

Accepted Manuscript

Envisioning the qualitative effects of robot manipulation actions
using simulation-based projections

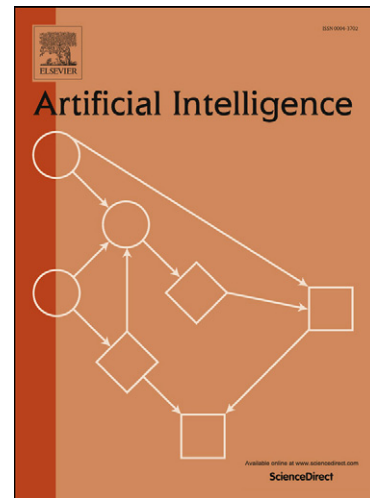
Lars Kunze, Michael Beetz

PII: S0004-3702(14)00154-4
DOI: [10.1016/j.artint.2014.12.004](http://dx.doi.org/10.1016/j.artint.2014.12.004)
Reference: ARTINT 2809

To appear in: *Artificial Intelligence*

Revised date: 11 December 2014

Accepted date: 20 December 2014



Please cite this article in press as: L. Kunze, M. Beetz, Envisioning the qualitative effects of robot manipulation actions using simulation-based projections, *Artificial Intelligence* (2015), <http://dx.doi.org/10.1016/j.artint.2014.12.004>

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

Envisioning the Qualitative Effects of Robot Manipulation Actions using Simulation-based Projections

Lars Kunze^a, Michael Beetz^b

^aIntelligent Robotics Laboratory, School of Computer Science, University of Birmingham, United Kingdom

^bInstitute for Artificial Intelligence & TZI (The Centre for Computing Technologies), University of Bremen, Germany

Abstract

Autonomous robots that are to perform complex everyday tasks such as making pancakes have to understand how the effects of an action depend on the way the action is executed. Within Artificial Intelligence, classical planning reasons about whether actions are executable, but makes the assumption that the actions will succeed (with some probability). In this work, we have designed, implemented, and analyzed a framework that allows us to *envision* the physical effects of robot manipulation actions. We consider *envisioning* to be a qualitative reasoning method that reasons about actions and their effects based on simulation-based projections. Thereby it allows a robot to infer *what could happen* when it performs a task in a certain way. This is achieved by translating a qualitative physics problem 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 sub-symbolic 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. By envisioning the outcome of actions before committing to them, a robot is able to reason about physical phenomena and can therefore prevent itself from ending up in unwanted situations. Using this approach, robots can perform manipulation tasks more efficiently, robustly, and flexibly, and they can even successfully accomplish previously unknown variations of tasks.

Keywords: Envisioning, Simulation-based Projections, Naive Physics, Everyday Robot Manipulation

1. Introduction

In recent years, we have seen substantial progress towards personal robot assistants which are able to perform everyday household chores such as cleaning a room¹ or preparing a meal (Beetz et al., 2011). However, designing and building robots that can autonomously perform an open-ended set of manipulation tasks in human environments remains an unsolved problem and poses many challenges to the field (Kemp et al., 2007). One of the challenges is enabling robots to learn how to perform novel tasks from natural instructions, for example interpreting natural language instructions (Tenorth et al., 2010b) or analyzing observations of a human performing the task (Beetz et al., 2010b). Based on such information, a robot has to understand the nature of the task, that is, it has to reason about *how the physical effects of a manipulation action depend on the way the action is executed*. In particular, to perform the task itself, the robot has to understand how its own manipulation actions produce physical effects. For example, how does the pose of the robot's end-effector affect the outcome of a pouring action when a liquid is poured from one container to another.

Within Artificial Intelligence (AI), the problem of reasoning about actions is often considered in the area of classical planning. In contrast to the question of *how the effects of an action depend on how it is executed?*, the question that classical planning typically considers is *what effects are caused by an action?* However, in the context of robotics, the problem has to be approached from a different direction, because *how* an action is executed has a major influence on its consequences. Furthermore, classical planning approaches are inadequate for everyday manipulation for two reasons: open-ended tasks makes classical planning intractable; and robot control programs cannot be adequately

¹<http://personalrobotics.stanford.edu>

represented by sequences of actions (McDermott, 1992). Logical axiomatizations for representing and reasoning about actions and their effects have also been developed for problems such as cracking an egg (Lifschitz, 1998; Morgenstern, 2001). However, when temporal projections are made on the basis of logical formalizations, the physical details of 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*. This requires them to handle an enormous amount of interdependent temporal data (Dean, 1989).

Evidence from cognitive psychology and neuroscience shows 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 determine how a human performs an action is 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: that behavior can be simulated; that perception can be simulated; and, finally, that 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. Evidence shows that even high-level cognitive processes are grounded in bodily-based simulations (Svensson and Ziemke, 1999).

Within this line of research we are interested in how robots can predict the possible outcomes of both single actions, like reaching for an object, and complex tasks, through *mental* simulations. We illustrate this with the following concrete example scenario.

1.1. Example Scenario: Making Pancakes

In this article, we use *making pancakes* as a running example to illustrate what physical knowledge robots need to competently accomplish everyday manipulation tasks. As mentioned above, 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 physical reasoning problems such as “cracking an egg” (Morgenstern, 2001) which is listed on the common sense problem page². In analogy to the problems listed on that page, we have formulated the task of making pancakes as follows:

“A robot pours 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 uses the spatula to move the pancake onto an upturned dinner plate.”

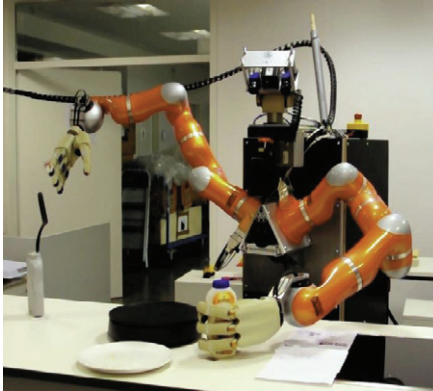
whereby a solution to the problem should also take the following variants into account:

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

By following the description of the challenge problem, a robot can acquire basic knowledge about the task. However, the robot does not know *what could happen* when itself performs the task in a certain way. This knowledge is obtained by what we call here *envisioning*³. We consider *envisioning* functionally similar to de Kleer (1977); de Kleer and Brown (1982). It denotes the kind of qualitative reasoning that predicts what might or could happen in a given

²Common Sense Problem Page: <http://www-formal.stanford.edu/leora/commonsense>

³Note, that we will also use the term *envision* to refer to the envisioning process



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 the spatula under the pancake?
- ◊ at what angle?
- ◊ with how much force?
- ◊ how to lift the pancake? ...

Figure 1: *What should I do? And how should I do it?*

situation. However, as we generate symbolic descriptions of what could happen based on simulation-based projections, the underlying principles are different from de Kleer (1977). In this work, we understand *envisioning* as an ability of a robot to predict and to reason about possible outcomes of its parameterized actions based on physics-based simulations. Through envisioning a robot can learn a mapping between different context conditions and qualitative action effects. In this sense, it shares similarities with classical and/or motion planning as it also makes predictions about future states. But at the same time it is different, as it is based on physics-based simulations. In fact, envisioning is complementary to planning methods as it takes a fully instantiated plan (a sequence of parameterized actions) as its input to reason about the effects and the dynamics of manipulation actions. By varying the initial conditions and the context, envisioning reasons about the generality (or the limitations) of a plan which could have been generated using classical and motion planning methods. Hence, envisioning provides supplementary reasoning capabilities that go beyond simple predictions, enabling robots to know what could happen when they themselves perform a task within a given context.

For a simplification of the task above, we consider the natural language instructions a human may use to describe the process of making a pancake:

- Pour the pancake mix into a frying pan.
- Flip the pancake around.
- Place the pancake onto a plate.

Humans can understand such instructions easily, and can immediately follow them. However, for robots these instructions are highly underspecified and therefore they need other means to acquire the relevant knowledge which enables them to perform the task. Figure 1 highlights some questions a robot has to answer in order to accomplish the task successfully.

To understand what makes this particular task interesting, let us first consider 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 instructions above are a pan (pancake maker), a pancake mix, a pancake, and a plate. Not mentioned in the description are some additional tools, for example, the container holding the mix and the spatula for flipping the pancake. Inferring these missing objects is straightforward for humans given their common sense. However, robots have to figure out these objects by other means. Overall, the task scenario contains objects with different physical properties, namely solid, liquid, and deformable objects. This task covers a range of manipulation problems since it involves a spectrum of different object types.

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 it, and turning the spatula.

Successfully pouring the mix onto the pancake maker requires that the container holding the mix is positioned at the right height over the pancake maker. At this position the container has to be tilted for some time at the right angle,

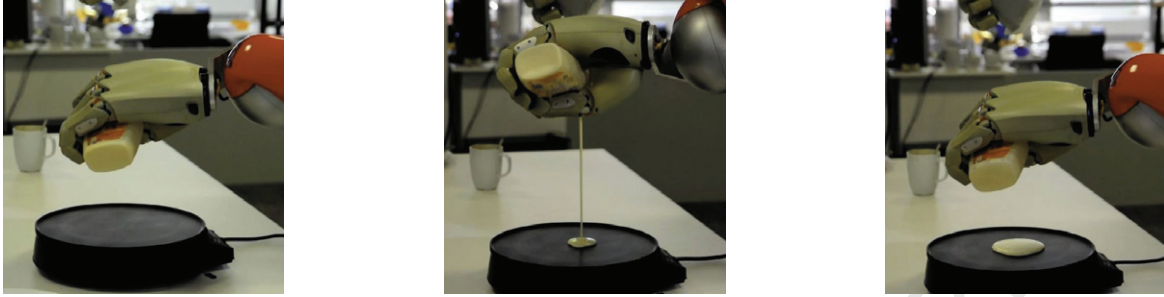


Figure 2: Pouring mix onto the pancake maker.

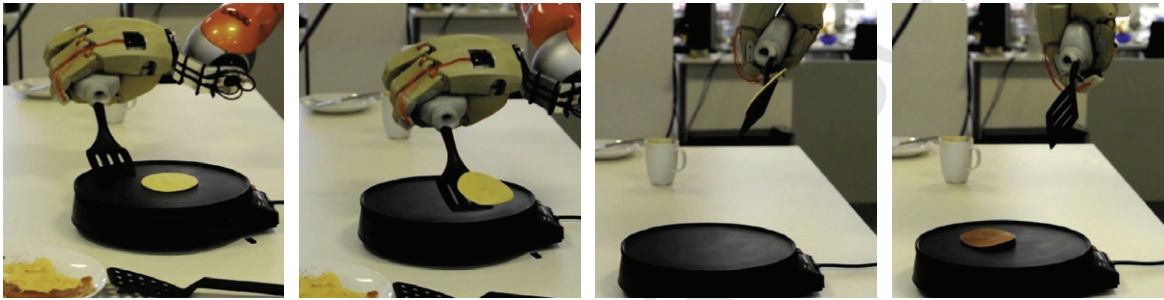


Figure 3: Flipping a pancake.

so that the mix flows out onto the pancake maker. Figure 2 shows our robot, Rosie, 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 held at an appropriate angle to get under the pancake. When lifting and turning the pancake the spatula has to be tilted at the correct angle so the pancake falls off and lands upside-down on the pancake maker. Figure 3 shows some aspects of Rosie performing the flipping action.

The pouring and the flipping actions performed by the robot could have various effects ranging from desired to undesired. Some of the undesired effects are depicted in Figure 4. During the pouring action, the robot could spill some pancake mix onto the table or it could pour the mix onto the pancake maker with lots of splashes. During the flipping action, the robot could damage the pancake by touching it from the top, or the pancake could get stuck to the spatula.

1.2. Physical Reasoning in AI

Figure 5 depicts an approach to using formal, logic-based methods to perform commonsense reasoning about problems like pancake making. Humans try to anticipate the appropriate sequence and configuration of actions that will lead to a desired outcome given an initial situation and an intended goal. However, it is not clear how we



Figure 4: Physical behavior flaws: pancake mixed spilled, spatula placed over the pancake, pancake stuck on spatula.

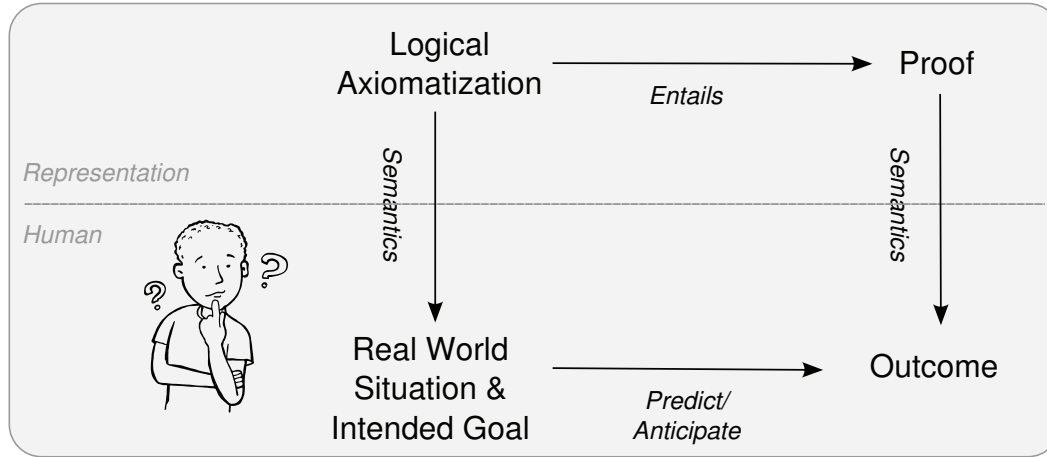


Figure 5: Commonsense reasoning and its formalization using first-order logic in traditional AI.

accomplish this kind of reasoning, although the simulation theory of cognition may partly provide an answer (Hesslow, 2012). In traditional AI, the initial situation and an action plan are described using a logical axiomatization. Then, a specialized calculus, for example, the situation or the event calculus, is applied in order to transform the axiomatization from the initial situation into a proof that resembles the intended goal. However, three major problems arise when applying such traditional AI approaches to robotics:

Level of abstraction The physical effects of actions strongly depend on the concrete parameterization of continuous variables. 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 leads to oversimplified and inadequate conclusions.

Interfering effects/concurrent actions The effects of serial 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. Such interfering effects are difficult to model using rules in a logical calculus.

Handling variants Robots should be able to handle variants of the original problem. An intrinsic feature of everyday manipulation tasks is that they will never be repeated under exactly the same conditions. For example, the ingredients or tools a robot has to use might differ; or if the pancake mix has a higher viscosity then the robot has to pour for a longer duration than usual. Not all variants of a problem can be foreseen, yet a robot should be able to cope with variants without the need to extend the underlying theory.

In conclusion, reasoning components for robots should be able to deal with the problems laid out above. That is, they should operate at a level of abstraction that considers the robot's actuators, sensors and control routines; handle interfering effects; and also cope with variants of a problem. In the next section we will outline the principle by which we enable robots to envision the outcome of their own actions adequately.

1.3. Our Approach

Our aim is to allow robots to reason semantically about objects and actions that rely on the richness of the continuous world. To do this we embed reasoning components deeply within robot control programs, allowing programmers to write more general control programs in a concise way. Our components allow task- and context-related decisions to be made, and action parameters determined, based on the underlying robotic components. The following excerpt of LISP pseudo-code illustrates the basic idea of our constraint-based action specifications, as applied to the example of pouring:

```
(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 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*.

The above example should give the reader only an idea about how we embed the use of naive physics and commonsense knowledge within cognition-enabled robot control. The aim of this work is not to realize this control mechanism, but rather to acquire the commonsense knowledge and generate the models that can be used within reasoning components. As will be shown later, we generate a model in form of a decision tree to determine the tilt angle of the container when pouring the pancake mix based on a desired pancake size and a given task context.

By considering the scenario of making pancakes, we aim to find the appropriate representations and inference mechanisms to enable robots to predict the effects of actions which depend on the way they 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 physics-based simulations (Kunze et al., 2011a,b; Kunze, 2014). Figure 6 shows the robot Rosie pushing the spatula under pancake and envisioning the action through mental simulation.

Though we have used the example of Rosie to motivate the overall problem of making pancakes, in the remainder of the article we use the PR2 robot to illustrate our ideas. This switch is done mainly for historical reasons as the original work on making pancakes started with Rosie (Beetz et al., 2011). However, more recently it has been transferred to the PR2 platform⁴.

Figure 7 shows how we extend the previously presented approach to commonsense reasoning. Based on a logical axiomatization (i.e., 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. These logged timelines allow the robot to perform logical queries on the outcome of the scenario. These queries play a key role in the constraint-based action specifications described above. Further, we show how decision trees can be learned from timelines and how they can be used for planning and action monitoring.

Despite using physics-based simulations, our research does not aim to determine the physical effects of action at a detailed level. Instead we aim to capture the *qualitative* effects and understand how these depend on the parameters of the respective manipulation actions. Furthermore, the developed representations and inference mechanisms should allow the robot to diagnose and revise executed actions; and in the context of learning, they should allow the robot to explore the parameter space of actions more efficiently.

1.4. Contributions

In this work, we have integrated methods from the fields of Artificial Intelligence and Robotics in order to envision and qualitatively evaluate the physical effects of robot manipulation actions. To this end, we have realized an open source programming environment which combines logic programming and physics-based simulation in a coherent framework. The main contributions of this work are as follows. We have:

⁴<https://www.youtube.com/watch?v=0eIryyzlRwA>

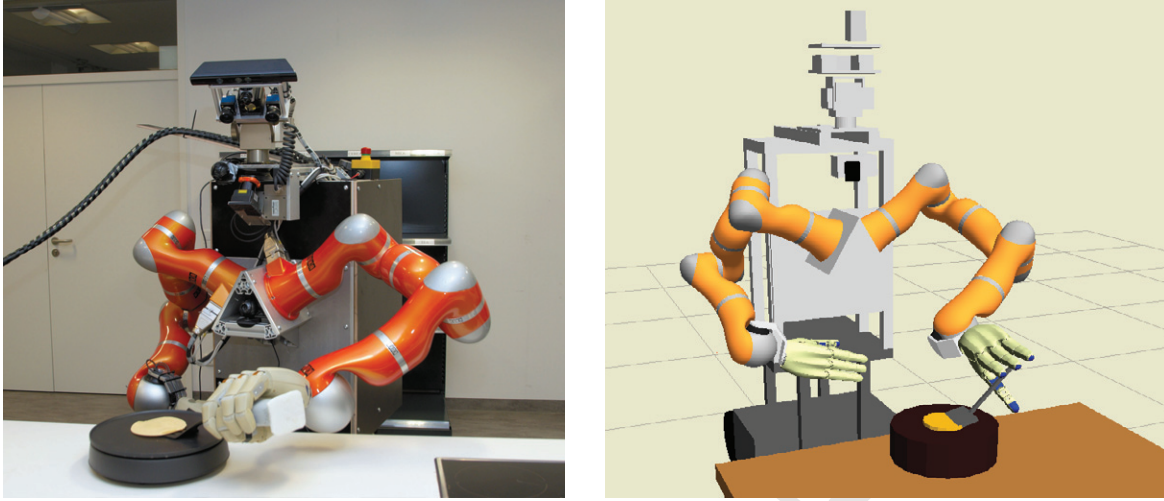


Figure 6: Left: Rosie pushing the spatula under the pancake. Right: Envisioning the *pushing* action using a physics-based simulation.

- established an interface for parameterizing and controlling physics-based simulations from the logic programming environment Prolog.
- linked first-order representations to physical object models that can be instantiated in simulation.
- started a library of physical object models and specialized physical behaviors which are not covered by rigid-body simulations, for example, the mixing of liquids.
- developed a monitoring and logging mechanism (configurable from Prolog) that observes data structures of interest within the simulator.
- introduced interval-based first-order representations (timelines) that tightly integrate sub-symbolic and symbolic information from logged simulations as a powerful means for reasoning about and learning from the consequences of robots' actions.

1.5. Outline

The rest of the article is structured as follows. Related work is reviewed in Section 2. We explain the envisioning framework in Section 3. In Section 4, we describe one example of a specialized physical behavior: how fluids are represented and simulated within the framework. Experimental results of various manipulation scenarios are reported in Section 5. We discuss our approach and the experiments in Section 6. Finally, we summarize the approach, give an outlook on future work and conclude in Section 7.

2. Related Work

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

Smith and Morgan (2010) have 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 such as *picking up an object* only at a very abstract level, whereas we focus on the physical details of such actions in order to recognize qualitative phenomena occurring during their execution.

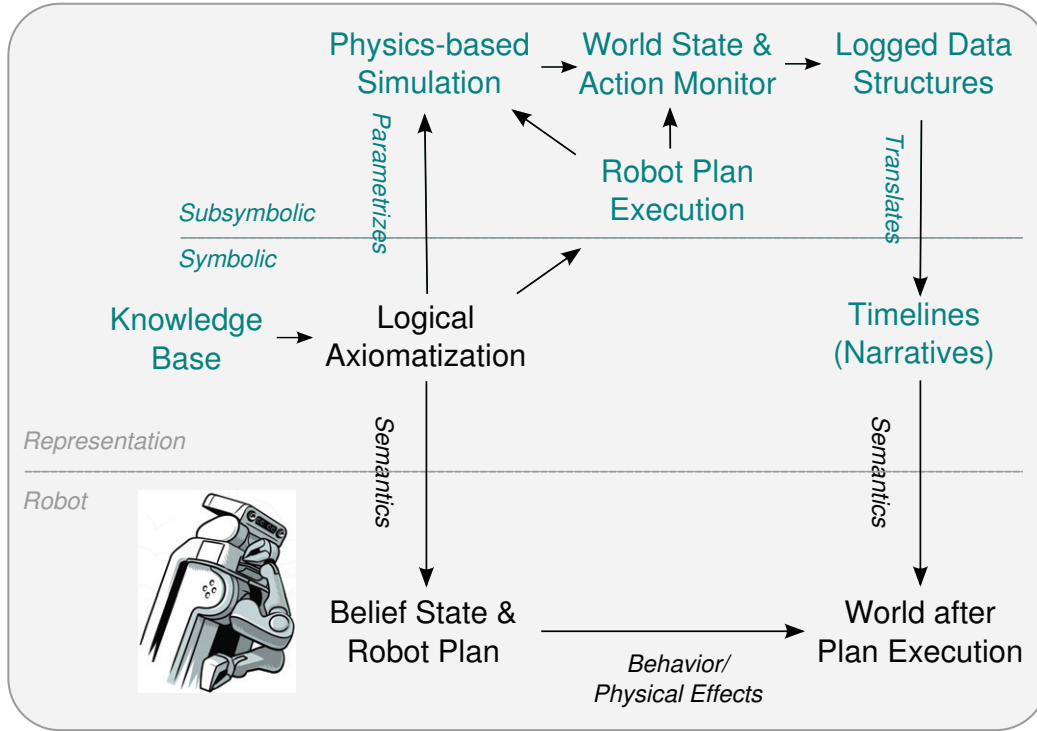


Figure 7: Envisioning of robot manipulation tasks based on physics-based simulations. (PR2 illustration by Josh Ellingson, Willow Garage)

In (Johnston and Williams, 2008), a general simulation framework and logic-based reasoning methods, in particular tableau-based reasoning, are integrated in order to establish a practical approach to commonsense reasoning. In contrast to their work, we are not aiming at commonsense reasoning in general but rather at reasoning for naive physics problems in the context of everyday robot object manipulation. Instead of looking at isolated problems, we aim for a tight integration between our proposed reasoning system and other processes such as planning, e.g., to predict whether a meal will be edible after executing a specific plan for cooking pasta.

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 have 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.

The interactive cooking simulator (Kato et al., 2009) is relevant for our work, since the research 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) and for navigating in environments with deformable objects (Frank et al., 2009). 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 (Miller and Morgenstern, 2009), were formulated based on logical axiomatizations (Lifschitz, 1998; Morgenstern, 2001). These approaches are limited in use for robotics as the physical details of the task are abstracted away and variants cannot

be handled very flexibly. It is to overcome such limitations that we propose a simulation-based approach. Please refer to our earlier work for a more detailed account on the problem of egg cracking using simulation-based techniques (Kunze et al., 2011b).

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 Lugin 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 no-one we are aware of has investigated a simulation-based approach for making predictions in the context of everyday robot object manipulation.

We ground logical predicates, such as $contacts(o_1, o_2)$, in the data from logged simulations. This is 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), where prediction is used to detect interfering effects of continuous and concurrent actions. Similarly, our work proposes a simulation-based approach for naive physics reasoning for robot manipulation tasks.

Lakshmanan et al. (2012) have developed a motion planning algorithm for folding clothes which they have evaluated in simulated and real robot experiments. Their work is complementary ours, as we assume that an action plan is given whereas they generate such a plan, i.e., a sequence of motion primitives. However, our framework could evaluate which of the generated motion plans could be successful and which could fail, for example, because the robot is not able to reach parts of the cloth from some of its poses.

Approaches to robot grasping have used planning and learning from experience to generate possible grasps (Dogar and Srinivasa, 2011; Mericli et al., 2013). These approaches provide complementary information to our work and could be integrated to some extent. In particular, it would be interesting to integrate the feedback from real world experiences into the simulation and thereby improve its accuracy.

3. The Envisioning Framework

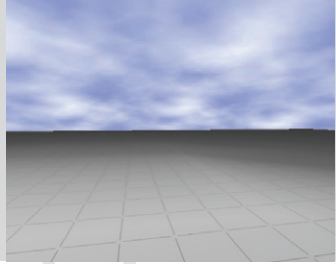
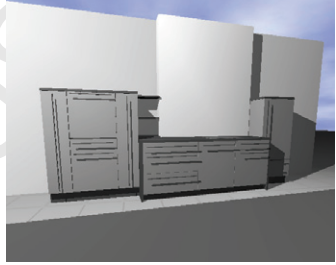
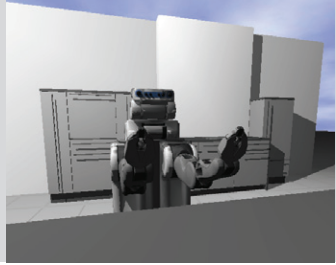
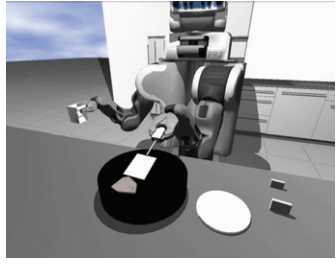
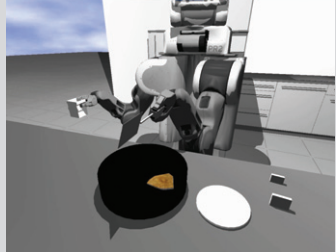
In this section we describe the overall envisioning framework. First, we give a general overview of the framework and its components, and second we explain the different components of the framework in detail.

3.1. Overview

The general principle of the framework is depicted in Figure 8, it 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 manipulation *Scenario* along with a parameterized *Action plan* that potentially solves the problem. The output of the envisioning framework is a *Timeline* which holds 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 such as 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 fully-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 given action plans we sample parameter values from the parameter space associated with a plan. In (Kunze et al., 2013), we have shown how a sensible range of parameter values can be extracted from observations of human demonstrations.

Table 1 illustrates the individual steps of the envisioning process with the example of flipping a pancake. In step (1) a new scenario called Pancakes is asserted using a logical language and thereby an empty simulation environment is set up. Steps (2) and (3) make further assertions to the scenario. The variables *kitchen* and *pr2* are instances in a given knowledge base that are now associated with the scenario. These instances have also links to physical models that are then instantiated in the simulation environment. In step (4), the parameter space (ParamSpace) of the action

Table 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.

Step	Logic Programming Environment	Physical Simulation
	<pre>(1) ?- assert_scenario(Pancakes).</pre>	
	<pre>(2) ?- assert_scenario(\$Pancakes,kitchen).</pre>	
	<pre>(3) ?- assert_scenario(\$Pancakes,pr2).</pre>	
	<pre>(4) ?- param_space(flip,ParamSpace), setof(TL,(member(P,ParamSpace), envision(\$Pancakes,flip(P),TL)), TLs).</pre>	
	<pre>(5) ?- 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).</pre>	

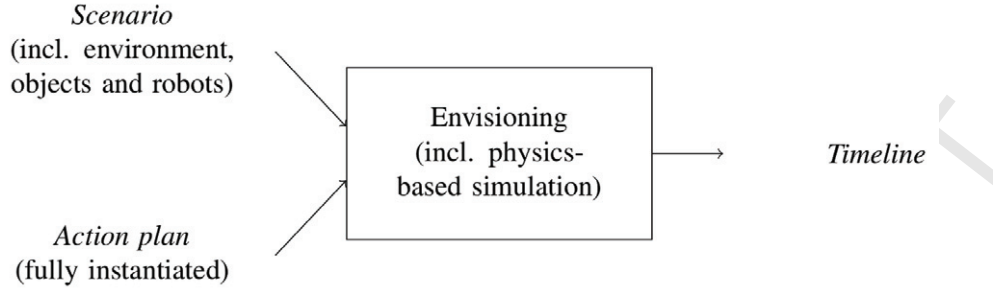


Figure 8: Input and output of the envisioning framework.

plan (`flip`) is retrieved from the knowledge base. For each parameterization P of the action plan (`flip`) envisioning is performed. Within the envisioning process the action plan is executed and the state evolution of the simulator is monitored and logged. Finally, the data structures of the logged simulations are abstracted and represented as timelines. Note that the `envision` predicate takes a *Scenario* (Pancakes) and a parameterized *Action plan* (`flip(P)`) as input as shown in Figure 8. Its output is a *Timeline* TL. Given Prolog’s backtracking mechanism a set of timelines TLs is retrieved using the `setof` predicate. Eventually, step (5) evaluates which timelines represent successful trials. Logical predicates such as `holds_tt` can be used to determine whether certain conditions hold within a time interval. For example, the query in step (5) 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.

Figure 9 visualizes how the process is embedded within Prolog’s backtracking mechanism. Given Prolog’s depth-first search strategy a proof tree is generated whereby the branches correspond to different parameterizations of robot action plans. The top-level predicate (`envision`) is based on two other predicates, namely `simulate` and `translate`. The `simulate` predicate executes the action plan within the simulator and logs the world state at each point in time. Afterwards the `translate` predicate converts the logged data structures into symbolic representations, called timelines. Eventually, the timelines are evaluated with respect to desired and undesired effects.

To cope with the uncertainty inherent in robot object manipulation, the same set of action parameters can be applied under varying conditions. To some extent, the physics-based simulation is already stochastic. Additionally, the poses of objects and the robot itself, and the initial state of objects can be varied. Thereby, the robustness of a single parameterized plan can be evaluated on the basis of the set of resulting timelines. Furthermore, it is possible to learn a joint probability distribution over the different outcomes of a robot plan. However, learning such a distribution is beyond the scope of this paper.

After we have looked at the general principle of the framework and its input and output specifications, we show how the envisioning functionality is achieved through the interplay of various components. Figure 10 shows the 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. We employ Prolog⁵ to realize the interface of the envisioning framework. Given domain knowledge (a 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 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 placed all components of the framework into context, we will explain how the individual components are realized.

⁵SWI-Prolog: <http://www.swi-prolog.org/>

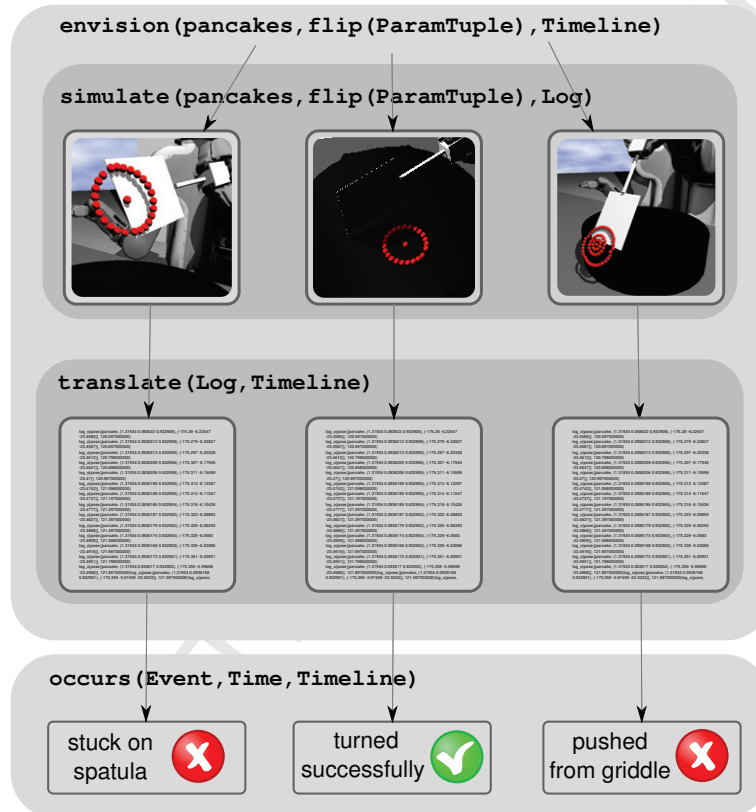


Figure 9: Prolog's proof tree for three different flipping plans.

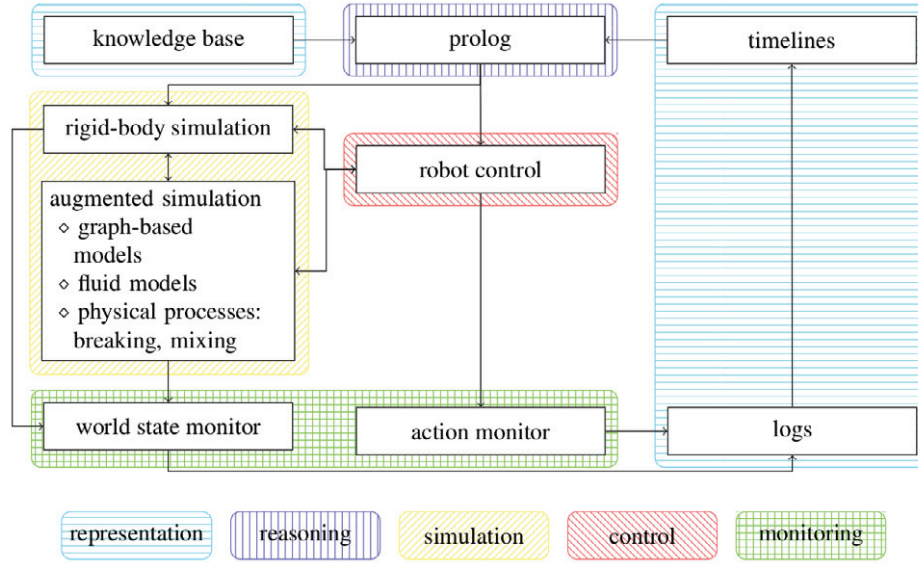


Figure 10: Envisioning framework overview.

3.2. Knowledge Base

The robot is equipped with knowledge about situated manipulation problems, action plans that describe how to solve these problems, and parameter spaces of primitive actions occurring in the plans. For representing the knowledge, we mainly use Description Logic (DL), in particular the semantic web ontology language OWL⁶. We build our representations on OpenCyc's⁷ 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 describe semantic categories like *cooking top* using an ontology. Similarly, the everyday objects that are to be manipulated by the robot are described within the 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 the environment is simulated, it makes sense to relate all objects to physical models. These models can be described in all formats that can be loaded into the physical simulator Gazebo⁸. In this work, we mainly make use of physical object models described in the Unified Robot Description Format (URDF)⁹. But other formats such as COLLADA¹⁰ are also feasible. Eventually, an instance of an object is linked to a physical model that can be instantiated within the Gazebo simulator.

Figure 11 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 instances and their relations among each other. The *in* relation between *PancakeMix* and *Container* is explained in Section 3.6.

The robot itself is specified by the Semantic Robot Description Language (SRDL) which we introduced in (Kunze et al., 2011c). The description includes the kinematic structure of the robot as well as a semantic description of its

⁶<http://www.w3.org/2004/OWL>

⁷<http://www.opencyc.org>

⁸<http://gazebo.org>

⁹<http://www.ros.org/wiki/urdf>

¹⁰<http://www.collada.org>

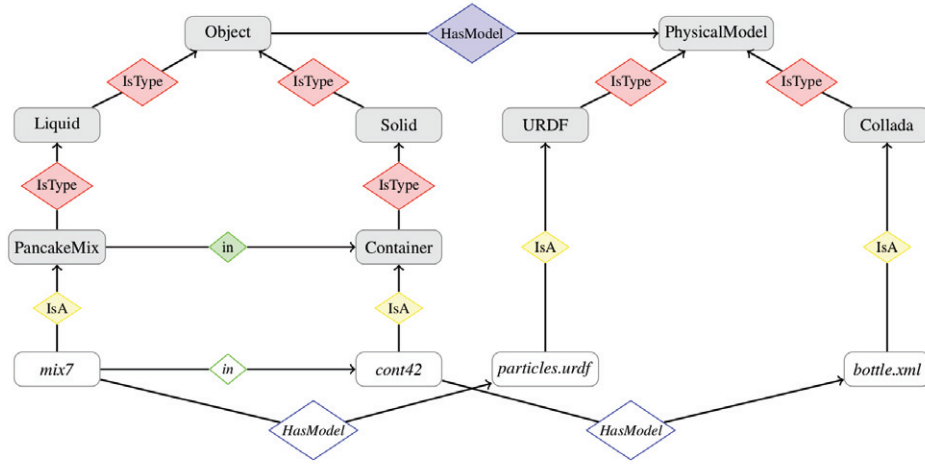


Figure 11: Simplified Ontology about “Making Pancakes”.

sensors and actuators. All of the aforementioned 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 12. Note that the order of actions is implicitly defined by a depth-first traversal strategy.

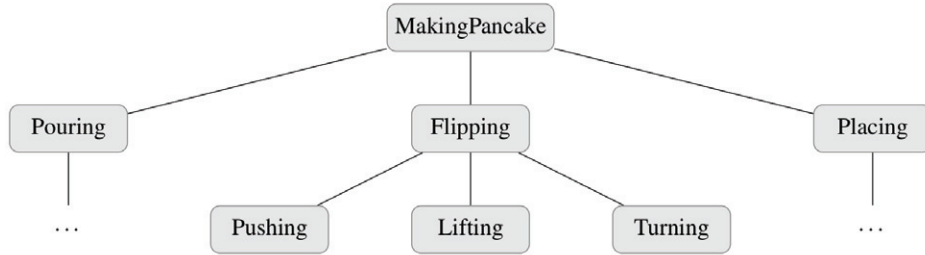


Figure 12: Ontological representation of an action plan for making pancakes.

3.3. Prolog — A Logic Programming Environment

Prolog is at 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. Within this language we use a notation similar to that used in the Event Calculus (Kowalski and Sergot, 1986). To support reasoning in our system we have developed the following temporal projection language in Prolog.

Temporal Projection Language. To initiate a projection, a new scenario is asserted and task-relevant descriptions are added to it, then a robot control program is executed using control parameters selected from a specified range of possible values. States that are traversed during simulation are monitored, logged, and translated into timelines. Eventually, the generated timelines are subject to further evaluations of specialized predicates. For example, a timeline can be evaluated with respect to desired (or undesired) outcomes, qualitative spatial relations, or other performance criteria such as the speed of execution.

The following Prolog query shows how the simulation-based temporal projection can be used. Terms starting with an upper-case letter such as `Scenario` denote variables, terms starting with a lower-case such as `kitchen_env` denote

concrete instances in the knowledge base, and the predicate `occurs` is an example of a specialized predicate that is evaluated on 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(T, (member(P,ParamSpace),
            envision(Scenario,actionplan(P),T)),Ts),
   member(Timeline,Ts), holds(occurs(event),Time,Timeline).
```

Values for the formal parameters, e.g. `P`, are selected from their respective range and are bound to the variable in order to make them accessible for further evaluations. The generation and selection of parameter values is an open problem. Intuitively, the parameters could be chosen depending on the qualitative outcomes of the simulation, however, this is beyond the scope of this article. The language elements for making temporal projections are as follows:

`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 and 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, the Gazebo simulator is launched and all entities that were added to the given scenario 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 timeline, by using the first-order predicate `Holds(f,t)`. To differentiate between the individual timelines, a unique ID (`Timeline`) is generated and attached to the individual fluents and events.

`holds(Fluent, Time, Timeline)` retrieves the given `Timeline` and evaluates it with respect to a fluent (`Fluent`) that might have held at a point in time (`Time`) during the simulation. If the specified fluent is found in the timeline the predicate evaluates to true.

`occurs(Event)` is evaluated with respect to an event (`Event`). The predicate is used within the `holds` predicate, i.e., `holds(occurs(Event), Time, Timeline)`, which retrieves the given `Timeline` and evaluates it with respect to an 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.

3.4. Physics-based Simulation

Within our approach, we utilize the physics-based simulator Gazebo¹¹ to compute the effects of robot actions, object interactions and other physical events. We augmented its rigid-body physics to simulate specialized behaviors.

¹¹<http://gazebo.org>

Rigid-Body Simulation. In principle 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 it updates the state of the simulated world according to physical laws. Within each update step, forces are applied to affected objects by considering both the object's current dynamic state and its properties such as mass and friction.

The initial configuration of the Gazebo simulator is based on an XML file, called *world file*. It describes properties of the simulation, specifies parameters for the physics engine (ODE¹²) 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 such as size, mass, and rigidity. When properties are not explicitly specified within the knowledge base, we simply assume default values.

Augmented Simulation. The Gazebo simulator is limited to rigid body dynamics. Since we want to simulate naive physics problems with phenomena such as 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 Johnston and Williams (2008) with graph-based structures which allow us to inspect physical aspects at a more detailed level. These model configurations are derived from the information stored in the knowledge base. The basic entities for modeling the shape of an object are *bodies* and *joints*, which are mutually connected. Properties of an object such as 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 such as being broken and being cooked can be computed. For example, if a force applied to an object exceeds a certain threshold over a period of time a controller will disconnect the affected joint between two bodies and thereby the object structure will break. The simulation of fluids and deformable objects will be explained in Section 4. To make objects subject to physical processes such as breaking and cooking we only have to add respective attributes to object instances in the knowledge base.

In previous work, we analyzed the diversity and variability of activities within the household domain (Nyga and Beetz, 2012). This showed that in a given domain there are only a limited number of activities and processes that have to be implemented, making our approach scalable. In Section 4, we provide more details on the augmented simulation. In particular, we will explain how fluids are represented and simulated within the framework. As the augmented fluid simulation is complex, we first continue with the remaining components of the envisioning system for the sake of a better understanding.

3.5. 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.

Actions of the robot are monitored as follows. Ideally, robot control programs would be written as plans. For example, using a plan language such as CRAM (Beetz et al., 2010a) allows a robot to reason about its own programs. However, in this work we treat a robot control program as a black box, so it can be implemented in any kind of 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. The begin and the end of an action as well as its parameters should be logged by the control program. This allows robots to relate 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 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.

¹²<http://www.ode.org/>

Table 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)), (ori- entation, (-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)), (ori- entation, (-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	–

The data structures of the world state we are monitoring are the position, orientation, linear and angular velocities, and the bounding boxes of objects and their respective parts. Furthermore, we observe the physical contacts between objects and log information such as contact points, contacts normals, and forces. All this information is constantly monitored and only changes are logged.

3.6. Fluents, Events and Timelines

Reasoning about everyday object manipulation requires robots to understand the spatial and physical configurations of objects and their parts over time. This section focuses on the knowledge representation part of our framework. A 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 simulation, all this information can be abstracted from the data structures of the simulator. Conceptually, the robot can access this information using the predicate *SimulatorValue* as follows:

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

where *position* is an example 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 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.

Another set of functions that can be accessed via the *SimulatorValue* predicate provides information about an object’s contacts. Contact information is 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 two objects as arguments.

Table 3: 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 $\langle x_{min}, y_{min}, z_{min}, x_{max}, y_{max}, z_{max} \rangle$

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, 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 4 shows functions that extract information about contacts between objects including contact positions, normals, penetration depths and forces.

Table 4: 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 $\langle t_x, t_y, t_z \rangle$

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 different types of spatial relations. Table 5 gives an overview of the implemented fluents.

Table 5: Fluents of Topological 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

Fluents describe conditions over time, for example the condition that object A is on object B . Therefore, fluents have to be related to time. In this work, we represent fluents as functions and use similar notations to the Event

Calculus (Kowalski and Sergot, 1986). 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*. The *Holds* predicate looks as follows:

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

where f is an arbitrary fluent from Table 5 and t a point in time. Note that Table 5 only lists example relations. A complete overview is given in (Kunze, 2014).

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

$$\begin{aligned} \text{Holds}(\text{on}(o_1, o_2), t_i) \Leftrightarrow \\ \text{Holds}(\text{contacts}(o_1, o_2), t_i) \wedge \\ \text{Holds}(\text{above}(o_1, o_2), t_i). \end{aligned}$$

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:

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

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

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

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:

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

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 *HoldsThroughout* predicate allows us to reason about 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). For example, the following logical sentence can be used to describe that the pancake mix was in the container before it was on the pancake maker:

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

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 *HoldsThroughout* predicates, i.e.:

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

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

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

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

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

Timelines. In this work, 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 13 visualizes the 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 12. The cooking event 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)*. The undesired effect of spilling some pancake mix onto a table is represented by the fluent *spilled*.

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 relate to either single or multiple timelines. We achieve this, by extending all previously introduced predicates by a timeline argument. Hence the *Holds* predicate is finally formalized as follows:

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

and the *HoldsThroughout* predicate as:

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

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. Furthermore, instead of linear search we use binary search methods to retrieve information from a timeline in logarithmic time.

4. Augmented Simulation of Fluids

Fluids play an important role in everyday cleaning and meal preparation tasks. Davis (2008) presents a formal solution to the problem of pouring liquids. In his work on the representation of matter (Davis, 2010) he investigated the advantages and disadvantages of various representations, including those for liquids. 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. Different approaches have been incorporated to simulate liquids depending on the required level of accuracy needed (Griebel et al., 2007). In (Klapfer et al., 2012), we proposed 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). Neither simulates liquids in all their aspects, but they provide enough information for making logical inferences about qualitative phenomena.

4.1. 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. In this approach, matter is represented as a hyper-graph 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.

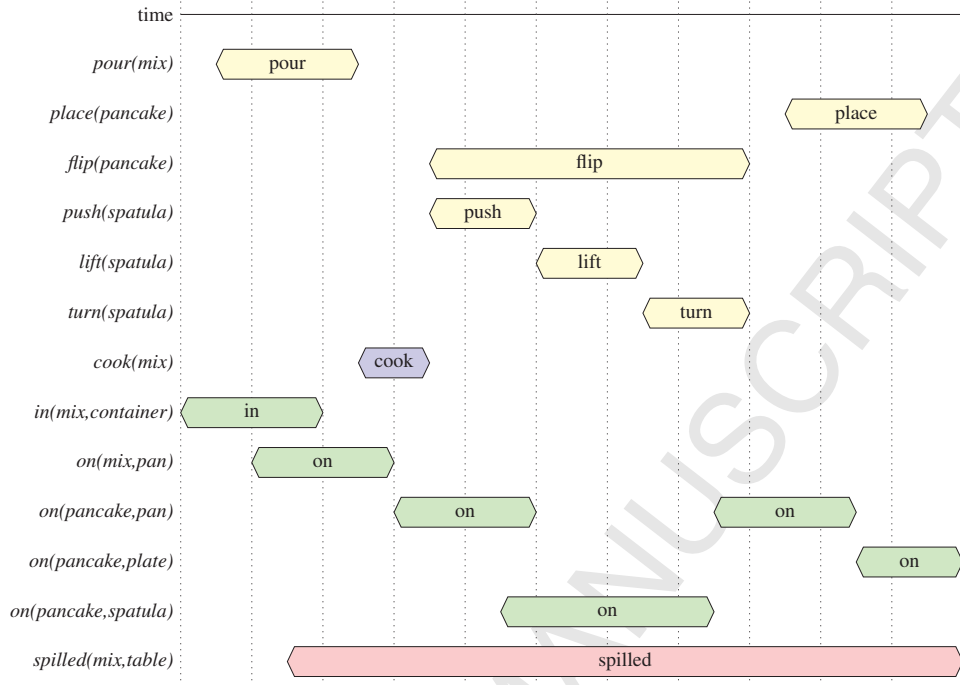


Figure 13: Timeline representation of making pancakes.

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 a graph with no connection between the vertices whenever the state is liquid. This means that the individual particles can freely move to some extent. This is useful for performing the pouring task. Due to the fact the particles are not connected with joints, the simulated liquid can be poured over the pancake maker where the particles disperse due to their round shape. A controller is attached to the spheres that applies small forces to the particles 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 occurs after a fixed time. We identify all particles on the pancake maker and create the pancake based on a fluid particle clustering algorithm (Figure 14).

4.2. Clustering of Fluid Particles

Let us assume that someone pours some pancake mix onto a pancake maker as illustrated in Figure 15. After pouring, some particles reside in the container, some are spilled onto the table, and others are on the pancake maker. If we want to address the particles in these three locations, it makes sense to group them in clusters. This reflects also how humans address fluids such as milk or sugar in natural language, e.g., *there is some milk spilled onto the table*. We use a Euclidean clustering strategy for 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. Whenever new particles become part of, or are separated from, a cluster we assign a new ID to them. That is, clusters of particles have only a limited time during which they exist. Hence, we can recognize which actions cause changes to clusters and their properties.

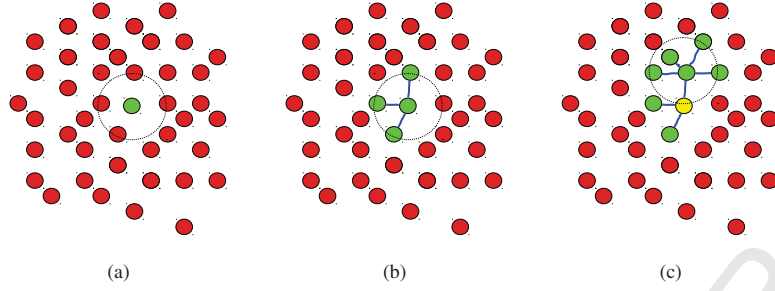


Figure 14: Generating a deformable pancake model from liquid particles. Illustration of the algorithm's procedure: (a) Radial search from the seed Point. (b) Creation of hinge joints to the neighbors. (c) Radial search and creation of joints in a recursive step.

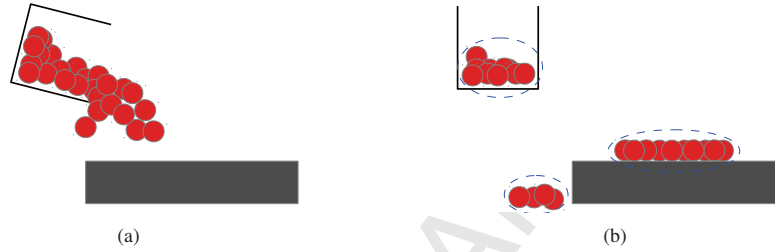


Figure 15: Basic idea of the clustering approach: during simulation we identify clusters of particles. For example, after pouring, one cluster resides still in the mug, a second is on the pancake maker and a third is spilled onto the table. We are able to extract information including contacts, position, extension, and size of the individual clusters.

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 processed and are part of a cluster $c_i \in Clst$
-

4.3. Monte Carlo Simulation of Fluids

Deformable bodies are a major challenge to simulate, requiring 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 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:

$$\frac{\partial C}{\partial t} = D \cdot \frac{\partial^2 C}{\partial x^2} \quad (1)$$

where C denotes the concentration and D the diffusion coefficient. It can be shown (Frenkel and Smit, 2001) that a *random walk* gives one particular solution for the above partial differential equation. Motivated by this idea we

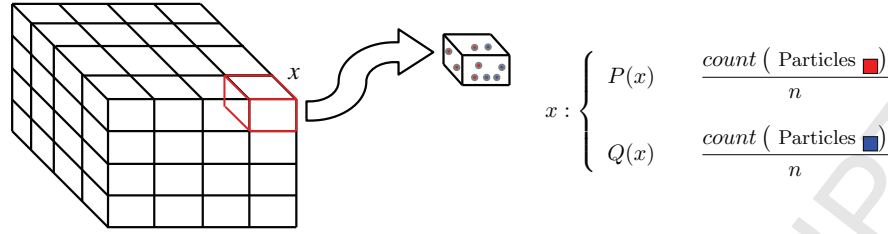


Figure 16: Density grid used for discretization and local density estimation of two classes P and Q (red and blue particles).

applied an algorithm proposed by Frenkel et al. to simulate this physical effect (Klapfer et al., 2012).

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 occurs 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 the stirring direction to the particles in the point cloud that are in reach of the cooking spoon.

4.4. Measuring the Homogeneity of Mixed Fluids

We are particularly interested in the homogeneity of a liquid after stirring. We 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 is known as the Jensen-Shannon divergence (Majtey et al., 2005) and used widely in information theory. It is defined as:

$$JS(P, Q) = \frac{1}{2} S\left(P, \frac{P+Q}{2}\right) + \frac{1}{2} S\left(Q, \frac{P+Q}{2}\right) \quad (2)$$

where $S(P, Q)$ is the Kullback divergence shown in equation 3, and P and Q are two probability distributions defined over a discrete random variable x .

$$S(P, Q) = \sum_x P(x) \log\left(\frac{P(x)}{Q(x)}\right) \quad (3)$$

We represent the point cloud as a three-dimensional grid. Each cell of the grid represents a discrete probability distribution x defined on the mixed probabilities of the two classes P and Q (red and blue particles), that could be computed as the relative frequency (Figure 16).

5. Experiments

For showing the feasibility of our approach, we have conducted several robot manipulation experiments in simulation including the problem of making pancakes as described in Section 1.1. The robot model used in our experiments is the PR2 robot platform developed by Willow Garage¹³. 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. It has 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. The entire setup is realistically modeled in the Gazebo simulator.

In this work, we present experimental results for the task of making pancakes. We have divided the task into three sub-tasks: mixing, pouring and flipping. These sub-tasks represent rather basic actions which are relevant for many meal preparation tasks, meaning the following results can be generalized to other tasks, for example making omelets.

¹³<http://www.willowgarage.com/pages/pr2/overview>

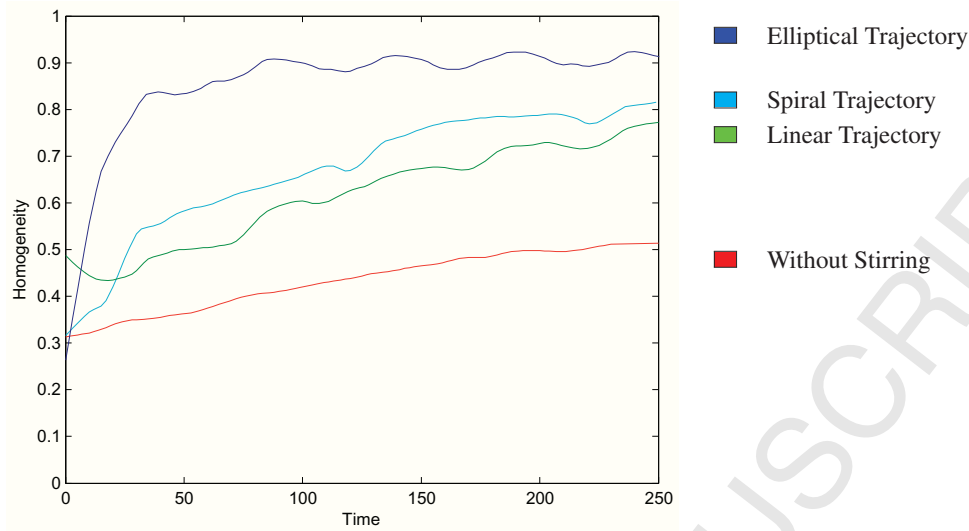


Figure 17: Homogeneity over time of different stirring trajectories. The graph shows the change of homogeneity on the vertical axis for different trajectories in direct comparison with the result of scenario of not stirring over time.

5.1. Mixing Fluids — Analysis of Homogeneity

In this experiment, the robot pours the contents of two containers into a bowl and stirs them. We used the Monte Carlo method described in Section 4.3 to simulate the physical effect of using different trajectories to mix fluids. We selected the coefficients to represent two viscous fluids and performed four experiments where the robot stirred the fluids using (1) an elliptical trajectory, (2) a spiral trajectory, (3) a linear trajectory, and (4) no trajectory (without stirring). The results are shown in Figure 17. They were evaluated on the basis of the homogeneity measure (Section 4.4). When the robot does not stir the fluids, the ingredients do not mix very well because only the diffusion process is influencing the homogeneity. Hence, the result of the experiment confirms our hypothesis that stirring increases the homogeneity of mixed fluids. Furthermore, our results show that an elliptical trajectory achieves the best mixing. Given the knowledge of homogeneous and non-homogeneous regions, a robot could adapt the trajectory dynamically using reinforcement learning. A qualitative interpretation could be based on a logical predicate that is true when the homogeneity is above a certain threshold and otherwise false. Thereby a robot could decide when to stop stirring with a particular trajectory.

5.2. 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. Using the resulting timelines, we analyze the qualitative outcome of the action and learn decision trees that can predict the action’s physical effects. The parameterization of the task includes the gripper position, the pouring angle and the pouring time. We also looked at different container types and fill levels. Table 6 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 the fill level and container type are perceptible by the robot. Effects of the action include the size of the pancake and the amount of spilled particles. Effectively all attributes, except the container type, are continuous by nature. However, as motivated earlier, we would like to learn interpretable action models. Therefore, we discretize the ranges of all continuous attributes to nominal concepts using Euclidean clustering methods.

The task was considered to be successful if no pancake mix was spilled, i.e. after pouring the liquid resides on the pancake maker or in the container, and not on other objects such the kitchen table. We used the resulting clusters and their corresponding contact and spatial information to examine the outcome. Figure 18 shows how clusters of pancake mix are spatially related to other objects before, during and after the pouring action on a single timeline.

The following Prolog expression shows how information about clusters can be retrieved from timelines (TL):

Table 6: Attributes of the *pouring* domain with their respective types and ranges.

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

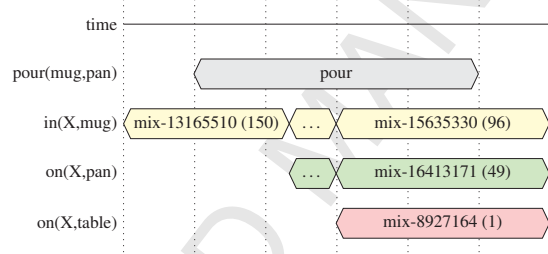


Figure 18: Visualization of the main clusters of particles before, during and after the pouring action. In this experiment the pancake mix was represented by 150 particles. The number of particles of a cluster is shown in parenthesis. During the simulation there were more than 20 clusters generated on this timeline.

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

where X denotes a cluster of pancake mix in contact with a pancake maker after a pouring action has been carried out. Figure 19 shows results from experiments in which the position of the container was varied in the x and y directions over the pancake maker. The figure shows the amount of pancake mix (particles) that were spilled onto the table. The tested positions are organized in a grid structure above the pancake maker (red dots). Given the symmetry of the pancake maker, for each of the positions the pouring direction was always pointing towards north. As explained earlier, we mapped the value of the *particle count* to a number of discretized classes (*Small*, *Medium*, and *Large*) to be able to interpret the results qualitatively. Note that a similar query to the one above can be embedded within the constraint-based action specification laid out in Section 1.3.

Additionally, we used logical queries such as the one above to extract data for learning decision trees in order to classify pancake sizes and pouring angles. The input data that we extract from each simulation run is a tuple as follows:

$\langle \text{container}, \text{particles}, \text{time}, \text{angle}, \text{size} \rangle$

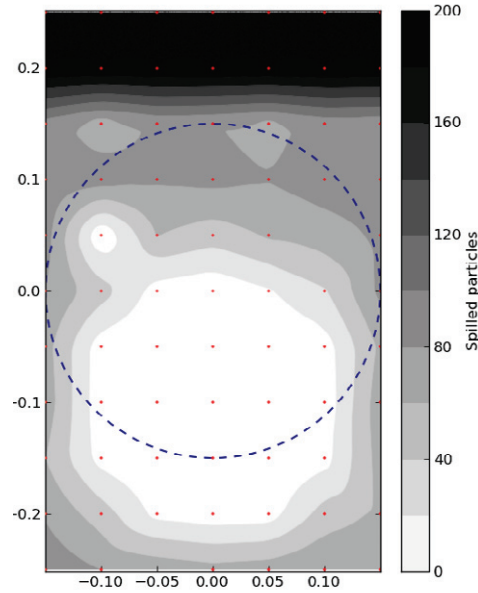


Figure 19: Number of spilled particles while pouring pancake mix from different positions. The blue dotted circle indicates the pancake maker. The tested positions are organized in a grid structure above the pancake maker (red dots).

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 20 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 for pouring, the right when a bottle was used. 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 21 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 can be observed that the size of a container's opening (*mug* vs. *bottle*) has a dramatic effect on number 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 clogged up. Furthermore, 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*.

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.

We use a decision tree to predict the pancake sizes that a particular (pre-pouring) configuration will achieve. Decision trees have the advantage that their internal structure is easily interpretable: each internal node of the tree is a test on an attribute; each branch represents a different outcome of such a test; and each leaf node represents a class label. By following the paths from the root node of the tree to the leaves the classification rules can be extracted. We ran Weka's *J48* algorithm (Hall et al., 2009) in its default parameterization on our experimental data. The resulting decision tree is visualized in Figure 22. Overall, the learned model achieves an accuracy of 92.41%. That is, out of

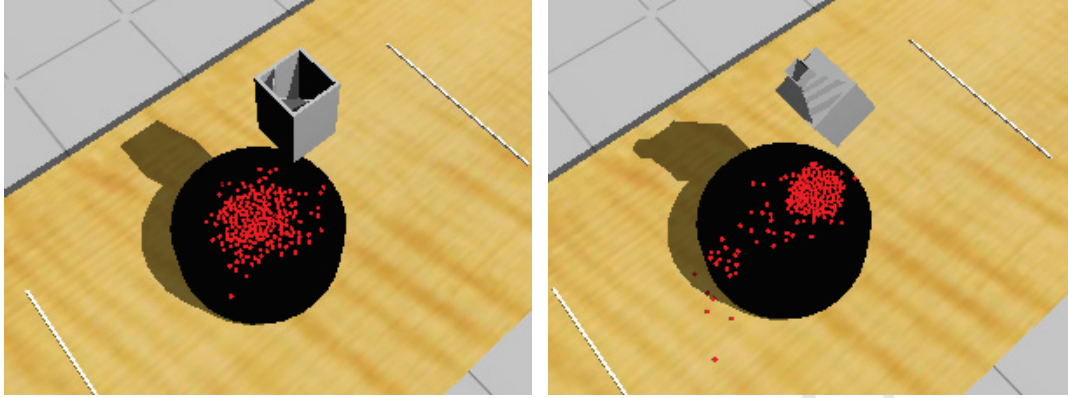


Figure 20: Different qualitative outcomes of a pouring action. Left: A mug with a large opening. Right: A bottle with a small opening.

Table 7: 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 8: 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

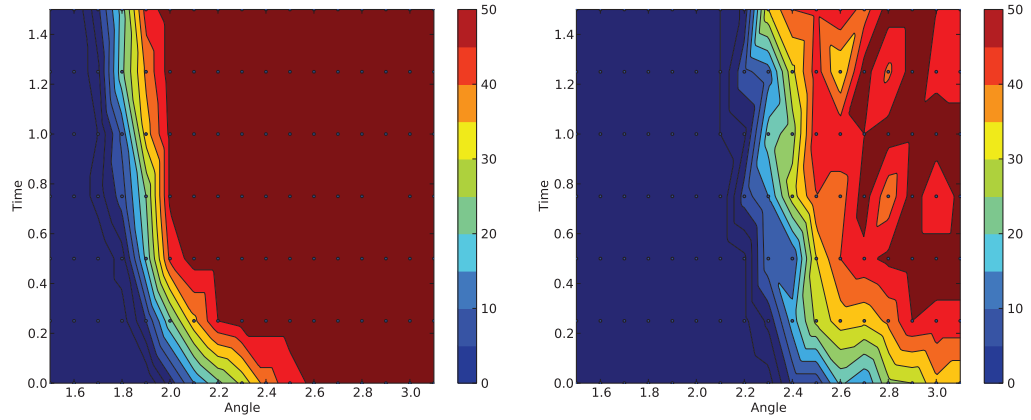
the 474 instances used for learning, 438 are classified correctly during 10-fold cross-validation. The most decisive attribute is the fill level. If there are only a few particles available, the robot can only make small pancakes. When many particles are available the size of the pancake depends first 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, can the robot make pancakes of different sizes by varying the pouring duration. The confusion matrix in Table 7 shows that mainly pancakes of medium size were misclassified.

We also learned an action model for determining the pouring angle required to produce a pancake of a desired size. Again we used Weka's *J48* algorithm. The learned decision tree, depicted in Figure 23, achieves an accuracy of 64.35% in the 10-fold cross-validation. As the confusion matrix in Table 8 shows, mainly the *mid* angles were misclassified.

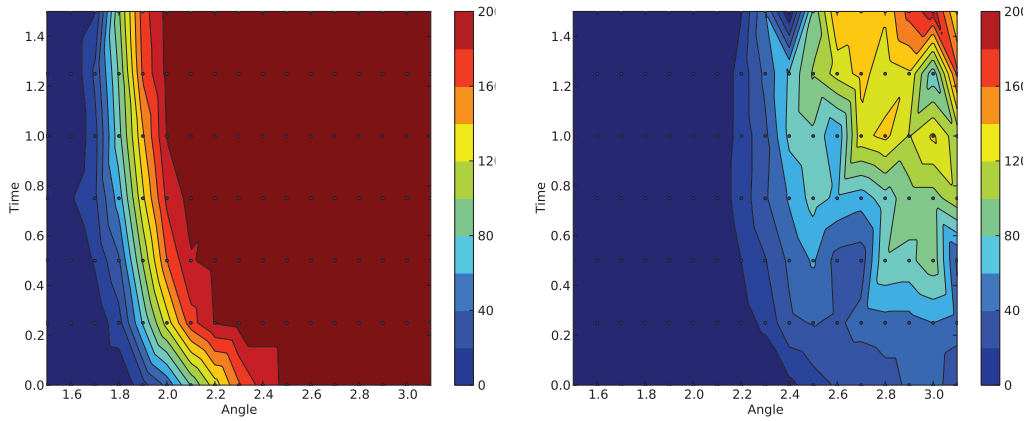
5.3. Flipping a Pancake

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 pancake simulation 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 25). During this scenario, the simulated 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. The qualitative results of the experiments performed in this scenario are shown in Table 9. The experiments were launched using the following query:



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



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

Figure 21: Relationship between pouring angle and time. The pouring results are shown for different types of containers and different fill levels. The tested positions are organized in a grid structure above the pancake maker (black dots). The color encodes the number of poured particles on the pancake maker.

```
?- param_space(flip_pancake,ParamSpace),
   setof(T, (member(P,ParamSpace),
            envision(flipping,flip_pancake(P),T)),Ts),
   member(Timeline,Ts).
```

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

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

That is, the pancake has to be on the pancake maker before and after the flipping action. In the query above, we used the overlaps relations as the temporal relation between the fluents and the flipping action. In general, all

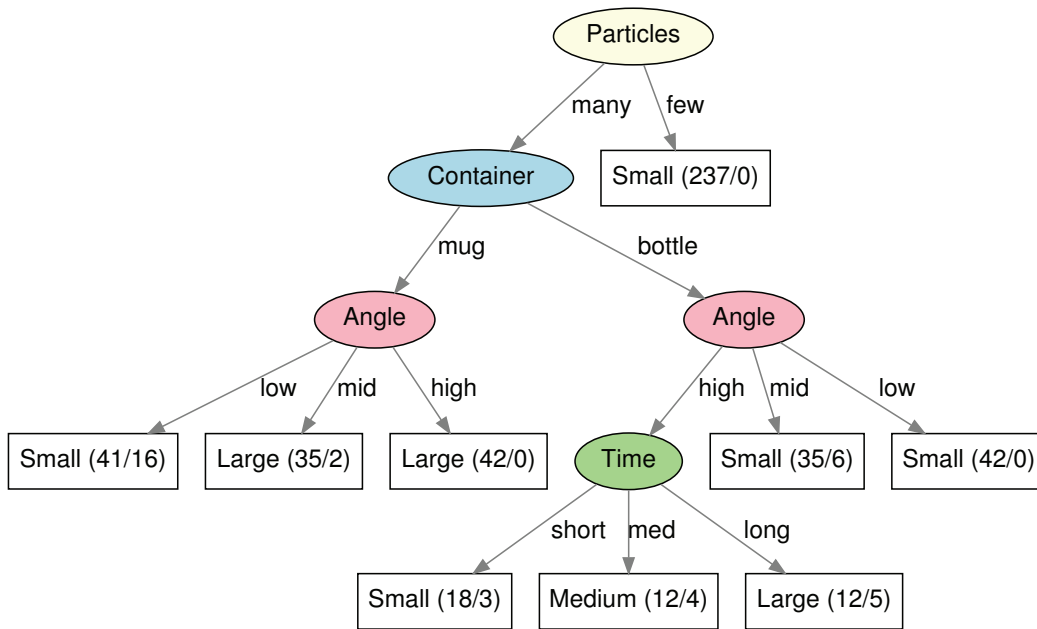


Figure 22: Decision tree for predicting the size of a pancake. The size is discretized in three classes, namely *Small*, *Medium*, and *Large*. The tree is learned from 474 instances and classifies 438 instances correctly (92.41%) and 36 incorrectly (7.59%).

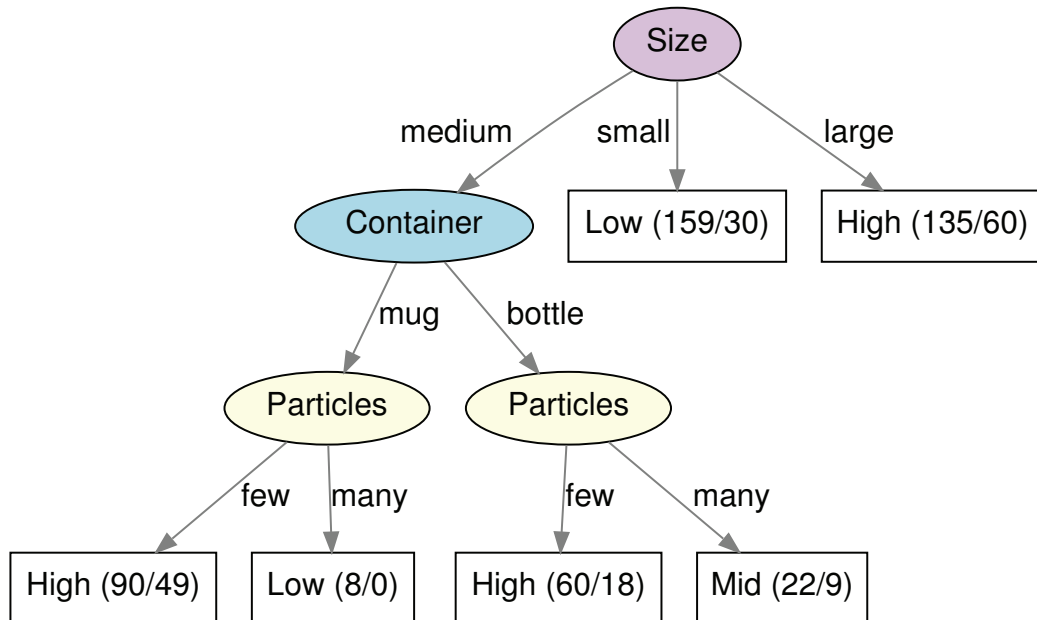


Figure 23: Decision tree for selecting an angle. The angle is discretized in three classes, namely *Low*, *Mid*, and *High*. The tree is learned from 474 instances and classifies 305 instances correctly (64.35%) and 169 incorrectly (35.65%).

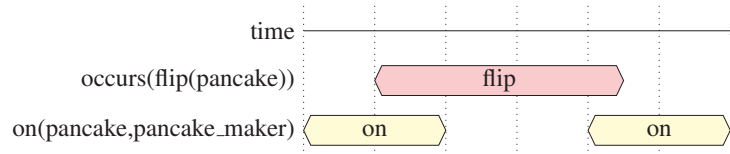


Figure 24: Timeline representation of flipping a pancake.

Table 9: Qualitative results: Flipping a pancake.

<i>Angle</i>	0.1	0.3	0.4	0.5	0.7	0.9
<i>Outcome</i>	ok	ok	ok	fail	fail	fail

13 possible temporal relations between time intervals can be used to constrain the query (Allen, 1983). Figure 24 visualizes the temporal relations between the *on* fluent and the flipping action.

Note that a query such as the one above that was used to find flipping events across timelines, can also be used to determine the objects the pancake is on top of. For example, during the execution of the flipping action, the pancake would be *on* a blade, which is a part of the spatula object.

Overall, such logical queries allow us to select and filter data of logged simulations at an abstract, semantic level. This is a distinct feature of the explicit logical abstractions developed in this work when compared to implicit models that can be learned through reinforcement learning and those that can be optimized using methods from control theory. This is not to say that our models would perform better than models from other fields or should replace them. On the contrary, the presented methods should complement approaches from the other fields as they have the advantage that they are interpretable and thereby allow robots to reason about the physical effects of their actions. In particular, robots can employ the qualitative information about physical aspects of their manipulation actions to answer questions in the following contexts:

Monitoring What is the expected outcome of an action?

Planning Which action will lead to the intended goal?

Diagnosis What has caused something to happen?

Question Answering Why has an action being performed?

Learning How to explore the parameter space of an action effectively?

This incomplete list of contexts and queries illustrates where the developed framework and learned models can be employed in order to adjust the behavior, and to improve the overall performance, of robots. Hence, we believe that the underlying idea and the developed methods of this work can have a broad impact in field of robotics.

6. Discussion

In this section, we discuss why autonomous robots should be endowed with methods allowing them to make temporal projections about naive physics problems. We provide arguments to base these methods on detailed physical simulations and elaborate on the necessary fidelity of these simulations. Furthermore, we outline how our 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,

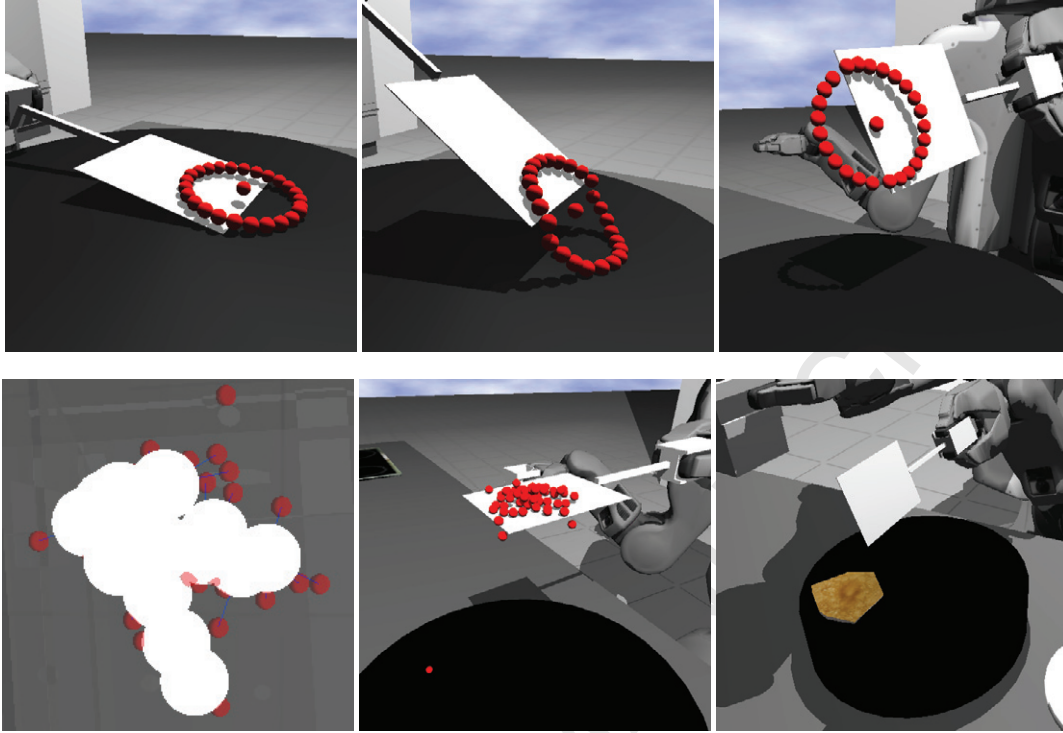


Figure 25: The pancake model in different views: a generic circular model (up), a generated model showing the joint structure and flexible mesh (down).

task and environment conditions change all the time. Therefore robots need flexible mechanisms to reason about the parameters of their control programs.

In the literature there exist approaches which use symbolic reasoning methods to make inferences about simple physical problems (Lifschitz, 1998; Morgenstern, 2001). The main limitations of these approaches in the context of robotics are threefold: (a) important details such as positions of manipulators and objects are abstracted away; (b) variants of problems such as 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. Moreover, to obtain a simulation that is as close as possible to the real world, parameters of the simulator 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, our system cannot make predictions in a 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 consider task failures.

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

Although it would be desirable to use the presented approach for action planning with realistic physical models, we are currently not aiming at either the high performance and or realistic models that this would necessitate. Instead,

the developed system represents a proof-of-concept of 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 animation industry (e.g. (Cho et al., 2007)), providing powerful technologies that robots could employ.

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 parameterized 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 ROS¹⁴ service, which takes arbitrary Prolog queries as a request and provides the respective variable bindings as a response. This allows naive physics reasoning for manipulation tasks to be flexibly integrated into control programs and planners in order to effectively change the robot's behavior as outlined by the example given in Section 1.3.

Planning is increasingly considering physical platforms in complex, real world environments. The presented framework could provide more precise guidance in the planning process since the simulation-based methods for making temporal projections are tightly linked to the technical details of platforms under question. However, currently this would only be possible offline as we have laid out above. Naive physics reasoning could also be used as a tool for developing robot control programs. Programmers could recognize and prevent problems from 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 straightforward way. In general, the use of the open source software such as Gazebo and ROS allow researchers and developers to apply 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.

It is clear that one would not want to adopt our approach 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 such as robot navigation in dynamic environments, where current methods are not effective.

7. Conclusions

In this paper, we presented a framework for envisioning the effects of everyday robot manipulation actions using physics-based simulations.

Within this framework, we 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. We conducted experiments for three scenarios in which the 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. We also discussed the demands of equipping robots with means to perform naive physics reasoning and provided arguments for basing this reasoning on detailed physical simulation.

In future work, we will continue our research on how we can extract information from human demonstrations performed in a virtual manipulation environment to automatically determine the parameter space of robot manipulation actions from timelines (Kunze et al., 2013). Furthermore, to cope with the uncertainty inherent in robot manipulation tasks we will extend our approach by integrating probabilistic representations over timelines as briefly outlined in Section 3.1. Finally, we will also integrate our system with a real robot. This integration will pose several challenges. For instance, a robot has to recognize task-related objects and it has to estimate their poses in order to assert and instantiate a new scenario within the framework. Secondly, during the execution of an action plan the robot has to observe the objects in the real world and it has to continuously estimate their states in order to allow the framework to

¹⁴<http://www.ros.org>

generate timelines. For some types of objects, such as fluids, this would be very challenging. However, as a first step towards an integrated system we will start with a rigid object manipulation scenario.

We believe that the presented framework provides important functionality for robots by giving them the ability to autonomously determine the action parameterizations for underspecified and ambiguous instructions by the means of logic programs using simulation-based temporal projections.

Acknowledgments

We would like to thank our colleagues and students in the department for their help: Mihai Emanuel Dolha, Andrei Haidu, Emitza Guzman, Johannes Mikulasch, Reinhard Klapfer and Nick Hawes.

This work has been supported by the EU FP7 Project *RoboHow* (grant number 288533). Additionally, it was partly funded by the *CoTeSys* cluster of excellence (Cognition for Technical Systems, <http://www.cotesys.org>), part of the Excellence Initiative of the German Research Foundation (DFG).

References

- Allen, J., 1983. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM* 26, 832–843.
- Beetz, M., Klank, U., Kresse, I., Maldonado, A., Mösenlechner, L., Pangercic, D., Rühr, T., Tenorth, M., 2011. Robotic Roommates Making Pancakes, in: 11th IEEE-RAS International Conference on Humanoid Robots, Bled, Slovenia. pp. 529–536.
- Beetz, M., Mösenlechner, L., Tenorth, M., 2010a. CRAM – A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, Taipei, Taiwan. pp. 1012–1017.
- Beetz, M., Tenorth, M., Jain, D., Bandouch, J., 2010b. Towards Automated Models of Activities of Daily Life. *Technology and Disability* 22, 27–40.
- Berleant, D., Kuipers, B., 1992. Qualitative-Numeric Simulation with Q3, in: Recent advances in qualitative physics, The MIT Press. pp. 3–16.
- Brown, J., Sorkin, S., Bruyns, C., Latombe, J.C., Montgomery, K., Stephanides, M., 2001. Real-Time Simulation of Deformable Objects: Tools and Application, in: Computer Animation, 2001: The Fourteenth Conference on Computer Animation, pp. 228–236.
- Cho, J.H., Xenakis, A., Gronsby, S., Shah, A., 2007. Anyone Can Cook – Inside Ratatouille’s Kitchen, in: SIGGRAPH ’07: ACM SIGGRAPH 2007 courses.
- Davis, E., 1990. Representations of Commonsense Knowledge. Morgan Kaufmann series in representation and reasoning, Morgan Kaufmann.
- Davis, E., 2008. Pouring liquids: A study in commonsense physical reasoning. *Artif. Intell.* 172, 1540–1578.
- Davis, E., 2010. Ontologies and Representations of Matter, in: Fox, M., Poole, D. (Eds.), Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11–15, 2010, AAAI Press.
- Dean, T., 1989. Using Temporal Hierarchies to Efficiently Maintain Large Temporal Databases. *J. ACM* 36, 687–718.
- Dogar, M., Srinivasa, S., 2011. A Framework for Push-grasping in Clutter. *Robotics: Science and Systems VII*.
- Faloutsos, P., van de Panne, M., 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. *Artif. Intell.* 24, 85–168.
- Forbus, K.D., Falkenhainer, B., 1990. Self-Explanatory Simulations: An Integration of Qualitative and Quantitative Knowledge, in: AAAI, pp. 380–387.
- Frank, B., Stachniss, C., Schmedding, R., Teschner, M., Burgard, W., 2009. Real-world Robot Navigation amongst Deformable Obstacles, in: ICRA’09: Proc. of the 2009 IEEE int. conf. on Robotics and Automation.
- Frenkel, D., Smit, B., 2001. Understanding Molecular Simulation, Second Edition: From Algorithms to Applications (Computational Science). 2 ed., Academic Press.
- Ghallab, M., 1996. On Chronicles: Representation, On-line Recognition and Learning, in: Principles of Knowledge Representation and Reasoning- International Conference-, Morgan Kaufmann Publishers. pp. 597–607.
- Gould, H., Tobochnik, J., 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., Zumbusch, G., 2007. Numerical Simulation in Molecular Dynamics: Numerics, Algorithms, Parallelization, Applications. 1st ed., Springer Publishing Company, Incorporated.
- Haazebroek, P., Hommel, B., 2009. Anticipative Control of Voluntary Action: Towards a Computational Model, in: Pezzulo, G., Butz, M., Sigaud, O., Baldassarre, G. (Eds.), Anticipatory Behavior in Adaptive Learning Systems. Springer Berlin / Heidelberg. volume 5499 of *Lecture Notes in Computer Science*, pp. 31–47.
- Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H., 2009. The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl.* 11, 10–18.
- Hayes, P., 1979. The Naive Physics Manifesto, in: Michie, D. (Ed.), Expert Systems in the Micro Electronic Age. Edinburgh University Press, pp. 242–270.
- Hayes, P., 1985. The Second Naive Physics Manifesto, in: Hobbs, J.R., Moore, R.C. (Eds.), Formal Theories of the Commonsense World. Ablex, Norwood, NJ, pp. 1–36.
- Hesslow, G., 2012. The Current Status of the Simulation Theory of Cognition. *Brain Research Reviews* 1428, 71–79.
- Ingram, J., Howard, I., Flanagan, J., Wolpert, D., 2010. Multiple Grasp-Specific Representations of Tool Dynamics Mediate Skillful Manipulation. *Curr Biol*.

- Johnston, B., Williams, M., 2008. Comirit: Commonsense Reasoning by Integrating Simulation and Logic, in: Artificial General Intelligence 2008: Proceedings of the First AGI Conference, IOS Press. p. 200.
- Johnston, B., Williams, M., 2009. Autonomous Learning of Commonsense Simulations, in: Int. Symposium on Logical Formalizations of Commonsense Reasoning.
- Kato, F., Hanaoka, Y., Ngoc, T.N., Keoki, D., Mitake, H., Aoki, T., Hasegawa, S., 2009. Interactive Cooking Simulator: To Understand Cooking Operation Deeply, in: ACM SIGGRAPH 2009 Emerging Technologies.
- Kemp, C., Edsinger, A., Torres-Jara, E., 2007. Challenges for Robot Manipulation in Human Environments. *IEEE Robotics and Automation Magazine* 14, 20–29.
- Klapfer, R., Kunze, L., 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.
- 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, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. pp. 299–304.
- de Kleer, J., Brown, J.S., 1982. Foundations of envisioning, in: Proc. of AAAI-82, Pittsburgh, PA. pp. 434–437.
- Kowalski, R., Sergot, M., 1986. A Logic-based Calculus of Events. *New Gen. Comput.* 4, 67–95.
- Kunze, L., 2014. Naïve Physics and Commonsense Reasoning for Everyday Robot Manipulation. Dissertation. Technische Universität München. München, Germany.
- Kunze, L., Dolha, M.E., Beetz, M., 2011a. Logic Programming with Simulation-based Temporal Projection for Everyday Robot Object Manipulation, in: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), San Francisco, CA, USA.
- Kunze, L., Dolha, M.E., Guzman, E., Beetz, M., 2011b. Simulation-based Temporal Projection of Everyday Robot Object Manipulation, in: Yolum, Tumer, Stone, Sonenberg (Eds.), Proc. of the 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011), IFAAMAS, Taipei, Taiwan.
- Kunze, L., Haidu, A., Beetz, M., 2013. Acquiring Task Models for Imitation Learning through Games with a Purpose, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Tokyo Big Sight, Japan.
- Kunze, L., Roehm, T., Beetz, M., 2011c. Towards Semantic Robot Description Languages, in: IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China. pp. 5589–5595.
- Lakshmanan, K., Sachdev, A., Xie, Z., Berenson, D., Goldberg, K., Abbeel, P., 2012. A constraint-aware motion planning algorithm for robotic folding of clothes, in: Desai, J.P., Dudek, G., Khatib, O., Kumar, V. (Eds.), Experimental Robotics - The 13th International Symposium on Experimental Robotics, ISER 2012, June 18–21, 2012, Québec City, Canada, Springer. pp. 547–562. URL: http://dx.doi.org/10.1007/978-3-319-00065-7_37, doi:10.1007/978-3-319-00065-7_37.
- 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., 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, ACM, New York, NY, USA. pp. 225–234.
- Majtey, A.P., Lamberti, P.W., Prato, D.P., 2005. Jensen-Shannon Divergence as a Measure of Distinguishability between Mixed Quantum States. *Phys. Rev. A* 72, 052310.
- McDermott, D., 1992. Robot Planning. *AI Magazine* 13, 55–79.
- Merikli, T.A., Veloso, M., Akin, H.L., 2013. Achievable push-manipulation for complex passive mobile objects using past experience, in: Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems, International Foundation for Autonomous Agents and Multiagent Systems. pp. 71–78.
- Miller, R., Morgenstern, L., 2009. Common Sense Problem Page. <http://www-formal.stanford.edu/leora/commonsense>.
- Morgenstern, L., 2001. Mid-Sized Axiomatizations of Commonsense Problems: A Case Study in Egg Cracking. *Studia Logica* 67, 333–384.
- Mösenlechner, L., 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., Sedgewick, R., 1992. Deterministic Skip Lists, in: SODA, pp. 367–375.
- Nyga, D., Beetz, M., 2012. Everything Robots Always Wanted to Know about Housework (But were afraid to ask), in: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vilamoura, Portugal.
- Oztop, E., Kawato, M., Arbib, M.A., 2006. Mirror Neurons and Imitation: A Computationally Guided Review. *Neural Networks* 19, 254–271.
- Siskind, J.M., 2001. Grounding the Lexical Semantics of Verbs in Visual Perception using Force Dynamics and Event Logic. *J. Artif. Intell. Res. (JAIR)* 15, 31–90.
- Smith, D., Morgan, B., 2010. IsisWorld: An Open Source Commonsense Simulator for AI Researchers, in: AAAI Workshops.
- Svensson, H., Ziemke, T., 1999. Making Sense of Embodiment: Simulation Theories and the Sharing of Neural Circuitry Between Sensorimotor and Cognitive Processes. *Learning*, 1309–1314.
- Tenorth, M., Kunze, L., Jain, D., Beetz, M., 2010a. KNOWROB-MAP – Knowledge-Linked Semantic Object Maps, in: 10th IEEE-RAS International Conference on Humanoid Robots, Nashville, TN, USA. pp. 430–435.
- Tenorth, M., Nyga, D., 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), Anchorage, AK, USA. pp. 1486–1491.
- Ueda, R., Ogura, T., Okada, K., 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, pp. 374–381.
- Vondrak, M., Sigal, L., Jenkins, O.C., 2008. Physical Simulation for Probabilistic Motion Tracking, in: CVPR.
- Weigelt, M., Kunde, W., Prinz, W., 2006. End-state Comfort in Bimanual Object manipulation. *Experimental Psychology* 53, 143–148.
- Weitnauer, E., Haschke, R., 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, Springer-Verlag, Berlin, Heidelberg. pp. 144–155.
- Weld, D.S., Kleer, J.d. (Eds.), 1990. Readings in Qualitative Reasoning about Physical Systems. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Zickler, S., Veloso, M., 2009. Efficient physics-based planning: sampling search via non-deterministic tactics and skills, in: AAMAS '09: Proc. of The 8th Int. Conf. on Autonomous Agents and Multiagent Systems.