Article

The International Journal of Robotics Research
1–25

DOI: 10.1177/0278364919850295
journals.sagepub.com/home/ijr

Library of actions: Implementing a generic robot execution framework by using manipulation action semantics

Mohamad Javad Aein[1], Eren Erdal Aksoy[2] and Florentin Wörgötter[1]

Abstract

When a robot has to imitate an observed action sequence, it must first understand the inherent characteristic features of the individual actions. Such features need to reflect the semantics of the action with a high degree of invariance between different demonstrations of the same action. At the same time the machine needs to be able to execute the action sequence in any appropriate situation. In this study, we introduce a new library of actions, which is a generic framework for executing manipulation actions on robotic systems by combining features that capture action semantics with a framework for execution. We focus on manipulation actions and first create a generic representation consisting of symbolic and subsymbolic components. To link these two domains we introduce a finite state machine allowing for sequential execution with error handling. The framework is developed from observing humans which provides us with a high degree of grounding. To quantitatively evaluate the scalability of the proposed approach, we conducted a large set of experiments involving different actions performed either individually or sequentially with various types of objects in different scene contexts.

Keywords

Library of actions, execution, manipulation action, semantic event chain

1. Introduction

Contemporary research in robotics aims at developing intelligent robotic systems with human-like skills. To perform an action with a robot, very often the action is parameterized and represented by a robotic-compatible encoding that allows execution. Various ways exist for doing this, for example one can employ methods for the imitation of human-demonstrated actions by using low-level continuous sensory-motor data streams (programming by demonstration (PbD); see, e.g., Inamura et al (2005), Aleotti and Caselli (2006), and Dillmann et al (2010)). All these methods have in common that they lead to parametric and, thus, executable action representations. They directly rely on "signals": for example, PbD needs perception signals from the observed human action and outputs action signals to reproduce it with the machine. The signal level, thus, allows action execution but, owing to its high degree of detail, easily suffers from deficiencies by not being able to generalize action concepts in a meaningful (semantic) way.

This, however, is needed as soon as the robot has to perform a complex task in different situations. For this it requires some conceptual understanding of the required action sequence and it needs to comprehend the general constraints of the individual sub-actions. Several frameworks exist that attempt action generalization and/or action conceptualization. In a nutshell, they range from generalization at the signal (trajectory) level all the way up to generalization of actions by symbolic (planning-compatible) descriptors (for more details see Section 2).

In this paper, we focus on manipulation actions because they allow for a rather rigorous ontological structuring, the germs of which had been discussed in an older paper (Wörgötter et al, 2013). We extend this approach by developing a library of manipulation actions that captures the essence of each action in an abstract way but remains compatible with robotic execution. To this end we use, as previously (Aein et al, 2013; Aksoy et al, 2011, 2015a), the

[1]Department for Computational Neuroscience, Bernstein Center Göttingen (Institute of Physics 3) and Leibniz Science Campus for Primate Cognition, Georg-August-Universität Göttingen, Germany
[2]Halmstad University (HH) School of Information Technology (ITE), Center for Applied Intelligent Systems Research (CAISR), Halmstad, Sweden

Corresponding author:
Florentin Wörgötter, Bernstein Center for Computational Neuroscience (BCCN),Third Physics Institute, Georg-August-Universität Göttingen, Friedrich-Hund-Platz 1, D-37077 Göttingen, Germany.
Email: worgott@gwdg.de

framework of semantic event chains (SECs) to encode the action *type*. SECs just analyze the sequence of touching and untouching events that happen during an action to do this, but they can, in this way, break the realm of manipulation actions only down into a few semantically similar classes (Pastra and Aloimonos, 2012; Wörgötter et al, 2013). The here-pursued approach enriches this by descriptive movement primitives that allow for two things. On the one hand, many more (possibly all single handed) manipulation actions are now represented by a unique set of symbolic descriptors, which on the other hand remain execution-relevant, because they can at run-time be filled with the required parameters for performing the different movements to execute the action. Therefore, this approach represents one possible way for linking a symbolic action representation in a grounded way with its corresponding signal-level description (derived from observations, hence from sensory experience). All in all, with this framework we hope to achieve a contribution towards closing, or at least reducing, the *signal-to-symbol gap* (Coradeschi and Saffiotti, 2003; Krüger et al, 2011) in robotics.

Thus, based on prior works from us and others, the main contribution of this paper is the rigorous structuring of a large set of manipulation actions into a three-layer representation starting from a high, symbolic level via a state machine-like encoding and ending at detailed movement primitives. We show that these representations can then act as library functions and that they can be parameterized in a situation-dependent way to execute them either alone or in a sequence.

The rest of the paper is organized as follows. We start with introducing the state of the art in Section 2. We then continue with a detailed description of action definition and execution in Section 3. The results of many experiments using this framework are finally shown in Section 4 followed by a discussion in Section 5.

2. State of the art

There exists a large corpus of work on action representation and execution (Calinon et al, 2007; Ijspeert et al, 2002; Lee and Nakamura, 2006; Simmons and Apfelbaum, 1998; Ude, 1993). Two distinct approaches are commonly preferred in order to represent and execute actions; one at the trajectory level (Ijspeert et al, 2002), the other at the symbolic level (Simmons and Apfelbaum, 1998). The former gives more flexibility for an execution-relevant definition of actions, while the latter defines actions at a higher level and allows for generalization and planning.

For trajectory-level representation there are several well-established techniques: splines (Ude, 1993), hidden Markov models (HMMs) (Lee and Nakamura, 2006), Gaussian mixture models (GMMs) (Calinon et al, 2007), dynamic movement primitives (DMPs) (Ijspeert et al, 2002; Kulvicius et al, 2012; Luksch et al, 2012). With trajectory-level encoding, one can investigate or learn different complicated trajectories, but it is difficult to use them in a more "*cognitive sense*." Generalization of the observed trajectories is the main challenge here (and often addressed in different ways in the above-cited papers), because even the same action can be demonstrated by following various trajectories.

High-level symbolic representations many times use graph structures and relational representations (e.g. Ekvall and Kragic, 2006; Pardowitz et al, 2007). Alternative methods, such as that of Lee et al (2013), described a syntactic approach for learning robot imitation by capturing underlying task structures in the form of probabilistic activity grammars. These approaches give compact descriptions of complex tasks, but they do not consider execution-relevant motion parameters (trajectories, poses, forces) in great detail.

In this work, our high-level action descriptor is based on the concept of SECs introduced by Aksoy et al (2011) and used also by others (Luo et al, 2011; Martinez et al, 2014; Vuga et al, 2014; Yang et al, 2013). SECs are generic action descriptors that capture the underlying spatio-temporal structure of continuous actions by sampling only decisive key temporal points derived from the spatial interactions between hands and objects in the scene. The SEC representation is invariant to large variations in trajectory, velocity, object type, and pose used in the action. Therefore, SECs can be employed for the classification task of actions as demonstrated in various experiments in Aksoy et al (2015b) and we have shown in Aein et al (2013) that human-demonstrated actions encoded by SECs can also be executed by robots, once low-level data (object positions, trajectories, etc.) are provided.

Many times trajectory-level descriptions of actions, object properties, and high-level goals of the manipulation were brought together through STRIPS-like planning (Beetz et al, 2015; Dillmann et al, 2010; Kunze et al, 2011), resulting in operational although not very transparent systems. The approaches in Ahmadzadeh and Kormushev (2016); Ahmadzadeh et al (2015) attempted to integrate symbolic action representation and planner with a motor skill learner. The robot learned the goal of the human-demonstrated actions by using a so-called visuo-spatial skill learning (VSL) method, which produced symbolic predicates. Such predicates were directly fed into a standard planner to encode skills in a discrete symbolic form. This framework also considered sensorimotor skills, such as the followed trajectory from the observed action. In contrast to the works in Ahmadzadeh and Kormushev (2016); Ahmadzadeh et al (2015), we do not immediately require any additional symbolic planner because SECs provide a fully observable state sequence. As long as we are only dealing with straightforward linear action sequences, planning is no longer needed. To show this here, we also perform evaluations on long and complex human manipulation actions.

Still the problem of how to bring the signal (trajectory) level together with the symbolic level remains a big challenge in robotics.

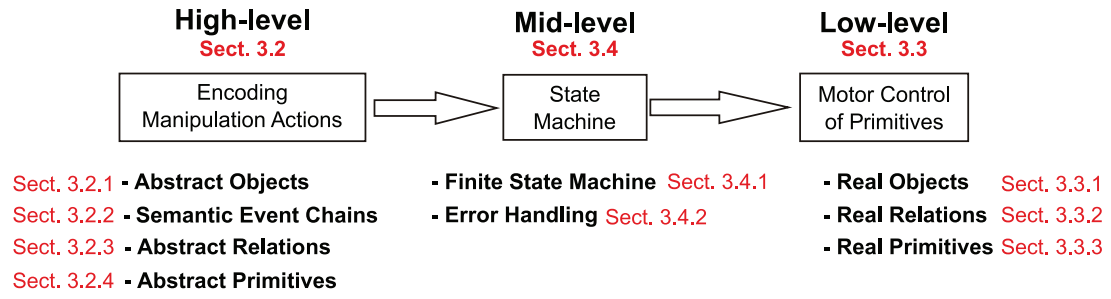

Fig. 1. Levels of action definition. The high-level components are symbolic and close to human language. The low-level components are in the signal domain. The mid-level fills the gap and makes execution possible. The red numbers refer to the sections in the text.

There are also many works concentrating on the execution of manipulation actions using cognitive agents. Yamaguchi et al (2014) designed a finite state machine (FSM) to execute a pouring action. Morante et al (2014) used guided motor primitives (GMPs) to perform painting and cleaning tasks on a simulated robot. Another approach to deal with the signal–symbol gap was to combine motion and task planning such as in the work of Srivastava et al (2014). They generated trajectories for tasks such as pick-up and put-down to solve problems in different domains. In Ghalamzan et al (2015), GMM and DMPs were integrated to learn robotic tasks from human demonstrations. He et al (2015) proposed a manipulation planning framework with linear temporal logic specifications. The system was demonstrated on a simulated robot to successfully perform some tasks, but the planning could take a long time as soon as the number of objects and locations in the environment increases. Kappler et al (2015) proposed a decision-making approach to perform robotic tasks. Here, multi-modal sensor data were processed to switch between several movement primitives called associative skill memories.

Morante et al (2014) proposed execution of actions using so-called continuous goal directed actions (CGDAs). They generated a library of GMPs in the joint space of the robot, and later used them in the execution phase. A simulated robot was introduced to perform cleaning and painting tasks. Lioutikov et al (2016) performed a bimanual cutting task by sequencing the learned DMPs. As only position data were used, the quality of execution is highly dependent on the placement of the knife in the robot hand.

In Rozo et al (2013) a pouring task was learned by using parametric HMMs. In addition, in Yamaguchi et al (2015) a pouring task was represented, planned and learned from human demonstration. In this work, pouring of liquids and granular material was modeled by a FM.

Although the existing works show promising results, they are usually limited in the number of actions and manipulated objects. One of the main goals of our study is to develop a generic scheme that allows robots to perform a much wider variety of actions on various object sets.

3. Methods

As illustrated in Figure 1, our proposed perception–action framework involves three main levels: *high-, mid-*, and *low-level* action units. To address this we will start with a detailed description of high and low levels together with their components. In the very end, the mid-level action unit that bridges the gap between high and low levels will be introduced. The figure provides in red the section numbers by which the road-map of this section is represented.

First, however, we provide a short overview of the domain in which we operate and also discuss which actions we have implemented. Then we describe the framework using the structure from Figure 1.

3.1. Domain and actions

In our experiments, we are focusing on tabletop manipulations related to cooking tasks, which can be performed with a stationary robot. Note, however, that by design the framework is not restricted to this domain, because the structuring of all actions in high-, mid- and low-level action-units allows transferring the same actions also to (for example) a workshop or other tabletop manipulation action domains.

We have analyzed and structured our library of actions for all 32 manipulation action types described in Wörgötter et al (2013), 10 of those we are investigating in depth performing also robotic experiments with them.

Note that action examples are kept simple to be able to show clearly the belonging trajectories, force, and tactile patterns (see the figures at the end of the Section 4). The same framework, however, had been used to analyze long and complex real-world manipulation actions (Aksoy et al, 2017). Hence, this framework can address much higher levels of scene complexity than shown here.

3.2. High-level action definition

In this section, we give a high-level action definition to extract and encode the semantics of manipulations. Note that, at this level, definitions are mainly symbolic (abstract) and close to human descriptions.

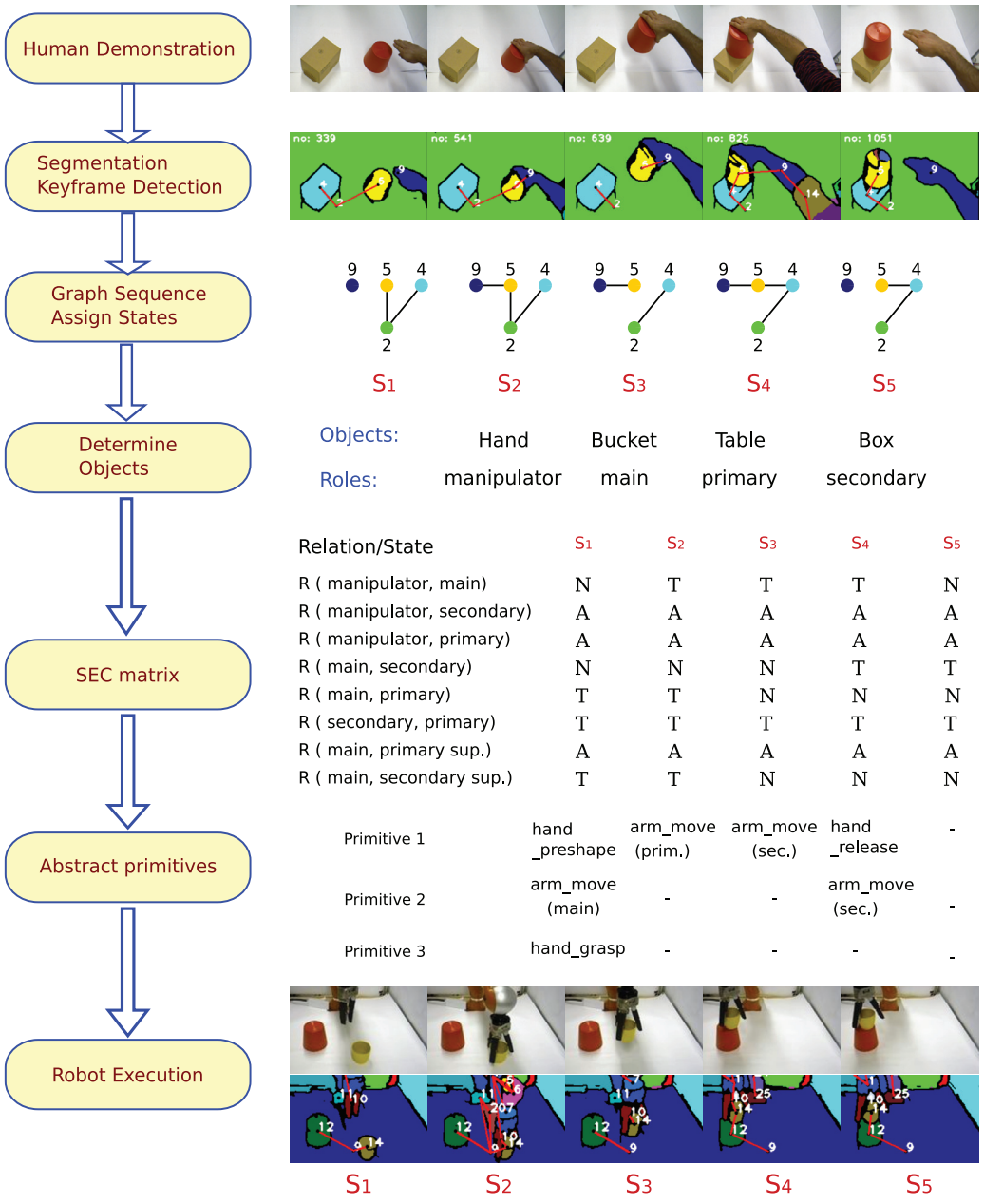

Fig. 2. A sample human demonstration and robot execution of a *put-on-top* action are shown to highlight different action components. At the top, snapshots and segmented images of the human demonstration are shown. Next, a relational graph sequence is computed. Each graph corresponds to one world state (S_1 to S_5). The objects in the scene are recognized and their roles in the action are determined. Abstract spatial relations and their values at each state are shown in the SEC matrix. Here, each row represents a pairwise object relation such as *N* and *T* that denote *Not touching* and *Touching*, respectively. Action primitives at each state are shown at the bottom of the SEC matrix. Finally, some snapshots and segmented images of the robot execution with different objects are shown.

Take the example of a manipulation action "*put a bucket on a box.*"Figure 2 shows some sample frames from human demonstration. This simple action may be described by a human as follows:

1. ***Approach*** the bucket;
2. ***Grasp*** the bucket;
3. ***Lift*** the bucket from table;
4. ***Place*** the bucket on the box;
5. ***Release*** the bucket.

This description is by no means unique. One could easily describe the same action in different words, with different number of steps and details. However, one could still extract some common and descriptive properties from such a naive description.

- **Property 1:** The definition is still valid even if the manipulated objects are (within reason) altered.
- **Property 2:** The action (here "put-on-top") can be broken into a sequence of smaller sub-actions (primitives) such as *Approach* and *Grasp*.
- **Property 3:** There are conditions to end one primitive and start with the next. In the above example these conditions are not spelled out explicitly.
- **Property 4:** As humans, we intuitively know how to perform these primitives, although our exact movements are only then produced when we see the objects and are adapted to the scene context while we perform the action.

The main features that we use here to describe a scene are the touching relations between its objects. During a manipulation action, these touching relations change from some initial state to a final state. A manipulation action is, therefore, represented by a sequence of changes in touching relations of the objects.

Our approach to represent and execute manipulation actions with robots has the following fundamental properties. We introduce a generic high-level definition of actions which is independent of the manipulated objects in the action (*Property 1*), and consists of a sequence of symbolic primitives (*Property 2*). The conditions to start and end each primitive are defined by considering the touching relation between objects in the action (*Property 3*). We also store the default action descriptive parameters (e.g. trajectory) to execute actions at the high-level with symbolic definitions. When novel physical objects are observed at each specific instance of an action, these parameters are adapted according to the situation to generate the required movements (*Property 4*).

To fully satisfy these four properties in our high-level action definition, we benefit from the *ontology of manipulation actions* introduced in Wörgötter et al (2013). This ontology structures human-demonstrated manipulation actions, e.g. *putting a bucket on a box*, as sequences of spatio-temporal interactions between objects (including the manipulator) in the scene by using the concept of SECs presented in Aksoy et al (2011). This ontology suggests about 30 fundamental and unique manipulations that allow complex and chained activities, e.g. "*making a salad*" or "*preparing breakfast*."

The ontology also introduces four constraints on the definition of manipulation actions, which are stated as follows.

- **Constraint 1:** The action is performed by one hand. This is true for most human actions, because the second hand is usually used only as a support.
- **Constraint 2:** The hand touches exactly one object in the course of the action and does not purposefully touch other objects in the scene unless the current action ends.
- **Constraint 3:** The hand is free at the beginning and at the end of the action.
- **Constraint 4:** The action must lead to some changes in the touching relations between objects and hands (e.g. human or robot hand). In other words, the hand must interact with at least one object.

These constraints had been discussed in great detail in Wörgötter et al (2013) but we would like to add some important notes here, too. (1) Considering one-handed actions is to some degree a simplification, because a *supporting hand* can also play an active role in a manipulation (for example, in creating counter-forces, etc., see also Section 5). (2) In case multiple objects need to be manipulated at the same time (e.g. pushing clutter away) this framework needs to be extended by a system that can reason about the semantics of single versus multi-objects. Such problems are related to the perceptual binding problem and cannot be solved without additional mechanisms. (3) Constraint 3 is very important to allow for a rigorous cut between each two manipulations. Actions that involve tools can be understood as a broken-up (interrupted) action chain without violating Constraint 3. Constraint 4 is evident without further comment.

From the first two constraints one concludes that in each action there are at least two entities: one hand and one object that is directly touched by the hand. This fact will be used in Section 3.2.1 to define object roles. The second and third constraints together define actions in a way that they cannot be further split into shorter actions. The last constraint assures that there is at least one change in the touching relations. This is essential because the whole framework relies on the touching relations between objects.

In the rest of this section, we describe several components of the high-level action definition that are required to reach these descriptive properties. We refer the interested reader to Wörgötter et al (2013) for details of the manipulation action ontology.

3.2.1. Object roles. There exist many objects in the real world and actions can be performed with different sets of object combinations. It is, however, not practical to define a separate action for each possible object set. Instead, as stated in *Property 1*, we represent manipulation actions in a generic way to make them applicable to any novel object. For this, we label objects by their roles exhibited in the action. First, recalling *Constraint 1*, we need an actor to perform the action, which is here called *manipulator.* As stated in *Constraint 2*, there exists exactly one object that is directly manipulated by the manipulator. This object is called *main*. Optionally, there are other objects in the action, which interact with the *main* object in different ways.

The object roles can be better explained in an example. In the action *"putting a bucket on a box"* depicted in Figure 2, the human hand is the *manipulator*, the bucket which is directly touched by the hand is the *main* object. There are two more objects whose relations with *main*

Table 1. Object roles defined based on spatial relations. Each role is defined and the constraints on the relations are presented. Note that main object is defined with regard to the manipulator, unless the action is performed using a tool. In this case, the main object is defined with regard to the tool.

Object role	Description	Relation constraints
Manipulator	The object that performs the action	Not touching anything at the beginning and the end of action. During the action, it touches at least one object.
Main	The object that is directly in contact with the manipulator (tool)	Not touching the manipulator (tool) at the beginning and the end of action. It touches the manipulator (tool) at least once.
Primary	The object from which the main object separates	Initially touches the main and makes a T to N transition.
Secondary	The object to which the main object joins	Initially does not touch the main and makes an N to T transition.
Load	The object that is indirectly manipulated	Does not touch the manipulator. During the action leaves the main and touches the container or vice versa.
Container	The object whose relation with load changes and it is not the main object	Touches or untouches the load object.
Main support	The object on which the main object is located	Touching the main object all the time.
Primary support	The object on which the primary object is located	Touching the primary object all the time
Secondary support	The object on which the secondary object is located	Touching the secondary object all the time.
Container support	The object on which the container is located	Touching the container all the time.
Tool	The object which is used by the manipulator to enhance the quality of some actions	Grasped by the manipulator at the beginning of action and released at the end.

change in the action: table and box. The relation of *main* and the table changes from touching T to not touching N. We call such objects *primary* or *source* object. Conversely, the relation of *main* and the box changes from not touching N to touching T. These objects are called *secondary* or *destination* object.

The complete list of object roles with their definitions are shown in Table 1. Some roles are defined by the changes in relations, such as *primary* and *secondary*, whereas others (such as support objects) are defined based on constant touching relations. For instance, *secondary support* is the object on which the *secondary* object is located. In the above example, the table also plays the role of *secondary support*. Note that not always all relations are needed to define an action.

The role of objects are automatically detected with the method described by Aksoy et al (2015b), which explores the temporal evolution of spatial object relations embedded in SECs.

3.2.2. SECs. At the highest symbolic level, actions are represented by the concept of SECs, which captures the essence of an action by employing computer vision techniques as described by Papon et al (2012) and Aksoy et al (2011). A summary of this process is shown in Figure 2 along with the *put-on-top* example. To calculate the SEC representation, an image sequence of an observed action is first represented by 3D image segments, each of which corresponds to one object in the scene and is consistently tracked during the action. Each frame in the sequence is then converted into a graph: nodes represent tracked segments, i.e. objects, and edges indicate the contact relation between a pair of objects. By employing an exact graph matching method, the continuous graph sequence is discretized into decisive main graphs, i.e. "states," each of which represents a topological change in the scene. The extracted main graphs form the core skeleton of the SEC, which is a matrix where rows are the spatial relations between object pairs in the scene. Each column of the SEC matrix is interpreted as a state of the scene, which is the combination of object relations when a new main graph occurs.

Possible spatial relations in the SEC matrix are *Not touching* (N), *Touching* (T), and *Absence* (A), where N corresponds to two spatially separated objects, T represents objects that touch each other. The value A occurs when there exists no information about the relation, e.g. one object is not visible in the scene.

Note that the SEC matrix will not unduly grow when there are many objects in the scene. This is due to the fact that only the (abstract) objects in Table 1 are considered for any possible action. Hence, relations between objects that do not partake in an action do not create additional rows in the SEC.

Thus, in a SEC, the progress of the action from the beginning to the end is stored in a compact way. In addition, the SEC matrix is invariant to large variations in trajectory, velocity, object type, and pose used in the action and, therefore, remains the same for different instances of the same action.

Figure 2 shows a *put-on-top* action from human demonstration to robot execution. The snapshots of the demonstration are shown together with the tracked segments (colored regions) and main graphs. The objects in the scene and the extracted SEC matrix are shown with the corresponding states and primitives. At the bottom, the snapshots and tracked segments of the robot execution are depicted.

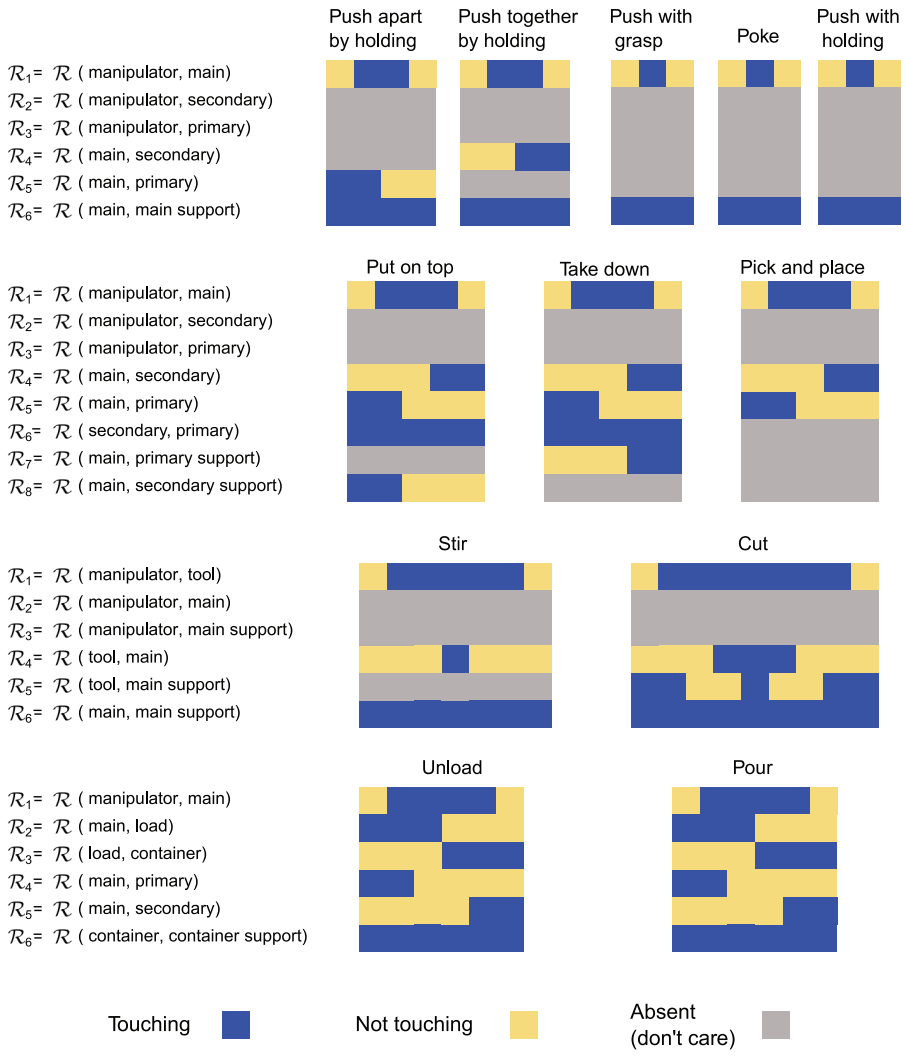

Fig. 3. Extracted SEC matrices for 10 single atomic manipulation actions. The abstract spatial relation associated to each SEC row is color coded in which blue and yellow represent *Touching* (*T*) and *Not touching* (*N*), respectively. The gray color shows either an *Absent* (*A*) or a *Don't care* relation. Note that the SEC matrix of three actions *push by grasp, poke*, and *push by holding* are the same, whereas their action primitives and parameters are different. In the action *Put on top* the *primary* object is the same as the *secondary support*, which makes the relations R_5 and R_8 identical. Similarly, relations R_4 and R_7 in the action *Take down* are identical, because the *secondary* object is the same as *primary support*.

Figure 3 depicts the event chain patterns of different actions in the library as color-coded images. These SEC patterns are stored as high-level action descriptors in the action library. Although SEC patterns are very distinctive, some are semantically identical as in *Push by grasp, Poke*, and *Push by holding* actions. This semantic similarity is natural since those actions have the same changes in the touching relation of objects. However, they have different primitives with different object poses, trajectories, and force parameters that are not captured by SECs.

This action descriptive object, trajectory, and force information is separately stored as primitives (see Sections 3.2.4 and 3.3.3).

3.2.3. Abstract relations. We continue with computing the spatial relations between each abstract object pair, e.g. between the *manipulator* and the *main* object. Table 2 shows the abstract relations for the action *"putting a bucket on a box"*, previously shown in Figure 2.

Each relation is defined by two attributes, namely *type* and *value*. The *type* of a relation is determined by the importance and variation of that relation throughout the action. For example, for the action in Figure 2, the relation between the *manipulator* and the *primary* is always not touching and does not affect the outcome of the action because they are not directly interacting with each other at all. The type of such relations is *don't care*.

Other relations, which are crucial for an action, are categorized as *variable* and *constant* relations. For example, the relation between the *manipulator* (i.e. hand) and the *main* object (i.e. bucket) in Figure 2 is *variable* because it naturally alters during the action. The variable relations encode the dynamics

Table 2. Abstract relations and their attributes for the action "*putting a bucket on a box*" shown in Figure 2.

Relation name	Abstract relation	Real relation	Type
$\mathcal{R}_1$	$\mathcal{R}$(manipulator,main)	$\mathcal{R}$(hand,bucket)	Variable
$\mathcal{R}_2$	$\mathcal{R}$(manipulator,secondary)	$\mathcal{R}$(hand,box)	Don't care
$\mathcal{R}_3$	$\mathcal{R}$(manipulator,primary)	$\mathcal{R}$(hand,table)	Don't care
$\mathcal{R}_4$	$\mathcal{R}$(main,secondary)	$\mathcal{R}$(bucket,box)	Variable
$\mathcal{R}_5$	$\mathcal{R}$(main,primary)	$\mathcal{R}$(bucket,table)	Variable
$\mathcal{R}_6$	$\mathcal{R}$(secondary,primary)	$\mathcal{R}$(box,table)	Constant
$\mathcal{R}_7$	$\mathcal{R}$(main,primary support)	—	Absent
$\mathcal{R}_8$	$\mathcal{R}$(main,secondary support)	$\mathcal{R}$(bucket,table)	Variable

of the action. On the other hand, the relation between the *secondary* object (i.e. box) and the *primary* (i.e. table) remains constantly touching, and hence is *constant*. We note that such constant relations highlight the necessary pre-conditions to perform an action and any unexpected change in these constant relations implies a failure of the action.

3.2.4. Abstract primitives. As stated in *Property 2* in Section 3.2, an action can be divided into several sub-actions or primitives. In our approach we define the following abstract primitives.

- *arm_move*(*object*): The robot arm moves to a pose relative to *object*.
- *arm_move_periodic*(): The robot arm moves periodically.
- *arm_exert*(): The robot arm exerts a force.
- *hand_preshape*(): The robot hand moves to a certain pre-shape.
- *hand_grasp*(): The robot hand performs a grasp.
- *hand_release*(): The robot hand releases the already grasped object.

These abstract primitives correspond to the basic functions of the robot manipulator, which can be implemented in many different ways. Our way of implementing such primitives at the lowest motor control level are presented in Section 3.3.3. The focus of our work is, however, not a specific implementation, but rather we would like to propose a way to combine them to seamlessly perform actions. In our approach a state transition in the SEC, i.e. a change from one column to the next, needs at least one of these unique primitives. All manipulations that we have analyzed have a strictly linear sequence of primitives between two subsequent SEC columns. Thus, an action is performed when all of its primitives are sequentially executed while the relations change according to the SEC matrix.

In Figure 2, the necessary primitives associated with each column of the SEC matrix of the *put-on-top* action are shown. The reason of having multiple primitives is that sometimes more than one primitive is required to induce the desired change in the spatial relation. For example, the combination of *arm_move*(*main*) and *hand_grasp*() primitives is necessary to change the relation of *manipulator* and *main* from *N* to *T*.

In general, which primitives to choose is determined by the column-to-column transition in a given SEC. Owing to the fact that we had in total analyzed 32 manipulation action types, which on average contain five SEC columns each, we were faced with only about 150 column-to-column transitions in total. It was, thus, possible to analyze all of those "by hand" and manually define the required primitives for every transition.

3.3. Low-level action definition

In this section, the abstract components of the high-level definition are related to their real-world counterparts at the signal level. This includes defining objects in the real world, calculating their spatial relations from the sensor data, and implementing low-level primitives such that proper commands are sent to the robot arm and hand control systems. In the rest of this section, these elements are described in more detail.

3.3.1. Real objects. In real-world experiments, abstract objects (i.e. *manipulator, main, primary*, etc.) are instantiated by real objects in the scene. For the "*putting a bucket on a box*" example depicted in Figure 2, these objects are hand (*manipulator*), bucket (*main*), table (*primary*), and box (*secondary*). We need to identify the real-world objects in the signal space in order to perform the low-level primitives.

For this task, we use our modular computer vision architecture described in Papon et al (2012), which segments each object in the scene by employing the color and depth cues fed from the RGB-D sensor. We further apply the instance-based object recognition method from Schoeler et al (2014) to identify extracted image segments. By incorporating the depth information, we also detect the background segment (supporting surface, i.e. Table), which is in the form of a planar surface.

Once real objects in the scene are detected, we compute each object pose in signal space. In our work, two pieces of information are required to represent an identified object: position and orientation. The position of each object

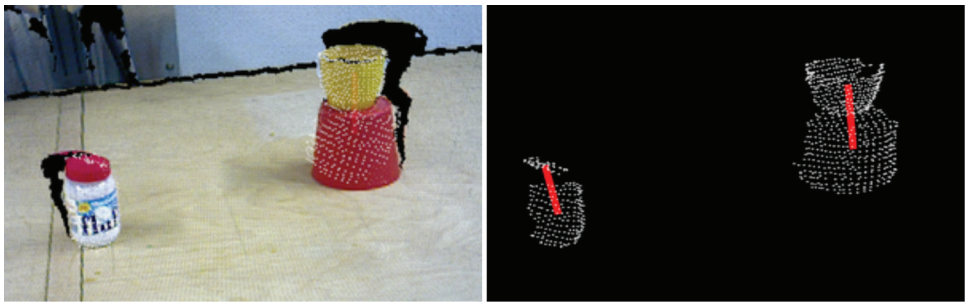

Fig. 4. Calculating the contact relation with our visual perception interface. The small red lid is touching the jar and the yellow cup is on top of the bucket (left). The red lines indicate the existing touching relations between objects (right).

is computed in Cartesian space. To associate a position to an object, we model the object with a single point located at the center of mass. The orientation of objects is defined as the angle that the main axis of the object makes with respect to the X-axis of the reference frame. Note that we extract the orientation information only for elongated objects (e.g. cucumber) but not for symmetric objects (e.g. apple). The abstract pose of the respective object is finally estimated from its major axis derived by principle component analysis (PCA). The orientation information is used to find the parameters of the primitives for elongated objects.

Note that the position of the manipulator, i.e. robot end effector, is directly calculated from position sensors and the kinematics of the arm.

3.3.2. Real relations. The *real relations* are the values of relations between pairs of objects in the scene. To detect these values, we use a combination of proprioceptive (e.g. position) and exteroceptive (e.g. tactile, force, and vision) sensors.

When it comes to detecting object relations, there are three phases: before, during, and after the action. In the first and last phases, the only source of information is the vision interface, which essentially computes the Euclidean distance between segmented object point clouds to decide whether they touch each other or not. An example of this detection is shown in Figure 4.

While the action is being performed, the data acquired by other sensors (position, force, and tactile) are used in addition to the vision system. The data collected from these sensors are fused using several *heuristic* rules, which are conjunctions of individual conditions on different sensor data. For example, the first rule to detect the relation of *manipulator* with *main* object is a combination of conditions on two sensors: position and tactile. This rule declares a touching relation when the Euclidean distance between the two objects is less than a threshold (denoted by $\mathcal{D}_1$) and the tactile sensor detects a grasp. These rules are listed in Table 3.

The rules of Table 3 use some intermediate signals that are abstractions of force and tactile sensor data: *contact* and *grasped*. These are flags showing when the robot arm is touching the environment (*contact*) or the robot hand is grasping an object (*grasped*). The *grasped* flag is set to one if the average values of tactile sensors on all three fingers exceed a threshold. An example of tactile sensor readings during an action are shown in Figure 5 (top). The solid red line shows the raised *grasped* flag at the times that the hand is grasping some object.

The *contact* flags raise when the external force applied to the end effector of the robot arm exceeds a threshold. In the rules of Table 3, we have only used the $contact_z$ flag which shows contact along the Cartesian Z axis. Figure 5 (bottom) shows an example of contact detected along the Z axis.

As multiple rules exist to detect the same relation in Table 3, we should assign the rules that need to be considered in each action. This is summarized in Table 4 for the actions in the library.

3.3.3. Real primitives. In Section 3.2.4 we defined the abstract primitives. Here, we re-introduce these primitives by adding their parameters and describe their implementations at the low level.

For the robot arm, we have the following primitives:

- $arm\_move(object, T_{off}, \mathcal{P})$;
- $arm\_move\_periodic(a_x, a_y, a_z, b_x, b_y, b_z, \omega)$;
- $arm\_exert(F_{des})$.

For the robot hand we have defined the following primitives:

- $hand\_preshape(q)$;
- $hand\_grasp()$;
- $hand\_release()$.

Here we explain these primitives in more detail and discuss their specific implementation in our system.

***arm_move*$(object, T_{off}P)$**

This primitive moves the end effector from the current pose to a pose relative to *object*. The offset of the target is stored in the homogeneous transformation T_{off}. Equation (1) shows how the goal of this primitive is calculated from the pose of the object and the offset transformation:

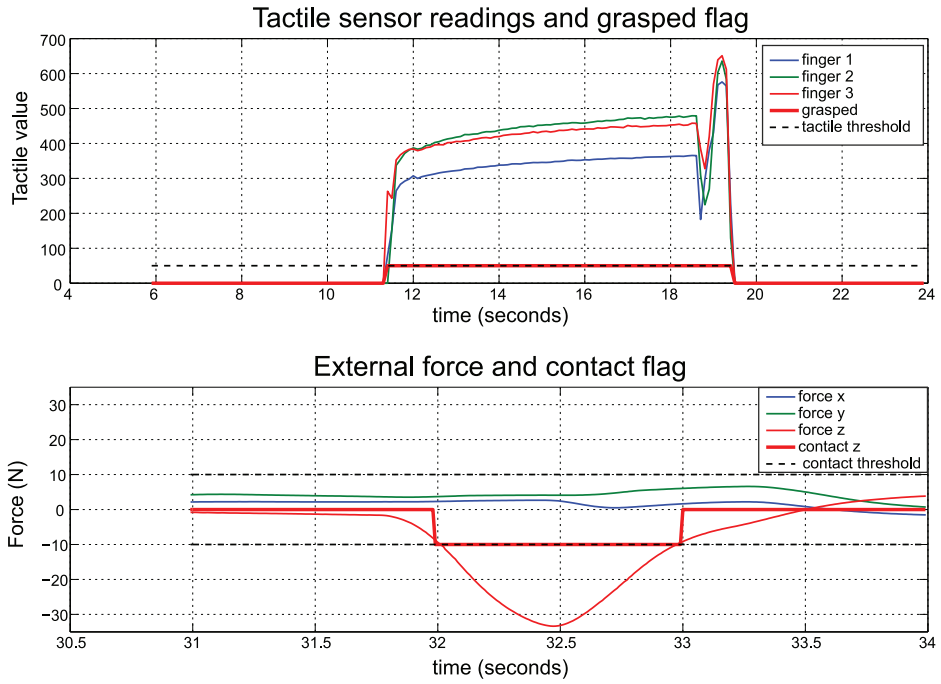

Fig. 5. Intermediate exteroceptive sensory input in the process of analyzing the spatial relation rules given in Table 3. The tactile sensor values are used to detect when an object is grasped by the hand (top). The external force signals are processed to detect the touching event (bottom). Here a contact in the Z direction is detected.

Table 3. Rules for detecting the spatial relational changes during action execution. Note that $\| manip, O_i \|$ represents the Euclidean distance between manipulator end-effector and position of object O_i. Similarly, $\| O_i, O_j \|$ is the distance between objects O_i and O_j. O_i and O_j are any pair of objects in the scene. $contact_z$ means that the sensor sensed a contact in Z axis, while $!contact_z$ means there is no such contact. *grasped* means the hand has grasped some object, while !*grasped* means the hand is empty. The parameters $\mathcal{D}_{ij}$ are the distance thresholds to decide whether two objects are touching or not.

Rule	Relation	Change	Vision	Conditions position	Force	Tactile	$\mathcal{R}(manip, O_i)$
1	$\mathcal{R}(manip, O_i)$	N to T	—	$\| manip, O_i \| < \mathcal{D}_{11}$	—	*grasped*	—
2	$\mathcal{R}(manip, O_i)$	N to T	—	$\| manip, O_i \| < \mathcal{D}_{21}$	$contact_z$	—	—
3	$\mathcal{R}(manip, O_i)$	T to N	—	—	-	!*grasped*	—
4	$\mathcal{R}(manip, O_i)$	T to N	—	—	$!contact_z$	—	—
5	$\mathcal{R}(O_i, O_j)$	N to T	$\| O_i, O_j \| < \mathcal{D}_{51}$	$\| manip, O_j \| < \mathcal{D}_{52}$	$contact_z$	*grasped*	T
6	$\mathcal{R}(O_i, O_j)$	N to T	$\| O_i, O_j \| < \mathcal{D}_{61}$	$\| manip, O_j \| < \mathcal{D}_{62}$	—	*grasped*	T
7	$\mathcal{R}(O_i, O_j)$	T to N	$\| O_i, O_j \| > \mathcal{D}_{71}$	$\| manip, O_j \| > \mathcal{D}_{72}$	—	*grasped*	T
8	$\mathcal{R}(O_i, O_j)$	N to T	$\| O_i, O_j \| < \mathcal{D}_{81}$	—	—	—	—
9	$\mathcal{R}(O_i, O_j)$	T to N	$\| O_i, O_j \| > \mathcal{D}_{91}$	—	—	—	—

$$P_{goal} = T_{off} P_{obj} \tag{1}$$

The parameters of the trajectory are stored in $\mathcal{P}$. We use DMPs introduced by Ijspeert et al (2002) and the joining method proposed by Kulvicius et al (2012) to generate smooth trajectories. To save space, the equations of DMP that generate trajectories are not repeated here. The outputs of DMP are the desired trajectory of robot end-effector in Cartesian space that move from the start pose to the goal pose.

These trajectories are fed as desired values to the low-level control system of the robot arm. In our setup, we have a KUKA LWR robot which has the following control policy to generate commanded joint torques τ_{cmd}:

$$\tau_{cmd} = J^{\mathrm{T}}(k_c(X^* - X)) + D(q) + f_{dyn}(q, \dot{q}, \ddot{q}) \tag{2}$$

where X^* is the desired pose, X is the measured actual pose of the robot. The coefficient k_c denotes the gain of the position control which determined the stiffness of the arm during motion. The terms $D(q)$ and $f_{dyn}(q, \dot{q}, \ddot{q})$ are the friction and dynamics of the robot arm which are used in the control system.

Table 4. Rule consideration for detecting spatial object relations (see Table 3 and Figure 3).

	Relation							
Action	$\mathcal{R}_1$	$\mathcal{R}_2$	$\mathcal{R}_3$	$\mathcal{R}_4$	$\mathcal{R}_5$	$\mathcal{R}_6$	$\mathcal{R}_7$	$\mathcal{R}_8$
Pick and place	Rule 1,3	—	—	Rule 5	Rule 7	—	—	—
Put on top	Rule 1,3	—	—	Rule 5	Rule 7	Rule 8,9	—	Rule 7
Take Down	Rule 1,3	—	—	Rule 5	Rule 7	Rule 8,9	Rule 5	—
Stir	Rule 1,3	—	—	Rule 6	—	Rule 8,9	—	—
Cut	Rule 1,3	—	—	Rule 5	Rule 6	Rule 8,9	—	—
Poke	Rule 2,4	—	—	—	—	Rule 8,9	—	—
Push with grasp	Rule 1,3	—	—	—	—	Rule 8,9	—	—
Push with holding	Rule 2,4	—	—	—	—	Rule 8,9	—	—
Push apart by holding	Rule 2,4	—	—	—	Rule 9	Rule 8,9	—	—
Push together by holding	Rule 2,4	—	—	Rule 8	—	Rule 8,9	—	—
Pour	Rule 1,3	Rule 8,9	Rule 8,9	Rule 7	Rule 5	Rule 8,9	—	—
Unload	Rule 1,3	Rule 8,9	Rule 8,9	Rule 7	Rule 5	Rule 8,9	—	—

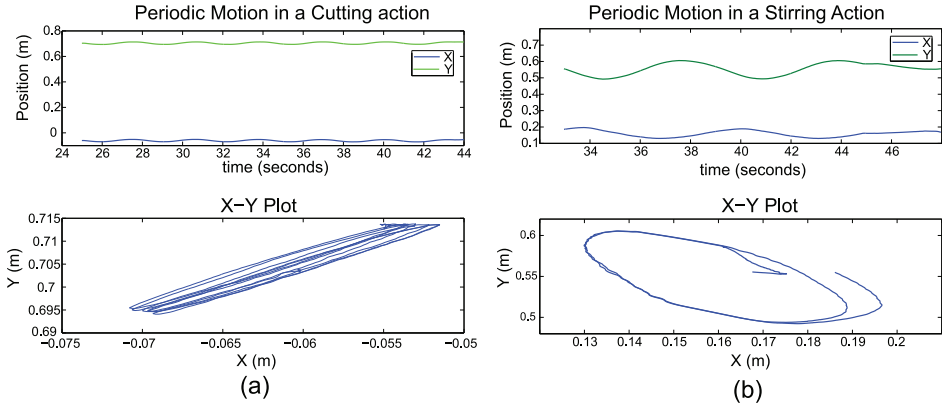

Fig. 6. Examples of periodic trajectories in actions. (a) In the cutting action the following parameters are used to generate a back-and-forth motion: $a_x = -0.008$ m, $a_y = -0.01$ m, and $\omega = 1.8$ rad/s. The initial position is $x(0) = -0.6$ m and $y(0) = 0.7$ m. (b) In the stirring action a circular motion is generated by using the following parameters: $b_x = 0.03$ m, $a_y = -0.05$ m, and $\omega = 1$ rad/s. The initial position is $x(0) = 0.187$ m and $y(0) = 0.55$ m.

***arm_move_periodic*($a_x, a_y, a_z, b_x, b_y, b_z, \omega$)**

For some actions we need to perform some simple periodic motions. There are comprehensive frameworks to create periodic (rhythmic) motions on robots such as rhythmic DMPs. However in our system we only need simple back-and-forth and circular motions, for which a combination of sine and cosine functions suffice. Therefore we implement *arm_move_periodic* primitive using the following equations:

$$x(t) = x(0) + a_x \sin(\omega t) + b_x \cos(\omega t) - b_x \quad (3)$$

$$y(t) = y(0) + a_y \sin(\omega t) + b_y \cos(\omega t) - b_y \quad (4)$$

$$z(t) = z(0) + a_z \sin(\omega t) + b_z \cos(\omega t) - b_z \quad (5)$$

These equations generate smooth trajectories from an initial position (which is $[x(0), y(0), z(0)]$). The a_x and b_x determine the strength of sine and cosine components on the X axis. There period of trajectory is determined by the parameter ω. Examples of periodic motion generated in actions are shown in Figure 6. The back-and-forth motion is used in actions such as cutting, while a circular motion is needed in stirring.

***arm_exert*(F_{des})**

Manipulation actions sometimes need more than just pure position control. In some tasks we need to also regulate the force exerted at the environment. Many times it is important to have both position and force control at the same time. For example, in a cutting action, the robot arm keeps a force between the knife and banana in the Z direction (constrained space), while moving the knife back and forth in the XY plane (unconstrained space).

This is possible by using parallel position–force control schemes such as that introduced by Chiaverini and Sciavicco (1993). The following control policy is used in this case:

$$\tau_{cmd} = J^{\mathrm{T}}(k_c(X^* - X) + F_{cmd}) + D(q) + f_{dyn}(q, \dot{q}, \ddot{q}) \quad (6)$$

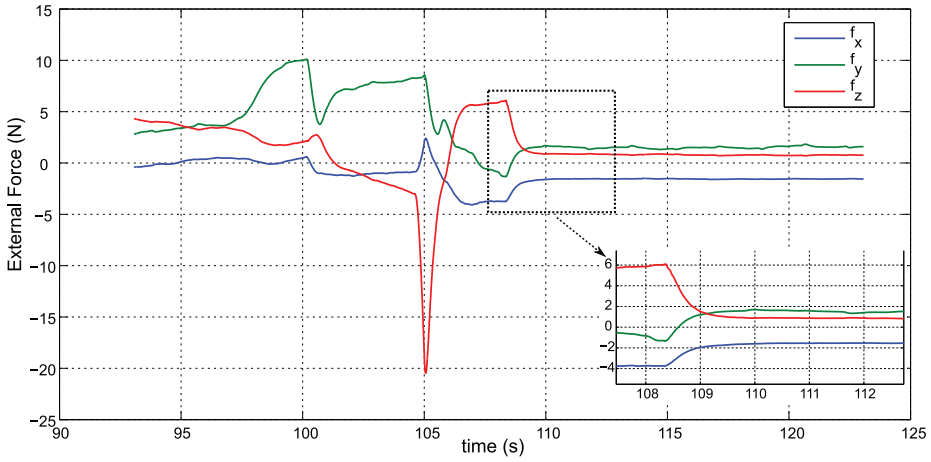

Fig. 7. Example of force control using the *arm_exert* primitive. The desired force is 1 N in the Z direction.

Fig. 8. Two pre-shape configurations are used in our system. The power grasp (right) is used for symmetric objects whereas the precision grasp (left) is for elongated objects.

Here the term F_{cmd} implements a force control that is feed-forward and PI terms:

$$F_{cmd} = F_{des} + K_p(F_{des} - f) + K_I * \int (F_{des} - f) \qquad (7)$$

An example of force control is shown in Figure 7 where the desired force is $F_{des} = [0, 0, 1]^{\mathrm{T}}$ N.

***hand_preshape*(*q*)**

This primitive is used to create a desired shape of the robotic hand. Our robot hand (Schunk SDH-2) has three fingers and in total seven degrees of freedom (DOFs). The control system of the hand has the ability to move to a desired configuration:

$$q = [q_1, q_2, q_3, q_4, q_5, q_6, q_7]^{\mathrm{T}} \qquad (8)$$

Two sample configurations are shown in Figure 8, which are the power and precision grasps used for round and elongated objects, respectively.

hand_grasp()

Manipulating objects usually requires grasping them first. For grasping, we use velocity control of finger joints together with feedback from tactile sensors on the fingers. The combination of *hand_preshape* and *hand_grasp* primitives enables us to grasp simple objects, which is enough to demonstrate the functionality of the proposed execution system. The complex problem of grasping arbitrary objects is out of the scope of this research.

hand_release()

This primitive is used to release a previously grasped object, which is simply opening the hand until the tactile sensors show that the object is released.

3.4. Mid-level action definition

So far, we explained high- and low-level action components. In this section, we present a mid-level component that acts as a bridge between those two levels and guides action execution. The core of the mid-level component is a FSM together with an error-handling protocol.

3.4.1. FSM. A FSM has a number of states, inputs, outputs, and transition rules. The states show different stages of the execution algorithm. The inputs are the real relations of the objects. The outputs are the robot primitives that are sent to the control system of robot arm and hand.

In the FSM we have some parameters that define the current action. The main parameters are the number of states, the desired relations and primitives at each state. To execute any action in the library, the proper set of parameters should be loaded into the FSM.

There are also some variables used during the FSM execution. For instance, variables *current_column* shows which column of the SEC matrix the current relations refer to. This variable is used to track the progress of the action.

The FSM is part of a software package to control the robot setup that is implemented using the Open Robot

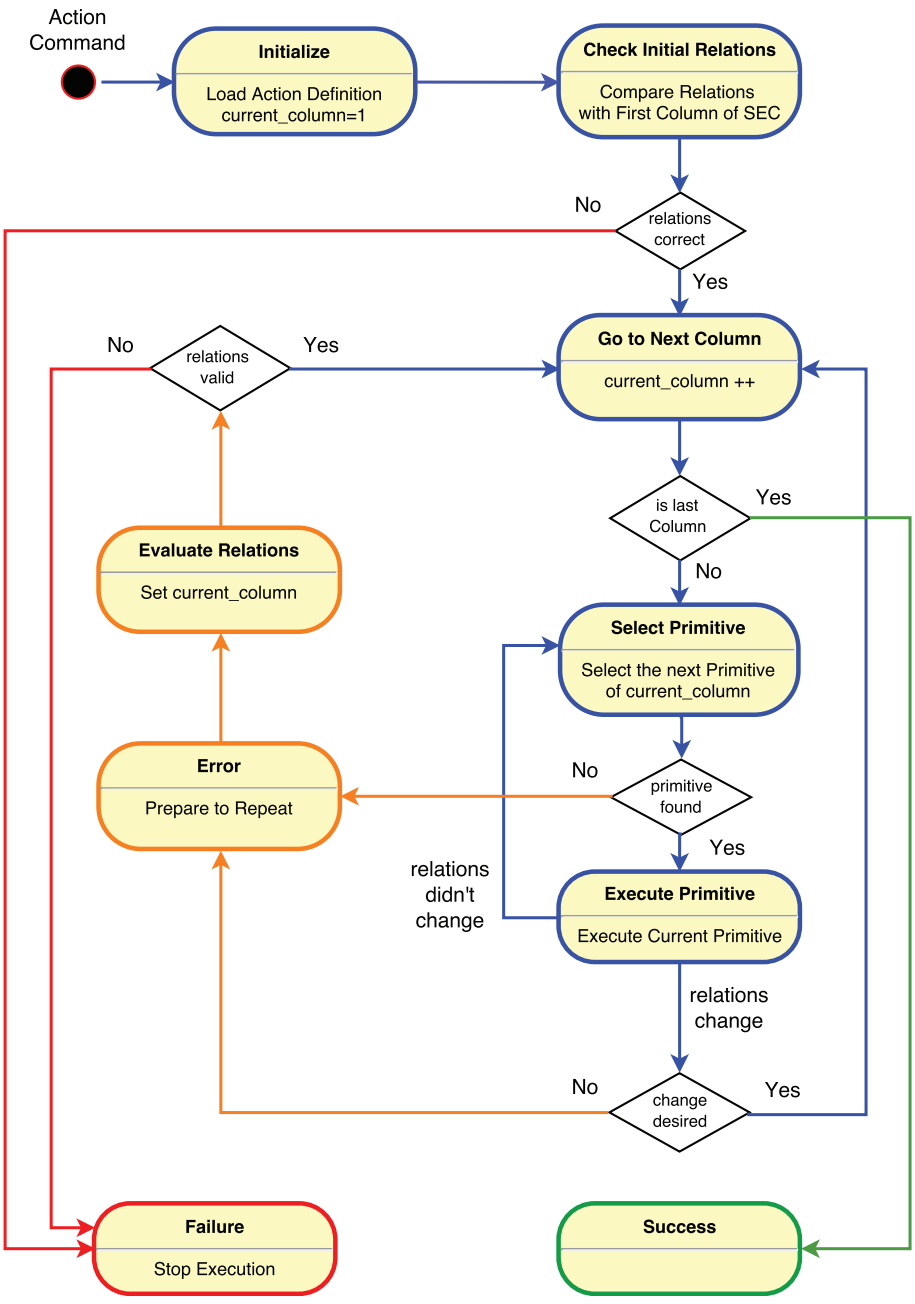

Fig. 9. State diagram of the FSM that controls the execution of actions. This state machine is the main component of the mid level. For clarity, different colors are used for normal execution (blue), error handling (orange), failure (red), and success (green) states and transitions.

Control Software (OROCOS) framework [Soetens(2013), Soetens(2006)]. The OROCOS framework provides tools to develop real-time robotic software including a useful FSM implementation. The overview of the mid-level FSM and execution process is shown in Figure 9. The details of states and transitions are as follows.

- **Initialize:** After receiving a new action command, the high-level definition of the desired action is loaded. The variable *current_column* is set to 1 to point to the first column of the SEC matrix. The action command consists of the action type, the main and other objects involved in the action.
- **Check Initial Relations:** Here the current relations of objects are compared with the first column of the SEC matrix of the commanded action. The pre-condition of executing the action is that the two are equal, otherwise the FSM transitions to the *Failure* state.
- **Go to Next Column:** This state increments the variable *current_column* and causes the action to progress from one column of the SEC matrix to the next. If we are already in the last column of the SEC matrix, it

transitions to *Success* state, which means the action is done successfully.

- **Select Primitive:** In this state the next primitive of the current SEC column is selected. If available, we transition to the *ExecutePrimitive* state. Otherwise, we are entering the *Error* state and we need to handle the error, because this implies that all primitives of the current SEC column are executed but the desired changes in relations did not happen.
- **Execute Primitive:** The selected primitive is executed here. This state has several sub-states, each performing one type of primitive. To keep the diagram simple, they are not shown in Figure 9. In this state, the relations of objects are monitored and if they change to the desired values, we transition to the *GotoNextColumn* state. If relations change to unwanted values, the next state will be the *Error* state. Finally if the primitive is done and no change in relations is detected, it transitions to the *SelectPrimitive* state, to look for the next primitive.
- **Error:** This state indicates that the execution of the current action is not progressing as expected. However, there is still hope to recover from the error, and continue the execution. There are two ways to enter this state. First, the primitives defined for the current SEC column are all executed but the desired change in relations has not occurred. Second, during the execution of primitives an unwanted change in relations happened.

In this state we try to go back to a previously known state of the action, and continue from that point. Usually this means that the robot arm retracts from the scene and receives new object poses and relations. After receiving the new perception, we transition to *EvaluateRelations* state. In Section 3.4.2 we show some examples of handling errors.

- **Evaluate Relations:** After receiving the new perception in *Error* state, we evaluate the current situation of the objects in this state. If for these relations, we could continue from the last known state, we transition to *GotoNextColumn* state and continue the execution. Otherwise, we go to *Failure* state since we are in an invalid state and can not proceed.
- **Failure:** If the relations of objects are in a way that there is no known way to proceed the execution, we transition to Failure state. At this point we stop the execution and announce failure. The failure is reported to the operator or the high-level planner, so that a proper decision can be made. Note that there is no high-level planner introduced here, since it is not in the scope of this work.
- **Success:** This state is entered if the action is successfully executed according to the SEC matrix.

3.4.2. Error handling. The execution of actions could fail due to different problems. Faults may happen in controllers and their interfaces, for which proper detection and recovery systems are necessary. Other errors may happen at more abstract levels like failure to properly grasp or push an object. These errors are detectable by observing the relations of the objects and we can deal with them in our execution engine.

In the previous section we described the *Error* state in the state machine, which is entered under these conditions:

1. the execution of the primitives does not result at the expected change in relations (from *SelectPrimitive* state);
2. the execution of the primitives causes unexpected changes in object relations (from *ExecutePrimitive* state).

To deal with these errors, first we undo the primitives of the current state (SEC column) to reach the previous known state. Then, we evaluate the object relations again and transition to the *CheckCurrentRelations* state and continue the execution. This results in a new perception of the position and relations of objects, by which the system decides which primitive should be executed next.

The first example shows an error when expected changes in relations do not happen. In Figure 10, the robot hand approaches the apple to grasp it, but fails, because the apple is not in the expected position. The manipulator retracts and receives the new position of the apple from the vision system and repeats the grasp.

An example of the second type of error is shown in Figure 11, in which after a successful grasp, the object is taken away from the robot hand. The robot detects the absence of the grasped object and reacts to it by opening the hand and moving up. After receiving the new position of the object, the grasp is repeated. The videos of error handling cases are shown in Extension 1 submitted with this paper.

There are cases where error handling cannot help, for example if we try to cut an uncuttable object (such as a cup). The error handling would try to repeat cutting the cup without success. After a few unsuccessful repetitions, the system transitions to the *Failure* state.

4. Experimental results

In this section, we present various experimental results of our proposed action execution framework. Results cover execution of both single atomic actions (e.g. *cutting, pushing*, etc.) and long chained activities such as "*making a salad*." Before presenting these results, we briefly introduce our hardware and software tools used in the experiments.

4.1. Hardware

Our setup consists of a robot manipulator, a three-finger robotic hand, and a vision interface.

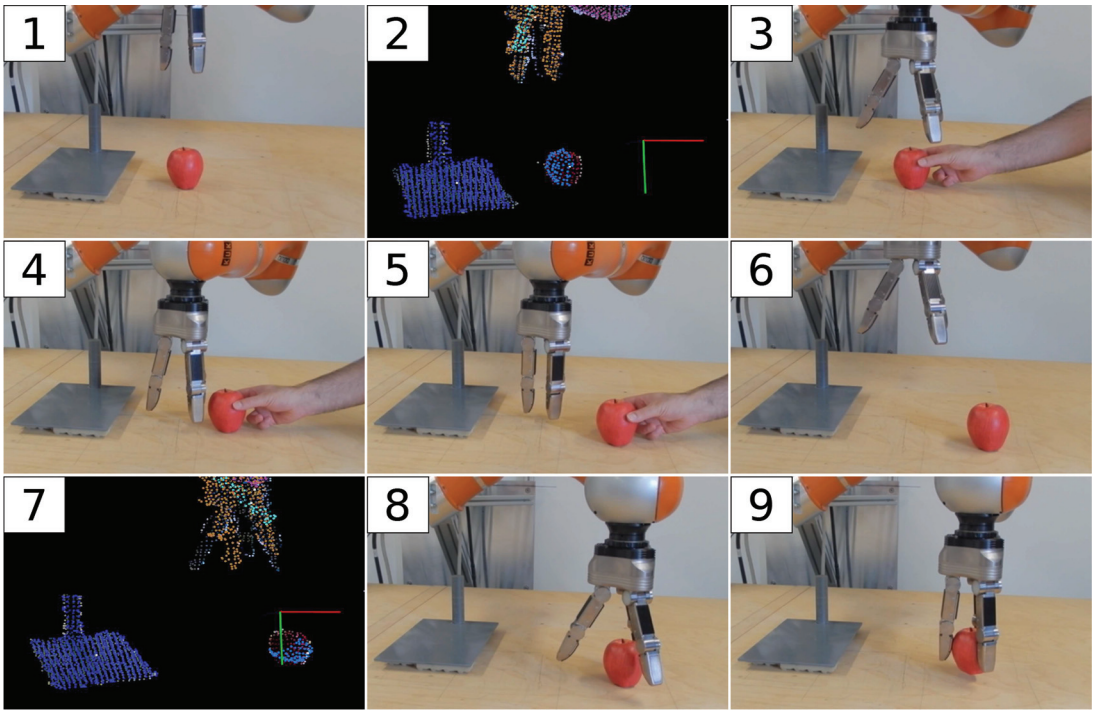

Fig. 10. Error handling after failure in grasping an object. 1. The initial scene. 2. Perception of the objects by vision system. 3–5. When the manipulator approaches to grasp the apple, we move it to cause the grasp to fail. 6. The robot hand opens and the robot arm moves up waiting for a new perception. 7. New perception of the objects by vision system. 8, 9. Approaching the apple in its new position and performing a grasp.

4.1.1. Robot arm. Our robot arm is a KUKA LWR (Light Weight Robot) IV manipulator. It is a kinematically redundant anthropomorphic manipulator developed jointly by KUKA Robot Group and the German Aerospace Center (DLR). It has 7 DOFs and is equipped with position and torque sensors at each joint. It estimates the external torques applied to each joint, which also gives an estimate of external force and torque at the end-effector. The robot can be controlled both in joint and Cartesian spaces with variable compliance and damping.

4.1.2. Robot hand. Our robot hand is a Schunk Dexterous Hand 2 (SDH-2) produced by the company Schunk. It has three fingers and 7 DOFs, which can be controlled in position or velocity modes. It is equipped with two tactile sensors on each finger, that provide feedback while grasping objects.

4.1.3. Vision system. Our vision system includes a static RGB-D (Asus Xtion) sensor and a DSLR camera (Nikon D7200). The RGB-D sensor provides both color and depth cues that are processed for image segmentation and tracking issues. The DSLR camera is further integrated into the vision system to capture high-resolution images of the scene for the purpose of object recognition. The vision system was developed using the ROS framework (Papon et al, 2012; Schoeler et al, 2014: see).

4.2. Single atomic actions

To quantitatively evaluate the proposed action execution framework, we conducted a large set of experiments with several types of actions and objects. The central goal here is to benchmark the success of the execution of actions provided in the library. Note, to arrive at a useful characterization of this framework all actions are analyzed *without error handling*. Only by this can decisive percent-success values be measured. We are not concerned with complex computer vision, thus, colored and textureless objects were mostly preferred in the experiments to cope with the intrinsic limitations of the imaging sensors and to have more reliable visual segmentation of perceived scenes. Figure 12 illustrates the set of manipulated objects, which contains in total 19 different samples from 8 categories, such as containers, round fruits, etc.

The first part of our experiments covers only single atomic actions, such as pushing, cutting, or stirring. The first 10 actions defined in Table 4 are considered as atomic actions, each of which is performed by the robot using objects of various types, sizes, shapes, and poses (see Figure 12). The executed atomic actions with their brief explanations and involved objects are listed in Table 5.

To evaluate the atomic actions, we provide two types of results. First, the success rate of execution of each action type is measured. These results give an overview of the execution performance of actions on different object categories

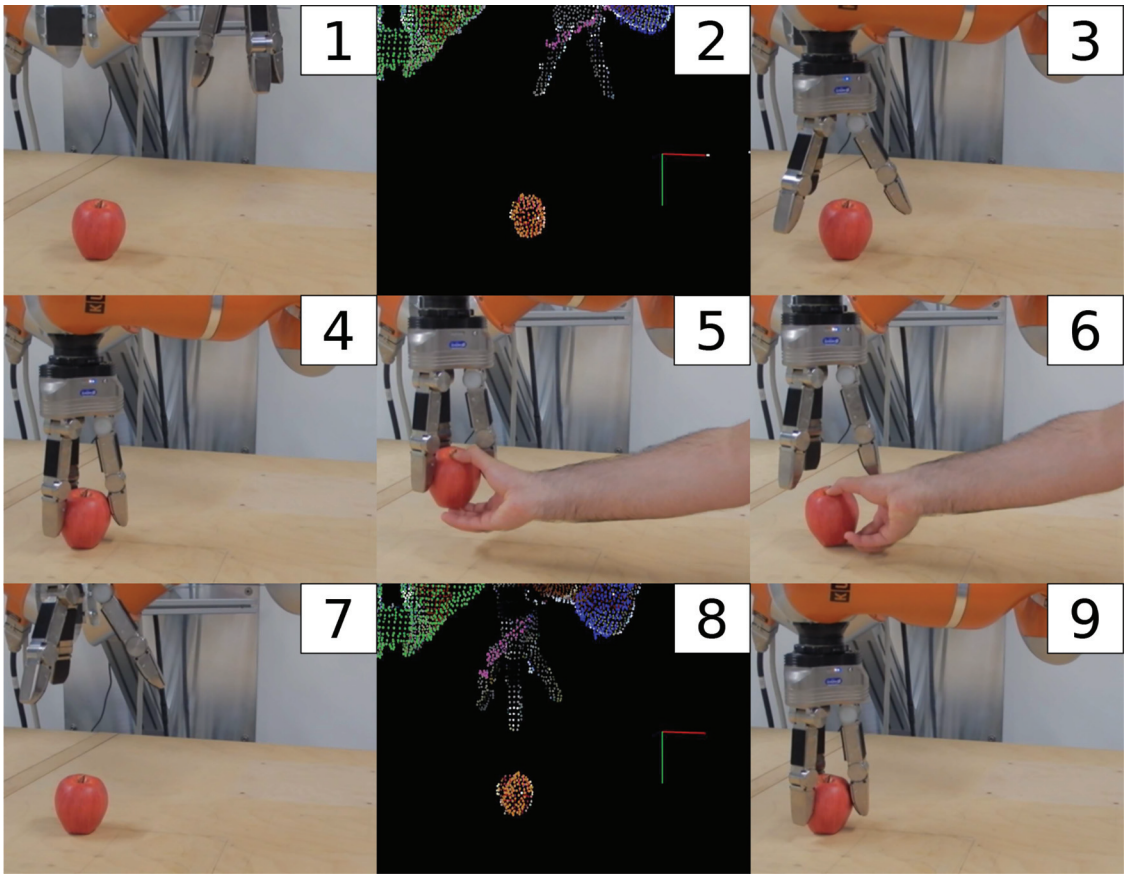

Fig. 11. Error handling after the grasped objects slips through the robot hand. 1- The initial scene. 2. Perception of the objects by vision system. 3, 4. The manipulator approaches the apple and grasps it successfully. 5. The grasped object is taken out of the robot hand to cause the error. 6, 7. The robot hand opens and the manipulator retracts. 8. New perception of the objects is received from the vision system. 9. The grasp is repeated successfully.

Fig. 12. The set of objects used in our experiments. There are in total 19 objects in 8 categories: 1. round fruits; 2. long fruits; 3. cubes; 4. cups; 5. containers; 6. plates; 7. spoons; 8. knives.

presented in various scene contexts. Thus, we can measure the robustness as well as the generalization capacity of the proposed action library. Second, we plot variations in the low-level sensory input, such as tactile, position, and contact signals, while the action is being executed. In these results, we can obtain information on the underlying perception mechanism in the execution framework and the discretization of the low-level continuous sensory data to reach high-level symbolic action representation.

To evaluate success rates, we repeated each atomic action 3 times on 10 different object sets, i.e. various object combinations with different poses. Thus, we obtained 30 trials for each action, i.e. in total 300 experiments. The overall success rate per action type is shown in Figure 13. Red bars in the figure depict the standard error of the mean.

The first result is that in 7 out of 10 actions, the success rate is equal or more that 50%. This shows that the system is able to cope with different objects and poses for most of the actions. The second impression that the figure conveys is that there is a prevalent failure, mostly observed in the execution of pushing actions which were mainly performed by just holding objects without applying any certain grasp, e.g. *push with holding* described in Table 5. The overall accuracy was measured as 64.5% and this value reached 75.8% in the case of excluding those failed pushing types. The main reason for this accuracy drop in pushing actions

Table 5. List of 10 atomic actions stored in the library and also used in experiments introduced in Section 4.2. The last two columns show sample objects used in each action.

#	Action Name	Explanation	Main	Tool	Primary	Secondary
1	Pick and Place	The main object is picked from primary and placed on the same object.	cup, apple, orange, cucumber, eggplant, bucket, plate, box	— — —	table	table
2	Push with Grasp	The main object is pushed to the goal position after being grasped.	box, apple, orange cucumber	— —	— —	— –
3	Push with Holding	The main object is pushed to the goal position after being held on top	box, apple, orange	—	—	
4	Poke	The main object is poked.	box, apple, orange	—	—	
5	Put on Top	The main object is put on top of the secondary object.	cup, cucumber, apple, orange	— —	table table	box, bucket, cup plate, board
6	Take Down	The main object is taken down from the primary object.	cup, banana, apple, orange	— —	box, bucket, cup board	table table
7	Push apart by holding	The main object is pushed apart from primary after being held from top.	orange, box	—	apple, cup	
8	Push together by holding	The main object is pushed to secondary after. being held from top.	apple, orange	—		box
9	Cutting	The main object is cut by the tool.	zucchini, cucumber banana	knife		
10	Stirring	The main object is stirred by the tool object.	bucket	spoon, knife		

is due to the shape of manipulated objects. For instance, while the robot was gently holding the object, e.g. an apple, to push it, the object slipped over the contact surface and, thus, led to a failure of the action. It is known that such types of actions are exceedingly difficult for robots but also for humans and we have often to reactively correct grasp and push to succeed. Hence, building in reactive correction mechanisms would certainly mitigate this problem.

We also observed a low success rate of about 50% for the cutting and stirring action types. In the stirring action, some of the failures occurred because the manipulated spoons were slightly too big for the containers. In the case of the cutting action, failures were due to inability to cut thick objects such as round fruits (apple or orange). Human cutting operations are heavily dominated by reacting to the "feel" of cutting and correcting force and angle.

A more detailed analysis on the execution of single actions is given in Figure 14. The results are separately computed for each individual action and object category. For those actions which initially require object grasping, the results are also categorized according to the grasp type. We here note that object grasping is not in the focus of this study and therefore in our experiments we only considered two types of grasps: power and precision. The average success rate for each grasp type and for the entire experiment are shown in the last three columns. For instance, in *Pick and place* action, 21 out of 30 trials were successfully performed with power grasp, which led to 67% average accuracy, whereas it was computed as 100% for the precision grasp.

A deeper look at the errors encountered for those 300 experiments shows that about one-third of them could have been recovered when switching error-handling on. The rest are non-recoverable. Given that the number of errors is in general relatively small, this estimate could, however, be incorrect, because not enough error data exist for strong statistical evaluation and many more experiments would be needed to achieve this.

Next, we take a closer look at the low-level sensory data including the position, tactile, and force contact signals of the robot arm during the experiments. Here, we aim at topological changes in the perceived scene by fusing data from several sensors, and to calculate object relations.

Figure 15 shows the position, tactile, and force sensor data together with detected relational changes between objects and the robot arm during the execution of a sample *Put on top* action. The first plot in Figure 15 confirms that due to the use of DMPs with joining, the robot arm seamlessly follows the desired goal positions which are indicated with circles. In the second plot in Figure 15 we also see that the tactile sensor is activated once the primary object is grasped. In a similar manner, the force sensor reports a contact when the object is placed. All these sensory data together with the visual feedback are fused to detect final spatial relational changes in the scene as described in section 3.3.2. Figure 15 at the bottom illustrates the extracted SEC representation over time for the *Put on top* action as a colored matrix which is identical to the one stored in the action library as shown in Figure 3. This plot confirms that the proposed action execution framework can successfully

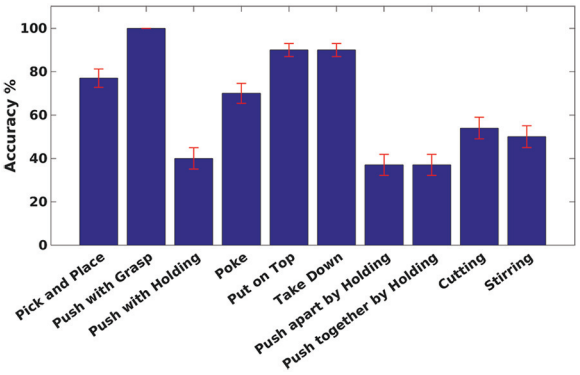

Fig. 13. Overall success rate of 10 atomic action executions after 30 trials for each.

process continuous sensory data and extract descriptive states in the scene, which yields compact high-level action representation, i.e. SEC.

In Figure 16, we show similar plots for a *Cutting* action in which the robot first grasps a knife and then cuts a cucumber into pieces. The position plot on the top highlights the oscillatory motion pattern of the robot arm during the actual cutting phase. Note that in some cases the actual robot position does not meet the goal position. This is expected, since whenever the desired relation changes happen, the current primitive is ended and the state machine moves to the next primitive. The second row shows how the tactile sensors detect the contact which happens at the *hand_grasp*() primitive (grasping the knife) and how this signal vanishes right after the *hand_release*() primitive.

The force signal in Z direction is used to verify the contact between the *tool* and *main* objects, in this case the knife and cucumber, which triggers the oscillatory motion. The extracted SEC is again the same as the one stored in the action library (see Figure 3).

In Figure 17 and Figure 18 we show the 3D trajectory of the robot arm for both *Put on top* and *Cutting* actions. The start and the end of the trajectory as well as position of objects in the action, are highlighted with red circles and text labels. Examples of single action executions are shown in Extension 2.

4.3. Chained actions

To demonstrate the scalability and strength of the proposed framework, we further benchmarked our system with execution of chained actions. For this purpose, we defined two scenarios. In the first scenario, the robot arm was given the

		Grasp type	*Main object category*						*Tool*		*Primary/ Secondary object category*									
Action Name	No. of Trials	Power / Precise	Round fruit	Long fruit	cup	container	cube	plate	knife	spoon	Round fruit	Long fruit	cup	container	cube	plate	knife	spoon	Success ratio	Success %	Average Accuracy (%)
Pick and Place	30	21	6/6	-	6/6	0/3	2/3	0/3	-	-	-	-	-	-	-	-	-	-	14/21	67	77
		9	-	6/6	-	-	3/3	-	-	-	-	-	-	-	-	-	-	-	9/9	100	
Push with Grasp	30	18	9/9	-	6/6	-	3/3	-	-	-	-	-	-	-	-	-	-	-	18/18	100	100
		12	-	9/9	-	-	3/3	-	-	-	-	-	-	-	-	-	-	-	12/12	100	
Push with Holding	30	-	4/6	2/9	0/6	-	6/6	0/3	-	-	-	-	-	-	-	-	-	-	12/30	40	40
Poke	30	-	6/6	6/6	0/6	-	6/6	3/6	-	-	-	-	-	-	-	-	-	-	21/30	70	70
Put on top	30	21	9/9	-	6/6	-	3/6	-	-	-	-	-	3/3	6/6	3/3	15/18	-	-	18/21	85	90
		9	-	9/9	-	-	-	-	-	-									9/9	100	
Take down	30	21	6/9	-	9/9	-	3/3	-	-	-	-	-	6/6	6/6	6/6	9/12	-	-	18/21	85	90
		9	-	9/9	-	-	-	-	-	-									9/9	100	
Push apart by holding	30	-	3/6	2/9	0/6	-	6/6	0/3	-	-	6/6	-	-	-	5/24	-	-	-	11/30	37	37
Push together by holding	30	-	3/6	2/9	0/6	-	6/6	0/3	-	-	6/6	-	-	-	5/24	-	-	-	11/30	37	37
Cutting	30	-	0/6	16/24	-	-	-	-	16/30	-	-	-	-	-	-	-	-	-	16/30	54	54
Stirring	30	-	3/12	12/12	-	0/6	-	-	-	15/30	-	-	-	-	-	-	-	-	15/30	50	50

Fig. 14. Success rate of executing actions in each object category. Each action is executed 30 times using different object sets. The ratio of successful trials are shown for each object category (middle columns). For actions involving grasp, the results are separately shown for each grasp type. The overall success rates on each grasp type and average success scores are shown in the last two columns. The values in the last column match the final average accuracy rates shown in Figure 13.

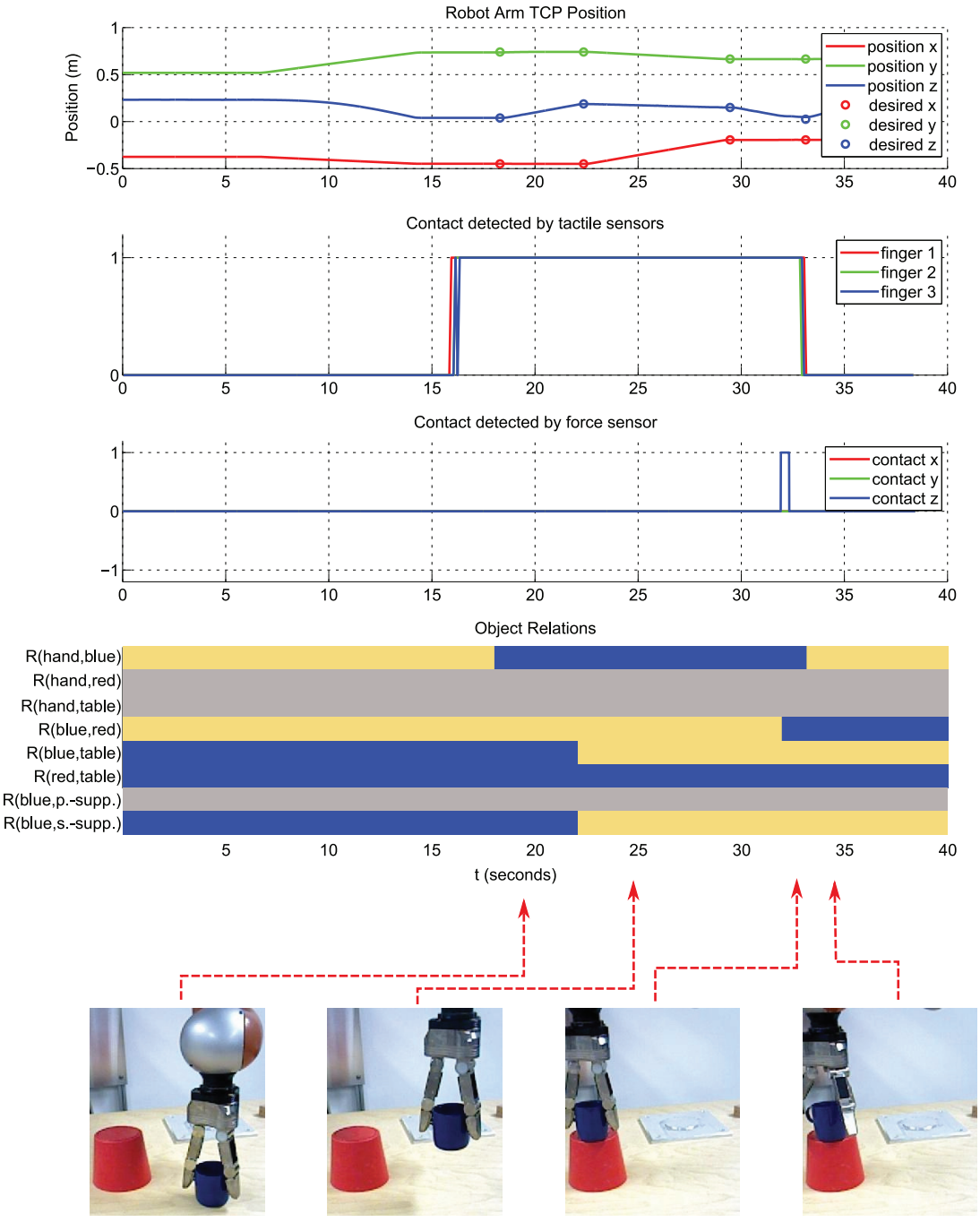

Fig. 15. Low-level sensory data in a sample *put on top* action. The position, tactile, and force contact signals are shown on the top. All changes in object contact relations are shown in the bottom plot as a color-coded SEC matrix. Here, blue and yellow represent *Touching* (*T*) and *Not touching* (*N*) respectively. The gray color shows either *Absence* (*A*) or relations which are not important (don't care). Note, primary support and secondary support are the same as primary itself (the table). From the definition in Figure 3 some relations can be ignored (don't care). Some sample snapshots at the bottom show the scene topology at each state of the action.

task of performing three atomic actions: *Take down, Push*, and *Put on top*. The second scenario is a more challenging task: *making a salad*.

Figure 19 shows the robot execution of the first chained action scenario. The first three plots depict the low-level sensory data. Due to having three atomic actions, there exist three peaks in the force sensor, whereas we obtain only two contact changes in the tactile sensor. In each action there is one interval at which the contact in *Z* axis is detected. However, we can see that in the second action there is no grasping.

In the second scenario, i.e. the salad making task, the robot performed a longer action sequence, in which we additionally introduced the last two actions defined in

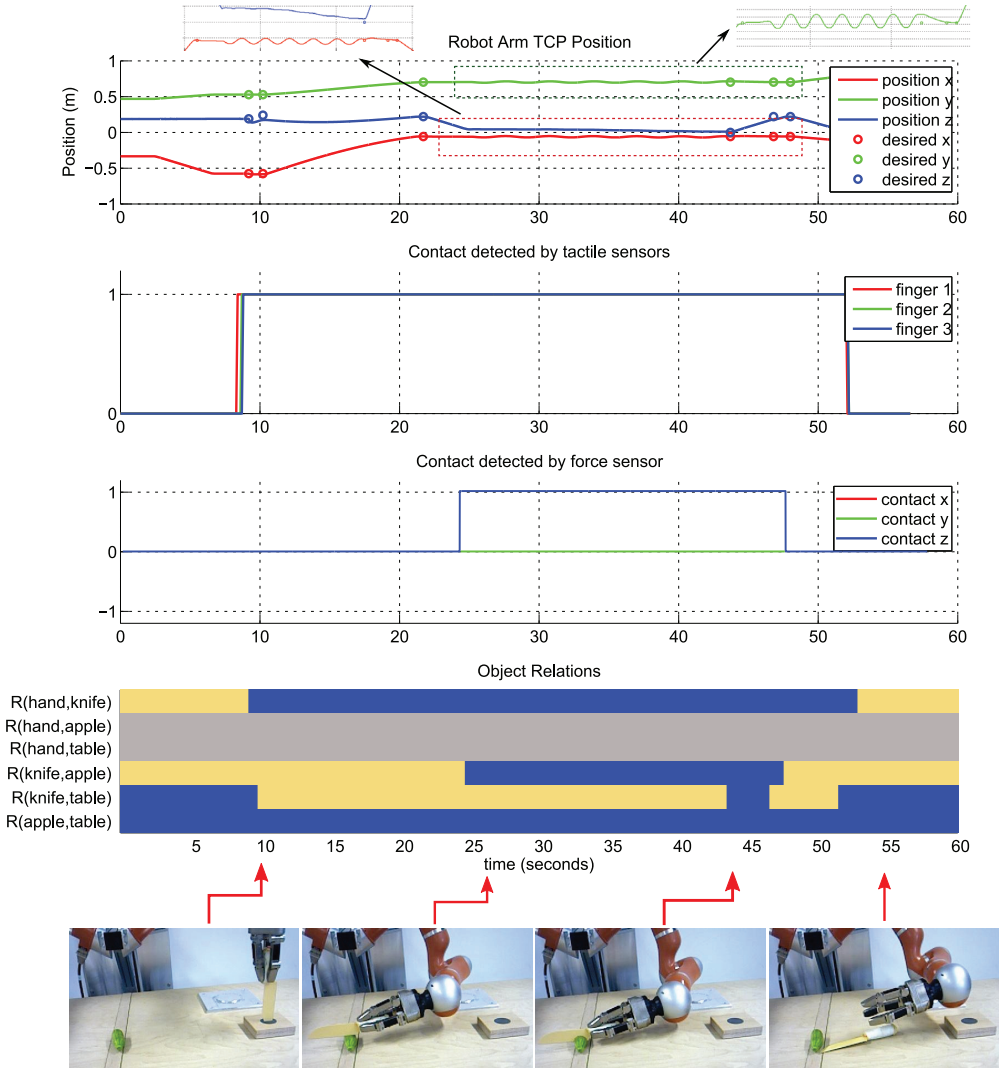

Fig. 16. Low-level sensory data in a sample *cutting* action. The position, tactile, and force contact signals are shown on the top. In the cutting action, a part of the trajectory corresponding to the back and forth motion of knife is zoomed in to show the oscillatory motion pattern. All changes in object contact relations are shown in the bottom plot as a color-coded SEC matrix. Here, blue and yellow represent *Touching* (T) and *Not touching* (N), respectively. The gray color shows either *Absence* (A) or relations which that not important (don't care). Compare with Figure 3. Some sample snapshots at the bottom show the scene topology at each state of the action.

Table 4: pouring and unloading. Consequently, the salad scenario contains the following steps:

1. pick up a cucumber and *put it on* a cutting board;
2. grasp the knife and *cut* the cucumber;
3. grasp the cutting board and *unload* the cucumber pieces into a bowl;
4. grasp the bottle and *pour* its content into the bowl;
5. grasp a spoon and *stir* ingredients in the bowl.

The final results of the salad scenario are shown in Figure 20. The low-level signals and high-level symbolic object relations are shown as usual. For the sake of clarity, sample snapshots for all five actions are shown vertically with horizontal arrows on the top showing the corresponding temporal interval of each action.

Note that in both scenarios, we assume that the high-level action plan is given in advance because we are not addressing any planning related issue in this study. Our only aim is to introduce a generic representation for the seamless execution of atomic and sequential actions independent from variations in the scene context.

The videos of chained action executions are shown in Extension 3.[1]

5. Discussion

The main contribution of this study was to give a thorough definition of manipulation actions at symbolic (high) and sub-symbolic (low) levels and link them through a mid-level FSM. The proposed state machine provides a mechanism to execute the actions on a robotic arm/hand system.

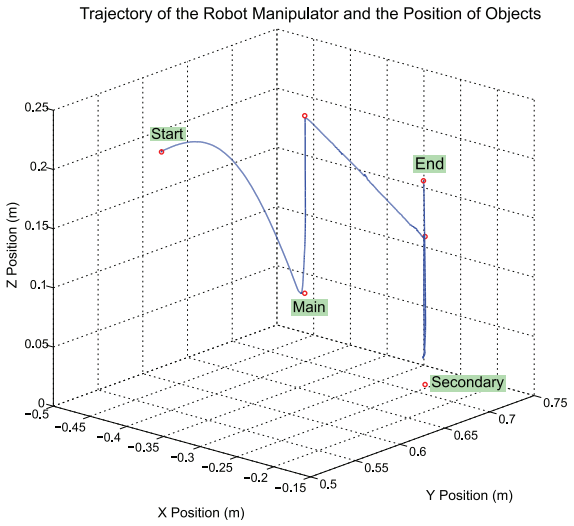

Fig. 17. Trajectory of robot arm during the *Put on top* action shown in Figure 15.

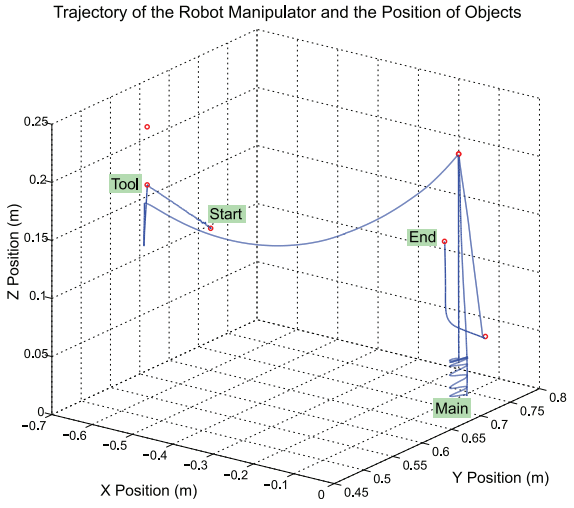

Fig. 18. Trajectory of robot arm during the *Cutting* action shown in Figure 16.

The proposed framework was tested on a wide range of actions and objects and we found satisfactory execution performance on various atomic and chained manipulation actions. Actions that produced problems are those notorious types that also humans find hard. So far our methods are excluding bimanual manipulation actions. True bimanual manipulations are rare, but also in a single-hand manipulation the supporting hand can help to control the force–torque patterns during the action. In Aksoy et al (2017) we performed a machine vision analysis also of manipulations, where both hands are used, which may pave the path for robotic execution.

5.1. Possible extensions

Note that all evaluations had been performed without engaging error correction so as to show the plain (feed-forward) properties of this system in a fair way. Performance increases to near perfect using error correction for all but the systematic errors; those, for example, where the chosen object lacks the required affordance for the action. It is the use of event chains that makes it possible to detect execution failures at all decisive action time points. If an expected object relation has not come into being, then an error must have happened. This event-based error detection is another advantage of our framework. Interestingly, a possible extension of this work would be to use it for active object affordance estimation. Repeated trials of an action will allow distinguishing random from systematic failures because random failures can be corrected, but systematic failures cannot. Hence, an uncorrectable error hints at a systemic lack of the required affordance (e.g. when repeatedly trying to cut a banana with a cup). This touches the field of developmental robotics (Pagliuca and Nolfi, 2015; Tikhano_ et al, 2013) where object affordance estimation remains a difficult issue.

Another way to extend this framework would be to create movement primitives automatically. At the moment we are working on an algorithmic framework that tries to achieve this using humans demonstration combined with DMPs and relying on the temporal chunking that the SECs provide. These preliminary results currently indicated that the here selected and hand-defined set of primitives is quite well reflected also by those that can be created by an automatic procedure.

Along the same lines, it is also conceivable to try to learn (some of) the parameter ranges needed for execution of a movement primitive for example by using reinforcement learning (RL) techniques. So far, parameter ranges have been set manually by us. In general, we observed that the rigorous chunking of the actions, using SECs and primitives, leads to large intrinsic robustness of our framework and it was never a problem to define the required parameter ranges. Still, using RL might lead to even more robustness, but would, very likely, require creating first a detailed simulation of the complete setup/framework to assure convergence of learning by allowing for enough iterations. Thus, implementing automatic primitive generation and/or parameterization via RL would, however, exceed the scope of this study by far.

In the context of a European project ACAT,[2] we had developed a massive XML schema called *action data table* (ADT) to capture actions generated by the framework proposed here. Essentially, this is the file format used to actually store the library data,[3] which are the high- and low-level data of an action as specified in the current paper. This data format enables us to execute new actions also on different robots and a main advantage of the action library framework is, thus, that it is (within reason) independent of the robot embodiment. Transfer to a different machine requires only the fine-tuning of a set of parameters (e.g. parameters related to the kinematic chain or to the sensor hardware). As a consequence, the framework proposed here has already been used as an action execution routine in different robotic applications (Agostini et al, 2015; Wörgötter

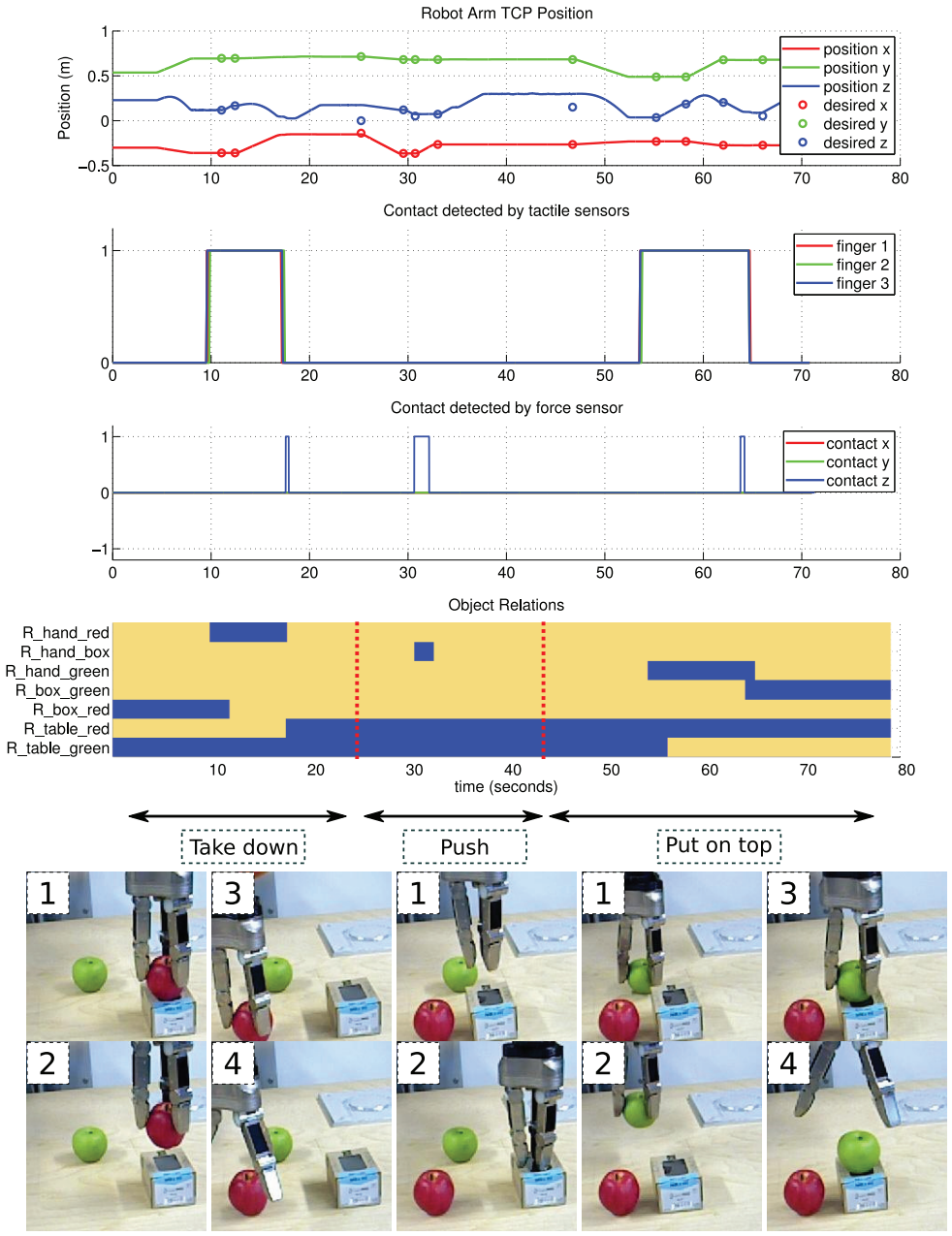

Fig. 19. Robot execution of three chained actions: 1. taking down the red apple from the box; 2. pushing the box by holding; 3. putting the green apple on top of the box. From top to bottom are shown the position, tactile, and force sensor data as well as the changes that are detected in the relation of objects in the scene. Sample snapshots for some SEC states are also depicted with numbers showing their order. Black arrows represent the temporal interval of each action.

et al, 2015). In the work of Agostini et al (2015), we showed that the robot can still generate similar actions by replacing tools in manipulations using the aspect of tool affordance. The work introduced in Wörgötter et al (2015) showed that robots can apply bootstrapping at different cognitive levels to improve their behavior based on the action representation and generation method proposed here. Therefore, we hope that the library of actions as proposed here can, in the long run, turn into a useful (and standardizable) robotics software tool also for other uses.

Funding

The research leading to these results has received funding from the European Union's Horizon 2020 Research and Innovation programme (grant agreement number 680431 (ReconCell); http://www.reconcell.eu).

Notes

1. All videos of this project including the extensions included here can be found at https://sites.google.com/site/aeinwebpage/actions/videos.

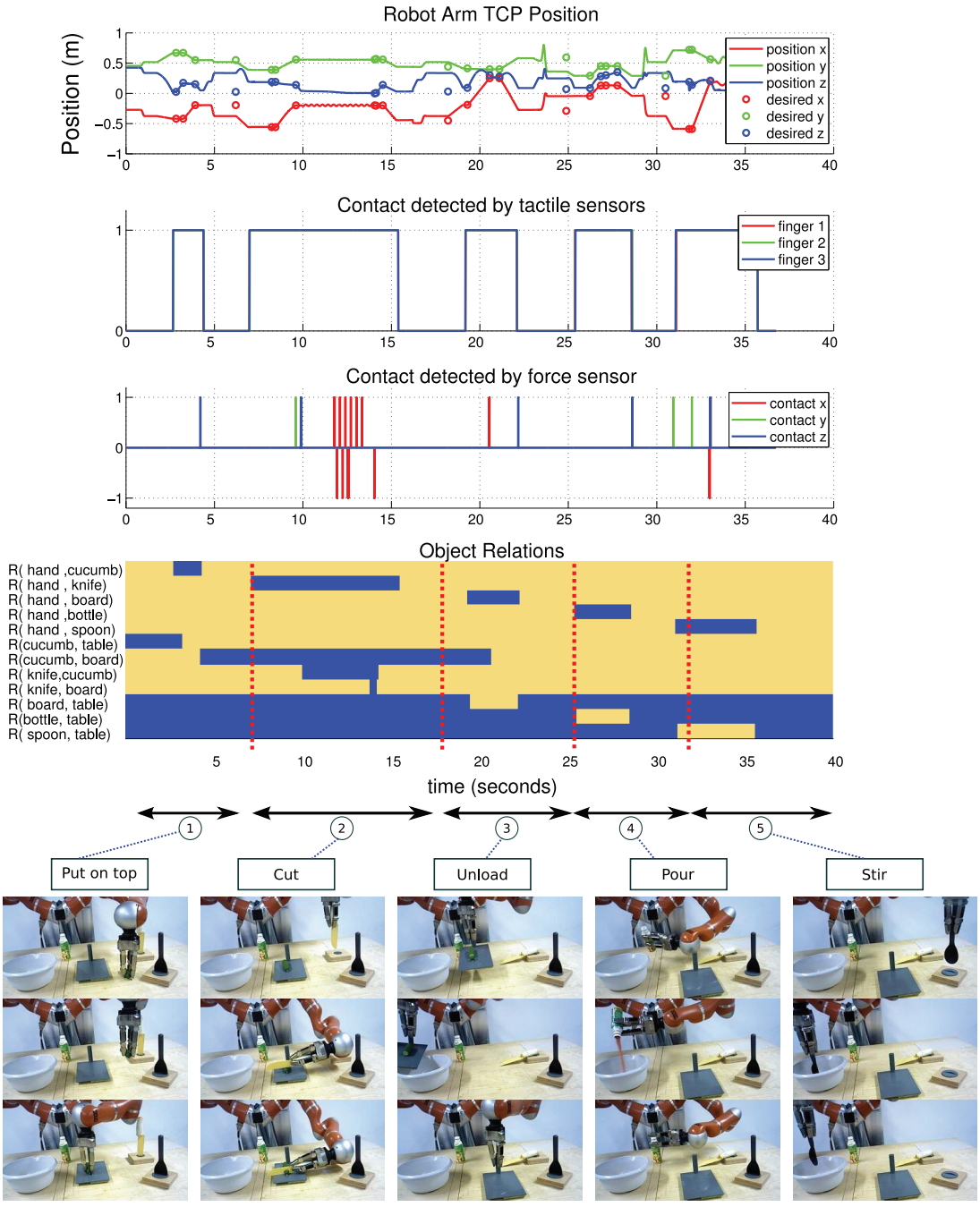

Fig. 20. Robot execution of a salad preparation scenario which involves five atomic actions: 1. put on top; 2. cut; 3. unload; 4. pour; 5. stir. From top to bottom are shown the position, tactile, and force sensor data as well as the changes which are detected in the relation of objects in the scene. Sample snapshots for some SEC states are also depicted at the bottom. Black arrows represent temporal intervals for atomic actions.

2. See http://www.acat-project.eu/.
3. See http://www.acat-project.eu/index.php?page=adt for examples of ADT.

ORCID iDs

Florentin Wörgötter https://orcid.org/0000-0001-8206-9738

References

Aein M, Aksoy E, Tamosiunaite M, Papon J, Ude A and Wörgötter F (2013) Toward a library of manipulation actions based on semantic object–action relations. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 4555–4562.

Agostini A, Aein MJ, Szedmak S, Aksoy EE, Piater J and Wörgötter F (2015) Using structural bootstrapping for object substitution in robotic executions of human-like manipulation tasks. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6479–6486.

Ahmadzadeh SR and Kormushev P (2016) *Visuospatial Skill Learning* (*Springer Studies in Systems, Decision and Control*). New York: Springer.

Ahmadzadeh SR, Paikan A, Mastrogiovanni F, Natale L, Kormushev P and Caldwell DG (2015) Learning symbolic representations of actions from human demonstrations. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3801–3808.

Aksoy EE, Abramov A, Dörr J, Ning K, Dellen B and Wörgötter F (2011) Learning the semantics of object-action relations by observation. *The International Journal of Robotics Research* 30(10): 1229–1249.

Aksoy E, Aein M, Tamosiunaite M and Wörgötter F (2015a) Semantic parsing of human manipulation activities using online learned models for robot imitation. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 2875–2882.

Aksoy EE, Orhan A and Wörgötter F (2017) Semantic decomposition and recognition of long and complex manipulation action sequences. *International Journal of Computer Vision* 122(1): 84–115.

Aksoy EE, Tamosiunaite M and Wörgötter F (2015b) Model-free incremental learning of the semantics of manipulation actions. *Robotics and Autonomous Systems* 71: 118–133.

Aleotti J and Caselli S (2006) Robust trajectory learning and approximation for robot programming by demonstration. *Robotics and Autonomous Systems* 54(5): 409–413.

Beetz M, Tenorth M and Winkler J (2015) Open-ease. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 1983–1990.

Calinon S, Guenter F and Billard A (2007) On learning, representing, and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* 37(2): 286–298.

Chiaverini S and Sciavicco L (1993) The parallel approach to force/position control of robotic manipulators. *IEEE Transactions on Robotics and Automation* 9(4): 361–373.

Coradeschi S and Saffiotti A (2003) An introduction to the anchoring problem. *Robotics and Autonomous Systems* 43(2): 85–96.

Dillmann R, Asfour T, Do M, et al. (2010) Advances in robot programming by demonstration. *KI - Künstliche Intelligenz* 24(4): 295–303.

Ekvall S and Kragic D (2006) Learning task models from multiple human demonstrations. In: *The 15th IEEE International Symposium on Robot and Human Interactive Communication, 2006 (ROMAN 2006)*, pp. 358–363.

Ghalamzan E, Amir M, Paxton C, Hager GD and Bascetta L (2015) An incremental approach to learning generalizable robot tasks from human demonstration. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 5616–5621.

He K, Lahijanian M, Kavraki LE and Vardi MY (2015) Towards manipulation planning with temporal logic specifications. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 346–352.

Ijspeert JA, Nakanishi J and Schaal S (2002) Movement imitation with nonlinear dynamical systems in humanoid robots. In: *Proceedings 2002 IEEE International Conference Robotics and Automation*, pp. 1398–1403.

Inamura T, Kojo N, Sonoda T, Sakamoto K and Okada Kand Inaba M (2005) Intent imitation using wearable motion capturing system with on-line teaching of task attention. In: *2005 5th IEEE-RAS International Conference on Humanoid Robots*. IEEE, pp. 469–474.

Kappler D, Pastor P, Kalakrishnan M, Wuthrich M and Schaal S (2015) Data-driven online decision making for autonomous manipulation. In: *Proceedings of Robotics: Science and Systems*, Rome, Italy.

Krüger N, Geib CW, Piater JH, et al. (2011) Object–action complexes: Grounded abstractions of sensory-motor processes. *Robotics and Autonomous Systems* 59(10): 740–757.

Kulvicius T, Ning KJ, Tamosiunaite M and Wörgötter F (2012) Joining movement sequences: Modified dynamic movement primitives for robotics applications exemplified on handwriting. *IEEE Transactions on Robotics* 28(1): 145–157.

Kunze L, Roehm T and Beetz M (2011) Towards semantic robot description languages. In: *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, pp. 5589–5595.

Lee D and Nakamura Y (2006) Stochastic model of imitating a new observed motion based on the acquired motion primitives. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4994–5000.

Lee K, Su Y, Kim TK and Demiris Y (2013) A syntactic approach to robot imitation learning using probabilistic activity grammars. *Robotics and Autonomous Systems* 61(12): 1323–1334.

Lioutikov R, Kroemer O, Maeda G and Peters J (2016) Learning manipulation by sequencing motor primitives with a two-armed robot. *Intelligent Autonomous Systems* 13: 1601–1611.

Luksch T, Gienger M, Mühlig M and Yoshiike T (2012) A dynamical systems approach to adaptive sequencing of movement primitives. In: *Proceedings of ROBOTIK 2012; 7th German Conference on Robotics*. VDE, pp 1–6

Luo G, Bergstrom N, Ek C and Kragic D (2011) Representing actions with kernels. In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2028–2035.

Martinez D, Alenya G, Jimenez P, et al. (2014) Active learning of manipulation sequences. In: *IEEE International Conference on Robotics and Automation*.

Morante S, Victores JG, Jardón A and Balaguer C (2014) On using guided motor primitives to execute continuous goal-directed actions. In: *The 23rd IEEE International Symposium on Robot and Human Interactive Communication, 2014 RO-MAN*. IEEE, pp. 613–618.

Pagliuca P and Nolfi S (2015) Integrating learning by experience and demonstration in autonomous robots. *Adaptive Behavior* 1059712315608424.

Papon J, Abramov A, Aksoy EE and Wörgötter F (2012) A modular system architecture for online parallel vision pipelines. In: *IEEE Workshop on Applications of Computer Vision (WACV)*, pp. 361–368.

Pardowitz M, Knoop S, Dillmann R and Zollner R (2007) Incremental learning of tasks from user demonstrations, past experiences, and vocal comments. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* 37(2): 322–332.

Pastra K and Aloimonos Y (2012) The minimalist grammar of action. *Philosophical Transactions of the Royal Society of London B: Biological Sciences* 367(1585): 103–117.

Rozo L, Jimenez P and Torras C (2013) Force-based robot learning of pouring skills using parametric hidden Markov models. In: *2013 9th Workshop on Robot Motion and Control (RoMoCo)*. IEEE, pp. 227–232.

Schoeler M, Stein S, Papon J, Abramov A and Wörgötter F (2014) Fast self-supervised on-line training for object recognition specifically for robotic applications. In: *International Conference on Computer Vision Theory and Applications (VISAPP)*.

Simmons R and Apfelbaum D (1998) A task description language for robot control. In: *Proceedings 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vol. 3*. IEEE, pp. 1931–1937.

Soetens P (2006) A software framework for real-time and distributed robot and machine control. PhD thesis, Department of Mechanical Engineering, Katholieke Universiteit Leuven, Belgium. Available at: http://www.mech.kuleuven.be/dept/resources/docs/soetens.pdf

Soetens P (2013) RTT: Real-Time Toolkit. http://www.orocos.org/rtt.

Srivastava S, Fang E, Riano L, Chitnis R, Russell S and Abbeel P (2014) Combined task and motion planning through an extensible planner-independent interface layer. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 639–646.

Tikhanoff V, Pattacini U, Natale L and Metta G (2013) Exploring affordances and tool use on the icub. In: *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. IEEE, pp. 130–137.

Ude A (1993) Trajectory generation from noisy positions of object features for teaching robot paths. *Robotics and Autonomous Systems* 11(2): 113–127.

Vuga R, Aksoy EE, Wörgötter F and Ude A (2014) Probabilistic semantic models for manipulation action representation and extraction. In: *Robotics and Autonomous Systems (RAS)*.

Wörgötter F, Aksoy EE, Krüger N, Piater J, Ude A and Tamosiunaite M (2013) A simple ontology of manipulation actions based on hand–object relations. *IEEE Transactions on Autonomous Mental Development* 5(2): 117–134.

Wörgötter F, Geib C, Tamosiunaite M, et al. (2015) Structural bootstrapping - a novel, generative mechanism for faster and more efficient acquisition of action-knowledge. *IEEE Transactions on Autonomous Mental Development* 7(2): 140–154.

Yamaguchi A, Atkeson CG, Niekum S and Ogasawara T (2014) Learning pouring skills from demonstration and practice. In: *2014 14th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. IEEE, pp. 908–915.

Yamaguchi A, Atkeson CG and Ogasawara T (2015) Pouring skills with planning and learning modeled from human demonstrations. *International Journal of Humanoid Robotics* 12(03): 1550,030.

Yang Y, Fermüller C and Aloimonos Y (2013) Detection of manipulation action consequences (mac). In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2563–2570.

Appendix. Index to multimedia extensions

Archives of IJRR multimedia extensions published prior to 2014 can be found at http://www.ijrr.org, after 2014 all videos are available on the IJRR YouTube channel at http://www.youtube.com/user/ijrrmultimedia

Table of Multimedia Extensions

Extension	Media type	Description
1	Video	Error handling cases
2	Video	Single action executions
3	Video	Chained action executions