Knowledge-based Specification of Robot Motions

Moritz Tenorth and **Georg Bartels** and **Michael Beetz**[1]

Abstract.

In many cases, the success of a manipulation action performed by a robot is determined by how it is executed and by how the robot moves during the action. Examples are tasks such as unscrewing a bolt, pouring liquids and flipping a pancake. This aspect is often abstracted away in AI planning and action languages that assume that an action is successful as long as all preconditions are fulfilled. In this paper we investigate how constraint-based motion representations used in robot control can be combined with a semantic knowledge base in order to let a robot reason about movements and to automatically generate executable motion descriptions that can be adapted to different robots, objects and tools.

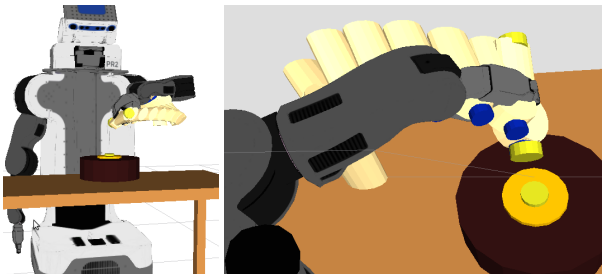

Figure 1. Two views of a pouring task that has been described in the proposed language and executed by the PR2 robot. The task description is composed of parts that are inherited from a motion library and can automatically be adapted to different objects.

1 Introduction

In AI planning, actions are commonly considered as "black boxes" that are described by their pre- and postconditions and possibly their composition from sub-actions. Based on this meta-information, planning methods can generate sequences of actions to get from a given initial state to a desired goal state. Such a representation abstracts away from everything that happens during the action: The assumption is that an action is successful if all preconditions hold at the time it is started, and that all postconditions will hold afterwards. While this strict model is somewhat relaxed when probabilistic models are used for planning (which are able to model uncertain action outcomes), actions are still considered as "black boxes". Rarely, there have been efforts towards more realism in modeling and reasoning. A famous example is the egg cracking action that has been described in predicate logic, allowing inference about the effects of actions performed with the egg [12]. However, the focus of this work was more on inferring "what happens if" and less on choosing parameters for successful action execution.

When planning robot manipulation tasks, such an abstracted view is often not sufficient. For many tasks, success or failure is determined by the choice of action parameters and the motions that are performed – for example for screwing a bolt into a hole, flipping a pancake, pouring liquids into a bowl, or stirring cookie dough (Figure 1). The representation therefore needs to combine semantic aspects, such as the goals of an action, with information about which motions to perform, and with rules for adapting them to the geometry of the involved objects. In this paper, we investigate how semantically rich (symbolic) task representations can be extended with meaningful descriptions of robot motions. Our aim is to build composable and semantically interpretable task models that can be parameterized with models of the involved objects and that are thus re-usable in different contexts. We describe motions using the task function approach [13] which analyses the kinematics and dynamics of scenes in terms of differentiable vector functions. Following the example of [3], we specify robot motions as robot-specific task functions with desired constraints over their outputs as set points. The task function approach enables us to automatically generate the corresponding feedback control laws at runtime (Section 5). In our previous work, we have developed a symbolic description language for motion tasks based on this approach [1, 9]. We proposed to assemble task functions out of sets of one-dimensional relations between coordinate frames, allowing the flexible and modular description of complex motions.

These control-level descriptions, however, lack semantic meaning: At the controller level, there is no notion of a "goal" to be achieved, or an "object" to be manipulated – movements are only described by a set of relations between coordinate frames. This impedes automatic adaption of the descriptions to situations involving novel objects and tools. We address this issue by integrating the constraint-based models with a formal knowledge representation language, automated inference and plan-based task execution. Based on semantic representations of motion templates, part-based object models and descriptions of the robot's structure, our approach can automatically generate motion descriptions that are grounded in the available tools, objects and robots. The main contributions of this paper are (·) motion descriptions that combine the strengths of semantic representations with constraint-based specifications for improved generalization and composability; (·) extraction of generic motion patterns of motions into a library from which particular movements can be derived; (·) methods for describing the motions in terms of relations between semantically meaningful object parts that can automatically be identified in object models and that generalize across particular object types; and (·) techniques for the automatic selection of the robot components to be used for a motion (e.g. LeftArm, RightArm) based on a semantic robot model.

[1] Institute for Artificial Intelligence and TZI (Center for Computing Technologies), University of Bremen, Germany
E-mail: {tenorth, georg.bartels, beetz}@cs.uni-bremen.de

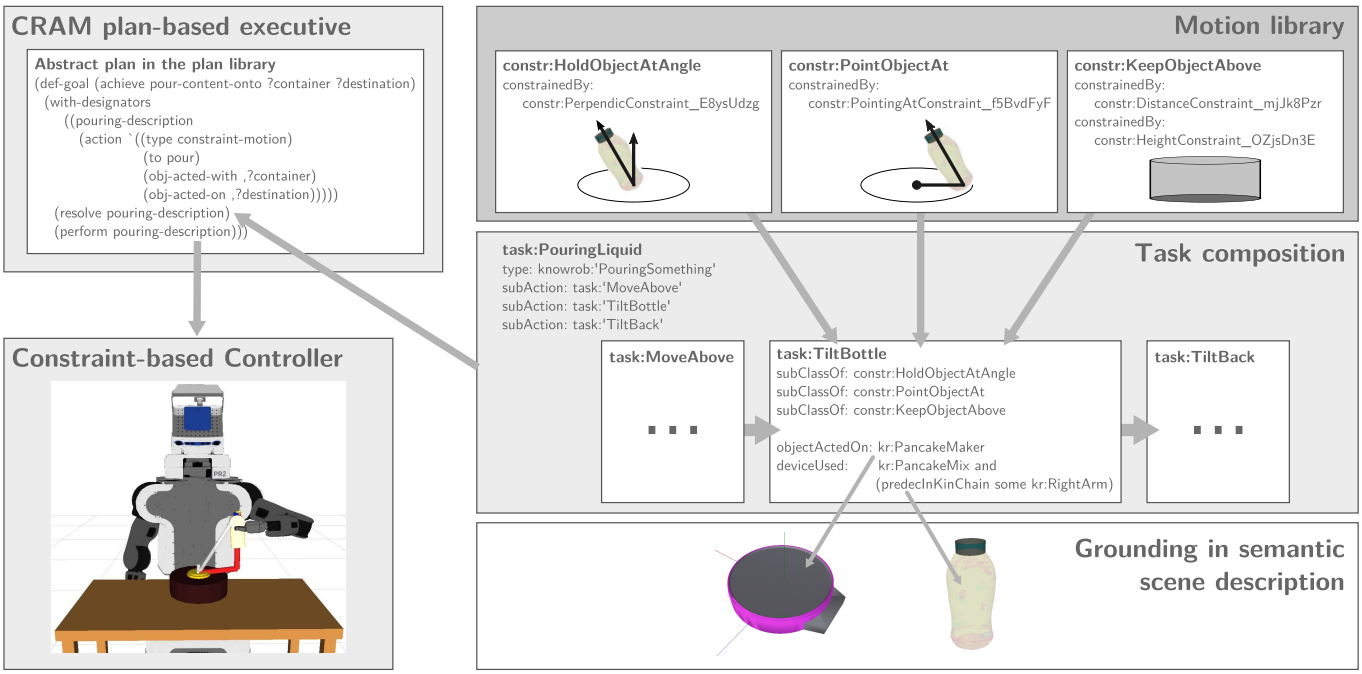

Figure 2. Overview of the proposed system: Programmers can define tasks by deriving motions from templates defined in the motion library. The abstract descriptions in these templates are grounded in concrete parts of objects that can be found in the scene. The resulting description is interpreted by the plan-based executive and sent to the constraint-based control framework for execution.

2 Related Work

While there has been substantial work in Artificial Intelligence research on the representation of actions, those approaches usually abstract the patterns away that we are interested in in this paper. For example, action languages based on the Situation Calculus, e.g. [17], allow reasoning about the effects of actions, but on a purely symbolic and logical level. In previous work, we have developed the RoboEarth language [15] as a representation for sharing task-, map- and object-related information between robots, though tasks are also only modeled on a symbolic level. In task-level planning approaches such as STRIPS [5], HTN [4] or PDDL [6], actions are described by their preconditions and effects, again abstracting away from the motions that "happen in between". Recently, different groups have explored ways to integrate task- with motion planning [21, 8] or with lower-level information sources [7]. While these approaches are to some extent similar, they focus primarily on achieving a given goal configuration, instead of the description of the motion for achieving it, and on the *generation* of a plan rather than the semantic representation of the involved steps. In robotics, efforts have been made to create domain-specific languages for robot behavior specification. Leidner et al [11] store manipulation-related information associated with the respective object models. Thomas et al. present an approach for describing high-level behavior using UML statecharts [18]. Vanthienen et al. propose a domain-specific language as a convenient way for creating constraint-based task descriptions [19], defining virtual kinematic chains between the tool and the manipulated object. In our previous work, we have created a similar language [1] that, however, uses sets of one-dimensional constraints instead of the virtual kinematic chains. These research efforts mostly aim at facilitating the life of a human programmer by providing higher-level program structures, rather than creating semantically rich machine-understandable descriptions which we strive for with this work.

3 Overview

Figure 2 explains the different components of our approach. Descriptions of single motions or parts thereof can be stored in a *motion library* in the robot's knowledge base (Section 4.1). These motions are modeled as action classes that are each linked to a set of constraints to be considered when performing the motion. The constraints refer to abstract descriptions of object parts, but do not specify which type of object is to be used. Programmers can describe novel movements as a combination of these motion templates by inheriting from the motion classes in the library and by adding task-specific information on the types of objects and the parts of the robot to be used (Section 4.2). This task description is still very abstract and refers to objects only at the class level. For execution, it has to be grounded in the actual situational context the action will be performed in. This includes the selection of appropriate object parts and robot components according to the class-level specifications, which the system does using geometric-semantic models of objects and their functional parts (Section 4.3). This grounding step, translating from abstract class descriptions to object parts, is key for the generalization capabilities of the system, as it allows the application of the same motion templates to very different objects as long as all required functional parts can be identified. The resulting task description can then be converted into a robot plan and be executed using the constraint-based controllers (Section 5).

4 Knowledge-based Motion Representation

As knowledge processing and inference framework we use KNOWROB [14], a knowledge processing system specially designed for being used on robots, which is implemented in Prolog and can load knowledge stored in the Web Ontology Language OWL [20]. The motion descriptions proposed in this paper are represented in OWL and extend the RoboEarth language [15] that provides struc-

tures for representing actions, object models and environment maps. KNOWROB provides sophisticated methods for including custom reasoning rules into the Prolog-based inference procedure that allow to perform inference beyond the capabilities of OWL, which we use for example for selecting the most appropriate object parts to be used for a task.

4.1 Motion Library

The motion library contains abstractly described *motion patterns* – partial descriptions of motions between objects such as "keep above" or "keep to the left of" that form the vocabulary for robot task descriptions. Each motion pattern is described by a set of constraints between a part of a tool and a part of an object in the world. The patterns do not yet refer to specific objects, but to generic object parts (e.g. the center or the main axis) that will later be grounded in the objects to be used for a task. Each constraint has the following properties: Its type (e.g. *HeightConstr*) describes the kind of relation encoded; the language currently includes the relations *left of*, *right of*, *above of*, *below of*, *in front of* and *behind*, as well as the 3D distance, the height above a surface, and the angle between two vectors (called *PerpendicConstr*). The *toolFeature* and *worldFeature* specify which object parts of the tool and of the manipulated object are related by the constraint. These object parts are described in terms of a class definition (e.g. *ObjectMainAxis*, *Handle*) – either by giving a named class (the *CenterOfObject* in the example below), or by using an OWL class restriction that describes a class by its properties. Depending on the type of constraint, a third feature in the world is needed to serve as reference, e.g. to define directional relations such as *leftOf* or *above*. In addition, each constraint specifies a lower and upper bound for its values. In the examples below, a postfix four-letter hash has been appended to the class names in order to obtain unique identifiers. The following listing shows the definition of the *KeepObjectAbove* motion pattern:[2]

```
Class: KeepObjectAbove
    SubClassOf:
        ConstrMotion,
        constrainedBy some HeightConstr_OZjs,
        constrainedBy some InFrontOfConstr_Sv4U,
        constrainedBy some LeftOfConstr_fePC,

Class: HeightConstr_OZjs
    SubClassOf:
        HeightConstraint,
        toolFeature some CenterOfObject,
        worldFeature some SupportingPlane,
        refFeature value "/torso_lift_link",
        constrLowerLimit value 0.25,
        constrUpperLimit value 0.3

Class: InFrontOfConstr_Sv4U
    SubClassOf:
        InFrontOfConstraint,
        toolFeature some CenterOfObject,
        worldFeature some SupportingPlane,
        refFeature value "/torso_lift_link",
        constrLowerLimit value -0.03,
        constrUpperLimit value 0.03

Class: LeftOfConstr_fePC
       [...]
```

4.2 Task Definition

Several of these motion patterns can be combined to form an action, and multiple actions can be combined to a multi-step task description. For example, the task of pouring liquids from a bottle into a pan consists of the three phases "moving the bottle over the pan", "tilting the bottle" and "bringing the bottle back into vertical position". Each sub-action combines different patterns, e.g. for holding the bottle above the pan and for keeping it upright.

This structure can be expressed elegantly in our language. The following task description for a pouring task first describes the class *PouringSomething* as a subclass of *Pouring* with three subactions *MoveAbovePan*, *TiltBottle* and *TiltBack*. Since OWL does not inherently describe the order of the subactions, we introduce pair-wise ordering constraints that impose a partial order among them. They are not to be confused with the motion constraints that describe spatial relations between object parts. The classes for describing the subactions are each derived from different motion patterns and inherit the constraints described for these patterns. For example, the class *MoveAbovePan* inherits the constraints *HeightConstr_OZjs*, *InFrontOfConstr_Sv4U* and *LeftOfConstr_fePC* from the class *KeepObjectAbove* and the constraint *PerpendicConstr_qpdE8yUz* from the class *HoldObjectUpright*. This very concise formulation is possible because constraints are a composable description of motions and thus allow the combination of different patterns.

```
Class: PouringSomething
    SubClassOf:
        Pouring,
        subAction some MoveAbovePan,
        subAction some TiltBottle,
        subAction some TiltBack,
        orderingConstraints value Pour01_h0t7,
        orderingConstraints value Pour02_3KER,
        orderingConstraints value Pour12_TE30

Class: MoveAbovePan
    SubClassOf:
        KeepObjectAbove,
        HoldObjectUpright,
        deviceUsed some BottleCapInLeftHand,
        objectActedOn some PancakeMaker

Class: TiltBottle
    SubClassOf:
        HoldObjectAtAngle,
        KeepObjectAbove,
        deviceUsed some BottleCapInLeftHand,
        objectActedOn some PancakeMaker

Class: TiltBack
    SubClassOf:
        KeepObjectAbove,
        HoldObjectUpright,
        deviceUsed some BottleCapInLeftHand,
        objectActedOn some PancakeMaker
```

4.3 Resolving Abstract Object Descriptions

The motion patterns only refer to abstract parts of (so far unknown) objects. The task definition adds information on which types of objects are to be used as tool (the *deviceUsed*) and as object to be manipulated (the *objectActedOn*). The information about the parts (from the pattern definition) needs to be combined with the description of which objects these parts belong to (from the task definition) and with models of the objects' geometry to identify the positions of the object parts to be used as *toolFeature* and *worldFeature*. For this purpose, we employ the part-based object representations proposed in [16] that combine geometric aspects (position of parts and geometric primitives approximating their shapes) with semantic properties (type and function of these components). The models can automatically be generated from CAD models using geometric segmentation techniques. In our experiments we used CAD models downloaded from online repositories such as the Google/Trimble 3D warehouse[3].

[2] In this paper, we use the OWL Manchester syntax and omit the OWL namespaces for better readability.

[3] http://3dwarehouse.sketchup.com/

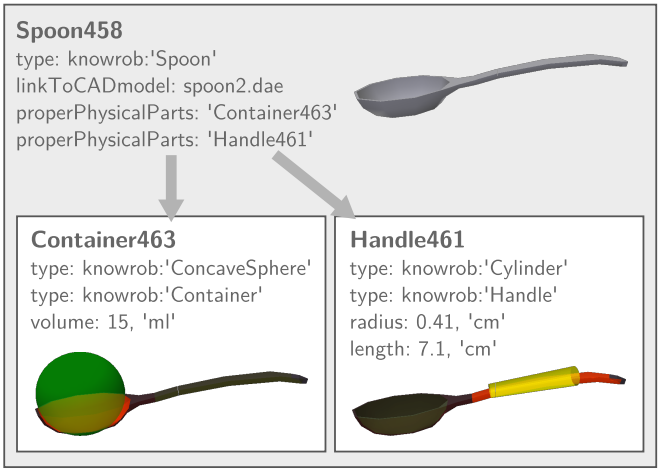

Figure 3. Part-based object representation generated from a CAD model. By geometric analysis, functional parts are identified and stored in the knowledge base. The resulting models combine geometric (poses, dimensions) and semantic aspects (types, properties).

The part-based model is represented in the knowledge base (Figure 3), enabling us to specify logical rules on the level of semantic object parts (e.g. a handle, the main axis, a bottle cap), that are evaluated on the geometric model. Using this approach, the system can make the link between abstract class descriptions of object components and the corresponding geometric parts.

More specifically, the robot has to determine a suitable part *FeatureInst* that matches the definition of the *FeatureClass* from the constraint description (e.g. *CenterOfObject*), and that is part of an object of type *ObjClass* given by the motion description. The following rules are examples of how this functionality is implemented. In the simplest case, the required parts can already be found in the part-based object model (first rule). The *owl_individual_of* predicate supports complete OWL inference, i.e. the classes do not have to be simple named classes, but can also be complex OWL class restrictions. However, some required parts may not be available in the precomputed object model and have to be identified by special rules that define features such as the main axis of an object (second and third rule below). This rule-based approach allows to easily extend the set of primitives that can be used for describing motions.

```
% Evaluate on pre-computed object model
object_feature(FeatureClass, ObjClass, FeatureInst) :-
    owl_individual_of(ObjInst, ObjClass),
    owl_has(ObjInst, properPhysicalParts, FeatureInst),
    owl_individual_of(FeatureInst, FeatureClass).

% Compute main axis of object
object_feature(FeatureClass, ObjClass, FeatureInst) :-
    rdf_equal(FeatureClass, 'ObjectMainAxis'),
    owl_individual_of(ObjInst, ObjClass),
    object_main_cone(ObjInst, FeatureInst).

% Compute object center: returns object instance itself
object_feature(FeatureClass, ObjClass, FeatureInst) :-
    rdf_equal(FeatureClass, 'CenterOfObject'),
    owl_individual_of(ObjInst, ObjClass),
    FeatureInst = ObjInst.
[...]
```

4.4 Adapting Motions to a Robot

Actions can include multiple simultaneous motions that are for instance performed by the left and right arm of a bimanual robot. This raises the problems of (a) assigning motion patterns to robot parts, and (b) selecting features on the correct object in case both hands use the same kind of tool. We therefore explicitly model the robot to be able to reason about the relation between the motions, objects and robot parts. For defining dependencies on robot components, we abstract away from the concrete kinematic structure of a particular robot and use abstract classes such as *LeftArm*. These abstract descriptions can be grounded in a detailed model of a robot's kinematic and sensory structure described in the Semantic Robot Description Language (SRDL [10]) that is also based on OWL and available in the same knowledge base. By combining the part-based object models with the robot model, we can specify in detail which part of which object is to be used for a motion. For example, the pouring task is described by the cap of the bottle of pancake mix that is attached to some sub-component of the robot's left arm using an OWL class restriction:

```
Class: BottleCapInLeftHand
    EquivalentTo:
        BottleCap and (physicalPartOf some
            (PancakeMix and (predecInKinChain some LeftArm)))
```

In order to resolve this abstract description, the robot selects an object that complies with this class restriction, i.e. one that is of the correct type and in this case attached to some part of the left arm. By exploiting the transitivity of the *predecInKinChain* relation, the robot considers all sub-components of the arm, including for instance its left gripper. The following object complies with the description, and the *Cone_7c7s* as part of the bottle is classified as bottle cap and selected:

```
Individual: mondamin-pancake-mix_SjoS
    Types:
        PancakeMix
    Facts:
        predecInKinChain pr2_l_gripper_palm_link,
        properPhysicalParts  Sphere_qEux,
        properPhysicalParts  Cone_7c7S,
        [...]
```

While this example used the simple named class *LeftArm* for describing which robot part to use, the system supports arbitrary OWL restrictions to describe the required properties, for instance support for force control or a minimum lifting force.

5 Execution of Motion Descriptions

The motion descriptions provide a generic plan for an activity with task-specific information about movements. Our robots are controlled by the CRAM plan-based executive that executes plans written in the CRAM Plan Language CPL [2]. CPL is an expressive Lisp-based language that provides sophisticated control structures for describing concurrent reactive robot behavior. A core concept of CPL are designators – partial descriptions of objects, movements or locations that are first-class objects in the language that the robot can reason about. In our case, an action designator, such as like the *pouring-description* in Figure 2 is filled by querying the knowledge base for a constraint-based motion description. The Prolog queries return a complete constraint-based specification in which the object instances and features have been resolved as described in Section 4.3. The resulting action designators contain effective, i.e. executable, motion descriptions with potentially several partially ordered motion phases.

The CRAM executive reconfigures the constraint-based motion controller by sending it a new motion phase description that contains a set of constraints. The controller translates each single constraint, e.g. *bottle cap above frying pan*, into a one-dimensional feature function $f(\mathbf{x_t}, \mathbf{x}_o, \mathbf{x}_v)$. $\mathbf{x}_t$ and $\mathbf{x}_o$ denote the Cartesian poses of the tool and object features, respectively. Additionally, $\mathbf{x}_v$ represents the reference frame w.r.t. which the constraint shall be evaluated. Stacking

the one-dimensional feature functions yields the feature function of the task: $\mathbf{y} = \mathbf{f}(\mathbf{x}_t, \mathbf{x}_o, \mathbf{x}_v)$. Note that this formalization requires known transformations between tool and object poses for all features and that all features share a common reference frame.
Assuming that the tool is controllable by the end-effector of the robot and that both the object and reference frame are quasistatic during one control cycle, i.e. $\dot{\mathbf{x}}_o \approx \dot{\mathbf{x}}_v \approx \mathbf{0}$, we derive the *interaction matrix* $\mathbf{H}$ as the partial derivative matrix of $\mathbf{f}(\mathbf{x}_t, \mathbf{x}_o, \mathbf{x}_v)$ w.r.t. $\partial \mathbf{x}_t$. $\mathbf{H}$ allows us to map from velocities in the feature space $\dot{\mathbf{y}}$ to the Cartesian tool twist $\mathbf{t}_t$:

$$\dot{\mathbf{y}} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}_t}\dot{\mathbf{x}}_t + \frac{\partial \mathbf{f}}{\partial \mathbf{x}_o}\dot{\mathbf{x}}_o + \frac{\partial \mathbf{f}}{\partial \mathbf{x}_v}\dot{\mathbf{x}}_v = \frac{\partial \mathbf{f}_t}{\partial \mathbf{x}_t}\dot{\mathbf{x}}_t = \mathbf{H}\mathbf{t}_t. \quad (1)$$

Further assuming that the tool is rigidly attached to the end-effector of the robot, we use the Jacobian $\mathbf{J}_R$ of the arm of the robot to map to the space of joint velocities $\dot{\mathbf{q}}$:

$$\dot{\mathbf{y}} = \mathbf{H}\mathbf{J}_R\dot{\mathbf{q}}. \quad (2)$$

A feedback controller calculates a desired $\dot{\mathbf{y}}_{des}$ from the current scene and the respective constraints in every control cycle. Multiplying $\dot{\mathbf{y}}_{des}$ with the weighted pseudo-inverse of $\mathbf{H}\mathbf{J}_R$ yields the instantaneous desired joint velocities $\dot{\mathbf{q}}_{des}$. For more details on the constraint-based controller we refer the reader to [1]. This programmable controller exposes a constraint language interface with few restrictions and allows run-time configuration. The constraint-based descriptions serve as interlingua between the motion controller and the plan-based executive and are grounded on both ends of the system (the CAD model reasoning and the joint state control, respectively). Furthermore, they allow meaningful feedback given by the controller in constraint space, for example that all constraints but the *have bottle cap at least 15cm above the pan* were fulfilled when the arm stopped.

6 Experiments

We evaluate the contributions of this paper by formulating the task of pouring liquids from a bottle (in our case pancake batter) in the proposed language and executing it on our PR2 robot. The representations and programs used for the experiments in this article have been released as open-source software.[4]. Despite being rather simple, the pouring task already combines different constraints on position and orientation in 6D space. And even in this simple task, the benefits of explicitly representing and inheriting motion properties can be seen: The three motion phases (approaching the goal area, tilting the bottle and tilting back) share the common motion patterns *KeepObjectAbove* and *HoldObjectUpright* which reduces redundancy in the descriptions and allows the robot to reason about common aspects of these motion phases. The following queries explain the main steps for reading a task description and grounding it in object models. The robot starts with reading the motion phases of the *PouringSomething* action and, for each phase, reading the constraints that are to be considered.

```
?- plan_subevents('PouringSomething', Phases).
Phases = ['MoveAbovePan','TiltBottle','TiltBack'].

?- motion_constraint('MoveAbovePan', C).
C = 'PerpendicularityConstraint_qpdE'
C = 'HeightConstraint_OZjs'
C = 'LeftOfConstr_fePC'
C = 'InFrontOfConstraint_Sv4U'
```

[4] Links to the source code repositories and a video of the experiments on the real PR2 robot can be found at http://knowrob.org/doc/motion_constraints

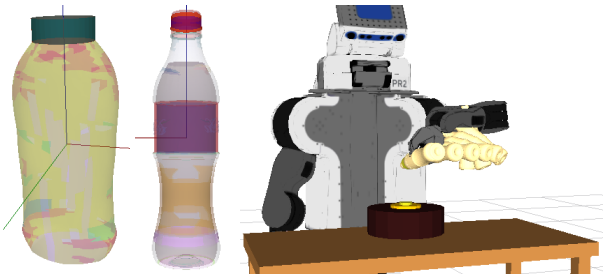

Figure 4. Segmented models for a bottle of pancake mix (left) and a soda bottle (center). The bottle cap was automatically identified in these very differently shaped bottles. This allowed executing the same task description using the novel bottle.

These abstract constraints are then combined with the description of the current scene in order to identify the object features that are to be controlled. The following query determines the values for the *HeightConstraint_OZjs* for the device and tool that are specified in the task description using the *object_feature* rules described in Section 4.3.

```
?- constraint_properties(
       'BottleCapInLeftHand', 'PancakeMaker',
       'HeightConstraint_OZjs',
        Type, ToolFeature, WorldFeature, RefFrame,
        Lower, Upper).
Type = 'HeightConstraint',
ToolFeature = 'Cone_7c7S',
WorldFeature = 'FlatPhysicalSurface_AEFl',
RefFrame = '/torso_lift_link',
Lower = 0.25,
Upper = 0.3.
```

Before applying the *object_feature* rules, the robot has to identify suitable objects that comply with the class restriction for the tool and world object, i.e. the *BottleCapInLeftHand* and the *PancakeMaker*. The selection of the tool object requires reasoning about the structure of both the robot and the tool. The inferences for answering the query for individuals of the *BottleCapInLeftHand* given below involve reasoning about the super-components of the gripper the candidate object is attached to (the *pr2_l_gripper_palm_link*):

```
?- owl_individual_of(O, 'BottleCapInLeftHand').
O = 'Cone_7c7S' .

?- sub_component(Super, pr2_l_gripper_palm_link),
   owl_individual_of(Super, 'LeftArm').
Super = pr2_left_arm
```

For each selected object feature, the system reads the properties needed for configuring the controller, namely their types, positions and directions with respect to a given coordinate frame:

```
?- feature_properties('Cone_7c7S', Type, Label,
                      TfFrame, Position, Direction).
Type = 'LineFeature',
TfFrame = '/pancake_bottle',
Position = [-9.733e-7,1.062e-6,0.457],
Direction = [-9.538e-11,4.656e-11,-0.008] ;
```

Since the motions are defined in terms of generic object parts, they naturally adapt to objects of different types. We demonstrate this by performing the pouring task using two different bottles. The task is defined using the center, the main axis and the cap of the bottle whose positions are determined by geometric object models. Figure 4 shows the segmented models for a bottle of pancake mix (left) and soda bottle (right). The motions performed with the pancake bottle are shown in Figure 1, the motions for the soda bottle in the right part of Figure 4.

7 Discussion & Conclusions

In this paper, we present a synergistic combination of a motion control framework and a robot knowledge base. From a top-down point of view, the methods add the capability for describing motions to abstract plan languages that so far have been limited to the description of atomic action blocks. From a bottom-up perspective, we gain flexibility in how motions can be described by exploiting the capabilities of a formal, logical knowledge representation language. This allows us to (a) extract re-usable patterns of motion descriptions and build up a "motion library"; (b) parameterize motion descriptions with models of objects and of the robot's kinematic structure; (c) make task descriptions more concise because common parts can be inherited from background knowledge in the motion library; and to (d) maintain full flexibility since task descriptions can also locally describe motion constraints in addition to or instead of inheriting them.

We present a hybrid solution for defining the relation between motions and object parts. It uses OWL class restrictions to represent task-dependent information (which tool) and motion-dependent information (which object part) separately and combines these aspects for a specific situation using Prolog rules. This is a very flexible approach since the rules can integrate on-demand computation to identify the object parts that best comply with a class description. For describing a new motion, the OWL description has to be extended, and possibly Prolog rules need to be added. Currently, these rules need to be defined manually. How many of them are needed in total depends on the application context, but given that many of the described concepts are very generic (main axis of an object, handle, top surface, ...), we expect that a reasonably small set of rules will be able to cover a wide range of movements.

The work presented in this paper is only the first step towards semantically rich motion description languages and still has some limitations that we plan to address in future work: For example, the constraint definitions still contain numbers that describe the allowed ranges, e.g. the maximum angle deviation, or the minimum and maximum height. These numbers limit the generalizability of task descriptions to rather similar cases – a description for pouring medicine from one test tube into another will not be usable for pouring water from a bucket into a bathtub. We plan to explicitly represent motion parameters that can be adjusted, such as the height from which something is to be poured, and distinguish them from those constraint values that need to remain fix because they define a motion, such as the alignment of the axes of a screw and a nut for a screwing motion. Based on such a representation, it would be easier to implement automated ways for determining these values, for example to scale them with the dimensions of the involved objects.

While the proposed representations are currently manually created, we expect that parts of them can be learned while still being similarly understandable. In fact, we hope that this explicit representation will help to learn good models: The constraints describe the essence of a task, and their parameter ranges represent the "screws" that can be tuned. We believe that learning in this parameter space will be more effective than learning models of the resulting motions because the constraints indicate which parts are relevant.

Acknowledgments

This work is supported in part by the EU FP7 Projects *RoboHow* (grant number 288533) and *SAPHARI* (grant number 287513).

REFERENCES

[1] G. Bartels, I. Kresse, and M. Beetz, 'Constraint-based movement representation grounded in geometric features', in *Proceedings of the IEEE-RAS International Conference on Humanoid Robots*, (2013).

[2] M. Beetz, L. Mösenlechner, and M. Tenorth, 'CRAM – A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments', in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1012–1017, (2010).

[3] J. De Schutter, T. De Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aertbeliën, K. Claes, and H. Bruyninckx, 'Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty', *Int. J. Rob. Res.*, **26**(5), 433–455, (2007).

[4] K. Erol, J. Hendler, and D.S. Nau, 'HTN planning: Complexity and expressivity', in *Proceedings of the National Conference on Artificial Intelligence*, pp. 1123–1123. John Wiley & Sons LTD, (1994).

[5] R. E. Fikes and N. J. Nilsson, 'STRIPS: A new approach to the application of theorem proving to problem solving', *Artificial intelligence*, **2**(3), 189–208, (1972).

[6] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins, 'PDDL–the planning domain definition language', *AIPS-98 planning committee*, (1998).

[7] A. Hertle, C. Dornhege, T. Keller, and B. Nebel, 'Planning with Semantic Attachments: An Object-Oriented View', in *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pp. 402–407, (2012).

[8] L. P. Kaelbling and T. Lozano-Pérez, 'Hierarchical task and motion planning in the now', in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1470–1477, (2011).

[9] I. Kresse and M. Beetz, 'Movement-aware action control – integrating symbolic and control-theoretic action execution', in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3245–3251, (2012).

[10] L. Kunze, T. Roehm, and M. Beetz, 'Towards semantic robot description languages', in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5589–5595, (2011).

[11] D. Leidner, C. Borst, and G. Hirzinger, 'Things are made for what they are: Solving manipulation tasks by using functional object classes', in *IEEE/RAS International Conference on Humanoid Robots*, pp. 429–435, (2012).

[12] L. Morgenstern, 'Mid-Sized Axiomatizations of Commonsense Problems: A Case Study in Egg Cracking', *Studia Logica*, **67**(3), 333–384, (2001).

[13] C. Samson, M. Le Borgne, and B. Espiau, *Robot Control, the Task Function Approach*, Clarendon Press, Oxford, England, 1991.

[14] M. Tenorth and M. Beetz, 'KnowRob – A Knowledge Processing Infrastructure for Cognition-enabled Robots', *International Journal of Robotics Research (IJRR)*, **32**(5), 566 – 590, (2013).

[15] M. Tenorth, A.C. Perzylo, R. Lafrenz, and M. Beetz, 'Representation and Exchange of Knowledge about Actions, Objects, and Environments in the RoboEarth Framework', *IEEE Transactions on Automation Science and Engineering (T-ASE)*, **10**(3), 643–651, (2013).

[16] M. Tenorth, S. Profanter, F. Balint-Benczedi, and M. Beetz, 'Decomposing CAD Models of Objects of Daily Use and Reasoning about their Functional Parts', in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5943–5949, (2013).

[17] M. Thielscher, 'A unifying action calculus', *Artificial Intelligence Journal*, **175**(1), 120–141, (2011).

[18] U. Thomas, G. Hirzinger, B. Rumpe, C. Schulze, and A. Wortmann, 'A New Skill Based Robot Programming Language Using UML/P Statecharts', in *IEEE International Conference on Robotics and Automation (ICRA)*, (2013).

[19] D. Vanthienen, M. Klotzbücher, J. De Schutter, T. De Laet, and H. Bruyninckx, 'Rapid application development of constrained-based task modelling and execution using domain specific languages', in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1860–1866, (2013).

[20] W3C, *OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax*, World Wide Web Consortium, 2009. http://www.w3.org/TR/2009/REC-owl2-syntax-20091027.

[21] J. Wolfe, B. Marthi, and S. Russell, 'Combined task and motion planning for mobile manipulation', in *International Conference on Automated Planning and Scheduling*, pp. 254–258, (2010).