2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

Toward a library of manipulation actions based on semantic object-action relations

M. J. Aein[1], E. E. Aksoy[1], M. Tamosiunaite[1], J. Papon[1], A. Ude[2], and F. Wörgötter[1]

***Abstract*— The goal of this study is to provide an architecture for a generic definition of robot manipulation actions. We emphasize that the representation of actions presented here is "procedural". Thus, we will define the structural elements of our action representations as execution protocols. To achieve this, manipulations are defined using three levels. The top-level defines objects, their relations and the actions in an abstract and symbolic way. A mid-level sequencer, with which the action primitives are chained, is used to structure the actual action execution, which is performed via the bottom level. This (lowest) level collects data from sensors and communicates with the control system of the robot. This method enables robot manipulators to execute the same action in different situations i.e. on different objects with different positions and orientations. In addition, two methods of detecting action failure are provided which are necessary to handle faults in system. To demonstrate the effectiveness of the proposed framework, several different actions are performed on our robotic setup and results are shown. This way we are creating a library of human-like robot actions, which can be used by higher-level task planners to execute more complex tasks.**

I. Introduction

It remains largely a puzzle how action knowledge can be efficiently structured for robotics applications. In a recent paper [1], we had discussed that human manipulation actions can be captured by a limited ontology-tree and we had argued that there is only a quite small number of basic actions existing (less than 30), which cover most if not all of our uni-manual manipulations.

This, we believe, offers the unique opportunity to structure the robot's manipulation action-space by defining all these actions in a default way to also store them in an action library. For any given robot (with not too different embodiment) execution of an action then means to access the required library function only having to re-parameterize it with respect to the *specific* action-situation (the scene) also taking into account all embodiment-specific constraints.

This idea as such is quite straight-forward but requires addressing many difficult problems. First and foremost we are asking for an appropriate action-representation from which an action library could be formed. Second, we need to define an architecture that can be used to access the library and actually execute an action.

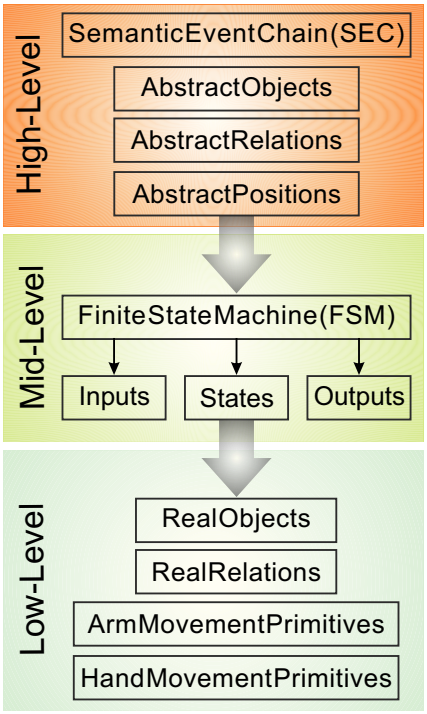

Fig. 1: An action is defined and executed using three levels.

To address both problems in a conjoint, process-oriented way, we suggest to use a layered action representation, which defines an action going from "generic" to "specific". We start at the top-level (Fig. 1, top) with a symbolic definition of actions proposed in [2] by which uni-manual human manipulations are defined as sequences of graphs that show how relations between objects change in the course of an action. From this a matrix representation of those graph sequences, called Semantic Event Chain (SEC), can be derived by which "the essence" of an action – the sequence of changes in a scene resulting from the action – is encoded . At this top-level objects and other action relevant parameters are encoded only in a very abstract way (described in Section II-A below).

One level down we define a finite state machine, which receives inputs from the SECs and produces an execution protocol of so-called "action primitives" (Fig. 1, middle). In general all robot arm- and hand-systems have low-level controllers, which make it possible to implement such action

*The research leading to these results has received funding from the European Community, Seventh Framework Programme FP7/2007-2013 (Specific Programme Cooperation, Theme 3, Information and Communication Technologies) under grant agreement no. 270273, IntellAct.

[1]Inst. Physics-3 & BCCN, University of Göttingen, Friedrich-Hund Platz 1, D-37077, Germany [aein,eaksoye,minija,jpapon,worgott] at physik3.gwdg.de

[2]Jožef Stefan Institute, Department of Automatics, Biocybernetics and Robotics, Ljubljana, Slovenia ales.ude at ijs.si

primitives. These low-level primitives are described in detail in Section II-B.3. Here the interfacing to the real world – to the currently observed situation – takes place and the *actual* execution parameters are determined from sensor information at the bottom level (Fig. 1, bottom). We use position and force sensors at the robot arm, a vision system to observe the scene, and tactile sensors of the robot hand.

By this, we achieve a systematic and general way of representing and executing all the actions of the ontology, which together form a *library* of human-like manipulations.

The proposed methodology and the resulting library of actions have some advantageous features. First there is the ability to perform actions on any robot manipulator platform that provides the required low-level primitive and sensor information. Second the framework generalizes well to different actions, objects and situations. And, third, it provides mechanisms to detect failures during execution and allows reacting to them.

The rest of the paper is organized as follows. Next follows Section I-A on the state of the art. Then, in Section II the proposed methodology is described. Section III is on failure detection and handling. In Section IV we present the results of experiments performed on our robotic setup. Finally, Section V provides a conclusion of the paper.

A. State of the Art

Commonly, there are two distinct approaches to represent actions, one at the trajectory level or the other at the symbolic level [3]. The former gives more flexibility for the 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 [4], Hidden Markov Models [5], Gaussian Mixture Models [6], Dynamic Movement Primitives [7], [8]. With trajectory level encoding, one designs or learns different complicated trajectories, but it is difficult to use them in a ”more cognitive sense”.

High-level symbolic representations many times use graph structures and relational representations, e.g., [9], [10], [2], [11], [12] (the last three are used in the current study). These representations give compact descriptions of complex tasks, but they do not consider execution-relevant motion parameter (trajectories, poses, forces) in great detail. However, in [12] it is already shown that actions defined by SEC can be executed once the 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 are brought together through STRIPS-like planning [3], resulting in operational although not very transparent systems. Thus, how to bring these two levels together remains a big challenge for robotics.

II. Description of the Proposed Method

In this section, the proposed methodology for executing actions is explained in detail. Fig. 1 shows the components of an action in three levels. In the next paragraphs, we explain these components along with one example, namely a *put-on* action, in which the goal is to lift an object and put it on another object. Then we discuss how the components of actions are obtained. At the end of this section, a summary is given that shows how these components are related.

Grammar:
Subj. + Verb + Direct Obj. + Indirect Obj. + Adv. of Place

Action:
tool + action + obj1 + obj2 + place

Example:
The knife cuts a cucumber which is on the table
The hand puts a box on top of another box on the floor

Fig. 2: Action-grammar analogy: The objects are abstracted to their roles in an action

TABLE I
Abstract relations and their attributes for the put-on action

#	abstract relation	type	sensor
1	r(tool,obj1)	variable	tactile
2	r(tool,obj2)	don't-care	vision
3	r(tool,place)	don't-care	force
4	r(obj1,obj2)	variable	vision
5	r(obj1,place)	variable	vision
6	r(obj2,place)	constant	vision

A. High-level Components

The high-level components define an action in an abstract way. At this level, the definitions are mainly symbolic and close to how a human would describe the action in words. The components of this level are as follows:

1) Abstract objects: There are many objects in the real world which makes it impossible to define actions for each of them. To deal with this problem, we use a property of actions in the ontology: A basic grammatical structure model of all the actions in the ontology, which is shown in Fig. 2.

In this structure the subject (tool) performs the verb (action) on the direct object (obj1) and (in some actions) on the indirect object (obj2). The adverb of place (place) is where the verb (action) happens e.g., table or floor.

By using this structure, objects are abstracted to their roles in the action. This helps to perform actions with different objects. Another important result is that the number of objects involved in an action is (at most) four. In the rest of the paper we call these abstract objects : tool, obj1, obj2 and place.

2) Abstract relations: Relations between objects play a key role in definition of actions. Since there are 4 abstract objects in an action, there exists 6 abstract relations. These abstract relations are assigned to pairs of abstract objects (See Table I).

Each relation is defined by three attributes, namely *type*, *value* and *sensor*. The *type* of a relation is determined by the importance and variation of that relation throughout the action. For example for the put-on action, the relation between tool and table, does not affect the outcome of

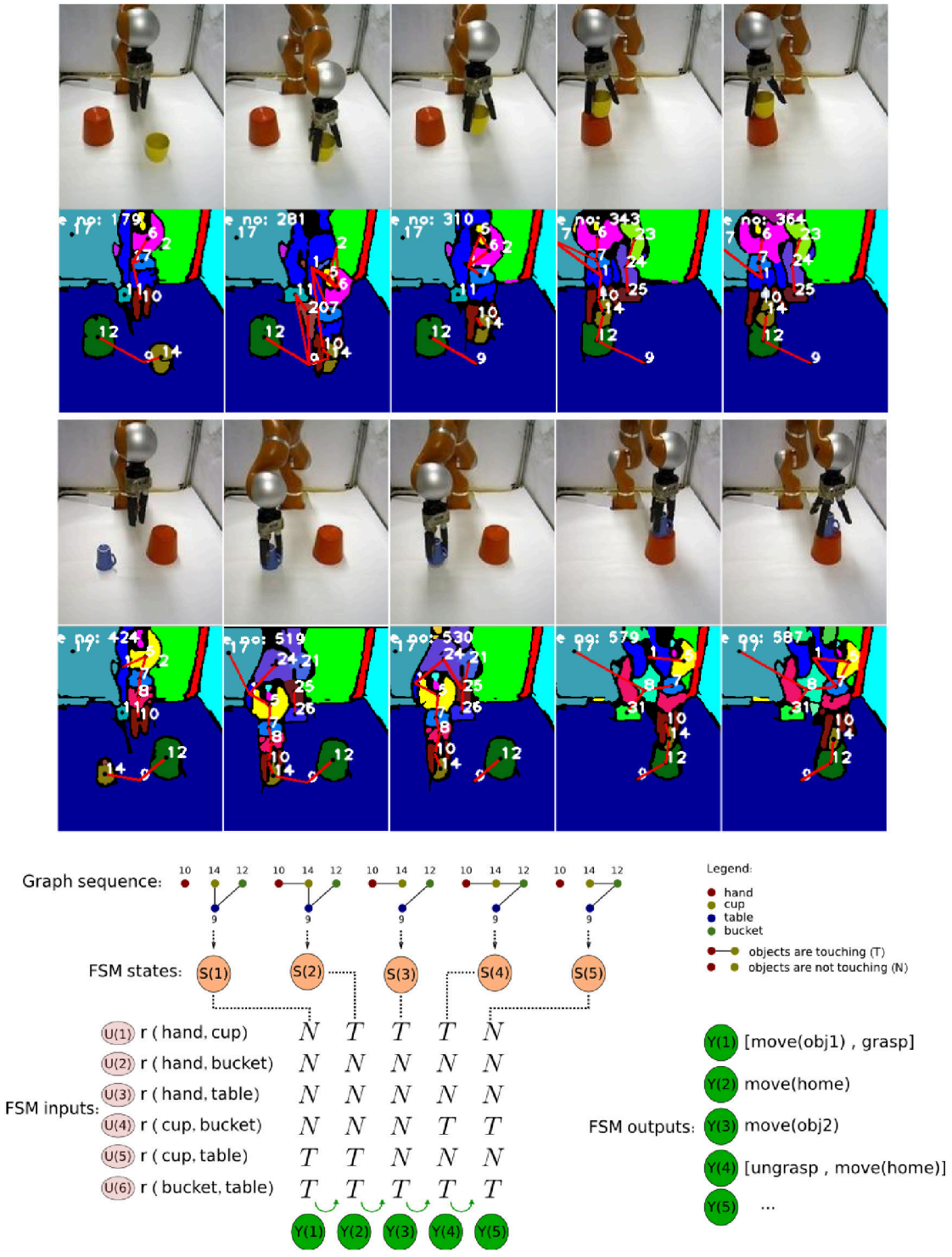

Fig. 3: Two instances of the put-on action, performed by our method, are shown with different objects and positions. The components of the action in three levels can be seen. The snapshots of the executed actions together with segmented images are shown (top) which indicate the real objects and relations. The arm and hand primitives are also listed (bottom right). The symbolic graph sequence (middle) and the SEC matrix (bottom) remain the same for both version. The states(S), inputs(U) and outputs (Y) of FSM and their relation to other components are shown.

the action. The type of such relations is *don't-care*. The importance of a relation is decided based on common sense.

Other relations, which are important for an action, are divided into *variable* and *constant* relations. For example, the relation between tool(hand) and obj1(cup) for put-on is *variable*, whereas the relation between obj2(bucket) and place(table) is *constant* (see Fig. 3). The constant relations show the necessary conditions to perform an action and a change in them implies failure of the action. On the other hand, variable relations encode the progress of the action and play the most important role in our methodology.

The *value* of a relation is either N (for Not touching), T (for Touching) and A (for absent). Absent value happens when we don't have information about the relation, e.g., one object is not defined or not visible.

The *sensor* attribute of a relation, determines the type of sensor that is used to detect that relation. We use position, tactile, force and vision sensors to obtain the value of a relation.

Table I shows the abstract relations and their attributes for a put-on action.

3) SEC: The actions in the ontology are defined as sequences of graphs. The nodes of these graphs are objects while the edges show the touching relation between a pair of objects. A SEC is a matrix derived from these graph sequences. Each row of the SEC matrix shows changes that occur in one abstract relation. Therefore, as we have four abstract objects we get six abstract relations and hence the SEC has 6 rows. The number of columns of SEC, depends on how many times the relations between objects change in the course of action, since there is at least one change in relation between adjacent columns.

The valid symbolic entries of SEC matrix are N (not touching), T (touching) and A (absence). In a SEC, the progress of the action from the beginning to the end, is seen in a compact way. In addition, the SEC matrix of different instances of one action remains the same. Two instances of a put-on action, together with graph sequences and SEC matrix, are depicted in Fig. 3.

4) Abstract positions: When performing actions, sometimes we move our hands toward points in space where there are no objects. One example is when we lift an object from a table to put it on top of another object. We move the object to a point that makes our next move easier. We call such points, abstract positions, and use them in our method.

In our method, we use two kinds of abstract positions, namely *home* and *goal* positions. The example above was a home position, while an example of a goal position is when we push an object on a surface to another point. The actual definition of abstract positions depends on the action and position of other objects. In Fig. 3 it is shown that the primitive $move(home)$ is used twice in this action.

B. Low-level Components

1) Real objects: In real world experiments, the abstract objects (tool, obj1, obj2 and place) are instantiated by real objects in the scene. For the put-on example depicted in Fig. 3, these objects are tool=hand, obj1=cup, obj2=bucket and place=table. We need to represent the real-world objects with numbers so that we can perform actions.

To this end, the objects are initially represented by a point in Cartesian space showing their center of mass. While this is enough for many actions, there are cases where more information is needed. For example in the stirring action we need to know how much the radius of the container (e.g. bucket or cup) is, in order to adjust the diameter of stirring.

In such cases, we extend the object model in the particular way required for the given action. In the stirring example, we use a cylindrical model which has a parameter that gives the radius of the object.

2) Real relations: The real relations are the values of abstract relations at each specific time obtained by sensors. In other words, real relations are the same as the *value* attribute of abstract relations discussed earlier. These relations are shown in graphs shown in segmented images of Fig. 3 for two instances of put-on action.

3) Arm/Hand primitives: An action is carried out through movements of the tool and its interaction with other objects. To imitate this in a robotic setup, we need to send commands to the low-level controllers of the robot arm and hand. These commands are called *primitives*. For the robot arm, we implement primitives for position and force control, namely $move(p_{des}, P)$ and $exert(f_{des}, P)$. For the robot hand, we have $open$, $close$, $grasp$ and $ungrasp$ primitives.

The $move(p_{des}, P)$ primitive, moves the end effector from the current position, to the position specified in p_{des}. The parameters of the trajectory are stored in P. We use Dynamic Movement Primitives (DMP) with joining [13] to generate trajectories. The parameters P are the parameters of its DMP and consist of weights of Gaussian kernels W, time T and number of repetitions N. The details of implementation are not covered here, but the final result is that by using the primitive $move(p_{des}, P)$, we can move the robot arm from the current position and orientation, to any desired position and orientation, along the desired trajectory. The trajectory shape is encoded in W, the duration of action is T and the number of times that the trajectory should be repeated is shown by N ($N > 1$ is only possible if start and end points coincide).

The $exert(f_{des}, P)$ performs force control with set-point f_{des}. In our setup, we use a parallel force/position scheme with dominant force control, similar to [14]. In this way we can achieve the desired force in the constrained space, and at the same time, perform desired motions in the unconstrained space, the trajectory of which is defined by P.

The controller for the robotic hand is able to perform $open$, $close$, $grasp$ and $ungrasp$ commands. $open$ and $close$ commands simply open and close the fingers of hand, whereas $grasp$ and $ungrasp$ use feedback from tactile sensors to make sure an object is grasped or released.

C. Mid-level sequencer: Finite State Machine

In order to perform actions defined by a SEC, the low-level primitives need to be executed in a sequence, taking

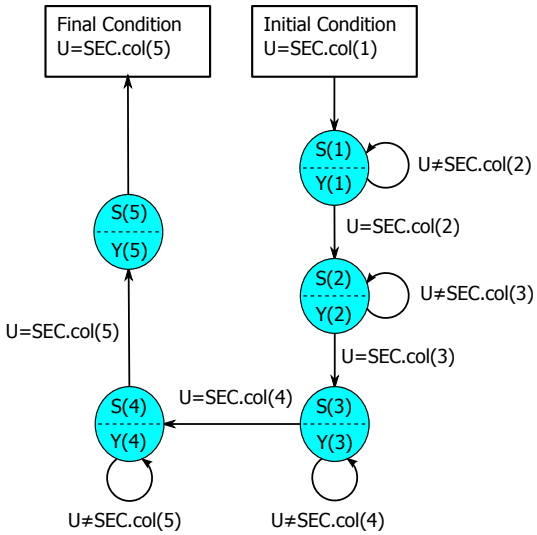

Fig. 4: A mid-level sequencer is implemented by a finite state machine (FSM) for put-on action. The action starts if initially inputs (real relations) match the first column of the SEC matrix. The transition from state i to $i+1$ occurs when the inputs match the i-th column of the SEC matrix. Otherwise, the next state is the same as the current state (loop transitions). This process is continued until reaching the final state which corresponds to the last column of the SEC matrix.

into account the relations between objects. There should be a connection between the high- and low-level, and this connection is made by introducing a mid-level sequencer. Here, it is implemented by a Finite State Machine (FSM).

An FSM is a logic unit, that determines outputs and the next state of a system, based on the current state and inputs. It is formally defined as a 6-tuple (S, U, Y, f_1, f_2, s_1). The following paragraphs show the definition of these variables. It is also shown how these variables are related to the high-level and low-level components. In particular, the variables for the put-on example are derived to complete the example. For the put-on action the resulting FSM is depicted in Fig. 4.

1) States (S): As the name suggests, there exists a finite set of states in an FSM denoted by S. The interpretation of these states depends on the application. Each column of SEC matrix shows a state of the system. Therefore, states of FSM are associated with the columns of a SEC. For example, the SEC matrix for put-on has five columns, therefore the FSM has five states : $S = \{S(1), S(2) \cdots, S(5)\}$ as shown in Fig. 3.

2) Inputs (U): The inputs of the FSM, are the inputs of the system. In action execution, the inputs are the abstract relations. There are six abstract relations, therefore we need six inputs $U = [U(1), U(2), \cdots, U(6)]^T$. For the put-on example, the inputs are highlighted in Fig. 3.

3) Outputs (Y): An FSM can produce outputs depending on current state and inputs. These outputs are defined in terms of defined commands of the application. In our application, the outputs are a combination of arm/hand primitives, that should get called in the appropriate sequence. In the put-on example there are five states, therefore there are five outputs $Y = \{Y(1), Y(2) \cdots, Y(5)\}$, shown in Fig. 3. Since the last state is already the goal state, the last output is always empty.

4) State transition function (f_1): The state transition function determines the next state, based on current state and inputs.

In action execution, the transition from state i to $i+1$ happens when the inputs are equal to the $(i+1)$th column of the SEC matrix. When checking the equality, the inputs that correspond to don't-care relations are not taken into account. Hence, the transition function for put-on example is as follows:

$$\underset{i=1,2,3,4}{f_1(S(i), U)} = \begin{cases} S(i+1) & \text{if } U = SEC(:, i+1) \\ S(i) & otherwise \end{cases} \quad (1)$$

5) Output function (f_2): In general, the outputs of an FSM depend on state and inputs. However, here we use a specific kind of FSM called Moore machine, in which outputs depend only on the states. Hence we have a simple output function which associates outputs to each state:

$$\underset{i=1,2,3,4}{f_2(S(i))} = Y(i) \quad (2)$$

6) Initial state ($S(1)$): The initial state is associated with the first columns of the SEC that represents the initial relations between objects. It is assumed that at the beginning the real relations match the first column of the SEC matrix, otherwise the action won't start.

D. Obtaining the components

For some of the above mentioned components we need to explain more about how they are obtained. One is the SEC matrix which defines actions at the highest level. This matrix is obtained by imitation of human behavior by two distinct methods. In the first method, we record several human demonstration of an action. Then by processing the recorded images the relations between pairs of objects are obtained. This includes segmentation of images, graph annotation and tracking which is done by our vision system [15]. By using learning algorithms and filtering out the noise, it is possible to achieve a clean and compact SEC matrix that gives a high-level definition of action (for details see [2] and [12]).

The other method is to use common sense, to draw the graph sequence of actions and derive the SEC matrices. This method is effective since for humans it is relatively easy to describe actions in terms of relations of objects. These methods result in similar SEC matrices. However, in this paper the second method is used since it is easier to generate structured SEC matrices.

The outputs of the FSM at each state, are also obtained by common sense. This means that we decide what low-level primitives should get called in each state, to achieve the desired change in relations.

E. Summary of the proposed methodology

The proposed methodology can be summarized as follows:

- The SEC matrix that defines an action is obtained from the graph sequences of actions.
- The relations and their attributes are identified. Here, it is decided which relations are important, and which are don't-care, based on common sense. The abstract relations are the inputs of the FSM.
- The states of the FSM are assigned to each column of the SEC.
- The low-level primitives that must be performed in each state are selected. These primitives form the outputs of the FSM.
- The objects involved in the action, namely tool, obj1, obj2 and place, are instantiated by real world objects.

At this point the FSM starts getting input (real relations), sending outputs (arm/hand primitives) and makes state transitions when necessary, until the action is done.

The FSM is implemented in C++ and for each action all the necessary data are stored in a human-readable data format called YAML (recursive acronym for YAML Ain't Markup Language). In this way, doing actions boils down to loading the proper YAML file and executing it with the robot.

III. Failure detection

In general it is currently not possible to achieve full fault tolerance for robot actions outside factory floors, because situations can vary widely in unconstrained (less constrained) environments leading to unforeseen contingencies. We can show that the definition of actions based on relations between objects, provides two specific ways to *detect* failures, which are not present in other action definitions.

As mentioned in Secion II, the important relations that define an action, are divided into two types, variable and constant. First we note that the constant relations serve as preconditions for executing an action. For example, for the put on action, where the goal is to put cup on top of bucket, which is on the table, it is necessary that bucket is indeed always and in a stable way on the table. If at some point during the action, the sensors detect that bucket is not on the table, a failure is detected.

Secondly, the variable relations can also be used to detect system failure. As described before, in each state of FSM, the output calls some low-level primitives. These primitives are supposed to change the object relations in a certain way as defined by the SEC. However, if a certain primitive has been executed and the variable relations do not change in the expected way (as predicted by the next column of the SEC), a failure is detected. An example of such failure in the put-on action is when the tool reaches the position of the obj1, and tries to grasp it, but finds nothing to grasp.

Both cases – missing precondition, case 1; or non-resulting postcondition, case 2 – contain specific information, for example about which specific transition expected from the SEC has gone wrong. Hence, deeper analysis of the encountered exception will immediately allow suggesting the required error correction mechanisms. A simple correction for non-resulting postcondition failure would be to perform the same primitive one more time.

TABLE II

List of actions executed by the proposed method

#	action name	tool	obj1	obj2	place
1	pushing	hand	box	-	table
2	put on	hand	cup	bucket	table
3	take down	hand	cup	bucket	table
4	stirring	spoon	liquid	container	table
5	poking	hand	box	-	table
6	pick&place	hand	cup	-	table
7	hiding	hand	cup	bucket	table

IV. Experiments and Results

To show the features of the proposed framework, several experiments are performed on our robotic setup. First, our setup is introduced. Then, the results of the experiments are presented. In addition, the videos of all experiments are available at our website `https://sites.google.com/site/aeinwebpage/actions/videos`.

A. Experimental Setup

To perform the experiments, a Kuka LWR IV robot manipulator is used which provides task-space position and force control. It has position sensors to locate the end effector of the arm, and the estimation of external force applied on the end effector.

A three-finger dexterous hand from Schunk (SDH2) is mounted on the robot arm to interact with objects, or grasp tools (knife, spoon, etc.). The tactile sensors located on the fingers, are used to detect the touching relation between hand and object.

The Oculus vision system [15] is used for real-time segmentation and extracting the spatial relation between objects, as well as to calculate the position, orientation and dimensions of the objects in the scene.

B. Results

Actions from the ontology that are executed with the described method are listed in Table II together with examples of real objects which could be used in them. This includes the put-on action that is used as an example throughout the paper.

The results of the put on action are already used in Fig. 3. The goal of this actions is to put the cup on top of the bucket. The results for two more actions namely, *pushing* and *pick and place* are presented here.

In pushing, the goal is to push an object by robot hand in the desired direction. Here, a box is used and the desired direction is perpendicular to the length of the box. Two versions of pushing action are demonstrated with different positions and orientations to show the generalization properties of the proposed method.

The goal of pick and place action is to move an object to a desired position. Here we show an experiment in which a cup is moved 20 cm to the right. The results of these experiments

are shown in Fig. 5 and Fig. 6. The results and videos of other actions can be seen in our website (the link is provided above).

V. Conclusion

In this paper a methodology is proposed to execute unimanual human actions defined in an action ontology by SECs. In this methodology, the actions are defined in a layered architecture that goes from generic level to specific. The method is applicable in any robotic arm/hand system that provides the low-level primitives and proper sensors to analyze the scene. Two methods to detect failure in actions by monitoring the relations are presented. The presented results show the ability of this method to perform several of these actions.

References

[1] F. Wörgötter, E. Aksoy, N. Krüger, J. Piater, A. Ude, and M. Tamosiunaite, "A simple ontology of manipulation actions based on hand-object relations," *IEEE Transactions on Autonomous Mental Development (in press)*, 2013.

[2] E. E. Aksoy, A. Abramov, J. Dörr, K. Ning, B. Dellen, and F. Wörgötter, "Learning the semantics of object-action relations by observation," *The International Journal of Robotics Research*, vol. 30, no. 10, pp. 1229–1249, 2011.

[3] R. Dillmann, T. Asfour, M. Do, R. Jkel, A. Kasper, P. Azad, A. Ude, S. Schmidt-Rohr, and M. Lsch, "Advances in robot programming by demonstration," *KI - Knstliche Intelligenz*, vol. 24, pp. 295–303, 2010, 10.1007/s13218-010-0060-0. [Online]. Available: http://dx.doi.org/10.1007/s13218-010-0060-0

[4] A. Ude, "Trajectory generation from noisy positions of object features for teaching robot paths," *Robotics and Autonomous Systems*, vol. 11, no. 2, pp. 113 – 127, 1993. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0921889093900155

[5] D. Lee and Y. Nakamura, "Stochastic model of imitating a new observed motion based on the acquired motion primitives," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, oct. 2006, pp. 4994 –5000.

[6] S. Calinon, F. Guenter, and A. Billard, "On learning, representing, and generalizing a task in a humanoid robot," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 37, no. 2, pp. 286 –298, april 2007.

[7] J. A. Ijspeert, J. Nakanishi, and S. Schaal, "Movement imitation with nonlinear dynamical systems in humanoid robots," in *Proc. 2002 IEEE Int. Conf. Robotics and Automation*, 2002, pp. 1398–1403.

[8] T. Luksch, M. Gienger, M. Mühlig, and T. Yoshiike, "A dynamical systems approach to adaptive sequencing of movement primitives," in *Proceedings of the 7th German Conference on Robotics (Robotik 2012)*, 2012, to be published.

[9] M. Pardowitz, S. Knoop, R. Dillmann, and R. Zollner, "Incremental learning of tasks from user demonstrations, past experiences, and vocal comments," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 37, no. 2, pp. 322 –332, april 2007.

[10] S. Ekvall and D. Kragic, "Learning task models from multiple human demonstrations," in *Robot and Human Interactive Communication, 2006. ROMAN 2006. The 15th IEEE International Symposium on*, sept. 2006, pp. 358 –363.

[11] E. E. Aksoy, A. Abramov, F. Wörgötter, and B. Dellen, "Categorizing object-action relations from semantic scene graphs," in *IEEE International Conference on Robotics and Automation (ICRA)*, may 2010, pp. 398–405.

[12] E. E. Aksoy, B. Dellen, M. Tamosiunaite, and F. Wörgötter, "Execution of a dual-object (pushing) action with semantic event chains," in *Proceedings of 11th IEEE-RAS International Conference on Humanoid Robots*, 2011, pp. 576–583.

[13] T. Kulvicius, K. J. Ning, M. Tamosiunaite, and F. Wörgötter, "Joining movement sequences: Modified dynamic movement primitives for robotics applications exemplified on handwriting," *IEEE Trans. Robot.*, vol. 28, no. 1, pp. 145–157, 2012.

[14] S. Chiaverini and L. Sciavicco, "The parallel approach to force/position control of robotic manipulators," *Robotics and Automation, IEEE Transactions on*, vol. 9, no. 4, pp. 361–373, 1993.

[15] J. Papon, A. Abramov, E. E. Aksoy, and F. Wörgötter, "A modular system architecture for online parallel vision pipelines," in *IEEE Workshop on Applications of Computer Vision (WACV)*, jan. 2012, pp. 361–368.

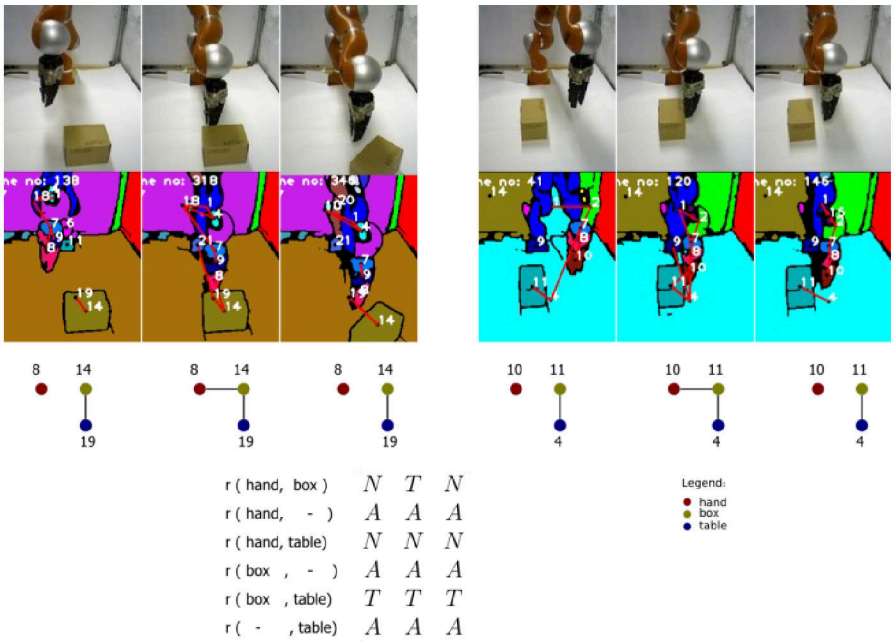

Fig. 5: The snapshots of two instances of *pushing* action are shown together with segmented images (real relations), symbolic graphs and the SEC matrix.

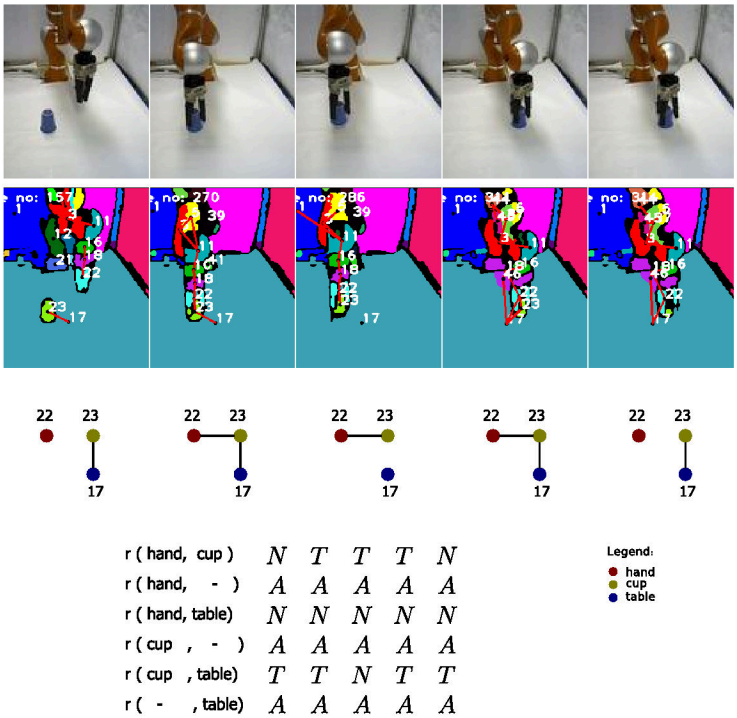

Fig. 6: The results of pick and place action are shown together with segmented images (real relations), symbolic graphs and the SEC matrix.