Execution of a Dual-Object (Pushing) Action with Semantic Event Chains

Eren Erdal Aksoy[1], Babette Dellen[1,2], Minija Tamosiunaite[1] and Florentin Wörgötter[1]
[1]Bernstein Center for Computational Neuroscience
University of Göttingen
Friedrich-Hund Platz 1, D-37077
Email: [eaksoye,minija,worgott]@physik3.gwdg.de
[2]Institut de Robòtica i Informàtica Industrial (CSIC-UPC),
Llorens i Artigas 4-6, 08028 Barcelona, Spain
Email: bdellen@iri.upc.edu

***Abstract*—Execution of a manipulation after learning from demonstration many times requires intricate planning and control systems or some form of manual guidance for a robot. Here we present a framework for manipulation execution based on the so called "Semantic Event Chain" which is an abstract description of relations between the objects in the scene. It captures the change of those relations during a manipulation and thereby provides the decisive *temporal anchor points* by which a manipulation is critically defined. Using semantic event chains a model of a manipulation can be learned. We will show that it is possible to add the required control parameters (*the spatial anchor points*) to this model, which can then be executed by a robot in a fully autonomous way. The process of learning and execution of semantic event chains is explained using a box pushing example.**

I. Introduction

If one wants to build robots which participate in everyday human life, learning from demonstration is perhaps the most practical paradigm. Although learning from demonstration has much advanced in recent years [1], [2], manipulation learning from demonstration has not yet come to its conclusion as here one has to bring together a sequence of demonstrated movements and task knowledge [3]. In our previous works [4], [5] we have introduced the so-called "Semantic Event Chain" (SEC) which is a compact and generic encoding scheme for manipulations. We have shown that the SECs can be used to allow an agent by observation to classify different manipulations and to categorize the manipulated objects based on their roles exhibited in the manipulation. Furthermore, we have demonstrated that an agent can learn an archetypical SEC model in an unsupervised way by watching about 10 demonstrations. The main advantage of this framework is that SECs link the signal domain (observed image sequences) to a symbolic rule-like domain encoding a manipulation in a highly invariant way, where – for a given manipulation – objects, poses, perspectives and trajectories can be interchanged to a very large degree. Thus, SECs provide one possible, quite efficient way to perform manipulation recognition and to learn a manipulation model.

SECs are essentially a symbolic representation encoding the manipulation by a temporal sequence of rules. Manipulations appear now in an abstract form, stripped from all pose and trajectory information and this makes it initially impossible "to invert the process" using a SEC for executing a manipulation. Thus, in the current paper we address the question how to actually do this and perform a manipulation starting from a SEC. Clearly, this requires that in the process of learning a manipulation model (learning the SEC) additional information must be stored, for example the start and endpoint of movement trajectories. But, because the SEC provides a temporal sequence of rules, we have well defined temporal anchor points when we have to store the additionally required trajectory information. Furthermore, as will be shown below, SECs also provide us with exact instructions, which spatial coordinates (spatial anchor points) are relevant for defining a movement.

The goal is to arrive at a fairly generic instruction set which allows "bringing two objects in close contact". Hence, to perform a basic dual-object action. Pushing two objects against each other but also pick one object up and placing it in contact to a second object fall into this category. To do this we will provide at the end a macro where we define an instruction set (D1-D8) which produces the basic movement segments for the execution. It is important to note that this macro will be applicable for all dual-object actions using the semantic event chains to structure and define the different execution primitives.

The structure of the paper is as follows. In Section II we discuss related works. In Section III, we briefly summarize our own prior work on how to learn a SEC from observation. Here we address the (new) problem, how to store information which is additionally required for execution. Next, in section IV, we describe how to actually execute a manipulation starting from a SEC showing simulation results. In Section V, the results are discussed and directions for future research are given. Finally, the work will be concluded in Section VI.

II. Related Work

In our previous works [4], [5] we have shown that SECs can both, classify manipulations and categorize manipulated objects in a model free way, needing prior representations neither for objects nor for actions. In [6] the authors built

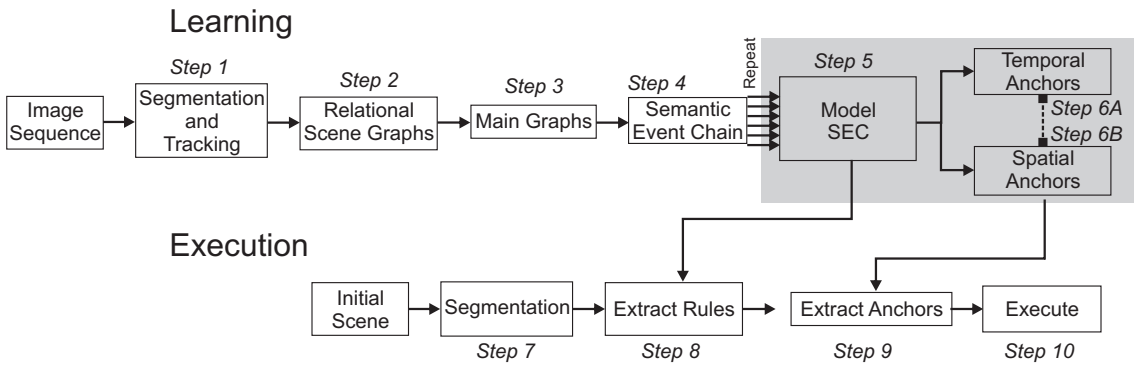

Fig. 1. Block diagram of the whole framework.

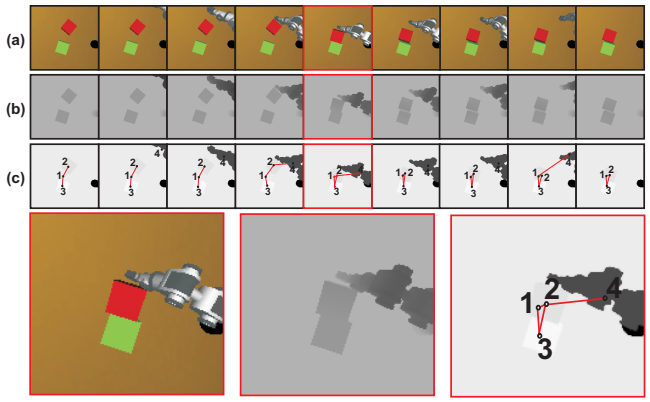

Fig. 2. Pushing action. **(a)** Original images from a movie recorded during the action. **(b)** Corresponding depth map from a range finder. **(c)** Corresponding HSV color based segmented images with extracted 3D scene graphs (See steps (1-2) in Fig. 1). Note that each object is represented by a unique segment label (e.g. 1, 2, 3, and 4 that represent *table*, *red box*, *green box*, and *robot arm*, respectively). Graph nodes represent the segments' centers and graph edges encode whether or not two segments touch each other in 3D. In red are indicated *Touching* relations between segments. The bottom part of the figure shows a magnified view of frames 5 from above.

a kernel based vectorial representation of event chains, which makes SECs more compatible with machine learning techniques. However, they have not addressed object categorization, learning and execution issues at all. Different from that, in the current work we will focus on execution of a learned pushing action by using SECs.

In the literature many works focus more on the (mechanical) aspects of controllability and planning of stable pushing actions [7], [8]. Such aspects are not in the core of our paper.

In [9] the authors showed how an agent can learn simple pushing actions on a toy object and then execute them as goal-directed behaviors. During the training phase, time evolution of the initial hand position and the direction of object displacement at the moment of contact were continuously recorded. As will be shown below, this is to some degree similar to our approach. In each trial the robot learns to map from initial hand position to the direction of object movement. However, the robot had only four possible initial positions which restricts the flexibility of manipulations in the execution phase of the learned maps. The high number of required trials (approximately 70) is another unrealistic drawback of this work.

In a different study [10] the problem of learning a general pushing rule has been addressed. The rule represents the relationship between the point and angle of push on the object's boundary and the observed object motion right after the pushing action. In the learning case the robot experimented with different pushing actions on different objects at different positions. The normalized retinal images of the experimental data served as input to a neural network to predict the object velocity in all directions. However, the input images had to be down-sampled to 20x15 pixels which causes much information loss. Moreover, in the testing case the robot has to drive an optimization process, the computational complexity of which is relatively high.

In [11] the authors described an on-line learning method for pushing an object to a desired (image) position. The system used past pushing operations to estimate future pushing actions. The main handicap of their approach is that the object is connected to the robot with a rotational point contact.

III. METHODS

In the current study, we perform execution of a pushing action by means of SECs. For this purpose we used the Webots software that simulates a 6 DOF Neuronics Katana robot arm. The experiment consists of two phases: Learning and execution (Fig. 1). In the first phase we perform manipulation demonstration. For this we already use the robot simulation and program it by hard-coding to push a red box to a green box on a table until they touch each other and then the robot is going to a home position. We could have used human demonstration instead, like in our older papers, but this is not of any relevance. The only thing needed is that demonstration is repeated using different setups.

Note, sophisticated object recognition is not part of this work. This is a difficult additional problem for which there are no generic solutions. In limited scenarios, such as those commonly used in state of the art robotics experiments, one could resort to conventional model-based object recognition methods. This step is simplified in our study and we get object

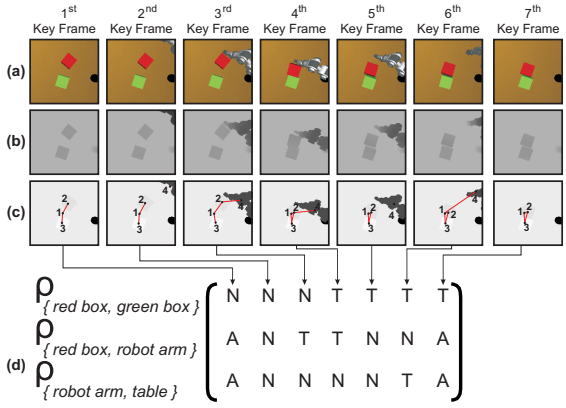

Fig. 3. Semantic event chain representation. **(a)** Original *"Key Frames"*. **(b)** Corresponding depth maps. **(c)** Corresponding HSV color based segmented images with extracted main graphs (See step 3 in Fig. 1). **(d)** Corresponding semantic event chain (See step 4 in Fig. 1), which is a sequence-table, where each entry encodes the spatial relations between each segment pair $\rho_{i,j}$ at each main graph. T means that segments touch (denoted by red edges), N means that there is no edge between two segments, and absence of a previously existing segment yields A.

identities (boxes, robot arm) by a unique color code, as more complex object recognition does not add to the relevant aspects of this study.

We also do not require that the red and green box should touch each other in some exact configuration. As a consequence, relative pose information can be neglected. However, it is also possible to store different touching types (i.e. pose information) between boxes in addition at the exact touching moment which is provided by the SEC. But, for the purpose of demonstrating the process of macronizing execution from a SEC it is sufficient to focus on the "ballistic push".

From these demonstrations a SEC-model is then learned. During learning also additional decisive information, for example relative coordinate frames and information about motion start and endpoints, is recorded. In the second phase (execution), we use the SEC and the additional information to let the robot execute a similar pushing action regardless of the initial state of the table.

A. Segmentation and SEC-generation (Steps 1-4)

Fig. 2 and Fig. 3 show a processing example of a manipulation resulting in its semantic event chain representation. We first extract all frames from the manipulation movie (Fig. 2 (a)) with corresponding depth maps from a range finder (Fig. 2 (b)). Frames are then segmented by a simple HSV color based segmentation algorithm, which allows for consistent marker-less tracking of each object (Fig. 2 (c)). Note that each object is represented by a unique segment label (e.g. 1, 2, 3, and 4). Once segments are calculated we drive a simple color based object recognition algorithm to replace those unique labels with object names (e.g. $table$, $red\ box$, $green\ box$, and tip of $robot\ arm$ instead of 1, 2, 3, and 4, respectively). After this step the agent knows which segment corresponds to which object. The scene is then represented by undirected and un-weighted graphs (Fig. 2 (c)). Nodes represent object center points and edges between nodes exist whenever two objects touch each other in 3D. Note, during a manipulation, graphs can change by continuous distortions (lengthening or shortening of edges) or, more importantly, through discontinuous changes (nodes or edges can appear or disappear). This happens when objects touch (or un-touch) each other. Such a discontinuous change represents, thus, a natural breaking point: All graphs before are topologically identical and so are those after the breaking point. Hence, we can apply an exact graph-matching method [12] at each breaking point and extract the corresponding topological main graphs. The sequence of these main graphs represents all structural changes (manipulation primitives) in the scene. The movie frames that hold such changes are called *"Key Frames"*. Fig. 3 (a-c) shows the *"Key Frames"* with corresponding depth map, segments, and main graphs for the action in Fig. 2. This type of graph representation is then encoded by the semantic event chain (Fig. 3 (d)), which is a sequence-table. Hence continuous time is now replaced by time-chunks where the same main graph persists until in the next chunk a new one appears. Each row in a SEC represents the temporally changing relations between one pair of objects in the scene, for example the first row in (Fig. 3 (d)) shows the relation between the red box and green box. There are three possible spatial relations defined between segments: *absence* (A), *no connection* (N), and *touching* (T). N means that there is no edge between two segments, corresponding to two spatially separated segments, and T represents segments that touch each other[1]. A special case exists when a segment has disappeared, which will be denoted by A.

Consequently, the complete image sequence, which has here roughly 320 frames, is represented by an event chain with a size of only 3×7. Note that several spatial relations, for example between *green box and robot arm* ($\rho_{3,4}$), are not included in this SEC since they do not contain any N-T or T-N transitions, which are decisive for a manipulation. Thus, we can always ignore such rows in the semantic event chain since they do not describe any manipulation relevant event.

These aspects are represented by steps (1-4) in Fig. 1, showing the block diagram of the complete algorithm.

B. Defining Temporal Anchor Points by Learning the SEC Model (Steps 5,6)

For learning we record the same pushing action from 10 different perspectives by changing the camera positions and extract the corresponding SECs. Fig. 4 shows the same, specific moment of the manipulation for each of the 10 different manipulation instances which allows us to get an impression of the level of perspective difference.

All movies used in this study can be found at http://www.dpi.physik.uni-goettingen.de/~eaksoye/movies.html. As described elsewhere [4], [5], we then calculate the pairwise percent-similarity between the manipulations. Normally this

[1]In our older papers [4], [5] we had also used an *overlapping* relation (O), which in general, however, is not needed and is omitted here.

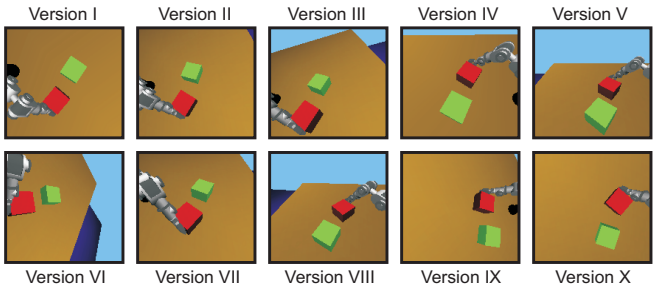

Fig. 4. Differences between the same moment of the pushing manipulation in all 10 different versions.

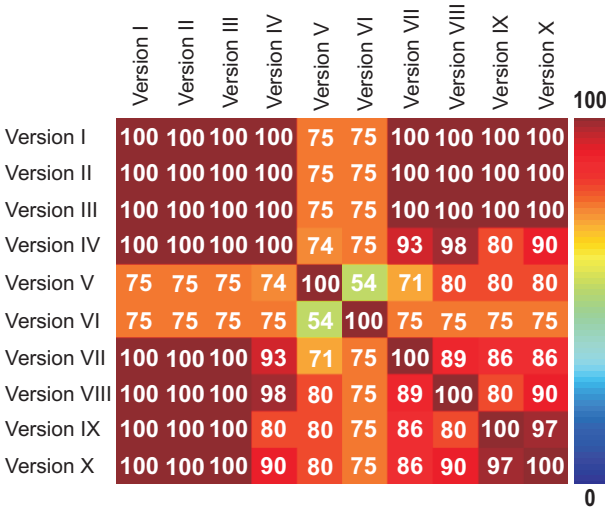

Fig. 5. Similarity values between 10 different versions of the pushing action.

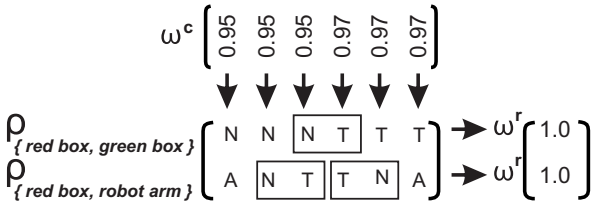

Fig. 6. The learned SEC model for the pushing action with corresponding normalized row (ω_i^r) and column (ω_i^c) weight values. Boxes indicate important transitions during this manipulation.

step is used to classify different manipulations; here we use it to show that indeed a high mutual similarity exists between those 10 repetitions (see Fig. 5 for the confusion matrix), where one outlier with only 54% similarity occurred due to some error in the image segmentation.

Having assured that the individual SECs represent indeed the same manipulation we are allowed to perform a weighted average and extract all re-occurring rows and columns in the ten SECs. The resulting SEC is shown in Fig. 6. Weights ω represent the normalized occurrence frequency of a given row or column and are sometimes less than 1.0 due to the situation that not all rows and columns are present all the time in the individual SECs. This is also visible by comparing model (Fig. 6) with the single SEC in Fig. 3. Thus, the model represents the archetype-SEC for this particular pushing action.

Details of all those steps (Fig. 1) can be found in our previous works [4], [5]. At this stage it is important to note that the start points of each temporal chunk, given by the time moments of the different columns in the model-SEC, represent *temporal anchor points* (Key Frames of the movie sequence). The SECs, thus, solve a difficult chunking problem in a natural way: By these anchor points the different motor primitives needed for a manipulation are defined. Furthermore, these moments are also decisive for defining the spatial anchor points at the objects, needed to define and actually execute an action.

C. Defining Spatial Anchor Points

The temporal anchors tell us "what happened when?", but they do not yet answer the question "how did it happen?". In the most general case, we would also have to know, (1) which objects are moved, (2) how the final spatial configuration (relative poses) of the objects looks like, and how the movement trajectories are shaped requiring (3) start- and endpoints and (4) trajectory shapes.

We will in the following section show that an analysis at the temporal anchor points, hence of the Key Movie Frames, suffices to extract components 1-3: 1) objects involved, 2) required poses, and 3) movement start and end-points, which define motion segments. Only when wanting information about (4) the complete movement trajectory we need to analyze also movie frames *between* the key frames.

To keep the algorithm general we assume that only one prime mover exists (usually the robot arm), bimanual manipulations need a different treatment. As the robot arm can only do "one thing at a time" we can in general state that for all possible manipulations the manipulation is started when the robot arm produces the first N-T (non-touching to touching) transition in the SEC and that it ends when the arm produces a T-N transition. Thus, we first have to find the prime mover by analyzing the image segments. Furthermore, it is evident that all other existing N-T transitions are decisive for the manipulation. Hence, we need to analyze those – one after the other – too. Somewhat more unusual, we will also make use of the fact that *continuous contact* of prime mover with another object effectively makes the other object a secondary mover allowing us to use vector addition in task space for defining the complete motion path. For example, as long as the robot arm remains in contact with the red object (unchanging T relation), we can neglect the robot arm and just consider the newly resulting spatial relations of red object with other

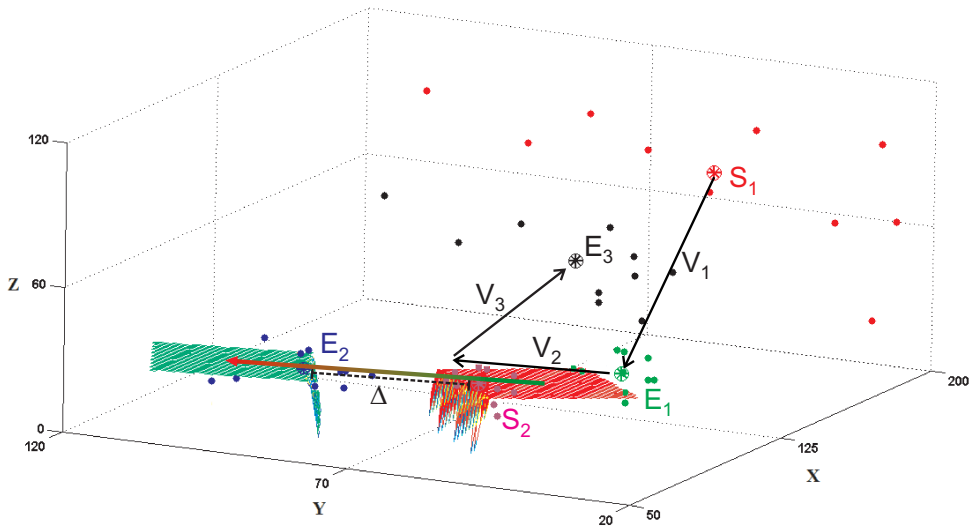

Fig. 7. Start (S_1, S_2) and end (E_1, E_2, E_3) point distribution as provided by the training dataset. Starred points are the average locations. Robot manipulator travels along the path given by vectors V_1, V_2, V_3. The coordinate origin is at the center of the red object. The motion vector for pushing is depicted by the color-changing vector that connects red with green object. Distance Δ is defines as $|E_2 - S_2|$ and vector V_2 has length Δ.

objects (here the green object) as spatial anchors[2]. Note, this reflects also the aspect of a robot using a tool. As long as the machine holds the tool, the robot's body is essentially extended and the tool defines now the end-effector of the machine [13].

Analyzing the Demonstration Examples: Let us first analyze the 10 demonstration pushes to see how the movement segments actually looked like. By D1-D8 we denote in the following those constraints which are used to actually define the movement segments for execution. As we do not need poses (ballistic push!), we did not implement any pose estimation steps.

To find the prime mover, we take the first N-T transition $(N_{2,2} - T_{2,3})$ in the model-SEC (Fig. 6). Here we use conventional row,column indices just for making it easier to find the entries in the SEC. Now we subtract the image segment configuration at key frame at $N_{2,2}$ from that at $T_{2,3}$, leading to a difference image only at the robot-arm image segment as the red box has not yet moved.

D1:*Thus, we obtain as prime mover the "robot arm segment".*

Then we consider the actual start points of the movement (red points in Fig. 7), which are widely distributed. The average is given by the starred red point.

D2: *Hence there are essentially no constraints on the starting point S_1 of the complete sequence.*

Next, we record at key frame at $T_{2,3}$ the coordinates of the red object at the touching point (see green points in Fig. 7).

D3: *This defines the endpoint E_1 for this specific motion segment V_1.*

We need to make sure that execution can cope with all kinds of different spatial configurations of robot arm and object. This requires defining a coordinate system which allows for such a generalization. To this end we use as the coordinate origin the segment center of the first touched object. This definition holds true for all conceivable basic single- or dual-object manipulation actions as *relations* between objects are decisive for the manipulation(s). Hence, we can always fix the origin on the first one touched and define coordinates relative to this[3], where we use any generic cartesian coordinate system just keeping it fixed for the remainder of the process.

D4: *Thus, the center of the first touched object defines the coordinate origin.*

The second found N-T transition concerns the red and the green object $(N_{1,3} - T_{1,4})$. As there is no change between the relation of robot arm and red object (relation remains $T_{2,4}$) we have indeed found a "secondary" mover (the red box) and a second touched object (the green box).

The segment center of the green object (the second touched object) defines together with the coordinate origin (center of red object) the so-called *dual object connection vector* (short: connection vector). Also this definition holds for all dual-object manipulations where a first object is supposed to make contact with a second object. In all these cases the first object must travel along a path (vector) that connects it to the second one. Clearly, in many dual-object manipulations additional difficult pose-constraints may arise, but the general connection vector will remain the same.

D5: *Thus, the connection vector is spanned between the*

[2]There might be some complicated manipulation actions where the arm (or hand) touches (or picks up) a second object before releasing the first. In this case, the argument about secondary mover would not hold. But such manipulations are uncommon and even for a human quite difficult. Hence, we do not consider them.

[3]Most basic, uni-manual manipulations are performed either at one object, leading to some configuration change at the object, or at two objects, where the first touched object is combined with the second one. Other manipulations, where more objects are directly involved are very rare (e.g. grasping two objects keeping both in the hand and combining them with a third one) or they can be considered as a chain of single- or dual-object manipulations.

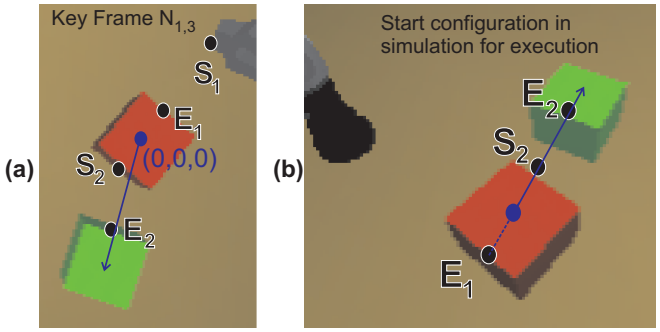

Fig. 8. Start and end points as given by demonstration (A), and as calculated for execution (B).

centers of the first and second touched object. Movement segment V_2 should follow the direction of this vector.

Now we need to define the path length. So far the definitions do not require any prior knowledge about the actual action to be performed. They hold for pushing as well as, for example, for pick-and-place actions. The fact that we want to perform a pushing action only comes in now: Similar to above, we record for key frame at $T_{1,4}$ the coordinates of the red *and* green objects at their touching point. They are shown back-projected onto the start frame in Fig. 7 (pink and blue). One can see that for a push, start and endpoints E_1, S_2, E_2 of the motion segments are roughly aligned with the connection vector (see Fig. 8 A and Fig. 7).

D6: *Points E_1, S_2, E_2 can be computed from the 3D-coordinates representing the intersection of the connection vector with the edges of the objects (Fig. 8 B).*

D7: *From this, we also note that the distance $\Delta = |E_2 - S_2|$ defines the length of the second motion segment V_2. Its direction is given by the connection vector.*

The core of the manipulation ends at the $T_{2,4} - N_{2,5}$ transition of robot arm with red object. The final homing motion of the robot arm, which follows thereafter, is not relevant for the manipulation and can be performed in any possible way. We look at the now following $N_{2,5} - A_{2,6}$ transition at the prime mover and plot the end points E_3 from $A_{2,6}$ (black points in Fig. 7) producing a set of actually observed final endpoints of the robot arm, which are also widely distributed.

D8:*Any endpoint for the motion can be used as long as the robot arm withdraws from the red object in a collision free way.*

D. Execution (Step 8)

The actual execution now is simple. For this, the model-SEC is used and every transition is treated like a rule. Constraints D1-D8 are attached to the transition rules as defined above.

For example the first N-T transition corresponds to a rule that demands that some movement by the prime mover should take place such that at the end the robot arm touches the red box and so on.

Thus, a new visual scene is presented to the system and segmented as usual. Robot arm, red box, and green box are recognized by their color or by any other object recognition algorithm. The model-SEC is split into its rules and the *actual* movement sequence is prepared by calculating the movement segments relative to the target objects. Start point S_1 is given by the momentary location of the robot arm. Again we use as coordinate origin the center of the red object and the connection vector points to the center of the green object. This origin- and vector-definition holds for all dual-object manipulations. For the specific purpose of pushing, and as explained above, the respective start and endpoints E_1, S_2, E_2 are now computed from the 3D-coordinates representing the cross-section of the connection vector with the edges of the objects (Fig. 8 B). The movement amplitude for the second N-T transition is in the same way given by $\Delta = |E_2 - S_2|$. We note that the touching point of the second object can be a bit over-estimated by this procedure if the diagonal of the object is aligned with the connection vector. In this case we will get a bit of a "push-second-object-away" when executing the action. This shows that pose estimation will at some point have to be added, too. Movement from S_1 to E_1 is defined by any collision free trajectory all other motion segments are straight. The last motion segment is a homing movement to any desired endpoint (E_3).

Without having to explain the details, execution now proceeds by following the N-T or T-N transitions from the model-SEC using conventional inverse kinematics for the Katana arm by vector addition of all motion segments until the sequence of motion segments has been consumed.

It is important to note the the robot has now immediately also a means to check whether the outcome of its actions are correct. After each movement of the arm, the machine needs to check the resulting relational changes between the image segments. These should match the changes in the model-SEC (see [5] for an example).

IV. Simulation Results

We let the agent realize the pushing action considering the learned model-SEC, and the motion segments as defined above. Fig. 9 (a-c) shows how the *robot arm* pushes the *red box* to the *green box* even if the object locations are different. In Fig. 9 (d-e) we used a red sphere and a bigger red box as pushable objects. In such cases the *robot arm* can still execute the manipulation. In Fig. 9 (f) a red cone and one more blue sphere were used. Finally, the robot arm chose and pushed the red conic to the bigger green box. All those simulation results, with corresponding depth, segment, and graph representations, can be found at http://www.dpi.physik.uni-goettingen.de/~eaksoye/movies.html. Finally we performed a self-check and Figure 10 shows the SEC obtained from the last execution example in Fig. 9, which is very similar to the model-SEC by which the robot can accept its own execution as correct. This is true for all executed examples.

These results show that with such a semantic representation the agent can learn and imitate a ballistic pushing manipulation independent of object shapes and positions.

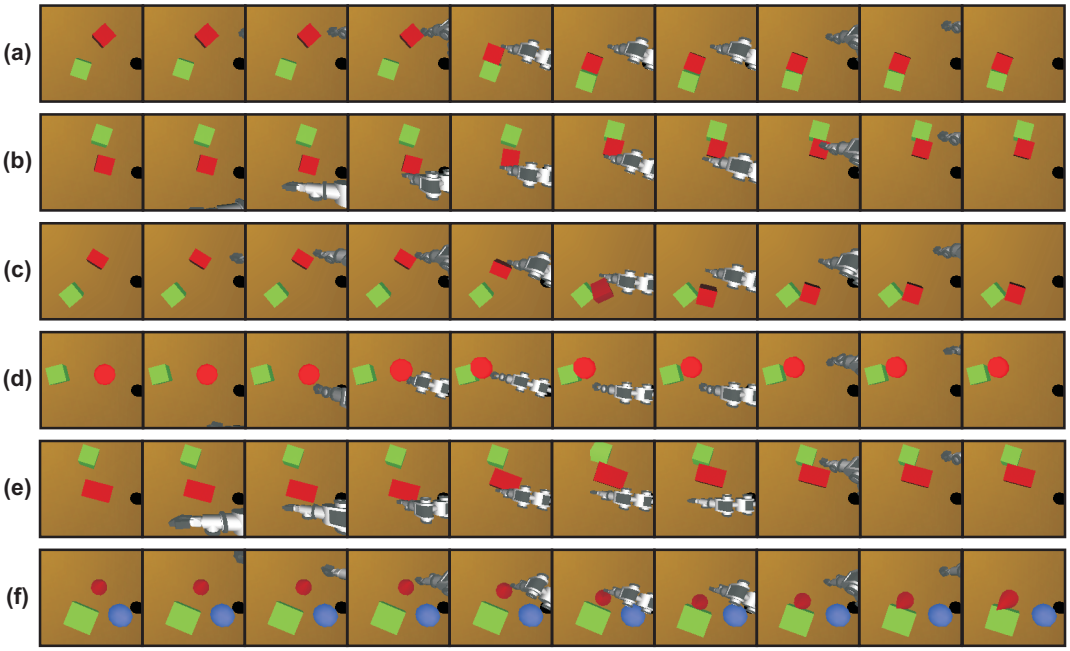

Fig. 9. Execution results. (a-c) *Robot arm* pushes the *red box* to the *green box* even if the object locations are different. (d-e) A red sphere and a bigger red box are used as pushable objects. (f) A red cone is used as a pushable object and one more blue sphere is added in the scene.

$$\begin{matrix}\rho_{\{\textit{red box, green box}\}}\\ \rho_{\{\textit{red box, robot arm}\}}\\ \rho_{\{\textit{robot arm, table}\}}\end{matrix}\begin{pmatrix}\mathrm{N} & \mathrm{N} & \mathrm{N} & \mathrm{T} & \mathrm{T} & \mathrm{T} & \mathrm{T}\\ \mathrm{A} & \mathrm{N} & \mathrm{T} & \mathrm{T} & \mathrm{N} & \mathrm{N} & \mathrm{A}\\ \mathrm{A} & \mathrm{N} & \mathrm{N} & \mathrm{N} & \mathrm{N} & \mathrm{T} & \mathrm{A}\end{pmatrix}$$

Fig. 10. SEC obtained from execution of the last example in Fig. 9.

V. Discussion

In this paper we have introduced a novel representation for the execution of manipulations by using the semantic event chains, which focuses on the spatial relations between objects in a scene. The representation generates column vectors in a matrix where every transition between neighboring vectors can be interpreted as an action rule, which defines which object relations have changed in the scene. In the first step, the approach learns from demonstration an archetypal event chain (model-SEC) consisting only of consistently repeated rows (spatial relations) and columns. Apart from the demonstration no other supervision is needed in this step, hence SECs are learned in a model-free way. In the second step, the learned rules are enriched by determining the movement segments by which the manipulation can be executed regardless of the configuration of the objects in a scene. Execution can then follow the enriched SEC rules and the robot can test its own success by checking the SEC, which results from execution, against the model-SEC.

To our knowledge this is the first approach which uses a learnt abstract symbolic representation for manipulation learning, execution, and self-recognition. We are aware of the fact that the algorithm uses some simplifications such as no object dynamics and pose estimations. Due to this fact, in some cases it was observed that the object could not be pushed in the desired direction, because of wrong object and/or gripper poses and the frictional restrictions both on the background and object surface.

It is important to note that this procedure can be macronized for all *dual-object manipulations*. In a very abbreviated form the instructions for such a macro would read:

1) `Identify prime mover.`
2) `Identify first touched object by first N-T transition and set coordinate origin.`
3) `Define first motion segment`
4) `(Extract relative poses between prime mover and first object, if needed).`
5) `Identify second touched object and fix connection vector and coordinate system.`
6) `Define second motion segment for second N-T transition relative to this coordinate system. (For pushing do this by cross-sectioning with object borders).`
7) `(Extract relative poses between objects involved, if needed).`
8) `Define third motion segment (home).`

Such a macro can be enriched by adding pose information from a pose estimation algorithm where required. This would be needed for a pick&place manipulation (which is also a dual object manipulation), where the final resulting relative pose of the two combined objects is most of the time important. Aspects of grasping an object (e.g. grasp preparation and the performing of a grasp) are not considered at all in this

Fig. 11. SEC for the action "cutting a carrot with a knife".

framework. Grasping is a very difficult technical problem but for manipulation actions it takes usually just a preparatory role. We do not wish to downgrade the importance of this role but the actual outcome of the manipulation is in most cases only in a secondary way affected by the way an object is grasped. Clearly, if the grasp is totally unsuitable a pick&place action will fail. But these considerations must happen before the first N-T transition in the SEC and are not part of this paper.

Note, the presented approach can also be extended to more general manipulation tasks, e.g. "cutting a carrot with a knife". In such a SEC we would observe more relational changes compared to the pushing example as the newly cut carrot pieces have to be also included into the representation. The SEC for cutting two pieces off the carrot is given in Fig. 11.

For execution purposes we again have to analyze N-T and T-N transitions, which in given cutting scenario will emerge only between *hand and knife*, *knife and carrot*, and *knife and the two new appearing pieces*. Therefore, fourth, sixth, and seventh rows of the SEC given in Fig. 11 can be ignored for clarity. In addition to the discussed pushing example, for the cutting scenario grasping of a knife will be required, where the SEC will provide the temporal anchor point for extracting relative pose between *hand and knife* at the hand-knife N-T transition. The relative pose between *knife and carrot* will be important at the transitions N-T in the relation knife-carrot. Also movement trajectory information for the cutting (actually, sawing) movement should be attached to the SEC between the N-T and T-N transitions in the relation knife-carrot, which will then be used in execution. This shows that the same anchoring process can also be performed for other actions. Clearly, this involves very difficult technical steps of pose estimation and trajectory shaping.

VI. Conclusion

In the current study we have shown how a learnt SEC can be used to "push one object against another one." We tried to make clear that this example generalizes to all dual-object actions, where for different instantiations different pieces of information (e.g. addition pose information, etc.) have to be stored. The anchor points for this information, however, exist as demonstrated in the current study and the inversion of an event chain is possible. Work has started to address this problem for a large ontology of manipulation actions [14].

Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 (Specific Programme Cooperation, Theme 3, Information and Communication Technologies) under grant agreement no. 270273, Xperience and grant agreement no. 269959, IntellAct. B.D. acknowledges support from the Spanish Ministry for Science and Innovation via a Ramon y Cajal Fellowship.

References

[1] A. Billard, S. Calinon, and F. Guenter, "Discriminative and adaptive imitation in uni-manual and bi-manual tasks," *Robot. Auton. Syst.*, vol. 54, pp. 370–384, 2006.

[2] T. Asfour, P. Azad, F. Gyarfas, and R. Dillmann, "Imitation learning of dual-arm manipulation tasks in humanoid robots," *Int. J. Hum. Robot.*, vol. 5, pp. 183–202, 2008.

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

[4] 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, ICRA2010 Alaska, USA*, 2010.

[5] 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 (IJRR), Special Issue on 'Semantic Perception for Robots in Indoor Environments' (In press)*, 2011.

[6] L. Guoliang, N. Bergström, C. H. Ek, and D. Kragic, "Representing actions with kernels," in *IEEE/RSJ International Conference on Intelligent Robots ans Systems (IROS)*, 2011, (To appear).

[7] K. Lynch and M. Mason, "Stable pushing: Mechanics, controllability, and planning," in *Algorithmic Foundations of Robotics*. Boston, MA: A. K. Peters, 1995, pp. 239–262.

[8] Q. Li and S. Payandeh, "Manipulation of convex objects via two-agent point-contact push," *Int. J. Rob. Res.*, vol. 26, pp. 377–403, April 2007. [Online]. Available: http://portal.acm.org/citation.cfm?id=1274664.1274673

[9] P. Fitzpatrick, G. Metta, L. Natale, S. Rao, G. Sandini, and G. S, "Learning about objects through action - initial steps towards artificial cognition," in *In Proceedings of the 2003 IEEE International Conference on Robotics and Automation (ICRA*, 2003, pp. 3140–3145.

[10] D. Omrcen, C. B. T. Asfour, A. Ude, and R. Dillmann, "Autonomous acquisition of pushing actions to support object grasping with a humanoid robot," in *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, Paris, France, 2009.

[11] M. Salganicoff, G. Metta, A. Oddera, and G. Sandini, "A vision-based learning method for pushing manipulation," in *In AAAI Fall Symposium Series: Machine Learning in Vision: What Why and*, 1993.

[12] M. F. Sumsi, "Theory and algorithms on the median graph. application to graph-based classification and clustering," Ph.D. dissertation, Universitat Autonoma de Barcelona, 2008.

[13] F. Wörgötter, A. Agostini, N. Krüger, N. Shylo, and B. Porr, "Cognitive agents - a procedural perspective relying on "predictability" of object-action complexes (oacs)." *Robotics and Autonomous Systems*, vol. 57(4), pp. 420–432, 2009.

[14] F. Wörgötter, E. E. Aksoy, N. Krüger, J. Piater, A. Ude, and M. Tamosiunaite, *Robotics and Autonomous Systems, (submitted).*