Single-Shot Learning and Scheduled Execution of Behaviors for a Robotic Manipulator

Christian Groth, Universität Bayreuth, christian.groth@uni-bayreuth.de, Germany
Dominik Henrich, Universität Bayreuth, dominik.henrich@uni-bayreuth.de, Germany

Abstract

In order to introduce robots into the household domain, they must be easy to program and be able to perform various tasks without being reprogrammed every time. In this paper, we address both issues. First, we present a behavior-based architecture, which is capable of storing, executing, and switching between various manipulation tasks of a robotic manipulator. Second, we present a single-shot learning from demonstration, to allow the user to extend the set of known behaviors by guiding the robot through new tasks.

1 Introduction and Related Work

To make robots take over useful work in the household domain, we are facing two issues. The first issue is, that a robot must be able to perform a lot of different tasks without the need of reprogramming. Since traditional programming is very time consuming, the robot should inherit some long-time memory and choose the appropriate action in new situations. This is addressed by behavior-based architectures.
The second issue is, that traditional robot programming is tedious and only feasible by experts. Therefore, there is a strong need for concepts to easily program a robot, in order to extend the set of executable tasks. This is addressed by the Programming by Demonstration (PbD) approach, which allows even non-experts to teach robots by guiding the robot through the task.
Regarding the first issue, we can distinguish two groups of behavior-based architectures for robotic manipulators, through which coordination of multiple behaviors is achieved. One uses action fusion, the other action selection. In the action fusion group, many behaviors are necessary to perform one specific manipulation task. Examples are the CBFM [1] or fuzzy-based approaches [2],[3] or force-based approaches [4], [5].
In the action selection group, a task can be performed by a single but complex behavior. The most well-known approach in this category is the subsumption architecture [6], on which several approaches rely [7] [8].
All of these systems have their drawbacks. None of the known approaches addresses the problem of several manipulating behaviors, which can safely be interrupted and resumed, so that the behaviors may be completed consistently later on. In this paper, we present a novel approach, where a behavior-based manipulator performs various tasks. All behaviors can be interrupted, resumed, and completed safely. We achieve this by transferring the concept of *Concurrent Sequential Processes* (CSP) from the computing domain to robotics.
Considering the issue of teaching new behaviors to the robot, there are several ways in PbD to extract relevant features of the task, in order to generalize a demonstration to a new situation. It depends on the number of demonstrations and on the level at which the task is learned. The demonstration can be provided on a symbolic level or on a trajectory level. On the symbolic level, predefined primitives are used like (dynamic) movement primitives or skills, which act as building blocks for complex tasks. The challenge is to identify these primitives and to find appropriate parameters. This has been investigated for just one demonstration in [9] [10] and [11]. When multiple demonstrations are available, the recognition of primitives may be improved and even alternative task executions can be provided [12].
On a trajectory level, information of given demonstrations is extracted and a generalized version of the trajectory under new conditions is reproduced.

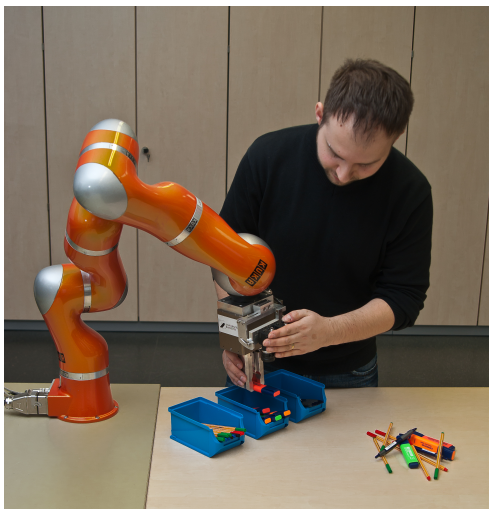

Figure 1: Demonstration of new behaviors to the system by kinesthetic teaching

A good survey on approaches using multiple demonstrations is provided by [13]. More recent research can be found in [14] or [15]. The few approaches that do learning from a single demonstration on trajectory level rely on strong context knowledge [16] or on additional infor-

mation [17] [18], or on manual post-processing of the trajectory [19]. Since we want to provide as little information as possible, we concentrate on a single shot learning on a trajectory level.
To sum up, in this work we will present a behavior-based approach for robotic manipulators, which allows concurrent execution and consistent switching between manipulation tasks. Every task is fulfilled by a single behavior. New behaviors can be added to the system by a single demonstration of the task.
The remainder of this paper is organized as follows: Our behavior-based archtitecture is described in Section 2. In Section 3, we describe, how new behaviors are taught to the system. In Section 4, we evaluate our system and conclude in Section 5.

2 Behavior-based manipulation

2.1 Behavior Model

Our approach consists of various behaviors running in parallel on a robot manipulator and performing different tasks. The focus lies on a mechanism to switch between multiple behaviors. The behaviors need to be in a consistent state, even if they are interrupted or resumed at a later time.
The behaviors react according to stimulation from the environment and the inner behavior status. In every time step the robot determines the behavior, which fits best to the current environmental situation. Each behavior is modeled by a Mealy Machine $B = [S, C, A, \delta, \omega, s_0, s_F]$, where S denotes a set of states, including the initial state s_0 and the final states s_F. A denotes a set of actions and C a set of conditions. The transition function δ is given as $\delta : S \times C \rightarrow S$ and the output function ω is given as $\omega : S \times C \rightarrow A$. For a behavior to change state from s_i to s_j, the conditions c_{ik} with $0 \leq k \leq n$ at a transition $T(s_i, s_j)$ have to match corresponding resources $r_{ik} \in R$, which are provided by the *observers*. The resources R of the system contain all stimuli from the environment Γ, like objects in the scene and all executing components ζ of the robot, like the arm or the tool. A condition $c \in C$ specifies a resource type, like a specific object or a robotic component and additional features of the resource. Therefore we can define a matching function $match(c, r)$, that evaluates to `true` if and only if a resource r satisfies all required features of c. If the condition is matched by a resource, the corresponding action of the transition is executed.
The action $a \in A$ depends on the the matching resources and the current robot configuration. Usually it is a robot trajectory T_k with additional tool commands and a frame f_k, to which the trajectory refers. After the action is completed, the current state is changed.
To enable the achievement of different goals, several competing behaviors are executed on the robot. Many of these behaviors have conflicting goals, which can not just be summed up. So we need a mechanism for consistent behavior execution.

2.2 Behavior Execution

To enable safe interruption, change, and resume of the active behavior, even within the execution of an action, we apply methods taken from modern operating systems. These are well known concepts within the computing domain and they are able to handle multiple processes on single-core platforms. This is analogous to one robot executing multiple behaviors. To do so, all behaviors are wrapped into processes [20]. I.e., we look at the former defined behavior as a program and execute one or more instances of it as processes. The processes are managed by four different lists, according to their process status: `ready`, `blocked`, `active`, `terminated`.
The resources $R = \Gamma \cup \zeta$ can be divided into two subsets $R = R_{np} \cup R_p$ with $R_{np} \cap R_p = \emptyset$. Here, R_p denotes all preemptive resources and R_{np} all non-preemptive. Preemptive resources can be withdrawn from a process and can be assigned to another process, like the robot or the gripper. Non-preemptive resources cannot be withdrawn from a process, like real-world objects, that are manipulated by the robot, so a former state may not be restorable. We define the set of resources held by process i as H_i and the set of free resources as V. Furthermore, there is a set of resources $N_{i,k} \subseteq R$, which is needed by process i to execute the action a of its current transition with condition c_k. The set $N_{i,k}$ is defined by the matching function of the conditions c_k with $k = 0 \ldots n$ of a transition. We define $\hat{H}_{i,k} \subseteq H_i$ as the subset of H_i through $H_i \xrightarrow{match(c_k, r_j), a} \hat{H}_{i,k}$. This means $\hat{H}_{i,k}$ consists of all resources r_j that satisfy the condition c_k and are held by process i. Further, we define analogous $\hat{V}_k \subseteq V$ as the subset of V though $V \xrightarrow{match(c_k, r_j), a} \hat{V}_k$, which consists of all free resources that satisfy the condition c_k. Now we can define $N_{i,k}$ as $N_{i,k} = \hat{V}_k \cup \hat{H}_{i,k}$.
Since there are n conditions for a transition, there may be some identical conditions. Imagine an action of a transition that requires two identical objects. Therefore we can group the identical conditions of a transition. Let the transition have groups G_g containing m_g similar conditions c_g. Since all conditions in a group are equal, any element c_g of a group is representative for the group. For every group G_g let $H_{i,g}$ be the set of the needed and already held matching resources. Also let V_g be the needed matching and free resources . Using these sets we can determine the status of a process.
A process is `active`, if the equation

$$m_g \leq |\hat{H}_{i,g}| + |\hat{V}_g| \tag{1}$$

for every group G_g, that contain m_g conditions of type c_g is satisfied. It means, that there are enough resources to satisfy every condition of every group of the current transition. The resources can either be already held ($\hat{H}_{i,g}$) or still available ($\hat{V}_g$). Since all conditions are satisfied, the process can immediately acquire all necessary free resources and can be executed.

A process is `ready`, if the equation

$$m_g \leq |\hat{H}_{i,g}| + |\hat{V}_g| + |\bigcup_{h \neq i} \hat{H}_{h,g} \cap R_p| \qquad (2)$$

for every group G_g, that contain m_g conditions of type c_g is satisfied. This means, we can match the conditions with already held resources ($\hat{H}_{i,g}$) and free resources ($\hat{V}_g$), like in Equation 1. But we would need to withdraw preemptive resources $\hat{H}_{h,g}$ from other processes.
A process is `blocked`, if neither Equation 1 nor Equation 2 can be satisfied.
A process is `terminated` if the current state s_c of the inherent behavior has reached the end state ($s_c = s_F$).
A scheduling algorithm can be used to choose which of the ready processes is executed. We evaluate different scheduling algorithms in Section 4.

2.3 Behavior Coordination

If an active process at time step $t-1$ is also active in time step t the current action is kept on execution. If the active process changes, we need to store the context of this process and restore the new process context. This means, we have to store the status of all resources H_i, that are held by process i in the process context.
This is straightforward for all kind of information and intrinsic resources, like the robot configuration, which can easily be stored. Remember that non-preemptive resources hold locks. So even if manipulated objects stay in the working area, no other process can use these. Preemptive resources may be withdrawn from the process. The preemption of manipulating devices is more expensive since they must be fully available for other processes. For example, if an object is held within the gripper, it must be stored reliably, so it can be restored later. Therefore special areas are provided, where objects can be deployed. The corresponding deploy position is also stored.
The context restoring of a process is done in reverse order. First eventually deployed objects are grasped again and the last pose is restored. If there is an incomplete action, it is resumed and execution is continued. If resources have been withdrawn from the process, then resources matching the corresponding condition within the state machine are acquired again.
To allow a temporal coordination of behaviors, we need to know, which behaviors were already applied to a resource. Therefore we extend the resource's feature vector. We store which behaviors were already applied and how often they have been applied. Now we can easily add already applied behaviors as required to a condition c and we can also define an upper limit of how often a behavior can be applied to a resource.

3 Programming by Demonstration

3.1 Trajectory Segmentation

To extend the set of known behaviors in a fast and easy manner, we provide these through programming by demonstration (Figure 1). The robot shall execute a generalized version of a task in a new situation under new constraints. The user provides just one demonstration for every behavior, which is adapted to the current situation and a generalized motion is generated. The motion should meet the constraints of the current situation and have similarity with the corresponding demonstration.
In this work a demonstration consists of a 6-dimensional trajectory $T = \{t_0, \ldots, t_{n-1}\}$ with $n \in \mathbb{N}$ of the robot's tool center point t_i in 3d space and a set of reference frames $F = \{f_0, \ldots, f_{m-1}\}$ with $m \in \mathbb{N}$. A sample $t_i \in T$ can carry additional tool operations, like opening or closing the gripper. A reference frame f_i is defined by each object's initial position in the workspace and the initial position of the robot's tool center point. It consists at least of a spatial coordinate system and may include further information such as time.
In order to generalize a demonstration to a new situation, we need a mapping of every sample $t_i \in T$ to the set of relevant frames $\tilde{F}_i$:

$$\forall t_i \; \exists \; \tilde{F}_i \subseteq F$$

We realize this mapping via a $m \times n$ weight matrix W where each matrix element w_{ij} carries the weight of a frame f_i with respect to a sample t_j. Thus, W provides the relevance of each frame w.r.t. every sample. Since we only take a single demonstration into account, we reduce the problem to

$$\forall_j : t_j \rightarrow f_i.$$

This will divide the trajectory into segments where every segment is exclusively related to exactly one reference frame.
We calculate the weight of a reference frame f_i with respect to a sample t_j by means of a distance function $dist(f_i, t_j) \geq 0$. The distance function may be based on spatial features like the Euclidean distance between t_j and f_i, temporal features, or even high-level features like instructions by speech or social cues.
We consider reference frames with smaller distances to be more important. Therefore we define a $m \times n$ reference matrix R with elements r_{ij} by three conditions each:

$$r_{ij} = \frac{1}{1 + dist(f_i, t_j)}$$

Using R we define a $m \times n$ segmentation matrix U with elements u_{ij} by

$$u_{ij} = \begin{cases} 1, & \text{if } r_{ij} = \min\limits_k(r_{kj}) \text{ and } r_{ij} < d_{i,\max} \\ 0, & \text{otherwise} \end{cases}$$

We assume $r_{ij} \neq r_{kj}$ for $i \neq k$. For the unlikely case of an identical distance of a sample to different frames, we

select at random. The threshold $d_{i,\max}$ discards the influence of a too distant frame f_i. This means, some of the samples cannot be mapped to a frame. Therefore, some columns will only consist of zeros. We assume that these trajectory samples belong to transfer motions between the preceding and the subsequent reference frame. Consequently, we have to assign these samples to the preceding and subsequent frame by adjusting the matrix elements which refer to these frames. Based on U, we can finally define W. For every c we define the column w_c of W by

$$w_c = \begin{cases} u_c, & \text{if } \sum_{i=0}^{m-1} u_{ic} = 1 \\ 0.5 \cdot u_p + 0.5 \cdot u_s, & \text{otherwise} \end{cases}$$

Here, u_p denotes the nearest preceding column and u_s the nearest subsequent column. These columns are defined by

$$\sum_{i=0}^{m-1} u_{ip} = 1, \quad |p - c| = \min, \quad p < c$$

and

$$\sum_{i=0}^{m-1} u_{is} = 1, \quad |s - c| = \min, \quad s > c$$

The columns u_p and u_s determine the relevant frames for the transfer. If there is no subsequent frame, the samples are assigned to the last referenced frame, since it could be some detach-movement from an object.

Since we assume that we can assign every sample to one reference frame, we will introduce *transfer frames*, which are calculated from the subsequent and preceding reference frame. The construction of these transfer frames will be shown in Section 3.2.

After applying these steps, we receive a segmented trajectory consisting of segments related to existing frames alternated with transfer segments (Figure 2). The generalization of the segments is described in the next sections.

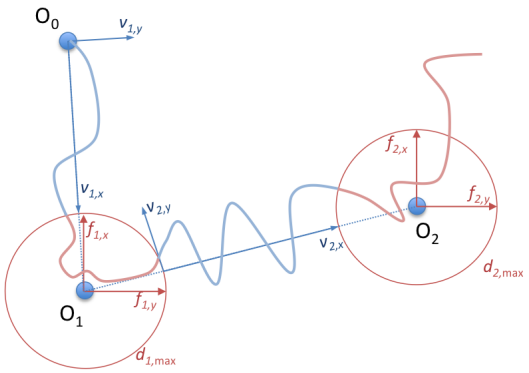

Figure 2: 2D example of a segmented trajectory into transfer frames (blue) and object frames (red).

3.2 Motion generalization

All samples within the distance threshold are assigned to an object frame f_i. As stated, every frame holds at least a spatial coordinate system. We use the coordinate system assigned to the manipulated object, which is unambiguously by construction. The system should reproduce the trajectory segments associated to objects very exact, since the object could be manipulated by this motion.

All samples that are not mapped to object frames, are mapped to transfer frames. We denote the set of all transfer frames as V. The reproduction of the transfer motion must satisfy the constraints of the new situation. Since a transfer motion depends on the preceding and subsequent frame, we calculate the transfer frame from the origins O_p and O_s of these frames. We also make use of the intersection $p_{\text{cut},s}$ and $p_{\text{cut},p}$ of the object frames' hulls, defined by $d_{i,\max}$, and the connecting line between O_p and O_s. The transfer frame is an orthogonal right-hand coordinate system x_V, y_V, z_V with its origin at $p_{\text{cut},p}$ and calculated as

$$x_\text{V} = p_{\text{cut},s} - p_{\text{cut},p}$$
$$x_\text{V} \perp y_\text{V}, x_\text{V} \perp z_\text{V}, y_\text{V} \perp z_\text{V}$$
$$\|y_\text{V}\| = 1, \|z_\text{V}\| = 1$$
$$\angle(z_\text{V}, g) = \min$$

where g is the negative gravity vector pointing up.

After having calculated the object and transfer frames, we transform each sample t_j of the demonstrated trajectory T into its frame. First we separate the trajectory into the segments of points $\langle T_k \rangle_{\text{frame}}$ by $T = \langle T_0 \rangle_{\text{world}} \langle T_1 \rangle_{\text{world}} \dots \langle T_{n-1} \rangle_{\text{world}}$. I.e. all points remain relative to the world coordinate system. Each segment T_k represents the membership to the same frame, which is provided by the weight matrix W.

Now we can transform the points into the object frames F and transfer frames V. We use the transformations ${}^{f_i}M_{\text{world}}$ for each frame $f_i \in F \cup V$ and denote the new transformed segments T_k as

$$\langle T_k \rangle_{f_i} = {}^{f_i}M_{\text{world}} \langle T_k \rangle_{\text{world}} .$$

In the reproduction phase we extract the reference frames F^r for all objects that correspond to the objects of the demonstration by means of a matching function, which is described in [21]. The transfer frames V^r are created analogous to the demonstration phase. Afterwards we extract the transformations ${}^{F^\text{r}}M_{\text{world}}$ for all frames F^r analogous. With this, we can reproduce the trajectory in the new situation by transforming the segments T_k from the frames into the world frame by

$$\langle T_k \rangle_{\text{world}} = {}^{f_i^\text{r}}M_{\text{world}}{}^{-1} \langle T_k \rangle_{f_i}$$

with $f_i^\text{r} \in F^\text{r} \cup V^\text{r}$.

Regarding our behavior based system from Section 2, an action $a \in A$ of a state stores the current subtrajectory with tool commands in the corresponding frame ($a_k = \langle T_k \rangle_{\text{f}_\text{k}}$). As transition condition c, we take the objects from the demonstration which were used to construct the corresponding frame ($c_k = \{f_k, [f_{k+1}]\}$). If it is an object frame, we set the second condition $[f_{k+1}]$ to the empty condition ε. We also define all needed robot resources, like arm or tool as a condition to this transition.

When adapting the demonstrated trajectory to a new situation, we may encounter some harsh changes due to strongly rotated or displaced objects. To generate smooth trajectories, we blend trajectories between the transfer frame and the object frames by means of a sigmoid function. Details can be found in [21].

4 Experimental Results

We evaluated our system with a Kuka LWR 4 mounted on top of a table. The objects are detected by a Microsoft Kinect using the Alvar Library. The user guides the robot through the task. The trajectory is recorded and stored along with the information about the objects. In the reproduction phase, the information is loaded again and the corresponding behaviors are generated. An example is depicted in Figure 3. But in particular, we investigated how efficient the scheduling algorithms from the operating systems domain are for robotic applications. The implemented scheduling algorithms are: priority scheduling (PS), shortest-job-first (SJF), and round-robin (RR). We regard the length of the trajectory of a task as the job length. Shortes-job-first is the only non-preemptive algorithm. For the round-robin algorithm we chose time slices of 20, 40, 60, 80 and 160 seconds.
We tested the scheduling algorithms with three pick-and-place behaviors in two settings L within a simulation environment. All behaviors were programmed by demonstration. In the first setting, enough resources to create two processes of every behavior are present from the beginning. In the second setting, the same resources are added one by one every 20 seconds. The priorities for PS are different for each behavior.

E	Scheduling	$\bar{t}_{\text{flow}}$(s)	$\bar{t}_{\text{wait}}$(s)	t_{total}(s)	n_{sw}
1	PS	238.2	152.0	1429.0	5
1	SJF	211.8	109.3	1271.0	5
1	RR(20)	360.7	44.4	2164.2	32
1	RR(40)	304.3	82.2	1825.9	15
1	RR(60)	284.4	122.1	1706.5	11
1	RR(80)	268.0	160.9	1608.2	9
1	RR(160)	261.4	179.9	1568.4	5
2	PS	239.8	87.3	1438.7	7
2	SJF	198.8	91.1	1192.7	5
2	RR(20)	319.3	17.7	1916.0	29
2	RR(40)	257.6	24.4	1545.8	16
2	RR(60)	235.8	57.8	1415.1	12
2	RR(80)	223.6	89.5	1341.4	10
2	RR(160)	178.8	106.2	1072.9	5

The results are shown in the table above. For each process we have measured: the average flow time $\bar{t}_{\text{flow}}$, the average duration $\bar{t}_{\text{wait}}$ between being `ready` and `active` for the first time, and the number n_{sw} of context switches. We have also measured the total execution time t_{total}, which is the time between the first process becoming `ready` and the last process terminating. Although all time measurements are affected by variations due to the generalization of the learned tasks, it is still obvious that round-robin results the longest flow and total execution times. In robotic applications, context switching is even more expensive than in operating systems. The tasks are executed by a robot, that can only move with a certain velocity limiting the speed for context changing. Additionally, objects may have to be grasped or deployed, resulting in an even more expensive context switch. With longer time slices, $\bar{t}_{\text{flow}}$ is reduced. At a length of 160s each process fits into a time slice, what results in the minimum of 5 context switches. The drawback of longer time slices is a slower reaction to new objects, which is indicated by a higher $\bar{t}_{\text{wait}}$. This can also be observed in the priority scheduling, when long tasks with high priority are present. Considering $\bar{t}_{\text{flow}}$ and $\bar{t}_{\text{wait}}$, shortest-job-first is a viable alternative to priority scheduling. Unfortunately, it is non-preemptive and, thus, short tasks may be heavily delayed due to long running task (starvation).

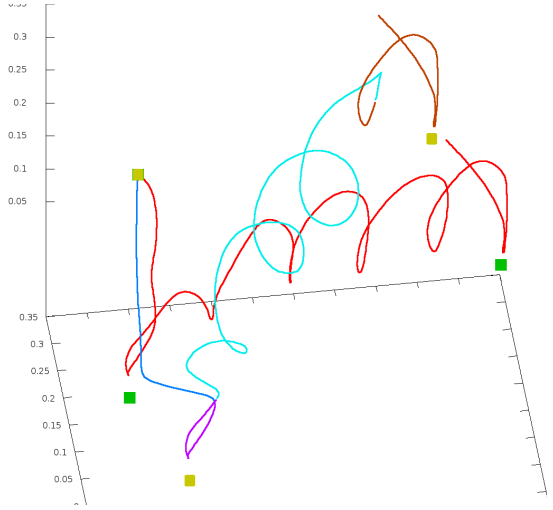

Figure 3: A pick-and-place task with a spiral transfer motion: The demonstration with trajectory (red) and objects (green) and the reproduction with trajectory (blue) and objects (yellow), both starting at the same position.

In conclusion, one can say, that context switches should occur as rarely as possible, but often enough to react to new stimuli from the environment. Therefore priority scheduling seems to be the best choice for robotic applications in the household domain. The system can immediately react to new and more important stimuli, but will not interrupt current tasks for less important ones. In our learning from demonstration context, the choice of the behavior priorities is still an open issue. Shortest-job-first can be an alternative for small enterprises, where e.g. the robot has to assemble batches of different parts. Choosing the shortest-job-first minimizes the average flow time.

5 Conclusions

In this work, we presented a behavior-based system for a robotic manipulator. All behaviors can be interrupted and resumed consistently, even if they are changing the environment. Furthermore, we presented a single-shot ap-

proach for learning by demonstration on trajectory level. Each demonstrated task can be added to the system as a new behavior to extend the set of known behaviors.
In our experiments we investigated the efficiency of scheduling algorithms for robotic applications. The results show that shortest-job-first and priority scheduling are simple yet promising choices for robotic applications in the household and small and medium enterprises.
Future work should improve the mapping between samples and reference frames. Also, the approach to map a sample to multiple reference frames should be extended. The integration of preview coordination for resources [22] may also be a possibility to improve scheduling by minimizing the number of context changes.

References

[1] S. Huang, E. Aertbeliën, and H. V. Brussel, "A Constraint-Based Behavior Fusion Mechanism on Mobile Manipulator," in *2008 ECSIS Symposium on Learning and Adaptive Behaviors for Robotic Systems (LAB-RS)*. IEEE, Aug. 2008, pp. 83–88.

[2] Z. Wasik and A. Safiotti, "A Hierarchical Behavior-Based Approach to Manipulation Tasks," in *2003 IEEE International Conference on Robotics and Automation*, Taipei, 2003, pp. 2780–2785.

[3] P. Dassanayake, K. Watanabe, K. Kiguchi, and K. Izumi, "Robot manipulator task control with obstacle avoidance using fuzzy behavior-based strategy," *Intelligent and Fuzzy Systems*, vol. 10, no. 3, pp. 139–158, 2001.

[4] A. C. Smith, E. Rafael, and T. Jara, "Sensitive Manipulation," Ph.D. Thesis, Massachusetts Institute of Technology, 2007.

[5] N.-H. Park, Y. Oh, and S.-R. Oh, "Behavior-based control of robotic hand by tactile servoing," *International Journal of Applied Electromagnetics and Mechanics*, vol. 24, no. 3-4, pp. 311–321, 2006.

[6] R. Brooks, "Intelligence without Representation," *Artificial Intelligence*, vol. 47, pp. 139–159, 1991.

[7] A. Edsinger and C. C. Kemp, "Two Arms are Better than One : A Behavior Based Control System for Assistive Bimanual Manipulation," *Artificial Intelligence*, pp. 345–355, 2008.

[8] T. Taipalus, "An Action Pool Architecture for Multitasking Service Robots with Interdependent Resources," in *Computational Intelligence in Robotics and Automation*, Piscataway, NJ, USA: IEEE Press, 2009.

[9] R. Dillmann, O. Rogalla, M. Ehrenmann, R. Zollner, and M. Bordegoni, "Learning from Human Demonstration and Advice: the machine learning paradigm," *Robotics Research*, vol. 9, pp. 229—-238, 2000.

[10] R. Zollner and M. Pardowitz, "Towards cognitive robots: Building hierarchical task representations of manipulations from human demonstration," *International Conference on Robotics and Automation (ICRA)*, 2005.

[11] R. Zollner, T. Asfour, and R. Dillmann, "Programming by demonstration: dual-arm manipulation tasks for humanoid robots," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 479–484, 2004.

[12] M. Nicolescu and M. Mataric, "Natural methods for robot task learning: Instructive demonstrations, generalization and practice," *International joint conference on Autonomous agents and multiagent systems*, 2003.

[13] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, May 2009.

[14] D. H. Grollman and O. C. Jenkins, "Incremental learning of subtasks from unsegmented demonstration," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 261–266, Oct. 2010.

[15] L. Rozo, S. Calinon, D. Caldwell, P. Jim, C. Torras, and I. D. Rob, "Learning Collaborative Impedance-based Robot Behaviors," *Proceedings of the AAAI Conference on Artificial Intelligence.*, 2013.

[16] A. Dey, R. Hamid, and C. Beckmann, "A CAPpella: programming by demonstration of context-aware applications," *Proceedings of the SIGCHI conference on Human factors in computing systems*, vol. 6, pp. 33–40, 2004.

[17] S. Iba, C. Paredis, and P. Khosla, "Interactive multimodal robot programming," *Robotics Research*, vol. 24, no. 1, pp. 83–104, 2005.

[18] P. E. Rybski, K. Yoon, J. Stolarz, and M. M. Veloso, "Interactive Robot Task Training through Dialog and Demonstration," *Forbes*, 2007.

[19] H. Mayer, I. Nagy, and A. Knoll, "Adaptive control for human-robot skilltransfer: Trajectory planning based on fluid dynamics," *Robotics and Automation*, pp. 10–14, 2007.

[20] A. S. Tanenbaum, *Modern Operating Systems*. Upper Saddle River, NJ, USA: Prentice Hall Press, 2007.

[21] C. Groth and D. Henrich, "Multi-Tasking of Competing Behaviors on a Robot Manipulator," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[22] E. Scioni et. al., "Preview coordination: An enhanced execution model for online scheduling of mobile manipulation tasks," in *IEEE/RSJ Intelligent Robots and Systems (IROS)*, 2013.