptional and border cases of manipulation actions, that are essential for generating robust and failure-aware models, are not reflected within the learned models. In this work we consider examples performed by humans as described in Chapter 6 and use them to estimate ranges of important parameters such as the angle and the duration of a pouring action. On this basis we semi-automatically instantiate simulations of robot manipulation tasks and thereby explore the parameter space of an action until its effects change qualitatively, for example, when too much pancake mix is spilled. Finally, we abort the robot simulation if we have explored the local region of the parameter space and its boundaries qualitatively enough.

Another limitation of many approaches in imitation learning is that they do not take intermediate configurations of objects and/or intentions of the agent into account. Work by Chella et al. (2006) addresses this challenge by incorporating symbolic knowledge about occurring actions and states of objects into the learning process. Similarly to their work, we use rich semantic descriptions of scenes (timelines) grounded in the data structures of robot and interactive human simulations to assign a meaning to intermediate situations.

The remainder of this chapter is structured as follows. First, we discuss different learning paradigms in Section 7.1. Secondly, we describe the problem domain and sketch our approach. Thirdly, we give an overview of the learning framework and explain how it processes data of human object manipulation tasks (Chapter 6) and the envisioning framework (Chapter 4) to learn compact action models in Section 7.3. Section 7.4 presents experimental results of learned models. Finally, we conclude with a short summary and a discussion in Section 7.5.

7.1 Learning Paradigms

Before we present our approach on learning physical behavior models for everyday manipulation in Section 7.3, we briefly discuss different types of learning. Basically one can distinguish between three different types, namely:

- supervised learning,
- unsupervised learning, and
- reinforcement learning.

In *supervised learning* a learner basically learns a function that maps an input to a corresponding output based on a set of training examples. If there is only a finite number of discrete output values, the resulting function is called a classifier. In supervised learning it is crucial that input and output of all training examples are known.

Unsupervised learning is mainly used to understand the underlying structure, similarities and patterns in data. In contrast to supervised learning, the training examples only include inputs but no outputs. This type of learning is also called clustering.

In *reinforcement learning* a learner only receives a feedback about its behavior. If the learner receives a positive feedback, its current behavior or strategy gets reinforced; otherwise, it gets downgraded. However, the learner receives no feedback on how to improve its behavior.

An extensive explanation of these different learning paradigms is beyond the scope of this thesis. Therefore, the interested reader is rather referred to a classical text book such as (Russell and Norvig, 2009).

In regards to the problem of “Making Pancakes”, all of the above mentioned learning approaches could be applied in one context or another. For example, unsupervised learning could be applied to distinguish between different types of pancakes. Looking at the resulting clusters a learner could possibly tell pancakes of different sizes and shapes apart. Please note, that an interpretation of the clustering results is always up to the learner. Reinforcement learning could be applied to learn a pouring behavior. Whenever the robot pours something onto the pancake maker without spilling it could receive a positive feedback, otherwise a negative. Thereby the robot could learn an appropriate parameterization of its pouring action. Finally, the robot could employ supervised learning to learn an action model that predicts the size of a pancake given a set of input variables, in particular action parameters and a context.

Although we have seen that all three types of learning could potentially be applied to problems in regards to “Making Pancakes”, in the remainder of this chapter we focus on methods based on supervised learning.

7.2 Problem Domain and Approach

The previous section already mentioned the problem of learning a model for predicting the size of a pancake. In this section we discuss this problem in more detail and sketch a solution to it using supervised learning methods.

Table 7.1 gives an overview of attributes and their respective ranges that are relevant for a pouring action. Some of the attributes are controllable parameters of the pouring action such as *angle*, *time*, and *position*. Others describe the context of the scenario. Context-dependent attributes such as fill level (*particles*) and container type (*container*) are perceptible by the robot, whereas the *viscosity* of the pancake mix is imperceptible. Effects of the action include the *size* of the pancake and the amount of *spilled* particles. Basically all attributes, except *container*, are continuous by nature. However, as motivated earlier, we would like to learn interpretable action models. Therefore, we discretized the ranges of all continuous attributes to nominal concepts. For example, instead of numerical values such as 0.2, 0.8, and 1.4 seconds, we now represent the pouring duration using the concepts *short*, *medium*, and *long*, respectively.

Having laid out a set of logical attributes with their respective ranges as in Table 7.1, we are able to define predicates for specifying the size of a pancake. Let us imagine that we only want to distinguish between *small* and *large* pancakes. Hence, we could specify two predicates as follows:

$$\forall x \; Size(x, Small) \Leftrightarrow P_1(x) \vee P_2(x) \vee \cdots \vee P_n(x),$$
$$\forall x \; Size(x, Large) \Leftrightarrow Q_1(x) \vee Q_2(x) \vee \cdots \vee Q_n(x)$$

whereby $P_i(x)$ and $Q_i(x)$ denote a conjunction of tests based on the logical attributes.

The logical predicates from above can also be represented as a decision list. A decision list is basically a logical expression of restricted form, namely a sequence of logical tests. Figure 7.1 shows an example of a decision list that we have learned on training data obtained through robot simulations. The advantage of such explicit models is that they can directly be interpreted and analyzed by humans. For example, it is very intuitive that *low* pouring angles applied to containers with a small opening (*bottle*) yield only *small* pancakes, whereas *high* pouring angles applied containers with a large opening (*mug*)

TABLE 7.1 Attributes of the *pouring* domain with their respective types and ranges.

Type	Attribute	Range	Description
Action Parameter	*Angle*	*Low, Mid, High*	The angle at which the container is hold during the pouring action
	Time	*Short, Med, Long*	Denotes the time during which the container is hold at a certain angle
	Position	*Left, Behind, Above, ...*	The position with respect to the pancake maker
Context (*perceptible*)	*Particles*	*Few, Many*	Number of particles, i.e., the amount of pancake mix in the container
	Container	*Mug, Bottle*	The type of the container
Context (*imperceptible*)	*Viscosity*	*Low, Mid, High*	The viscosity of the pancake mix
Effect	*Size*	*Small, Medium, Large*	The size of the pancake (particles that are on the pancake maker)
	Spilled	*Small, Medium, Large*	The amount of pancake mix that has been spilled after the pouring action

yield *large* pancakes. A robot could use a decision list that predicts the size of a pancake, for example, for monitoring the outcome of an action. Alternatively, it could use a decision list for selecting appropriate parameters that lead to a desired outcome, for example, a small pancake.

After we have explained the pouring domain and have introduced our approach on learning action models from narratives, we continue by describing the overall framework in the next section.

7.3 Learning Framework

In this section we provide a brief overview of the learning framework. Figure 7.2 shows the main components of the system. The data for learning is acquired from two sources: the interactive simulation and the envisioning framework described in Chapter 6 and Chapter 4, respectively. That is, data from both human and robot object manipulation functions as input to the system. As described in Chapter 3, narratives about object manipulation

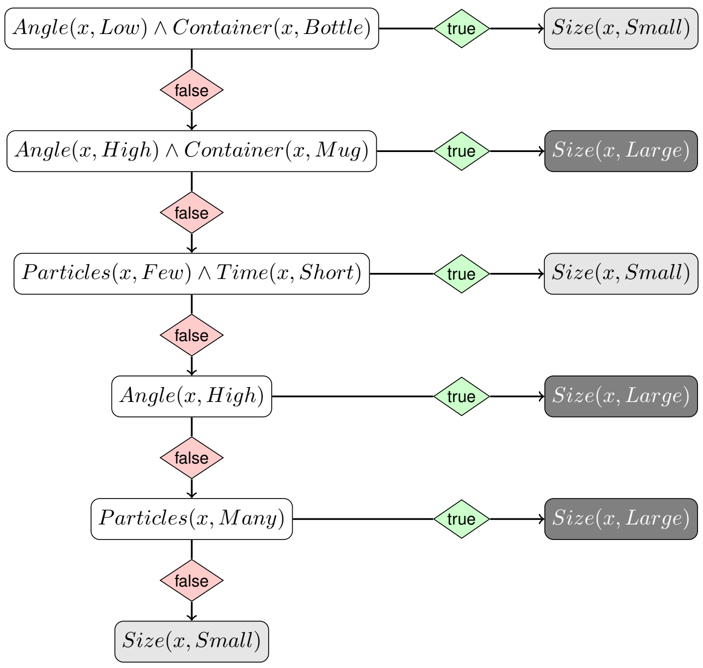
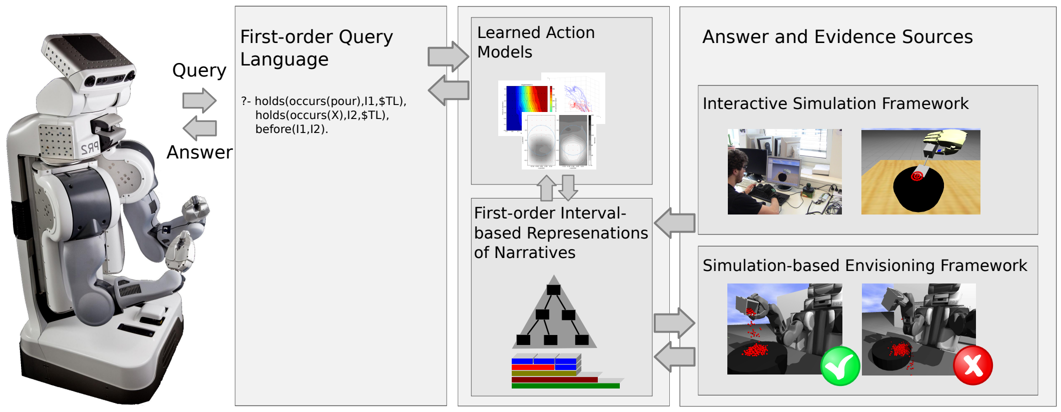


FIGURE 7.1 Decision list predicting the size of a pancake.

tasks are represented as timelines. However, timelines are not used as raw input for the learning algorithms. We rather use specialized predicates that extract relevant information from timelines. For example, in order to predict the size of a pancake we extract the size of the cluster of particles on the pancake maker as well as the angle and duration of the tilting action from a timeline. In addition, we extract context information such as the type of a container and its fill level. This information is then used as the input to different learning algorithms. In this work we used algorithms implemented in the Weka library (Frank et al., 2005), to learn compact models of actions such as decision trees.


Query
Answer
First-order Query Language
Learned Action Models
First-order Interval-based Represenations of Narratives
Answer and Evidence Sources
Interactive Simulation Framework
Simulation-based Envisioning Framework

```
?- holds(occurs(pour),I1,$TL),
   holds(occurs(X),I2,$TL),
   before(I1,I2).
```

FIGURE 7.2 Framework for learning action models from narratives.

7.4 Experimental Results

In this section we present learned action models obtained through the framework as described above. But before we look at the resulting models, we first summarize how the data was acquired.

7.4.1 Data Acquisition

We acquired data from human object manipulation tasks through Games with a Purpose as described in Chapter 6. A human subject manipulated objects by controlling a robot hand in a virtual environment. In particular, the subject performed the task of pouring pancake mix onto a pancake maker under various conditions. In these experiments we mainly paid attention to the movements of the container of pancake mix. That is, we extracted position and orientation information of the container within an interval during which certain conditions hold, namely, when the container was tilted and above the pancake maker. The duration of the interval corresponds to the time span that the container was tilted. The information extracted from the human object manipulation was used to setup the envisioning framework (Chapter 4). Basically, we derived a reasonable range of parameter values for all action parameters from the observed data. In addition, we included more extreme values to cover border cases which had not been observed. Figure 7.3 shows averaged positions in the x-y-plane that the human subject used for pouring the contents of the container onto the pancake maker. In all trials the pouring direction was always towards the center of the pancake maker.

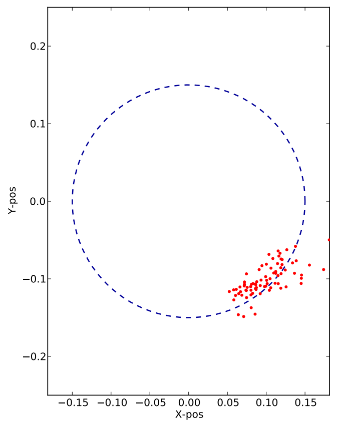


FIGURE 7.3 Averaged pouring positions above the pancake maker extracted from games played by a human subject (Chapter 6). Courtesy: Johannes Mikulasch.

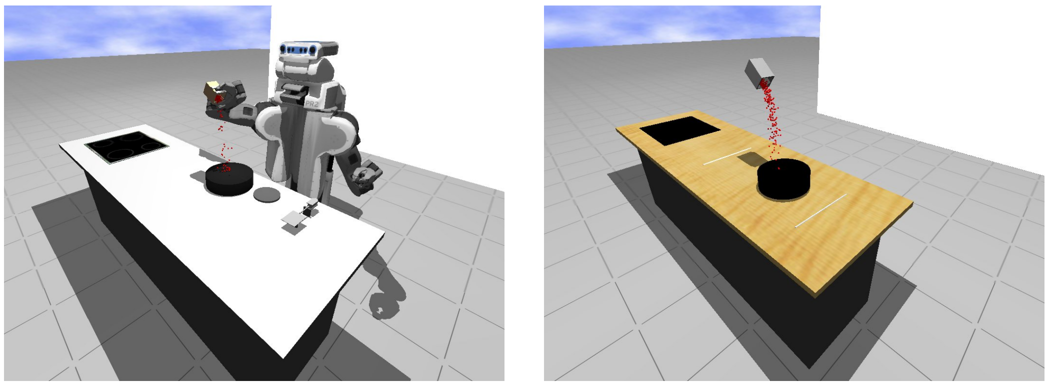


FIGURE 7.4 Simulation of a pouring action. Left: The PR2 pours pancake mix onto the pancake maker. Right: Efficient simulation of the action whereby the PR2 robot is abstracted away.

Using the information extracted from human performances we set up the envisioning framework and carried out hundreds of robot simulations. In order to speed up the performance of the simulator, we abstracted the robot away and directly controlled the container. However, the pouring action was performed as if the robot was present. Figure 7.4 shows images of the simulated pouring action with and without the robot.

Figure 7.5 shows results from experiments in which the position of the container was varied in x and y direction over the pancake maker. The figure shows the amount of pancake mix (particles) that are poured onto the pancake maker. The tested positions (red dots) are organized in a grid structure above the pancake maker. Given the symmetry of the pancake maker, for each of the positions the pouring direction was always pointing towards north. Otherwise we would have duplicated some results. The experiments demonstrate that the averaged pouring positions of the human (Figure 7.3) almost correspond to the optimal positions of the robot (center of the grey-shaded area). The effect of different levels of viscosities on the number of spilled particles can be observed in Figure 7.6. Noticeably, the diameter of the white area, meaning no or almost no spills, increases as the viscosity increases.

7.4.2 Learned Action Models

In this section we describe how we learn qualitative action models on the basis of quantitative robot simulations. First, we present some quantitative results of the simulations, and second, we explain how qualitative action models are obtained through learning.

In the following, we explain how we learn two different models: one that predicts the size of a pancake, and a second that predicts the pouring angle. The input data that we extract

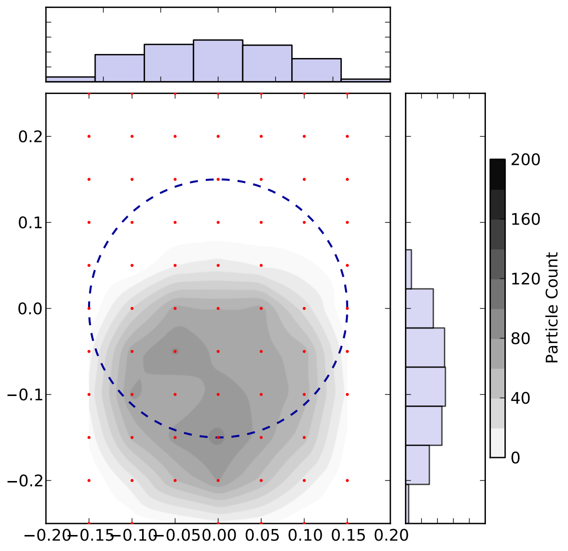


FIGURE 7.5 Amount of particles on the pancake maker while pouring from different positions. Courtesy: Johannes Mikulasch.

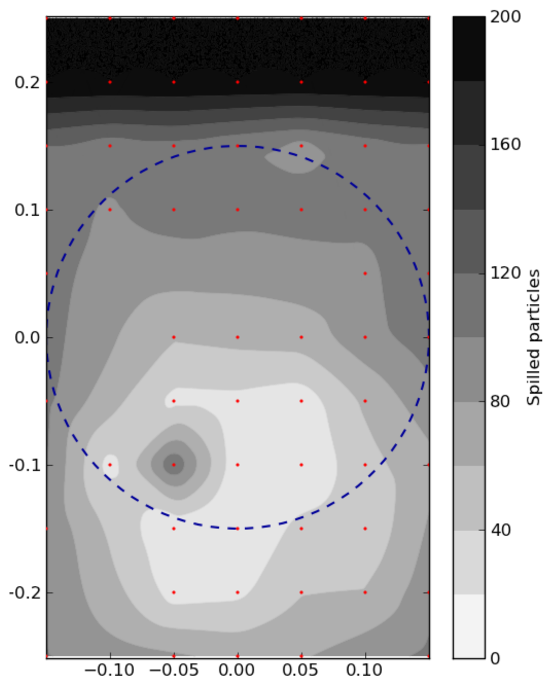
from each simulation run is a tuple as follows:

⟨*container, particles, time, angle, size*⟩

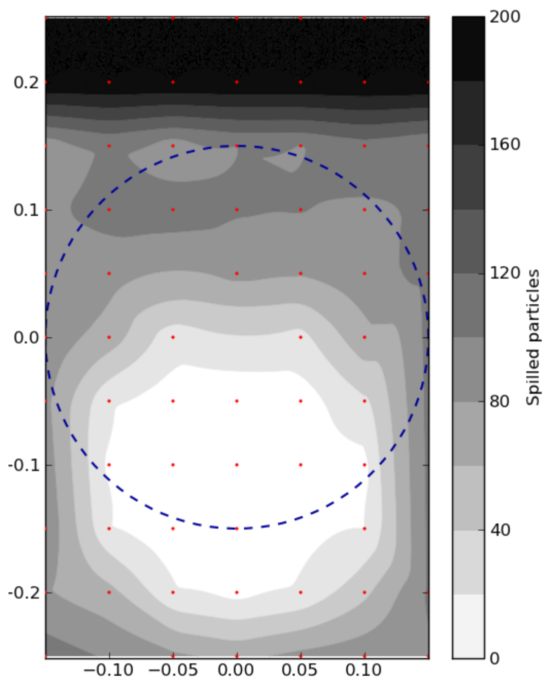
whereby *container* denotes the type of container (*mug* or *bottle*), *particles* denotes the fill level of the container (*few* or *many*), *time* denotes the duration of pouring (*short, medium, long*), *angle* denotes the angle of the container while pouring (*low, medium, high*), and *size* denotes the size of the resulting pancake (*small, medium, large*).

Figure 7.7 shows two situations after a pouring action has been performed using the same parameterization. The left image depicts the situation when a mug was used, the right when a bottle was used for pouring. The distinctive distributions of particles on the pancake maker show how the outcome of a pouring action depends qualitatively on the context, that is, on the type of the container.

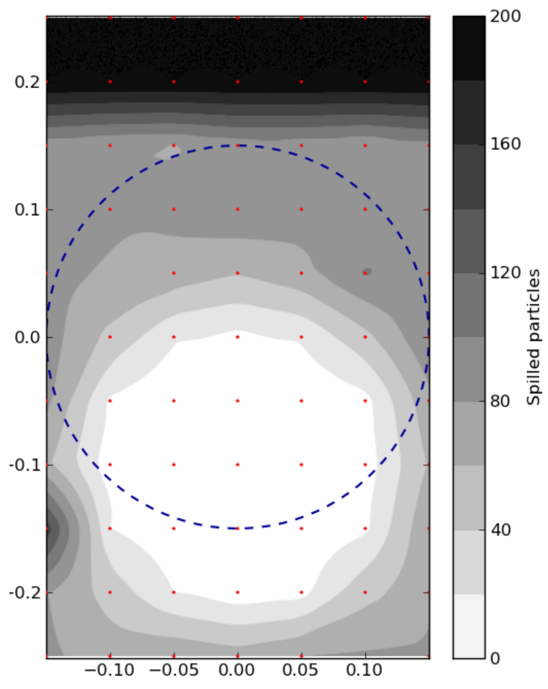
Figure 7.8 visualizes the relation of pouring *angle* and duration (*time*) quantitatively. The top row of the figure shows results when the container contains only a *few* particles (50). The bottom row visualizes the results for *many* particles (200). The left column shows the results for the *mug*, the right for the *bottle*. Looking at the results, it is obvious that the size of a container's opening (*mug* vs. *bottle*) has a dramatic effect on amount of particles that are poured onto the pancake maker. Additionally, the type of the container has also a noticeable effect on the continuity of the function describing the amount of pancake mix. The discontinuity results from the fact that the opening of the bottle occasionally got



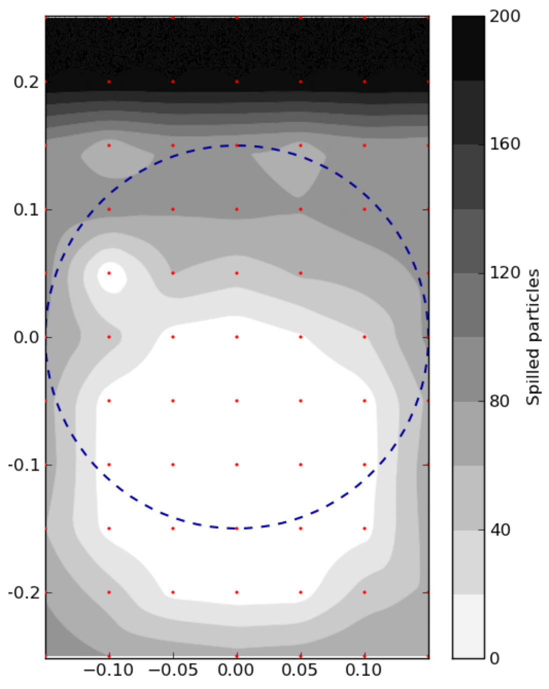
(a) Viscosity level 1 (lowest).



(b) Viscosity level 2 (second lowest)



(c) Viscosity level 3 (second highest)



(d) Viscosity level 4 (highest)

FIGURE 7.6 Amount of spilled particles while pouring pancake mix with different viscosities from different positions . Courtesy: Johannes Mikulasch.

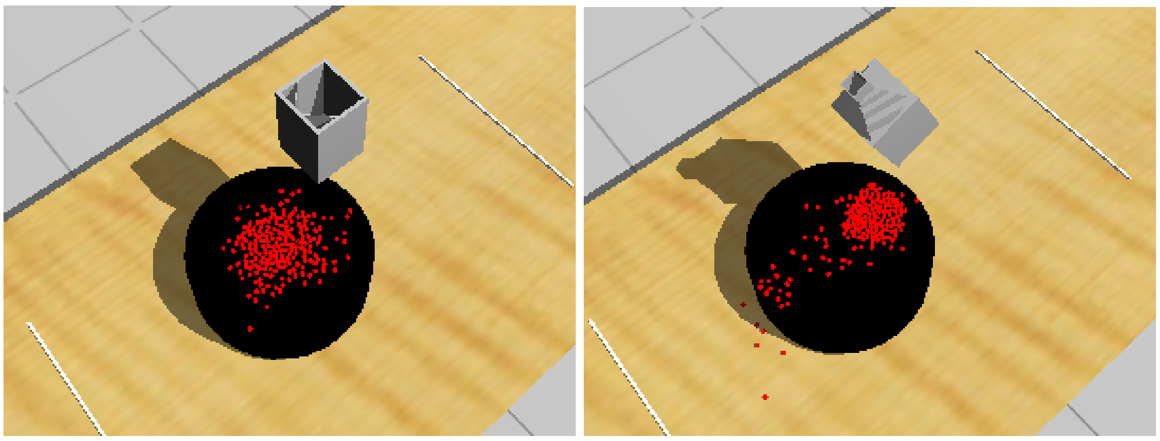


FIGURE 7.7 Different qualitative outcomes of a pouring action. Left: A mug with a large opening. Right: A bottle with a small opening.

TABLE 7.2 Mapping between the qualitative and quantitative size of a pancake.

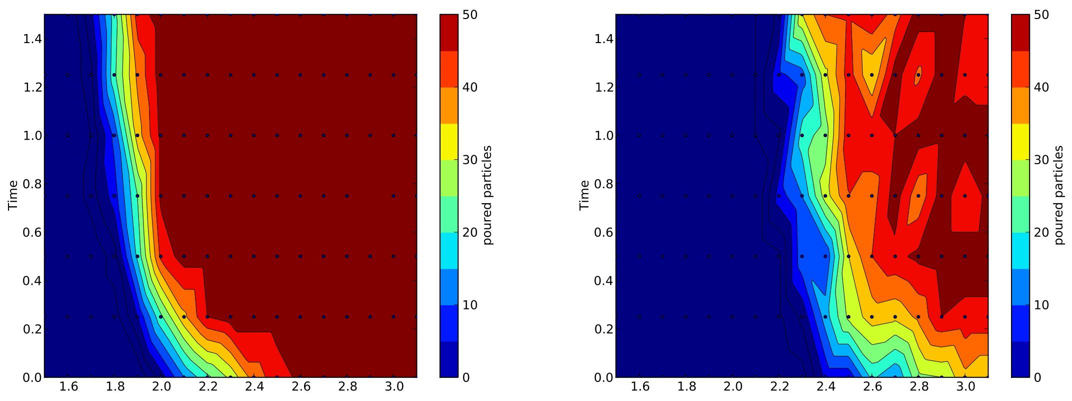
Size	Number of particles
Small	$(0, 66]$
Medium	$(66, 133]$
Large	$(133, +\infty)$

clogged up. Further, it can be noted that a different fill level (*few* vs. *many*) has more impact on the *bottle* than on the *mug*. In general, it can be seen that the pouring *angle* is more important for controlling the amount of pancake mix than the *time*.

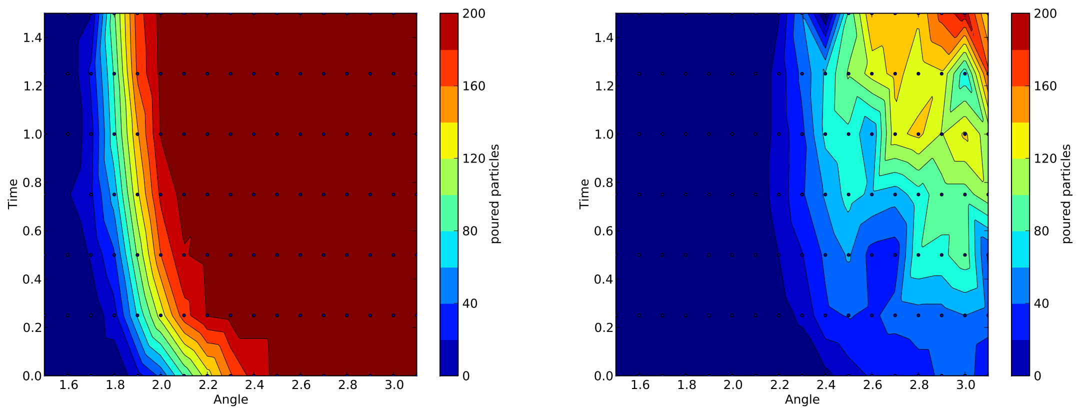
As we have discussed some of the quantitative results, we now proceed by explaining how we learn the qualitative models.

Whenever it is desirable to describe quantitative measurements by qualitative concepts one has to find an appropriate mapping between both. For example, if we want to distinguish between three different sizes of pancakes, namely *Small*, *Medium* and *Large*, we have to provide a mapping that relates, for example, each size to a certain number of particles. Such a mapping can either be based on thresholds or it can be learned using methods we present in Chapter 8. However, out of simplicity, we assume a static mapping as stated in Table 7.2.

As mentioned above, we have chosen a decision tree for the classification of pancake sizes. Although decision trees are rather simple models, they have the advantage that they are interpretable by humans. For learning the size of a pancake we used Weka's *J48* algorithm in its default parameterization. The resulting decision tree is visualized in Figure 7.9. Overall, the learned model achieves an accuracy of 92.41%. That is, out of the 474 instances used for learning, 438 are classified correctly withing the 10-fold cross-



(a) Fill level: 50 particles. Left: mug. Right: bottle.



(b) Fill level: 200 particles. Left: mug. Right: bottle.

FIGURE 7.8 Relationship between pouring angle and time. The pouring results are shown for different types of containers and different fill levels. Courtesy: Johannes Mikulasch.

validation. The most decisive attribute is the fill level (particles). If there are only a few particles available, the robot can only make small pancakes. In case of many particles, the size of the pancake depends first on the type of container, and second on the tilting angle. Only if the container is a bottle with a small opening and the pouring angle is high, the robot can make pancakes of different sizes by varying the pouring duration (time). The confusion matrix in Table 7.3 shows that mainly pancakes of medium size were misclassified.

In a second experiment we have learned an action model for predicting the pouring angle. Here we assumed an alternative mapping between the number of particles and the qualitative size of a pancake as shown in Table 7.4. The reason for using a different mapping was due to an unbalanced distribution of samples in regards to the different sizes of pancakes as can be derived from Table 7.3. Again, we used Weka's *J48* algorithm for learning. The learned decision tree, depicted in Figure 7.10, achieves an accuracy of

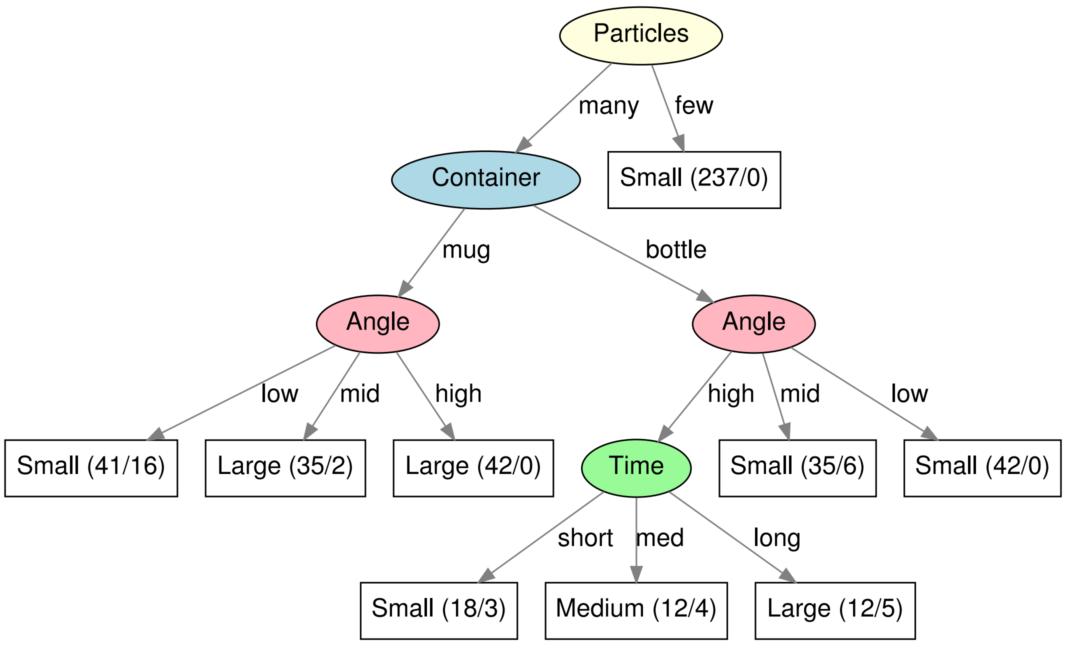


FIGURE 7.9 Decision tree for predicting the size of a pancake. The size is discretized in three classes, namely *Small*, *Medium*, and *Large*. The tree is learned from 474 instances and classifies 438 instances correctly (92.41%) and 36 incorrectly (7.59%).

TABLE 7.3 Confusion matrix for classifying the size of the pancake.

Class	**Classified as** Small	Medium	Large
Small	348	2	0
Medium	17	8	7
Large	8	2	82

TABLE 7.4 Alternative mapping between the qualitative and quantitative size of a pancake.

Size	Number of particles
Small	$(0 - 15]$
Medium	$(15 - 50]$
Large	$(50, +\infty)$

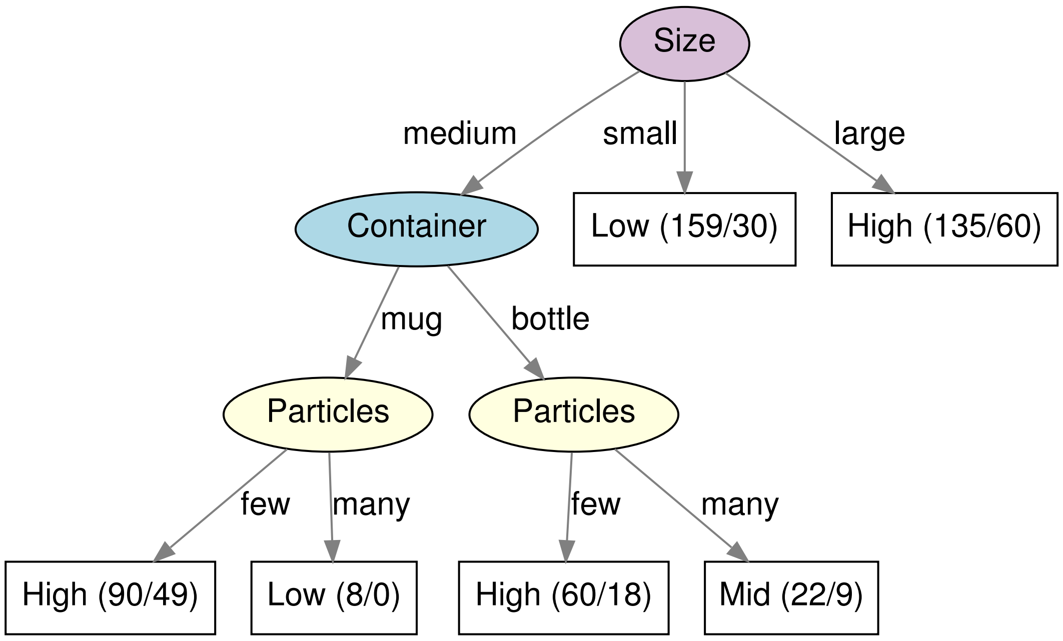


FIGURE 7.10 Decision tree for selecting an angle. The angle is discretized in three classes, namely *Low*, *Mid*, and *High*. The tree is learned from 474 instances and classifies 305 instances correctly (64.35%) and 169 incorrectly (35.65%).

64.35% in the 10-fold cross-validation. A robot can basically use the model for figuring out what angle to use. Given a desired size of a pancake and a context determined by the container's type and its fill level, the robot can infer an appropriate angle. As the confusion matrix in Table 7.5 shows, mainly the *mid* angles are misclassified.

7.5 Discussion

In this chapter we have presented work on learning from narratives. We have motivated that imitation learning often only copies the behavior without considering the underlying structure of tasks. In this work, we have shown how low-level data obtained through Games with a Purpose and robot simulations can be abstracted to learn more descriptive task models. In particular, we have learned decision trees for predicting the size of

TABLE 7.5 Confusion matrix for classifying the pouring angle.

Class \ **Classified as**	High	Mid	Low
High	158	9	0
Mid	100	10	30
Low	30	0	137

a pancake and the pouring angle dependent on the context. However, although the presented models are efficient for reasoning their expressiveness is limited since they are only propositional. For a better integration with timelines, it would be interesting to investigate powerful first-order models such as Markov Logic Networks (Richardson and Domingos, 2006) for learning compact representations of activities as, for example, described by Biswas et al. (2007). Another direction of research one could look at, is a tighter coupling between the data acquisition from human observations and an automatic exploration and exploitation of the parameter space through systematic robot simulations.

CHAPTER 8

Crowdsourcing Common Sense through Games with a Purpose

As we have continually motivated throughout this thesis, autonomous robots need to be equipped with common sense to understand and accomplish everyday manipulation tasks. However, a fair amount of commonsense knowledge is only tacit knowledge, that is, it cannot directly be extracted from any comprehensive source of knowledge such as the World Wide Web. In fact, much of this knowledge has never been described because either its information always seemed obvious or would be difficult to express explicitly. Parts of this knowledge can be obtained by putting humans in situations where they have to apply their common sense. By analyzing and interpreting the resulting performance data, common sense can be revealed and modeled.

In this work we focus on commonsense knowledge about spatial relations between objects. For example, imagine a situation where a robot has perceived various objects on a table as shown in Figure 8.1. After the robot has segmented and detected some of the objects on the table, it is asked to describe the scene semantically relying upon spatial concepts. The robot needs discriminative models of spatial knowledge in order to state, for example, *the blue cup is right of the plate*. In another situation a robot might be asked to set a table for breakfast. The robot receives instructions such as

- *put the cereals next to the milk*, and
- *place the spoon right to the bowl*.

Interpreting these instructions is difficult because they are both ambiguous and context-depended. For instance, the instruction *put the cereals next to the milk* implies a certain relative distance between the cereals and the milk. However, it is not explicitly specified.

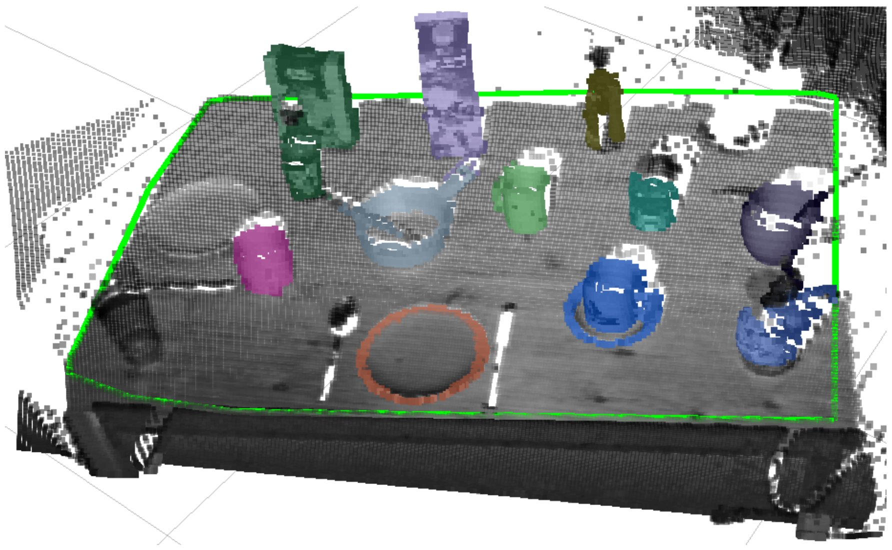


FIGURE 8.1 Segmented and detected objects on a breakfast table. Courtesy: IAS.

Similarly, the area which is characterized by *right to the bowl* is not clearly defined either. This is in particular interesting because the expression *right to the bowl* refers to different meanings (areas) in different contexts. In the context of setting a table, the space denoted by *right to the bowl* is restricted by additional implicit constraints. What this actually means is that the spoon should be placed in an appropriate distance and orientation on the right side of the bowl. In order to come up with potential locations where to place objects the robot needs generative models of spatial knowledge.

As humans are experts in applying their spatial knowledge to everyday situations, we propose a crowdsourcing approach using Games with a Purpose (Ahn, 2006). Games with a Purpose (GWAP) are used to acquire valuable information on the basis of games which would be hard and/or expensive to obtain otherwise. We have designed and implemented a framework that captures human knowledge about spatial relations using GWAP. Figure 8.2 visualizes the overall idea of the framework. Multiple users play games and thereby provide spatial information about everyday situations. This information is stored in a database, analyzed and used for learning compact models that can be employed by robots. Robots can also contribute made-up and/or perceived situations such as shown in Figure 8.1 by uploading these to the game database. Thereby they can actively acquire knowledge about specific spatial aspects.

Within the developed game, a user is confronted with different scenes of a breakfast table. For each scene, the user has to describe the relationships between objects which are stored in a database. Figure 8.3 shows an example of a rendered scene of a breakfast table. In the game, users have to describe the relationships between objects from their

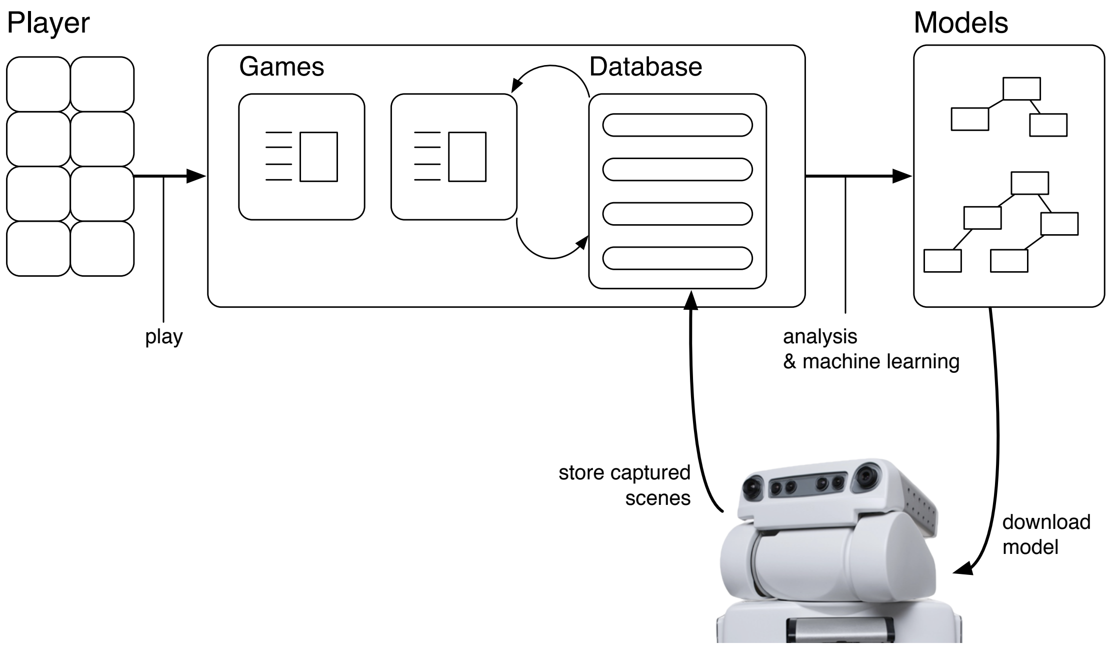


FIGURE 8.2 Crowdsourcing framework using Games with a Purpose: PLAY4ROBOTS. Courtesy: Raphael Teßmer.

own viewpoint. With respect to the figure, a description of the scene could include that the spoon is *in front* of and *close to* the box of cereals and that it is *right, behind* and rather *distant* to the butter dish.

The computer game we have developed is browser-based. Thereby it can be played by many users over the Internet. Hence, the information about spatial relations cannot only acquired from a limited number of well chosen people, but rather from a large number of unknown people, the *crowd*. Hence, we belief that the obtained information should reflect the common sense of people to a reasonable degree.

The remainder of this chapter is structured as follows. First we provide background information and discuss related work about crowdsourcing and reasoning about spatial relations in Section 8.1. Thereafter, we explain the crowdsourcing framework that we have designed and developed in Section 8.2. Third, we present experimental results in Section 8.3. Finally, we conclude with a discussion and an outlook on future work in Section 8.4.

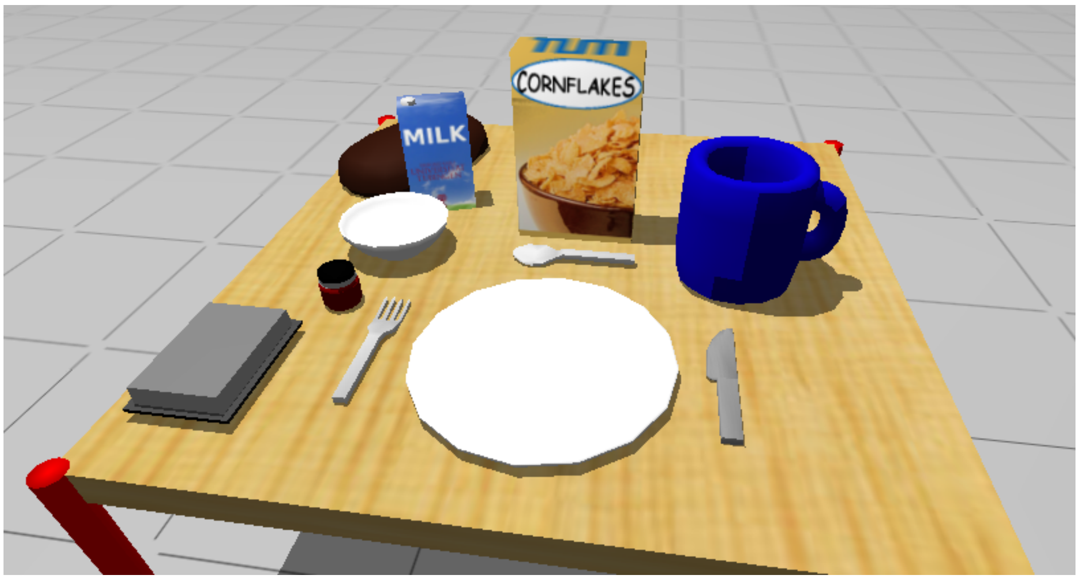


FIGURE 8.3 Rendered scene of a breakfast table.

8.1 Background

8.1.1 Crowdsourcing

The term *crowdsourcing* was coined by Howe (2006b). In a related blog post (Howe, 2006a) to his *Wired* magazine article he gave a definition of the term which reads as follows:

> “Simply defined, crowdsourcing represents the act of a company or institution taking a function once performed by employees and outsourcing it to an undefined (and generally large) network of people in the form of an open call. This can take the form of peer-production (when the job is performed collaboratively), but is also often undertaken by sole individuals. The crucial prerequisite is the use of the open call format and the large network of potential laborers.”

Given this definition, the term crowdsourcing applies to a broad range of activities in which services, information, funding and so on are obtained from a large group of unknown people. People who provide a service can be of any of the following groups: experts, amateurs, or lay men. The type of group basically depends on the task to solve and to whom the call for participation is addressed.

When looking at current crowdsourcing platforms, one can also distinguish between different motivations why people participate in crowdsourcing activities:

- One of the primary motivations for people to participate and contribute in crowdsourcing activities is simply to make money and/or to obtain compensations in other forms such as vouchers. For example, platforms like Amazon's Mechanical Turk[1] pay compensations for accomplished tasks called HITs (human intelligence tasks). Other platforms such as OMICS (Gupta and Kochenderfer, 2004) issued vouchers to very active users who provided information about commonsense facts.

- Another source of motivation is fame. For example, within some art projects of Aaron Koblin[2] people retained their authorship of a created piece of art.

- People also participate in crowdsourcing projects out of altruistic reasons. For example, in the project SETI@Home[3] people download a software package and thereby simply provide computational resources for the analysis of large amounts of data. Other examples include LabelMe[4] where people label parts of images and Comirit Objects[5] where people build simple 3D models of objects just for the sake of science.

- And finally, people participate in crowdsourcing just for fun. "Games with a Purpose" (GWAP) (Ahn, 2006) obtain valuable information about a subject area based on gaming data of their participants. For example, Artigo[6], a social image tagging platform, collects labels for artworks. Similarly, the social language tagging platform Metropolitalia[7] acquires information about regional dialects by mapping Italian language expressions to geographical areas.

In this work we use one particular form of crowdsourcing, namely Games with a Purpose, to acquire information about spatial concepts. We use GWAP because they are an excellent means to acquire information from a large group of motivated people over the Web without paying a compensation. After we have provided some background information about spatial relations in the next section, we continue by explaining the developed crowdsourcing framework based on GWAP in Section 8.2.

[1]https://www.mturk.com
[2]http://www.aaronkoblin.com
[3]http://setiathome.berkeley.edu
[4]http://labelme.csail.mit.edu
[5]http://www.comirit.com/objects
[6]http://www.artigo.org
[7]http://www.metropolitalia.org

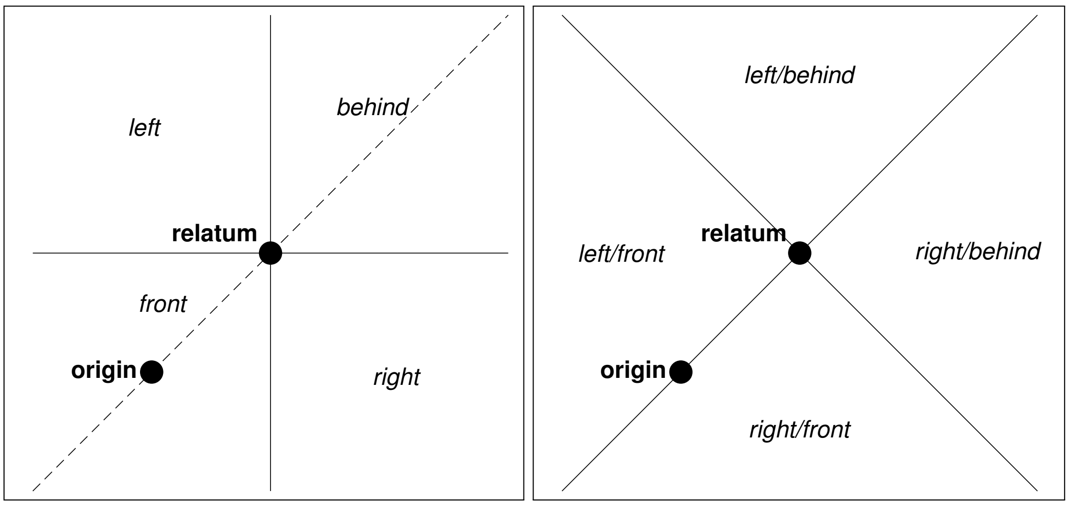


FIGURE 8.4 Partitioning of two different reference systems for relative relations.

8.1.2 Spatial Relations

As discussed in Chapter 3, spatial relations between objects can be grouped into different categories, namely topological, directional, and distance relations. In Chapter 4, we investigated mainly topological relations such as *on* and *in*. This chapter focuses rather on directional and distance relations. However, the presented methods can also be used to acquire information about other spatial concepts.

In their work, Moratz et al. (2003) developed a qualitative positional calculus based on ternary relations for the task of robot navigation. The three positions in the calculus are referred by *origin*, *relatum* and *referent*. The *origin* corresponds to the position of the robot. *Origin* and *relatum* define the reference axis which partitions the surrounding space. Then, the spatial relation is basically defined by the partition in which *referent* lies with respect to the reference axis. Figure 8.4 shows the partitioning of two different reference systems.

In order to determine the directional relation Moratz et al. (2003) calculate the relative angle ϕ_{rel} as follows:

$$\phi_{rel} = \tan^{-1} \frac{y_{ref} - y_{rel}}{x_{ref} - x_{rel}} - \tan^{-1} \frac{y_{rel} - y_{orig}}{x_{rel} - x_{orig}} \qquad (8.1)$$

Figure 8.5 shows which angle is represented by ϕ_{rel}. Basically, it is the angle between the reference axis, defined by *origin* and *relatum*, and the *referent* point. Table 8.1 shows how areas characterized by angle limits are mapped to the spatial relations.

Moratz et al. (2003) use the relative radius r_{rel} to describe the distance between *origin*, *relatum* and *referent*. It is defined as follows:

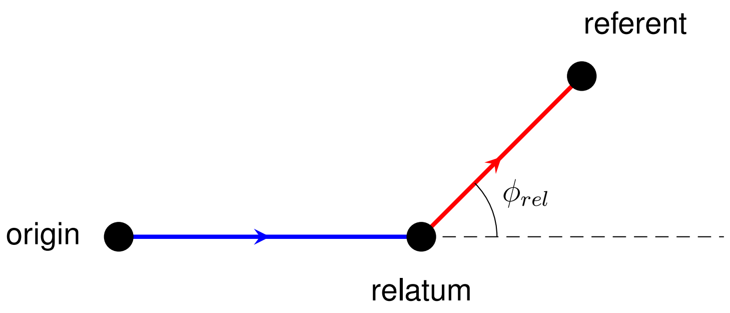


FIGURE 8.5 The relative angle ϕ_{rel} is defined by the reference axis, which is specified by *origin* and *relatum*, and the *referent*.

TABLE 8.1 Partitioning of space for determining directional spatial relations.

Relation	Partition
$referent$ behind $relatum$	$-45° \leq \phi \leq 45°$
$referent$ left $relatum$	$45° < \phi < 135°$
$referent$ front $relatum$	$135° \leq \phi \leq 225°$
$referent$ right $relatum$	$-45° > \phi > -135°$
$referent$ left - back $relatum$	$0° < \phi < 90°$
$referent$ left - front $relatum$	$90° < \phi < 180°$
$referent$ right - front $relatum$	$-180° > \phi > -90°$
$referent$ right - back $relatum$	$-90° > \phi > 0°$

$$r_{rel} = \frac{\sqrt{(x_{ref} - x_{rel})^2 + (y_{ref} - y_{rel})^2}}{\sqrt{(x_{rel} - x_{orig})^2 + (y_{rel} - y_{orig})^2}}. \tag{8.2}$$

The relative radius is calculated as the ratio of the distance between *referent* and *relatum* and the distance between *relatum* and *origin* as visualized in Figure 8.6. If the ratio is smaller than one (< 1) the relation is classified as *close*, otherwise as *distant*.

As presented above, Moratz et al. (2003) have defined partitioning functions based on thresholds for the classification of both directional and distance relations. In this work we investigate how parametric models such as Gaussian distributions can be learned on the basis of crowdsourced information about spatial relations.

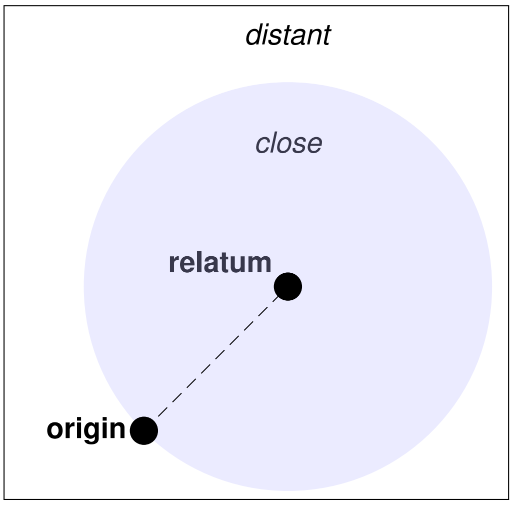


FIGURE 8.6 Distance relations are calculated as ratio of the distance between the *referent* and the *relatum* and the distance between the *relatum* and the *origin*.

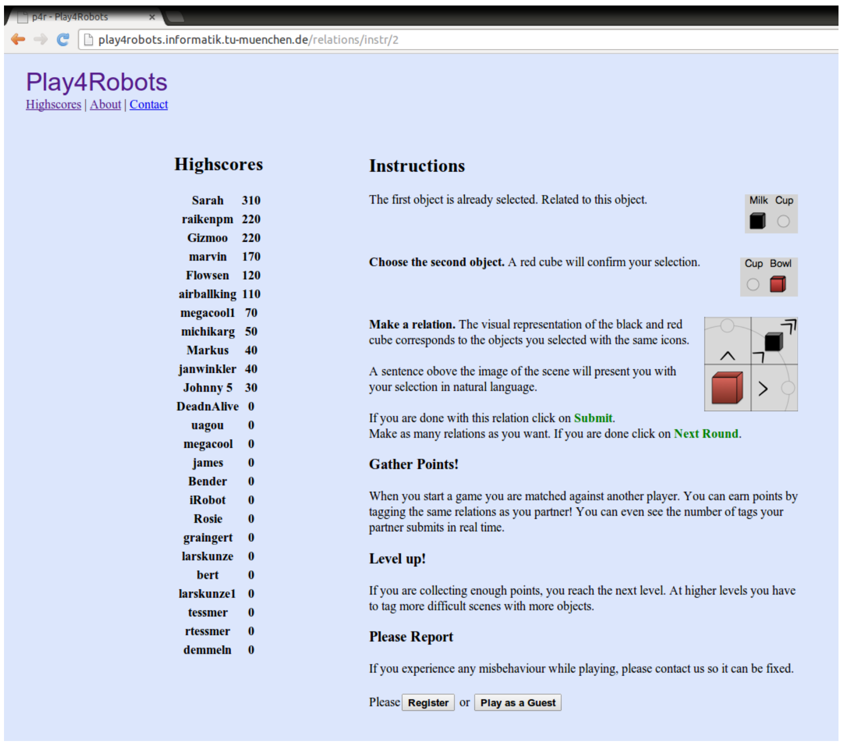
8.2 Crowdsourcing Framework: PLAY4ROBOTS

The overall idea about the crowdsourcing framework has already been presented in Figure 8.2. Users play web-based games in a browser and thereby contribute information about spatial relations. This information is stored in a database and used for learning parametric models. In this section we provide details about the game interface, the game logic and the game database. For more details about the technical implementation of the framework please refer to the work of Teßmer (2012).

8.2.1 A Game Walkthrough

In this section we explain the developed game with a quick walkthrough.

Whenever a user starts a game, the instructions of the game are displayed on the screen as shown in Figure 8.7. In this particular variant of the game a first object (*referent*) has already been selected by the game logic. The user's task is to select a second object (*relatum*) and then to define the spatial relation between these two; assuming that the user's viewpoint is at the camera position (*origin*). Afterwards the user should submit the specified relation. For each scene, the user can submit as many relations as he desires. If the user would like to label a different scene he can press the *Next Round* button. Finally, the user is informed that his game is matched with a partner and that he will only score points if he and his partner submit the same relations for a scene. After having read the game instructions, the user can continue with different options dependent on his status: a



Play4Robots
Highscores | About | Contact

Highscores

Sarah	310
raikenpm	220
Gizmoo	220
marvin	170
Flowsen	120
airballking	110
megacool1	70
michikarg	50
Markus	40
janwinkler	40
Johnny 5	30
DeadnAlive	0
uagou	0
megacool	0
james	0
Bender	0
iRobot	0
Rosie	0
graingert	0
larskunze	0
bert	0
larskunze1	0
tessmer	0
rtessmer	0
demmeln	0

Instructions

The first object is already selected. Related to this object.

Choose the second object. A red cube will confirm your selection.

Make a relation. The visual representation of the black and red cube corresponds to the objects you selected with the same icons.

A sentence obove the image of the scene will present you with your selection in natural language.

If you are done with this relation click on **Submit**.
Make as many relations as you want. If you are done click on **Next Round**.

Gather Points!

When you start a game you are matched against another player. You can earn points by tagging the same relations as you partner! You can even see the number of tags your partner submits in real time.

Level up!

If you are collecting enough points, you reach the next level. At higher levels you have to tag more difficult scenes with more objects.

Please Report

If you experience any misbehaviour while playing, please contact us so it can be fixed.

Please Register or Play as a Guest

FIGURE 8.7 Game instructions.

registered user can simply start playing the game; an unregistered user can either register himself or start playing the game as a guest.

Let us assume that the user starts playing the game. Figure 8.8 shows the main user interface of the game. It is basically divided into three columns. In the leftmost column, the current status of the game is displayed. Every game consist of seven rounds whereby each round lasts at most 50 seconds. As stated earlier, a user can finish a round and immediately switch to the next round by pressing the *Next Round* button. In addition, the current score and all labeled relations are displayed on the left hand side. In the center column mainly the scene is visualized. Above the rendered scene there is a phrase in natural language which reflects the user's input as specified in the rightmost column. In the column on the right hand side, the user has first to select an object which is used as *relatum*. Secondly, the user has to specify a spatial relation such as *left*, *right*, *behind*, *in front*, *close* and *distant* that holds between the *referent* and the *relatum* using the graphical user interface and submit his selection. If the labeled relation matches one of the partner's labels, the user receives some points and the labels are stored in the database. If the user has finished a round he continues with the next one and so on.

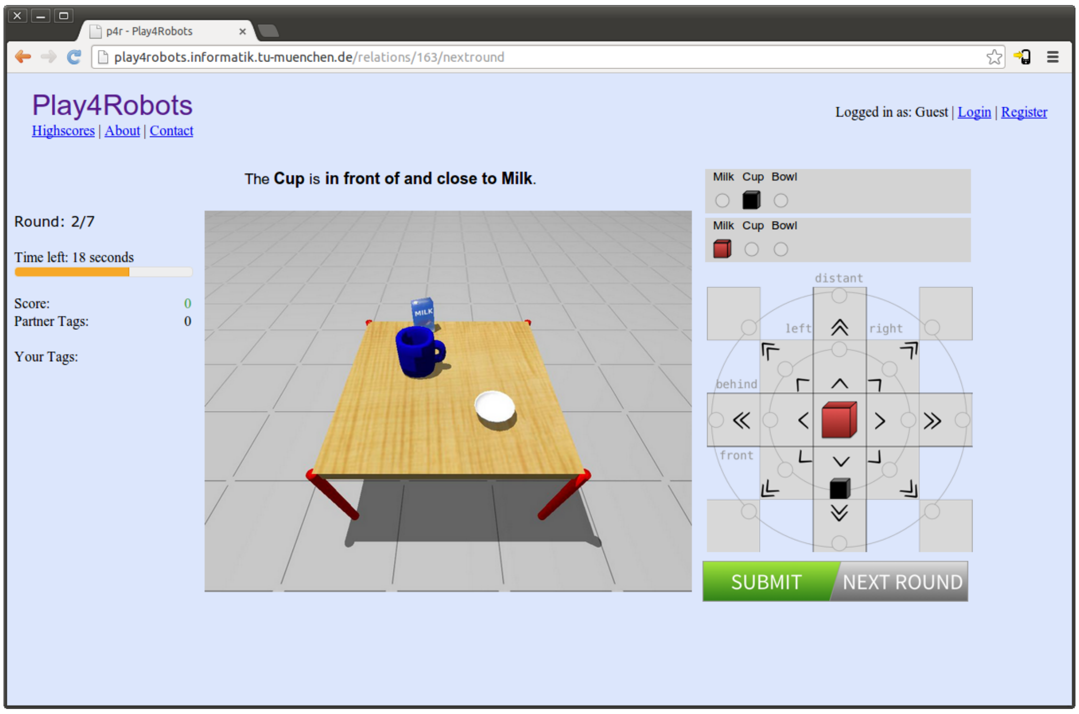

p4r - Play4Robots
play4robots.informatik.tu-muenchen.de/relations/163/nextround
Play4Robots
Highscores | About | Contact
Logged in as: Guest | Login | Register
The Cup is in front of and close to Milk.
Round: 2/7
Time left: 18 seconds
Score: 0
Partner Tags: 0
Your Tags:
Milk Cup Bowl
Milk Cup Bowl
distant
left
right
behind
front
SUBMIT
NEXT ROUND

FIGURE 8.8 PLAY4ROBOTS interface for labeling directional and distance relations of objects in a scene. Q: Where is the *Cup*? A: The *Cup* is *in front of* and *close* to the *Milk*. The user provides its answers by selecting objects and their respective relations to the *Cup*.

8.2.2 Game Logic and Database

Whereas the last section explained the game from the user's perspective, this section provides details about the underlying game logic and data structures of the framework.

The most import data structure in the framework is a *game*. A *game* basically combines all relevant information in one place: the *user*, a *scene*, and its associated *labels*. Within the game play, games are matched with a prerecorded game of a *partner*. Thereby only labels which have already be assigned to a scene are taken into account. This is an important aspect of the game play to keep the number of false positives labels low.

Labels that are associated with a *scene* are basically triples that relate two objects and a spatial relation. A triple stating that a cup is left of the milk looks as follows: *(left, cup, milk)*. Using the interface as shown in Figure 8.8, a user specifies usually multiple relations with a single submission. However, all relations are stored and matched individually. That is, the relation currently selected in the figure would yield the following triples: *(front, cup, milk)* and *(close, cup, milk)*.

A *scene* consists of a number of items that are placed on a table. The scenes for the

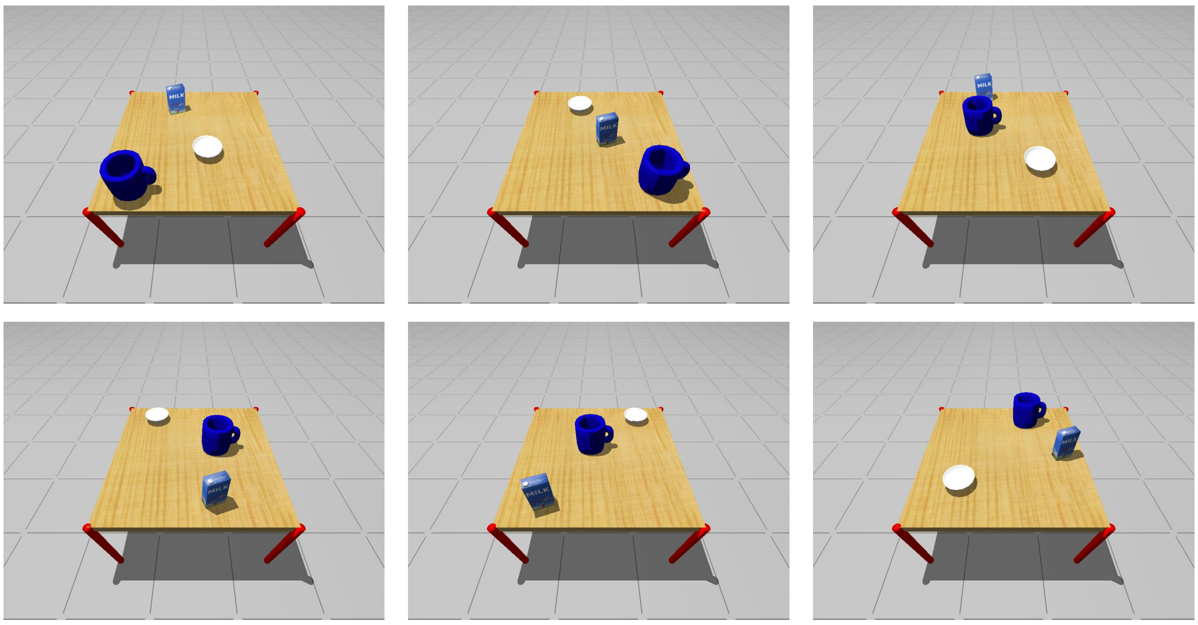


FIGURE 8.9 Scenes from the game database. Users have to answer a question about a particular object. For example, *where is the cup?*

game have been automatically generated using the Gazebo[8] simulator. A script was used to place a set of items (at least three) randomly on the table. Figure 8.9 shows some examples of generated scenes that are used within the game. In order to ground the learned models of spatial relations between objects within geometric data structures, we extract the following information from Gazebo's world state for each item:

- position (x, y, z)
- orientation $(roll, pitch, yaw)$
- bounding box $(min_x, min_y, min_z, max_x, max_y, max_z)$,

and store it along with the rendered scene in the database. Although our current approach only considers the position of objects, it would be interesting to include also the information about an object's orientation and occupied space.

8.3 Experiments

The web-based game for acquiring knowledge about spatial relations was deployed on a publicly available web server[9]. However, since the framework was still under development,

[8]`http://gazebosim.org`
[9]`http://play4robots.cs.tum.edu`

we only made a call for participation within the Intelligent Autonomous Systems group[10], that is, among colleagues and students. In the following, we firstly provide some statistics about the game, and secondly, we present the obtained data and describe the derived parametric models for reasoning about spatial relations.

Game Statistics Within the announced testing phase of two weeks, 24 users have registered and 106 guests participated in the game. However, only 23 of the 106 guests have actively played the game. Overall, 777 games have been played during this period. Within these games 2367 labels have be assigned to the 145 scenes from the game database. We now continue by explaining the obtained and analyzed data.

Spatial Relations As we have motivated in the beginning of this chapter, we would like to acquire commonsense knowledge about directional and distance relations. In Section 8.1 we explained the concepts of *origin*, *relatum*, and *referent* and described how Moratz et al. (2003) use these concepts to determine spatial relations such as *left*, *right*, *in front*, *behind*, *close*, and *distant*.

Figure 8.10 visualizes the labeled data points for the directional relations *left*, *right*, *behind* and *in front* in 2D. Please note, that the visualized data was revised in a post-processing step, because some users interpreted the spatial directions the other way round. On average 11% of the labeled data points have been discarded. The coordinate systems are centered at the position of the *relatum* which is represented by the red circle. The blue circles represent the respective *referent* objects. The *origin*, that is, the user's viewpoint, is not explicitly represented in this figure. It can approximately be imagined at the bottom center of each sub-figure. The mean angles and standard deviations for the respective directional relations are shown in Table 8.2. Moreover, the table lists also artificial values of the model depicted in Figure 8.11(a) for a better comparison. Figure 8.11(b) visualizes normal distributions of the crowdsourced parameters $(\mu_{gwap}, \sigma_{gwap})$. In comparison to the artificial model it can be noted that the angle means of the crowdsourced data reflect the idealized directions very well. However, the learned standard deviations are generally smaller than those of the artificial model. On average the standard deviation is 0.76 radians, which is approximately $\frac{1}{4}\pi$ radians or 45 degrees. This finding provides a plausible explanation for both reference models presented in Figure 8.4. As stated by the three-sigma rule in statistics, about 68.27% of the values lie within 1 standard deviation of the mean. This means that the left model of Figure 8.4 would already cover 68.27%

[10]`http://ias.cs.tum.edu`

TABLE 8.2 Comparison of crowdsourced angles with an artificial model (in radians).

Direction	$\mu_{artificial}$	μ_{gwap}	$\sigma_{artificial}$	σ_{gwap}
Behind	$0\ (0.00)$	0.24	1.00	0.86
Left	$\frac{1}{2}\pi\ (1.57)$	1.58	1.00	0.81
Front	$\pi\ (3.14)$	3.23	1.00	0.59
Right	$\frac{3}{2}\pi\ (4.71)$	4.30	1.00	0.78

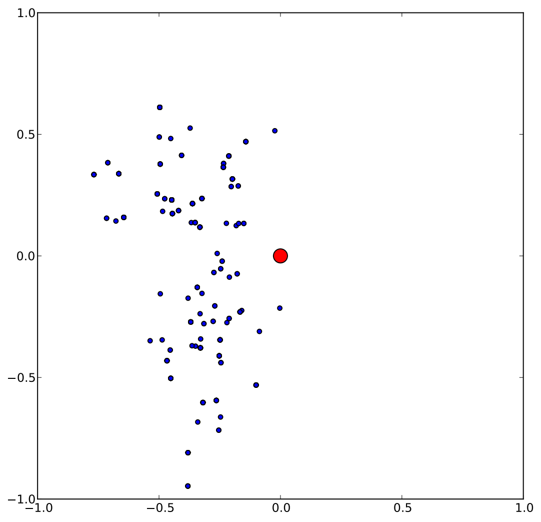
of the data people would classify as a particular direction. About 95.45% of the values lie within 2 standard deviations of the mean, that is, the right model of Figure 8.4 would cover almost all data points.

Overall, we think that the acquired parametric models are reasonable representations for directional relations which can be used for the discrimination as well as the generation of spatial relationships. Moreover, the Gaussian distributions of different directional relations such as *left* and *behind* can be combined for generating a position which fulfills both requirements.

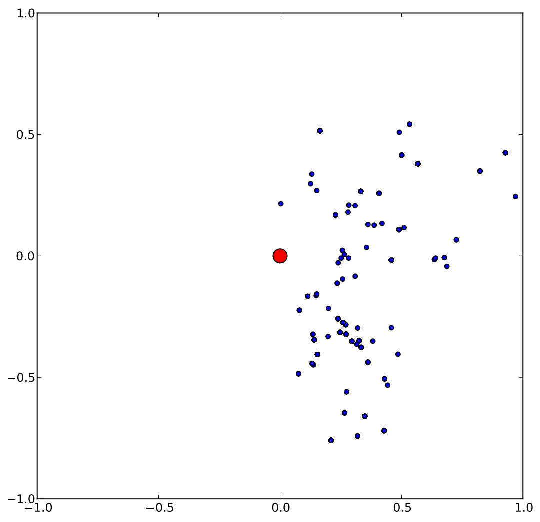
In the following we present the results with regards to the distance relations *close* and *distant*. Figure 8.12 visualizes the labeled data points for both relations. The blue circle in the center of the coordinate system represents the relatum. The red and yellow circles represent *distant* and *close referent* objects, respectively. Some of the data points have actually been labeled as both *close* and distant. An analysis of the absolute distance in meters provides the following numbers: The mean distance of the *referent* object from the *relatum* is 0.42 meters for the *close* relation and 0.66 meters for the *distant* relation. The standard deviation of the *close* relation is 0.14 meters, which is slightly smaller than the standard deviation of the *distant* relation with 0.17 meters. However, it would not be reasonable to only interpret these numbers in absolute terms, because they might depend on the size of the table on which the objects were represented. Therefore we also looked at the ratio between two distance measurements. The distance between *referent* and *relatum* and the distance between *origin* and *relatum*. We have calculated this measure according to Moratz et al. (2003):

$$r_{rel} = \frac{distance(referent, relatum)}{distance(origin, relatum)}$$

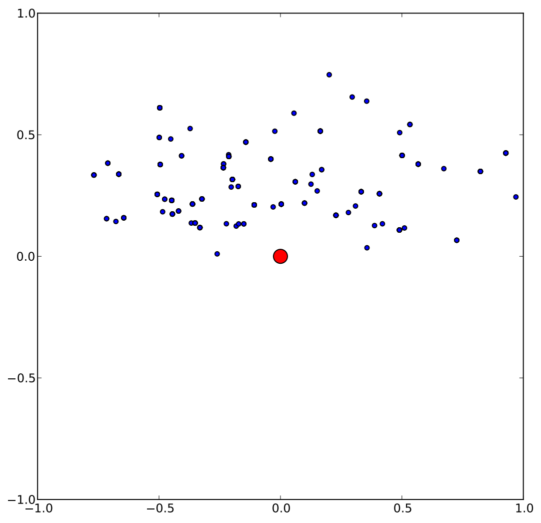
whereby *distance* is simply a function that calculates the Euclidean distance between two points in 2D space. Figure 8.13 visualizes the parametric models based on the relative radius r_{rel}. As can be derived from the figure, the obtained models suggest a different classification boundary between *close* and *distant* as proposed by Moratz et al. (2003).



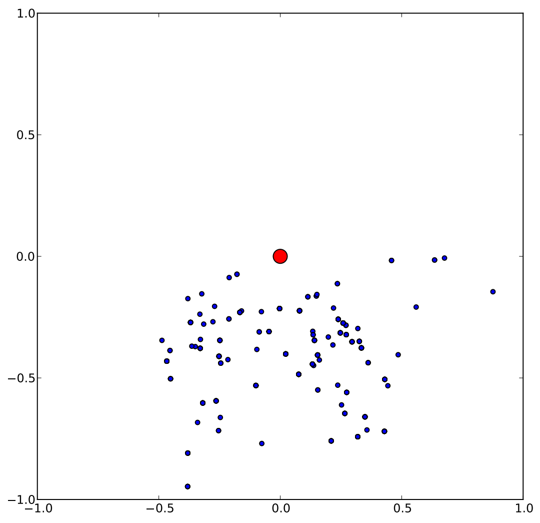
(a) Left.



(b) Right.

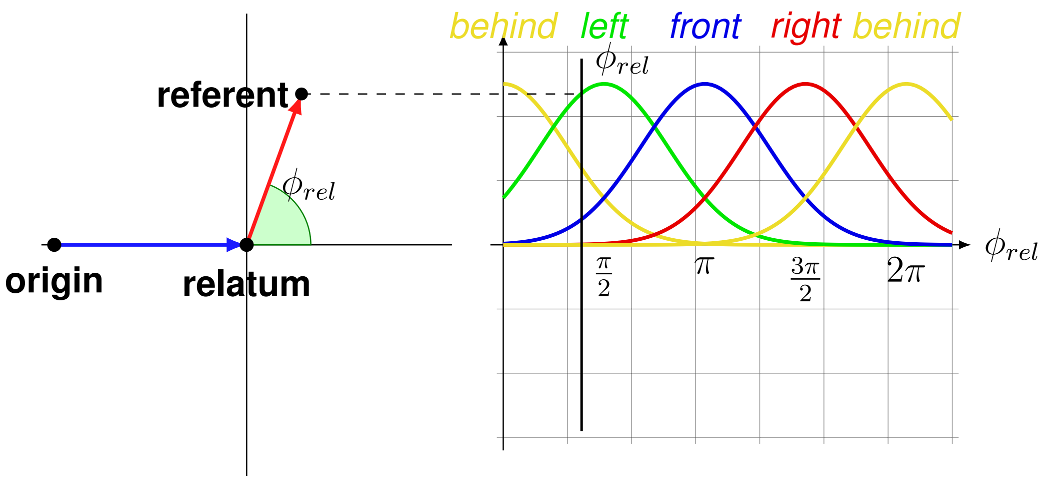


(c) Behind.

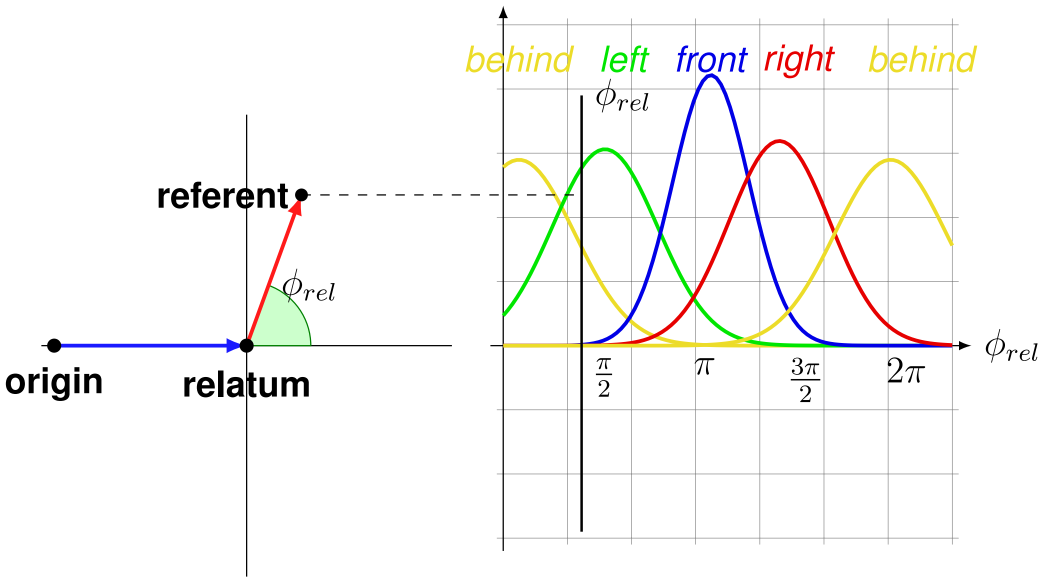


(d) In front.

FIGURE 8.10 Directional relations: left, right, behind, and in front. Courtesy: Raphael Teßmer.



(a) Idealized artificial models.



(b) Crowdsourced models.

FIGURE 8.11 Parametric models of directional relations.

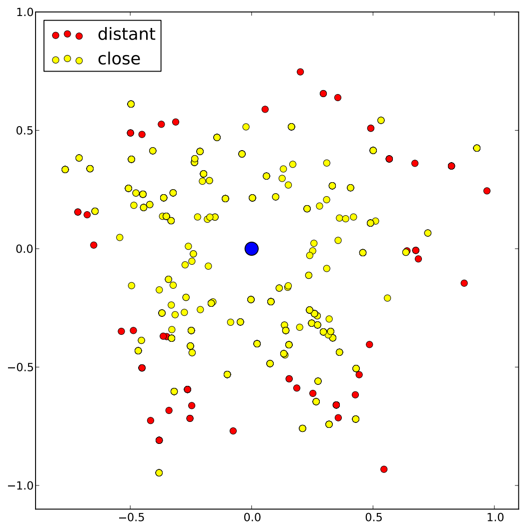


FIGURE 8.12 Distance relations: *close* and *distant*. The blue circle in the center denotes the relatum, whereas the red and yellow circles denote *distant* and *close referent* objects, respectively. Courtesy: Raphael Teßmer.

Whereas the proposed boundary of the relative radius r_{rel} was 1, we found a boundary at approximately 0.75. However, we do not believe that our model is more general than the model of Moratz et al. (2003). We rather think, that our result suggests that a mapping between vague spatial concepts, such as *close* and *distant*, and geometric properties is very context-sensitive. Hence, it would be necessary to learn a spatial model for each particular context of interest. To this end, it would be interesting to evaluate how the size of the table as well as number and the size of the present objects would affect the learned models.

8.4 Discussion

In this chapter we have presented a framework for crowdsourcing spatial knowledge using GWAP. At first, we provided background information about crowdsourcing and spatial relations. Afterwards we gave an overview of our approach based on GWAP. The developed framework, called PLAY4ROBOTS, uses a web-based interface and stores information about object relations provided by users in a relational database. Within the conducted experiment the obtained information about scenes was post-processed, analyzed and used for learning parametric models of directional and distance relations. The learned models can be considered as a proof-of-concept which demonstrates the feasibility of our approach. Given that the learned models are based on the data structures of the Gazebo

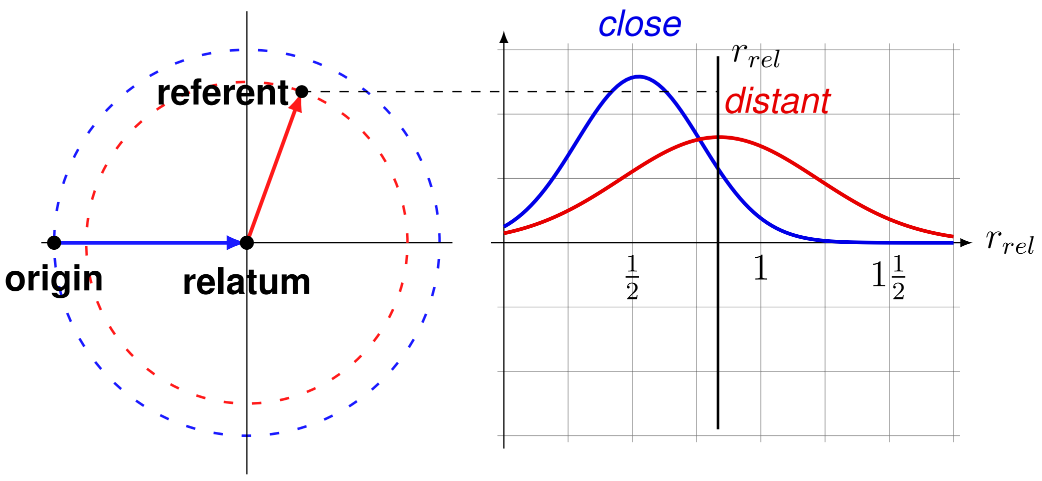


FIGURE 8.13 Parametric models of the relative radius r_{rel}.

simulator, they can directly be employed within the envisioning framework presented in Chapter 4.

However, our results are only a first step into the direction of using crowdsourcing as a means for the acquisition of commonsense knowledge. Overall, we see two major advantages of using GWAP for this purpose. One is the ability to obtain a large number of labeled training examples through the crowd for free of charge. And second, is the possibility that robots can actively learn spatial and other commonsense concepts by making up situations within the simulator and feeding them into the game database. Hence, we belief that PLAY4ROBOTS can be a powerful tool for robots with respect to the acquisition of commonsense knowledge in general.

CHAPTER 9

Conclusions

In this chapter we first briefly summarize the conducted research in Section 9.1. Second, we evaluate the results in regards to the “Making Pancakes” problem and highlight the major contributions of this work in Section 9.2. Third, we discuss potential impacts of this work on Cognition-enabled Robotics in Section 9.3. Finally, we give an outlook on future directions of this research in Section 9.4.

9.1 Summary

The aim of this work is to equip robots with naive physics and commonsense reasoning capabilities. By analyzing the example of “Making Pancakes” in Section 1.2, we illustrated why this type of reasoning is absolutely essential for mastering everyday manipulation tasks. Having motivated that reasoning about robot capabilities, spatial concepts, and actions and their effects are prerequisites for understanding everyday manipulation, we proceeded by describing how we approached this inherently complex problem from various directions in Section 1.4.

However, before we presented our solutions to the problem, Chapter 2 laid out a conceptual apparatus for addressing the problem properly. The concept of a *Scenario* has been introduced for describing an everyday manipulation problem by four components: a *Task*, an *Environment*, a *Robot*, and a *Plan*. The *Task* is a complete and formal specification of a manipulation problem. A *Robot* accomplishes a *Task* by executing a fully instantiate robot *Plan* in a particular *Environment*. Executions traces of the robot as well as physical configurations of objects in the environment are captured within *Narratives*. A *Narrative* is basically a story about a manipulation task that was performed by an agent.

After we presented a common terminology for describing a manipulation problem, we introduced the underlying formal representations used in this work in Chapter 3. We explained how objects and relations among them are represented in space and time using logical representations. We introduced the concept of *timelines* as the fundamental data structure for representing narratives of agent performances in Section 3.5. Finally, we explained how components of robots, that is, sensors, actuators, and control programs, can be represented using the semantic robot description language SRDL in Section 3.6. Furthermore, we showed how these semantic robot descriptions can be matched against formal task descriptions to reason about the capabilities of a robot. Both *timelines* and SRDL are major building blocks on which the contributions of this thesis rely.

In Chapter 4, we presented a framework for envisioning the effects of everyday robot manipulation actions using physics-based simulations. Within this framework, we have designed and implemented components for asserting the initial conditions of a manipulation scenario and for utilizing a simulation-based approach for making temporal projections about parameterized robot control programs. In Chapter 5 we extended the framework and integrated representations and methods for handling fluids in everyday manipulation. Overall, we conducted experiments for various scenarios, namely grasping an egg, mixing fluids, pouring a pancake mix, and flipping a pancake, in which formal parameters of robot control programs were systematically selected from ranges of possible values. These experiments, or more precisely, their resulting timelines, were evaluated with respect to specified performance criteria, e.g. desired and undesired effects. Moreover, we discussed the need for naive physics reasoning in the context of everyday manipulation and provided arguments for basing this kind of reasoning on detailed physical simulations. The simulation-based envisioning framework can be considered as the main contribution of this thesis.

Chapter 6 showed how narratives and timelines are used to capture examples of manipulation tasks performed by humans. We have developed a framework where humans can perform everyday manipulation tasks in a virtual environment. Humans have performed these tasks by controlling a virtual robot hand in an interactive simulator. Within this framework, we have developed two games where humans have to pour some pancake mix onto a pancake maker. We used methods from statistics to analyze and interpret the acquired data from human performances. The contribution of this part of work is a powerful framework for acquiring examples of everyday manipulation actions performed by humans through Games with a Purpose.

In Chapter 7 we explained how robots can learn from information obtained from envisioning (Chapter 4, 5) and observing (Chapter 6) everyday manipulation. We showed how data can be extracted semantically from timelines in order to learn logical representations such as decision lists and/or decision trees. A major advantage of such models is, that their inner structure is explicitly represented using logical predicates. Thereby, logical rules can easily be extracted and/or interpreted. The contribution with regards to learning is an established pipeline that allows robots to learn from robot as well as human performances.

Chapter 8 demonstrated how spatial knowledge can be crowdsourced from Internet users on the Web. In everyday language, the meaning of concepts such as *left of* and *close to* are often vague, ambiguous and/or dependent on the context. Therefore, we have developed a Game with a Purpose to acquire commonsense knowledge about spatial relations over the Web. Users described and assessed spatial relations between objects in artificially rendered scenes showing a breakfast table. Whenever two users agreed on a relation for the same scene, it was stored in a database. Thereby information about spatial relations between objects has been accumulated. Using methods from statistics and machine learning we were able to generate models for spatial relations that are grounded in the data provided by human users reflecting their common sense. The developed framework PLAY4ROBOTS is another important contribution of this thesis which allows robots to acquire commonsense knowledge through web-based Games with a Purpose.

9.2 “Making Pancakes” Evaluated

In Section 1.2 we have presented “Making Pancakes” as a challenge problem for reasoning about everyday robot manipulation. Although it is not the main aim of this thesis to solve this particular problem in all its aspects, we discuss and evaluate how the presented methods address the various subproblems of “Making Pancakes”. First, we look at the original problem. Thereafter, we investigate its numerous variants. Eventually, we highlight the contributions of this work in the light of this challenge problem.

Table 9.1 presents the subproblems of “Making Pancakes” sentence-by-sentence, asks the most relevant questions, and shows how these questions can be answered using the methods developed in this thesis. The table also gives references to the related chapters. Similarly, Table 9.2 steps through and discusses the variants presented in Section 1.2.

Overall, the example of “Making Pancakes” demonstrates how the contributions of this thesis provide answers to questions raised by the challenge problem. We have designed and realized the following components:

- an envisioning framework that allows robots to make temporal projections of parameterized robot control programs (plans) based on detailed physical simulations. The resulting physical effects are abstracted and described qualitatively using logical representations. Thereby, robots can reason about their actions and effects at a symbolic level.

- an interactive simulation framework that allows robots to observe and interpret manipulations episodes performed by humans. The framework can be used to let robots acquire initial information about manipulation actions and/or deepening their knowledge about boarder cases by setting up specialized scenarios.
- a crowdsourcing framework that allows robots to acquire commonsense knowledge from Internet users. In this work, we acquired information about directional and context-depended spatial relations in the context of a table-setting task. However, generally the framework is flexible to collect all kinds of commonsense knowledge through web-based Games with a Purpose.
- a semantic robot description language that allows robots to assess their own capability to perform a certain task and/or action. By matching robot descriptions with task specifications semantically, robots can reason about their about their ability and missing components.

TABLE 9.1 Evaluating Subproblems of "Making Pancakes".

Problem, Questions and Evaluation	**Chapter**
Problem: *A robot pours a ready-made pancake mix onto a preheated pancake maker.* **Questions:** Can the robot pour the pancake mix (at all)? How does it perform the pouring action? **Evaluation:** We can address the first question by reasoning about the robot's capabilities using SRDL. Basically, the robot has to be able to detect and localize a container of pancake mix in the environment, grasp it, and hold it at a certain angle. If it has the capabilities to do this, the robot is generally able to perform the action. A detailed example of a *pick up* action was given in Section 3.6.1.4. The second question asks how the robot exactly executes the action. For answering this questions the robot can envision the task and evaluate the resulting timelines to determine an appropriate action parameterization. It could also reason about how a human would perform the action.	3, 4, 5, 6, 7
Problem: *Properly performed, the mix is poured into the center of the pancake maker without spilling where it forms a round shape.* **Questions:** When is the mix centered on the pancake maker? And when does it have a round shape? **Evaluation:** Again, we can address this questions using the envisioning framework and compute the truth value of the fluents *centered* and *round* respectively. Within the work we used simple heuristics for grounding both predicates in the data structures of the simulator (Section 4.5.2.1). It would also be possible to learn models through crowdsourcing from the Web.	4, 5, 6, 7, 8

TABLE 9.1 Evaluating Subproblems of “Making Pancakes”. *(continued)*

Problem, Questions and Evaluation	**Chapter**
Problem: *The robot lets it cook until the underside of the pancake is golden brown and its edges are dry.* **Questions:** How can the robot recognize that a pancake is cooked sufficiently? **Evaluation:** In this work we ignore the problem of assessing the state of the pancake from the robot’s perception. Within our physical simulation we have used a cooking routine that generates a deformable pancake model given the particles in contact with the pancake maker. The robot itself simply waits for a certain amount of time. It would be necessary ingrate perception routines that assess the state of the pancake from its texture. But in regards to this integration, we refer the reader to Section 9.4.	4, 5
Problem: *Then, the robot carefully pushes a spatula under the pancake, lifts the spatula with the pancake on top, and quickly turns its wrist to put the pancake upside down back onto the pancake maker.* **Questions:** Can the robot flip the pancake (at all)? How does the robot perform the flipping action? **Evaluation:** Again, for answering the first question we can employ the capability reasoning based on SRDL. Similar to the pouring action, we employ our envisioning framework to determine appropriate parameters for the flipping action as we did in Section 4.5.2.2.	3, 4
Problem: *The robot waits for the other side of the pancake to cook fully.* **Questions:** How can a robot detect that the pancake is cooked fully? **Evaluation:** As mentioned above, we do not base the action on the robot’s perception system but rather use a simple waiting operation in the robot’s control program	NA

TABLE 9.1 Evaluating Subproblems of "Making Pancakes". *(continued)*

Problem, Questions and Evaluation	**Chapter**
Problem: *Finally, it places the pancake using the spatula onto an upturned dinner plate.* **Questions:** Can a robot place a pancake on a plate (at all)? How does it perform the action? **Evaluation:** In our implementation the placing action is almost the same as as the flipping action. Only the target location differs. Therefore the same kind of reasoning methods as above are employed.	3,4

TABLE 9.2 Evaluating Variants of "Making Pancakes".

Variant and Evaluation	**Chapter**
Variant: *the robot pours too much pancake mix onto the pancake maker? too little?* **Evaluation:** In case that the robot pours too much pancake mix, it either spills some pancake mix or the resulting pancake is too large to be flipped. In case the robot pours too little of pancake mix, the resulting size pancake might not match the size specified in the task description. Such behaviors can be detected using the envisioning framework.	4, 5
Variant: *the robot pours the mix close to the edge of the pancake maker?* **Evaluation:** We observed problems when flipping the resulting pancake. A robot and/or a human might push the pancake off the pancake maker. To determine whether a pancake is too close to the edge one could employ the crowdsourcing framework. Using this framework we learned models for determining the closeness of objects.	4, 5, 6, 7, 8

TABLE 9.2 Evaluating Variants of “Making Pancakes”.*(continued)*

Variant and Evaluation	Chapter
Variant: *the robot flips the pancake too soon? too late?* **Evaluation:** If the robot flips the pancake too soon, i.e. the cooking routine has not finished yet, the robot pushes the particles off the pancake maker. We have not simulated the burning of a pancake in our cooking routine. Therefore, the robot could not flip the pancake too late in our framework. However, it would be interesting to investigate this in future work.	4, 5
Variant: *the robot pushes only half of the spatula’s blade under the pancake?* **Evaluation:** Again, an answer to this problem can be generated using the envisioning framework.	3, 4
Variant: *the robot turns its wrist too slow?* **Evaluation:** We have observed that the pancake got folded when the robot turned its wrist to slowly. This behavior can also be easily tested and observed in the interactive simulator. Moreover, SRDL allows the robot to reason about its own mechanical specifications such as the forces, torques and limits of its wrist (Table 2.1 and 2.2).	3, 4, 6
Variant: *the robot uses a knife/fork/spoon to flip the pancake?* **Evaluation:** We have not evaluated different tools within envisioning framework. However, we have tested different versions of a spatula within the interactive simulator. In future work, it would be interesting to see how humans would flip a pancake when confronted with different tools such as knifes, forks and spoons.	6

TABLE 9.2 Evaluating Variants of “Making Pancakes”.*(continued)*

Variant and Evaluation	Chapter
Variant: *the pancake mix is too thick? too thin?* **Evaluation:** We investigated this variant in both the envisioning framework and the interactive simulation by adjusting the viscosity of the pancake mix. From the episodes performed by the robot one could observe how the effects change when the viscosity is varied. On the other hand, the episodes performed in the interactive simulation framework show how the human adapt his/her behavior to account for the different levels of viscosity.	4, 5, 6
Variant: *the ingredients of the mix are not homogeneously mixed?* **Evaluation:** We have simulated the mixing of two substances and measured the homogeneity of the resulting mixture. So far, we have not investigated the effects when pouring and/or cooking an inhomogeneous mixture. Whereas one could directly observe the effects of pouring a mixture without any changes to the simulator, we would have to adapt our cooking routine to account for differences in the behavior.	4, 5

9.3 Impact on Cognition-enabled Robotics

We believe that the present work is an important step in extending the reasoning capabilities of cognition-enabled robots in regards to everyday manipulation. For example, through this work robots can envision the outcome of their own manipulations actions, interpret human performances at a semantic level, crowdsource commonsense knowledge from the Web, and evaluate their own manipulation capabilities.

Envisioning is not only useful to determine an appropriate parameterization of an action but also in other contexts. For example, during the execution of an action it is important to predict the expected outcome in order to monitor the progress of an action. In the context of planning it is important to select an action that lead to desired effects. When diagnosing a problem, it is important to analyze what action has caused something to happen. When explaining a procedure, it is also important why an action has been performed and what is its intended outcome. All these examples above illustrate interesting contexts where the method of envisioning can potentially be employed.

Narratives of human manipulation actions can inform robots about intermediate states and configurations of objects within manipulation tasks. Furthermore, they could provide valuable information about the intentions of human agents. By integrating and exploiting this kind of knowledge in imitation learning robots could not only slavishly copy the behavior of humans but rather understand the inherent structure of a task to be learned.

The Web is valuable resource for robots. However, most of the commonsense knowledge of people has never been written down. Therefore dedicated platforms such as OMICS and Amazon Mechanical Turk are needed to accumulate the information about common facts. In the same line, we established the framework PLAY4ROBOTS where Internet users provide information about spatial relations by playing games. Users evaluate scenes of artificial rendered scenes. We believe that the impact of crowdsourcing can be quite huge, if we enable robots to generate novel problems in such a framework. Whenever a robot misses information about a relation and/or it wants to verify a hypothesis, it could generate a set of scenes that will be evaluated by the crowd. By analyzing the provided answers robots could autonomously bootstrap their common sense.

The semantic robot description language (SRDL) originally proposed in (Kunze et al., 2011c) has already been adapted and integrated in the RoboEarth[1] project. Similar to semantic web services, robot descriptions are matched with task specifications in order to assess and possibly update the capabilities of service robots. Thereby, SRDL can play a key role in bridging the gap between abstract tasks specification and concrete robot platforms by grounding high-level instructions into available hardware and software components of robots. Since reasoning about robot capabilities is increasingly important in the areas of cloud robotics and multi-robot scenarios SRDL can also have a considerable impact in these contexts, too.

Given the wide applicability of the underlying ideas and methods developed in this thesis we believe that this work can have a broad impact in field of cognition-enabled robotics.

9.4 Future Directions of Research

The goal of this thesis is to advance the state-of-the-art in regards to naive physics and commonsense reasoning for everyday robot manipulation. And although this work makes considerable progress in this area, it is still a long way towards cognition-enabled robots

[1] `http://www.roboearth.org`

that competently master everyday manipulation tasks. Given the rich variety of common-sense reasoning problems this work can be extended in multiple directions. In the following we discuss some possibilities of future work in each of the main directions covered in this thesis. But first we outline how the developed methods can be integrated into a real robot system.

Integration into Real Robot System The integration of the developed methods into a real robot system would be a logical and important next step to support the findings of this thesis. A major aspect of this integration is a tight coupling between the robot's perception and action routines with its logical abstractions.

The evaluation of the "Making Pancakes" problem in Section 9.2 has already reveled that a grounding of logical formulas with the robot's perceptual system is an open issue. For example, in order to detect that a pancake is ready for flipping the robot should be able to assess the state of the pancake through its perception. That means that a logical representation of the pancake's color, e.g., that it is golden brown, is grounded in the data structures of the robot's perception.

Similarly the action plans of the robot should be integrated with the real robot platform. That is, actions should be both executable in simulation and on the real platform. Within the simulation probabilistic models of sensors and actuators should account for noise and failures of the real robot components. Although many control programs we use within the simulator can be directly employed on the real PR2 platform, we expect that the simulated and the real outcome would differ to some degree. Therefore, it is important that a feedback from the real world is propagated to the simulator whereby it can learn to reflect the actual behaviors as demonstrated in (Johnston and Williams, 2009).

Envisioning The envisioning framework presented in Chapter 4 and 5 can be extended and improved in various ways. Here we elaborate on two possibilities, namely the generality of objects and the selection of parameters values.

Currently the system uses only a fixed set of objects to generate the environment of a manipulation scenario. To cover a greater variety and further types of objects it would be possible to utilize CAD models for their generation. Thereby existing repositories of CAD models such Google's 3D Warehouse[2] could be integrated. Additionally, this interface would also facilitate the integration with the robot's perception.

[2]`http://sketchup.google.com/3dwarehouse`

A second possibility in which the system could be extended is the selection of parameters and their values in the envisioning process. The current approach simply calculates the cross-product of all possible parameter assignments in a plan. A better strategy could already reflect the intermediate results of robot simulations. Generally, the search strategy should make a trade-off between exploitation and exploration of parameter values. Thereby, the parameter space could evaluated more efficiently.

Learning from Narratives As narratives represent performances of robots (Chapter 4) and humans (Chapter 6) in the same way, namely using timelines, it seems reasonable to combine both for learning (Chapter 7). We have already started to determine initial parameters of robot simulations based on human performances, e.g., the pose for pouring. However, a tighter integration of both approaches would be desirable. Robots should learn only from a couple of human demonstrations and then automatically explore the qualitative parameter space in simulation. How the inner structure of manipulation tasks could be exploited in imitation learning is another interesting problem.

Furthermore, action models could be learned using first-order representations such as Markov Logic Networks (MLNs) in a similar way as Biswas et al. (2007). Such representations could better account for actions, objects and their relationships than propositional models.

Crowdsourcing from the Web Crowdsourcing has proven to be successful for information acquisition in various projects such as Amazon's Mechanical Turk[3], LabelME[4], OMICS (Gupta and Kochenderfer, 2004), and this thesis (Chapter 8). However, the effects of crowdsourcing could be enhanced if robots are able to upload their own problems that are to be answered by the crowd. We envision a system where a robot can upload artificially generated and/or perceived scenes from its environment associated with a particular question. For example, which tool to use for flipping a pancake whereby a spoon, fork, and a spatula are shown on a table in a rendered scene. Further, we are interested in how these models can be learned incrementally as more and more information is continuously provided by users. Finally, as another future direction of research one could investigate how to represent and learn context-dependent models of space and actions.

[3]`https://www.mturk.com`
[4]`http://labelme.csail.mit.edu`

Bibliography

Ahn, L. v. (2006). Games with a purpose. *IEEE Computer*, 39(6):92–94.

Albrecht, S., Ramirez-Amaro, K., Ruiz-Ugalde, F., Weikersdorfer, D., Leibold, M., Ulbrich, M., and Beetz, M. (2011). Imitating human reaching motions using physically inspired optimization principles. In *11th IEEE-RAS International Conference on Humanoid Robots*, Bled, Slovenia.

Allen, J. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843.

Allen, M. P. and Tildesley, D. J. (1989). *Computer simulation of liquids*. Clarendon Press, New York, NY, USA.

Azad, P., Asfour, T., and Dillmann, R. (2007). Toward an unified representation for imitation of human motion on humanoids. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Beetz, M., Bandouch, J., Jain, D., and Tenorth, M. (2009). Towards automated models of activities of daily life. In *First International Symposium on Quality of Life Technology – Intelligent Systems for Better Living*, Pittsburgh, Pennsylvania USA.

Beetz, M., Jain, D., Mösenlechner, L., Tenorth, M., Kunze, L., Blodow, N., and Pangercic, D. (2012). Cognition-enabled autonomous robot control for the realization of home chore task intelligence. *Proceedings of the IEEE*, 100(8):2454–2471.

Beetz, M., Klank, U., Kresse, I., Maldonado, A., Mösenlechner, L., Pangercic, D., Rühr, T., and Tenorth, M. (2011). Robotic roommates making pancakes. In *11th IEEE-RAS International Conference on Humanoid Robots*, Bled, Slovenia.

Beetz, M., Mösenlechner, L., and Tenorth, M. (2010). CRAM – a cognitive robot abstract machine for everyday manipulation in human environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan.

Berleant, D. and Kuipers, B. (1992). Qualitative-numeric simulation with q3. In *Recent Advances In Qualitative Physics*, pages 3–16. The MIT Press.

Bicchi, A., Bonsignorio, F., Brändle, A., Burger, W., Christensen, H., Darko, O., Dillmann, R., Eberst, C., Graeser, A., Hägele, M., Khan, A., Kuntze, H. B., Lafrenz, R., Levi, P., Milighetti, G., Ollero, A., Penders, J., Prenzel, O., Siegwart, R., Steinbauer, G., Zielinski, C., Bischoff, R., de Sousa, J. B., Burgard, W., Chella, A., Dario, P., Dias, J., Dindo, H., Frigola, M., Gray, J., Kececi, F., Kraetzschmar, G., Kyriakopoulos, K., Laugier, C., Lima, P., Michel, O., Pegman, G., del Pobil, A., Röning, J., Smithers, T., Vincze, M., and Zimmermann, U. (2007). Research roadmap. Technical Report DR.1.3, EURON – European Robotics Network.

Billard, A., Epars, Y., Calinon, S., Cheng, G., and Schaal, S. (2004). Discovering optimal imitation strategies. *Robotics and Autonomous Systems*, 47(2-3):69–77.

Biswas, R., Thrun, S., and Fujimura, K. (2007). Recognizing activities with multiple cues. In *Proceedings of the 2nd conference on Human motion: understanding, modeling, capture and animation*, pages 255–270, Berlin, Heidelberg. Springer-Verlag.

Bollini, M., Barry, J., and Rus, D. (2011). Bakebot: Baking cookies with the pr2. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), PR2 Workshop*. Extended abstract.

Brown, J., Sorkin, S., Bruyns, C., Latombe, J.-C., Montgomery, K., and Stephanides, M. (2001). Real-time simulation of deformable objects: Tools and application. In *Comp. Animation*.

Chatterjee, R. and Matsuno, F. (2005). Robot description ontology and disaster scene description ontology: Analysis of necessity and scope in rescue infrastructure context. *Advanced Robotics*, 19(8):839–859.

Chella, A., Dindo, H., and Infantino, I. (2006). A cognitive framework for imitation learning. *Robotics and Autonomous Systems*, 54(5):403–408.

Cho, J. H., Xenakis, A., Gronsky, S., and Shah, A. (2007). Anyone can cook – inside ratatouille's kitchen. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*.

Compton, M., Henson, C., Neuhaus, H., Lefort, L., and Sheth, A. (2009a). A survey of the semantic specification of sensors. In *2nd International Workshop on Semantic Sensor Networks, at 8th International Semantic Web Conference,*.

Compton, M., Neuhaus, H., Taylor, K., and Tran, K. (2009b). Reasoning about sensors and compositions. *Proc. Semantic Sensor Networks*, page 33.

Davis, E. (1990). *Representations of commonsense knowledge*. notThenot Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann.

Davis, E. (1997). Cracking an egg. Commmon Sense Problem Page, `http://www-formal.stanford.edu/leora/commonsense/#eggcracking`.

Davis, E. (2008). Pouring liquids: A study in commonsense physical reasoning. *Artificail Intelligence*, 172(12-13):1540–1578.

Davis, E. (2010). Ontologies and representations of matter. In Fox, M. and Poole, D., editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press.

Davis, E. (2012). Qualitative spatial reasoning in interpreting text and narrative. *Spatial Cognition and Computation*. Forthcoming.

de Kleer, J. (1977). Multiples representations of knowledge in a mechanics problem-solver. In *Proceedings of the 5th international joint conference on Artificial intelligence - Volume 1*, pages 299–304, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

De la Torre, F., Hodgins, J., Montano, J., Valcarcel, S., and Macey, J. (2009). Guide to the Carnegie Mellon University Multimodal Activity (CMU-MMAC) Database. Technical report, CMU-RI-TR-08-22, Robotics Institute, Carnegie Mellon University.

Dean, T. (1989). Using temporal hierarchies to efficiently maintain large temporal databases. *J. ACM*, 36(4):687–718.

Faloutsos, P., van de Panne, M., and Terzopoulos, D. (2001). Composable controllers for physics-based character animation. In *Proc. of the 28th annual conf. on Computer graphics and interactive techniques*.

Forbus, K. D. (1984). Qualitative process theory. *Artificial Intelligence*, 24(1-3):85–168.

Forbus, K. D. and Falkenhainer, B. (1990). Self-explanatory simulations: An integration of qualitative and quantitative knowledge. In *AAAI*, pages 380–387.

Frank, B., Stachniss, C., Schmedding, R., Teschner, M., and Burgard, W. (2009). Real-world robot navigation amongst deformable obstacles. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Frank, E., Hall, M. A., Holmes, G., Kirkby, R., Pfahringer, B., Witten, I. H., and Trigg, L. (2005). Weka - a machine learning workbench for data mining. In Maimon, O. and Rokach, L., editors, *The Data Mining and Knowledge Discovery Handbook*, pages 1305–1314. Springer.

Frenkel, D. and Smit, B. (2001). *Understanding Molecular Simulation, Second Edition: From Algorithms to Applications (Computational Science)*. Academic Press, 2 edition.

Galindo, C., Fernández-Madrigal, J., González, J., and Saffiotti, A. (2008). Robot task planning using semantic maps. *Robotics and Autonomous Systems*, 56(11):955–966.

Ghallab, M. (1996). On chronicles: Representation, on-line recognition and learning. In *Principles of Knowledge Representation and Reasoning-International Conference-*, pages 597–607. Morgan Kaufmann Publishers.

Gil, Y. and Ramachandran, S. (2001). Phosphorus: a task-based agent matchmaker. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 110–111, New York, NY, USA. ACM.

Gould, H., Tobochnik, J., and Wolfgang, C. (2005). *An Introduction to Computer Simulation Methods: Applications to Physical Systems (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Griebel, M., Knapek, S., and Zumbusch, G. (2007). *Numerical Simulation in Molecular Dynamics: Numerics, Algorithms, Parallelization, Applications*. Springer Publishing Company, Incorporated, 1st edition.

Gupta, R. and Kochenderfer, M. J. (2004). Common sense data acquisition for indoor mobile robots. In *Nineteenth National Conference on Artificial Intelligence (AAAI)*, pages 605–610.

Haazebroek, P. and Hommel, B. (2009). Anticipative control of voluntary action: Towards a computational model. In Pezzulo, G., Butz, M., Sigaud, O., and Baldassarre, G., editors, *Anticipatory Behavior in Adaptive Learning Systems*, volume 5499 of *Lecture Notes in Computer Science*, pages 31–47. Springer Berlin / Heidelberg.

Hayes, P. (1979). The naive physics manifesto. In Michie, D., editor, *Expert Systems in the Micro Electronic Age*, pages 242–270. Edinburgh University Press.

Hayes, P. (1985). The second naive physics manifesto. In Hobbs, J. R. and Moore, R. C., editors, *Formal Theories of the Commonsense World*, pages 1–36. Ablex, Norwood, NJ.

Hesslow, G. (2012). The current status of the simulation theory of cognition. *Brain Research Reviews*, 1428:71–79.

Hollerbach, J., Mason, M., and Christensen, H. (2009). A roadmap for US robotics – From internet to robotics. Technical report, Computing Community Consortium (CCC).

Howe, J. (2006a). Crowdsourcing: A definition. Crowdsourcing Blog, `http://crowdsourcing.typepad.com/cs/2006/06/crowdsourcing_a.html`. Retrieved March 24, 2013.

Howe, J. (2006b). The rise of crowdsourcing. Wired, `http://www.wired.com/wired/archive/14.06/crowds.html`. Retrieved March 24, 2013.

Ingram, J., Howard, I., Flanagan, J., and Wolpert, D. (2010). Multiple grasp-specific representations of tool dynamics mediate skillful manipulation. *Curr Biol*.

Johnston, B. and Williams, M. (2009). Autonomous learning of commonsense simulations. In *Int. Symposium on Logical Formalizations of Commonsense Reasoning*.

Johnston, B. and Williams, M.-A. (2008). Comirit: Commonsense reasoning by integrating simulation and logic. In *Proceedings of the 2008 conference on Artificial General Intelligence 2008: Proceedings of the First AGI Conference*, pages 200–211, Amsterdam, The Netherlands, The Netherlands. IOS Press.

Kato, F., Hanaoka, Y., Ngoc, T. N., Keoki, D., Mitake, H., Aoki, T., and Hasegawa, S. (2009). Interactive cooking simulator: to understand cooking operation deeply. In *ACM SIGGRAPH 2009 Emerging Technologies*.

Kemp, C., Edsinger, A., and Torres-Jara, E. (2007). Challenges for robot manipulation in human environments. *IEEE Robotics and Automation Magazine*, 14(1):20–29.

Khansari-Zadeh, S. M. and Billard, A. (2010). Imitation learning of globally stable nonlinear point-to-point robot motions using nonlinear programming. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 2676–2683.

Klapfer, R., Kunze, L., and Beetz, M. (2012). Pouring and mixing liquids — understanding the physical effects of everyday robot manipulation actions. In *35th German Conference on Artificial Intelligence (KI-2012), Workshop on Human Reasoning and Automated Deduction*, Saarbrücken, Germany.

Kowalski, R. and Sergot, M. (1986a). A logic-based calculus of events. *New Gen. Comput.*, 4(1):67–95.

Kowalski, R. and Sergot, M. (1986b). A logic-based calculus of events. *New generation computing*, 4(1):67–95.

Kunze, L., Dolha, M. E., and Beetz, M. (2011a). Logic programming with simulation-based temporal projection for everyday robot object manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Francisco, CA, USA. Best Student Paper Finalist.

Kunze, L., Dolha, M. E., Guzman, E., and Beetz, M. (2011b). Simulation-based temporal projection of everyday robot object manipulation. In Yolum, Tumer, Stone, and Sonenberg, editors, *Proc. of the 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, Taipei, Taiwan. IFAAMAS.

Kunze, L., Haidu, A., and Beetz, M. (2012). Making virtual pancakes — acquiring and analyzing data of everyday manipulation tasks through interactive physics-based simulations. In *Poster and Demo Track of the 35th German Conference on Artificial Intelligence (KI-2012)*, Saarbrücken, Germany.

Kunze, L., Roehm, T., and Beetz, M. (2011c). Towards semantic robot description languages. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5589–5595, Shanghai, China.

Kunze, L., Tenorth, M., and Beetz, M. (2010). Putting people's common sense into knowledge bases of household robots. In *33rd Annual German Conference on Artificial Intelligence (KI)*, pages 151–159, Karlsruhe, Germany. Springer.

Lakshmanan, K., Sachdev, A., Xie, Z., Berenson, D., Goldberg, K., and Abbeel, P. (2012). A constraint-aware motion planning algorithm for robotic folding of clothes. In *Proceedings of the 13th International Symposium on Experimental Robotics (ISER)*.

Lifschitz, V. (1998). Cracking an egg: An exercise in commonsense reasoning. Presented at the 4th Symposium on Logical Formalizations of Commonsense Reasoning.

Lugrin, J.-L. and Cavazza, M. (2007). Making sense of virtual environments: action representation, grounding and common sense. In *IUI '07: Proceedings of the 12th international conference on Intelligent user interfaces*, pages 225–234, New York, NY, USA. ACM.

Majtey, A. P., Lamberti, P. W., and Prato, D. P. (2005). Jensen-shannon divergence as a measure of distinguishability between mixed quantum states. *Phys. Rev. A*, 72:052310.

Martin, D., Burstein, M., Mcdermott, D., Mcilraith, S., Paolucci, M., Sycara, K., Mcguinness, D., Sirin, E., and Srinivasan, N. (2007). Bringing semantics to web services with owl-s. *World Wide Web*, 10(3):243–277.

McDermott, D. (1992). Robot planning. *AI Magazine*, 13(2):55–79.

Moratz, R., Nebel, B., and Freksa, C. (2003). Qualitative spatial reasoning about relative position. *Spatial cognition III*, pages 1034–1034.

Morgenstern, L. (2001). Mid-sized axiomatizations of commonsense problems: A case study in egg cracking. *Studia Logica*, 67(3):333–384.

Mösenlechner, L. and Beetz, M. (2009). Using physics- and sensor-based simulation for high-fidelity temporal projection of realistic robot behavior. In *ICAPS 2009*.

Munro, J. I., Papadakis, T., and Sedgewick, R. (1992). Deterministic skip lists. In Frederickson, G. N., editor, *SODA*, pages 367–375. ACM/SIAM.

Nilsson, N. J. (1984). Shakey the robot. Technical Report 323, AI Center, SRI International, Menlo Park, CA, USA.

Nyga, D. and Beetz, M. (2012). Everything robots always wanted to know about housework (but were afraid to ask). In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vilamoura, Portugal.

Oztop, E., Kawato, M., and Arbib, M. A. (2006). Mirror neurons and imitation: A computationally guided review. *Neural Networks*, 19(3):254–271.

Preece, A., Gomez, M., de Mel, G., Vasconcelos, W., Sleeman, D., Colley, S., Pearson, G., Pham, T., and Porta, T. (2008). Matching sensors to missions using a knowledge-based approach. *SPIE Defense Transformation and Net-Centric Systems*, 2008.

Richardson, M. and Domingos, P. (2006). Markov logic networks. *Machine Learning*, 62(1-2):107–136.

Russell, S. and Norvig, P. (2009). *Artificial intelligence: a modern approach*. Prentice hall.

Schaal, S., Ijspeert, A.-J., and Billard, A. (2004). *The neuroscience of social interaction*, chapter Computational approaches to motor learning by imitation, pages 199–218. Oxford University Press.

Schlenoff, C. and Messina, E. (2005). A robot ontology for urban search and rescue. In *KRAS '05: Proceedings of the 2005 ACM workshop on Research in knowledge representation for autonomous systems*, pages 27–34, New York, NY, USA. ACM.

Siskind, J. M. (2001). Grounding the lexical semantics of verbs in visual perception using force dynamics and event logic. *Journal of Artificial Intelligence Research (JAIR)*, 15:31–90.

Smith, D. and Morgan, B. (2010). Isisworld: An open source commonsense simulator for ai researchers. In *AAAI Workshops*.

Stulp, F., Fedrizzi, A., Mösenlechner, L., and Beetz, M. (2012). Learning and reasoning with action-related places for robust mobile manipulation. *Journal of Artificial Intelligence Research (JAIR)*, 43:1–42.

Svensson, H. and Ziemke, T. (1999). Making sense of embodiment : Simulation theories and the sharing of neural circuitry between sensorimotor and cognitive processes. *Learning*, pages 1309–1314.

Sycara, K., Widoff, S., Klusch, M., and Lu, J. (2002). Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. *Autonomous agents and multi-agent systems*, 5(2):173–203.

Tenorth, M. and Beetz, M. (2009). KnowRob – knowledge processing for autonomous personal robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4261–4266.

Tenorth, M., Kunze, L., Jain, D., and Beetz, M. (2010a). KNOWROB-MAP – knowledge-linked semantic object maps. In *Proceedings of 2010 IEEE-RAS International Conference on Humanoid Robots*, Nashville, TN, USA.

Tenorth, M., Nyga, D., and Beetz, M. (2010b). Understanding and executing instructions for everyday manipulation tasks from the world wide web. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1486–1491, Anchorage, AK, USA.

Teßmer, R. (2012). Acquiring commonsense knowledge for robots using games with a purpose. Technical report, Ludwig-Maximialians-Universität (LMU), Munich.

Ueda, R., Ogura, T., Okada, K., and Inaba, M. (2008). Design and implementation of humanoid programming system powered by deformable objects simulation. In *Proceedings of the 10th International Conference on Intelligent Autonomous Systems*, pages 374–381.

Vondrak, M., Sigal, L., and Jenkins, O. C. (2008). Physical simulation for probabilistic motion tracking. In *CVPR*.

Weigelt, M., Kunde, W., and Prinz, W. (2006). End-state comfort in bimanual object manipulation. *Experimental Psychology*, 53:143–148.

Weitnauer, E., Haschke, R., and Ritter, H. (2010). Evaluating a physics engine as an ingredient for physical reasoning. In *Proceedings of the Second international conference on Simulation, modeling, and programming for autonomous robots*, SIMPAR'10, pages 144–155, Berlin, Heidelberg. Springer-Verlag.

Weld, D. S. and Kleer, J. d., editors (1990). *Readings in qualitative reasoning about physical systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Zickler, S. and Veloso, M. (2009). Efficient physics-based planning: sampling search via non-deterministic tactics and skills. In *Proc. of The 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*.