A ROS-integrated architecture to learn manipulation tasks from a single demonstration

[illegible]

Lorenzo Peppoloni, Alessandro Di Fava, Emanuele Ruffaldi and Carlo Alberto Avizzano

Abstract— **In the robot programming by demonstration (PbD) framework, the high-level representation of a skill in a series of action units gives an intuitive method to program and instruct robot behaviors. In this context we present a ROS (Robot Operating System) integrated architecture for learning households manipulation tasks by one observation. The user is observed during the execution of everyday tasks, every action is analyzed and its effect is translated into changes in the environment state. During the observation a strip-like map of the task is built and stored as a sequence of actions. From the map obtained the task can be performed. A planner robustly adapts the execution both to different environment initial conditions and to possible faults, occurring during the operations. We test the capability of the chosen approach to autonomously learn and robustly perform complex tasks, such as setting up and clearing a table in a real kitchen-like environment.**

I. Introduction

In the last decade robotics community is envisioning the possibility of programming robots with intuitive and natural methods for human user. Obviously this is a non-trivial task considering the potential vastness of behaviors that people would want from a general-purpose home robot. Robot programming by demonstration (PbD) research field is focusing on giving solutions to this challenge [2]. One of the main trend in PbD research field focuses on how to represent skills and how to generate the representation starting from the skill observation. Currently the problem has two different approaches:

- low-level representation of the skill, where the focus is on learning trajectories from the human demonstrator
- high-level representation, where the focus is on learning how to represent the task as a sequence of predefined motion elements (*primitives*).

The first approach focuses on representing generalization of taught gestures. Usually it is done using statistical methods [3], [4], [5], [6] or dynamical systems [7], [8], [9]. While offering the possibility to encode human movements, this approach is not suitable for learning complex high-level tasks. The symbolic approach usually relies on a large amount of prior knowledge, in particular regarding important cues used for example for the task segmentation. Nevertheless it gives the capability of efficiently learning high-level complex skills through a relatively easy process. For that reason this approach have been widely studied and several methods have been presented. In [10] the authors present a graph-based representation of a skill. Each node of the graph represents a different behavior described in terms of preconditions (activation conditions) and postconditions (effect of the behavior). Every behavior is mapped to a robot *primitive behavior* which reaches from the same preconditions the desired postconditions. The architecture is capable of generalizing from multiple demonstrations and the execution is refined with sequences of supervised performance. Command cues are given by the user to facilitate learning, in particular during the task segmentation and the supervised execution phases. In [11] the authors start from a hierarchical representation approach. During multiple demonstrations the relevance of a feature for every action is estimated. This is done according to background knowledge, action occurrence probability in all demonstrations and possible user's vocal comments. The work focuses mainly on manipulation tasks (where at least one grasp is included). The approach shown needs one demonstration for the task, but uses prior knowledge, in the form of previous demonstrations to model the action. The system performance is shown focusing on the household task of setting up a table. The household manipulation task can also be found in [12]. The authors present an approach in which a run time planner is used to choose best action policy according to environment state. The approach is task-oriented, thus the robot replicates the results of the human demonstration by planning at execution time a sequence of actions. Constraints in task execution are understood from multiple demonstration of the same task. The validity of the system is assessed with tests, both in a virtual environment and real-life scenarios. The task is setting a table up, colored cube blocks are used instead of real objects. A different approach is presented in [13], where the authors, starting from the analysis of learning among social animals, explore more in-depth the role of the teacher during the learning. A hierarchical structure for *sequences*, *tasks*, and *behaviors* is generated for learning. Elements at every level of the hierarchy can be directly taught to the robot putting it through the execution by means of a teleoperation control. The approach presented is proved to scale well from simple to moderately complex tasks. A hybrid approach, including both high-level and low-level representation is presented in [14]. The authors propose a framework in which multiple observations of the same task are used to understand the essential interactions and to build a model. This model is used as a priori knowledge to generalize the trajectories for the manipulated objects. Experimental results are presented using a humanoid robot and a pouring task as benchmark. All the works presented uses more than one demonstration

Lorenzo Peppoloni, Alessandro Di Fava, Emanuele Ruffaldi and Carlo Alberto Avizzano are with Scuola Superiore Sant'Anna, TeCIP Institute, PERCRO laboratory `l.peppoloni@sssup.it`

or combine one demonstration of the task with a following supervised execution, to refine the task model. This is very time-consuming for the user, taking into account the potentially infinite number of different behaviors he would want to program. To be effective, robots should allow the users to program new behaviors on the fly and then let the system execute the task autonomously as quickly as possible. For that reason, we strongly believe that, to be usable in a real life scenario, a learning by demonstration framework should employ as few demonstrations as possible. Moving from that assumption, we propose an architecture capable of learning from only one human demonstration, without further intervention of the human user. In doing so the main problem is how to cope with task execution variability due to different teacher's behaviors and different initial conditions of the environment. We start from the basic idea of understanding the user effects on the environment and matching them with a robot primitive having the same effects. To cope with those problems we support the learning architecture with a deterministic finite state machine-based planner. The goal of the planner is twofold:

- it analyzes the representation of the task and rearranges it taking into account the end goal, the actions order and possible new initial conditions
- it provides the robot with a fault tolerant behavior during the execution of the task

All the functionality are integrated in ROS [1]. All the robot primitives are implemented using existing ROS stacks and the architecture itself have been implemented in a stack, thus resulting in a platform-independent framework available for every ROS-based mobile platforms.

The paper is organized as follows. Section II defines the basic assumptions of the architecture. Section III explains how the task is demonstrated, segmented and represented. Section IV explain how the execution phase is planned and autonomously supervised. Lastly section V reports how the system performs in learning how to set up and clear a table in a real kitchen-like scenario and section VI discusses the results obtained and further development for the presented architecture.

II. Primitives

We start from the assumption that a task can be built as a sequence of existing behaviors, called *Primitives*. We focus on manipulation tasks, which comprise all the activities consisting on moving objects between certain locations in the work environment. The goal for this type of tasks is having n objects in a particular regions of the space, moved following a desired sequence. Tasks as ordering a room, setting a table up or clearing a table can all be included in this category. Manipulation tasks can be decomposed as a series of the following actions:

- locating a shown zone of interest in the environment, where with zone of interest we mean locations where the actual manipulations take place
- moving between multiple zones of interests
- observing environment state
- interacting with the environment to change its state, in practice manipulate objects.

Those can be considered the predefined motion elements of human demonstrator, thus his primitives. Those human primitives can be mapped to robot skills considering the effects of every primitive on the environment [10]. In particular we define two sets of primitives P^S and P^A. The first set are *Support Primitives*, that are behaviors which do not involve actual manipulation but are necessary to model the task. The second set is *Action Primitives* and comprises manipulation-related behaviors.

The sets are defined as follows:

$$\begin{aligned} P^S &= \begin{cases} locate\ human\ demonstrators \\ analyze\ environment\ status \end{cases} \\ P^A &= \begin{cases} pick\ object \\ place\ object \end{cases} \end{aligned} \tag{1}$$

P^S set is used to instruct the system on where the actions is taking place at every step of the demonstrations. P^A set is used to construct the actual sequence of behaviors needed to model correctly the task. It is to be noted that we consider the case in which P^A behaviors act on one object at time.

III. Task representation and modeling

Since we focus on only one demonstration of the task, we cannot rely on autonomous segmentation processes based merely on task observation. For that purpose we decide to use social cues to instruct the system. In particular we use vocal commands to guide the learning phase.

The commands used are:

Follow me/Stop: to indicate zone of interests location, this is done during the initial phase of both learning and executing phases

Watch: to focus on one zone of interest to infer its new state

Two additional commands **Hello** and **Completed** are used to start and end the learning phase. Commands are translated by the system in behaviors from P^S set. After every step of the task demonstration the changes in the environment are assessed by the system. More specifically the state of the environment is defined as the state of each zone of interest. Considering the $i-th$ zone of interest at step k the state S_k^i is defined as the number and type of objects O it contains and their exact locations pos in the space.

$$S_k^i = \begin{Bmatrix} O_1 & O_2 & \cdots & O_n \\ pos_1 & pos_2 & \cdots & pos_n \end{Bmatrix} \tag{2}$$

Given two consecutive states S_k^i and S_{k+1}^i the transition between them is defined as

$$S_{k+1}^i = P_l(S_k^i, O_s) \tag{3}$$

where P_l is the $l-th$ primitive taken from the P^A set and O_s is the object of the state S_k^i being manipulated in the last step of the demonstration. P_l is inferred autonomously by the system comparing the S_k and S_{k+1} in terms of number and type of objects and their positions.

Three are the possible cases, for every object.

- A new object O_j^* is in S_{k+1}^i, without being in S_k^i, in this case the system will infer a *Place* primitive for O_j^* on location of interest i.
$$O_j^* \notin S_k^i \; and \; O_j^* \in S_{k+1}^i \rightarrow Place$$
- An object O_j^* is in S_k^i, without being in S_{k+1}^i, in this case the system will infer a *Pick* primitive for O_j^* on location of interest i.
$$O_j^* \in S_k^i \; and \; O_j^* \notin S_{k+1}^i \rightarrow Pick$$
- An object O_j^* is in S_k^i, and in S_{k+1}^i, but in a different position, in this case the system will infer a sequence of a *Pick* and a *Place* primitive for O_j^* on location of interest i
$$O_j^* \in S_k^i \; and \; O_j^* \in S_{k+1}^i$$
$$and$$
$$pos_k \neq pos_{k+1} \rightarrow Pick \; and \; Place$$

Every step of the demonstration is saved in a strip-like map, as shown in Figure 1 for a two steps example. Also the final states of the location of interest are saved. Since the system is using only one demonstration, no inference can be done at this level about the generalization of the task.

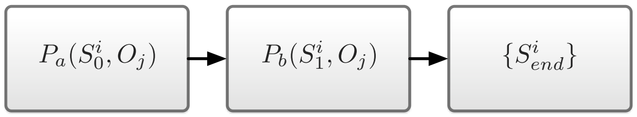

Fig. 1. Example of the representation for a two step task. Starting from initial state S_0, the first step is the execution of the primitive P_a on object O_j of location i. The second step is primitive P_b on object O_j of location i. Lastly final state S_{end} for every location is saved.

IV. Task planning and execution

During every execution phase the planner starts from the map of the task and creates a new ad-hoc map (*Execution map*) for this particular execution. At the beginning of the execution phase, the system assesses the initial conditions of the environment. In particular the initial states S_0^i for the locations of interest are detected and compared to the desired final states in the map. From this comparison all the unnecessary steps are pruned from the execution map. In a second phase all the remaining steps are analyzed to assess their necessity to achieve the final state. All the steps considered non influential are pruned from the map. For non influential we mean, for example, manipulation sequences of objects which are not present in the final state of the map. At this point the system starts executing the steps in the obtained execution map. If during execution the state step $i-th$ is not feasible the planner modifies the execution map adding new steps, to be performed before the current step. In particular, during manipulation tasks, three are the possible scenarios:

- the object is already on the right location of interest but in the wrong position and its final position is free
- the object is already on the right location of interest but in the wrong position and its final position is not free
- an object not present in the final configuration is on the location of interest. In this case the object is moved to a free space in another location of interest.

How the first two situations are managed by the planner is shown by the diagram in Figure 2. When a pick/place sequence is detected for the same object, if the object is already present and in the desired position the primitive is removed from the execution map. Otherwise, if the object is present but in the wrong position it is moved. If the desired position for the object is not free, primitives are added to move first the obstructing object, which will be later managed by other primitives.

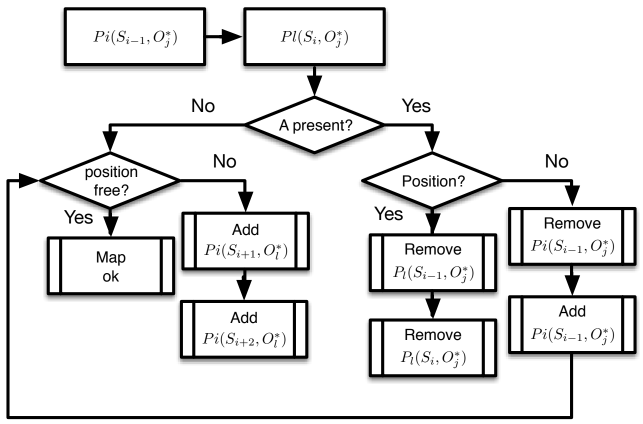

Fig. 2. Diagram showing the behavior of the planner regarding objects already present or in the case of obstruction by other objects. P_i and P_l are *Pick* and *Place* primitives. When a *Pick/Place* sequence for the same object is in the action list, the primitives are eventually changed to manage already in place objects, already present objects but in the wrong position and obstructing objects .

After the execution of every step, its success is assessed and in case of failure the step is performed again or new primitives are added to achieve the desired state. An example of how the map in Figure 1 can be modified in an execution map is shown in Figure 3.

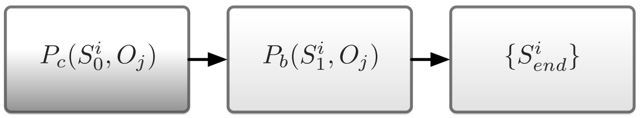

Fig. 3. Example of possible execution map. Starting from the original task map, primitive P_a have been pruned, because not necessary while the primitive P_c have been introduced to make the P_b primitive feasible.

An basic real example of how the execution map is obtained from a learning map is shown in Figure 4, where only one location of interest and three objects are considered.

Starting from the showed map and the desired final state, the planner compares the new initial state S_0 with the final state. Since object A is already present and in the correct

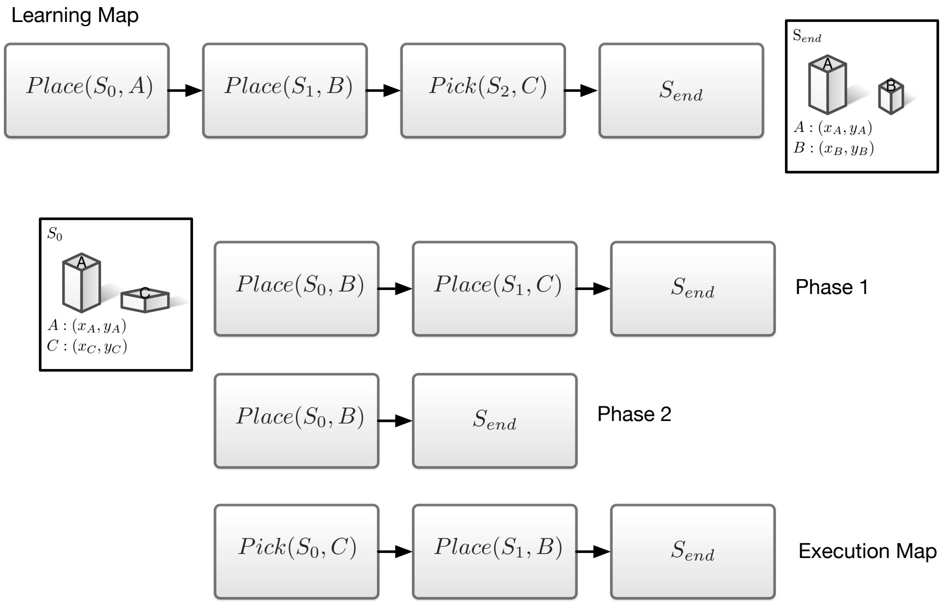

Fig. 4. Example of planner action in a three objects, one location of interest case. Starting from the original task map, primitive $Place(S_0, A)$ is removed because it is not needed comparing initial state S_0 an desired finale state S_{end}. Then primitive $Place(S_1, C)$ is removed because not necessary considering S_{end}. Lastly primitive $Pick(S_0, C)$ is introduced to make the primitive $Place(S_1, B)$ feasible, since the object B desired spot is occupied by object D. The resulting execution map is shown.

position the first *Place* primitive is pruned (Phase 1). Then analyzing the final state the second *Place* primitive is pruned since it acts on object C, which is not present in the final state (Phase 2). Lastly a new primitive is introduced since to place object B in its final position, object C must be first moved to solve the cluttering problem.

V. Experimental tests

To show the performance of the system we carried out experimental tests using as benchmark the manipulation tasks of setting up and clearing a table. We have already presented the actual implementation of the system in the ROS framework. The interested reader can refer to [15] for details. The platform used for the test is a Kuka YouBot [16], which is ROS compatible. For the tests a kitchen-like environment has been set up, in which the locations of interest are the sink and the table. The objects involved in the tests are shown in Figure 6 and are already known by the robot. This means that the platform is capable of recognizing and manipulating the objects. The platform is equipped with two Kinect cameras (Microsoft Corp., Redmond Washington) to navigate in the environment and recognize and manipulate the objects. Pictures of the robot performing the task learned in the environment are shown in Figure 5. The location of interest are equipped with markers to aid the YouBot navigation and to express objects positions in sink/table own reference system. We carried on one test for the learning and two for the performing. In the learning test the robot understands how to set up the table for one person. Then the acquired knowledge is used to set the table up from two different starting conditions.

In the first test we started with all the objects on the sink. In the second one an object is on the table in the wrong position and the other is on the sink.

A. Learning phase test

The learn phase, as explained in section III, is guided by user vocal commands. First the robot performs the recognition handshake, in which it recognizes and tracks the user, then commands are given to show the locations of interest for the task to be learned. In this phase the initial state of both table and sink is saved. Then the user changes the furniture configurations and after every change tells the robot to navigate to table/sink and infer which primitive has been applied from the comparison between previous and current state. The process ends with the "**Completed**" command, after which, the robot saves the actions sequence.

In the saved sequence, first the final states for the table and sink are declared:

```
table status:
bowl: 1
obj_id: 0 model_id: 18851
x: -0.0432 y: 0.0075 z: -0.0212
glass: 1
obj_id: 0 model_id: 18849
x: -0.2226 y: 0.0148 z: -0.0164
sink status:
bowl: 0
glass: 0
```

As it can be seen the final state is one bowl and one glass in specific positions on the table, while the sink is empty. After the final states the actual sequence of primitives is listed

```
actions:
```

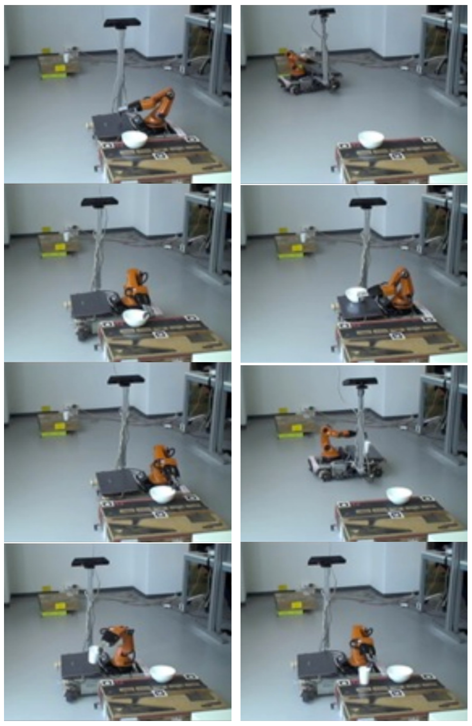

Fig. 5. Screenshots from the second perform phase test. After having watched the initial status of the furniture, the platform shifts the bowl (already on the table) to its correct position, then performs pick and place on the glass.

Fig. 6. Meshes of the object used for the tests in the real life scenario.

```
type: begin
 x: 0 y: 0 z: 0
 withobject: 0
 furniture: no_furniture obj_id: 0 model_id: -1
type: pick
 x: -0.1326 y: 0.0051 z: 0.0100
 withobject: 1
 furniture: sink obj_id: 0 model_id: 18851
type: place
 x: -0.2226 y: 0.0148 z: -0.0164
 withobject: 1
 furniture: table obj_id: 0 model_id: 18851
type: pick
 x: -0.0512 y: 0.0123 z: -0.0256
 withobject: 1
 furniture: sink obj_id: 0 model_id: 18849
type: place
 x: -0.0432 y: 0.0075 z: -0.0278
 withobject: 1
 furniture: table obj_id: 0 model_id: 18849
```

From the sequence we can see that first a bowl has been picked from the sink and placed on the table, and then a glass has been picked from the sink and placed on the table. It is to be noted that the positions saved are expressed in the sink/table reference frame, so that the sequence is independent from different table/sink positions in the environment. While the objects positions are relevant for the *Place* primitive, for the *Pick* primitive only which object has been picked is meaningful for the task modeling, positions are saved for logging reasons.

B. Performing phase test

During the perform phase, firstly, the location of the table and the sink is showed to the platform. From that point the task execution is autonomous. The first step consist on checking the furniture initial states. Then the best way to act is chosen. In this phase irrelevant and inconsistent steps of the demonstration are removed by the planner taking into account the initial state and the desired final configuration, as detailed in Section IV. In the first scenario, the initial conditions are similar to the one of the demonstration, only objects initial positions on the sink are different. In that case the execution map is a copy of the task map and no modification by planner are needed. The robot executes the 4 primitives in the same exact order. First the bowl is moved on the table in the desired position, then the same is done with the glass. In the second test, the bowl is already on the table in the correct position. The planner modifies the task map obtaining a different execution map, as shown in Figure 7. The third and the fourth primitives of the sequence are removed, and only the first and second ones are executed.

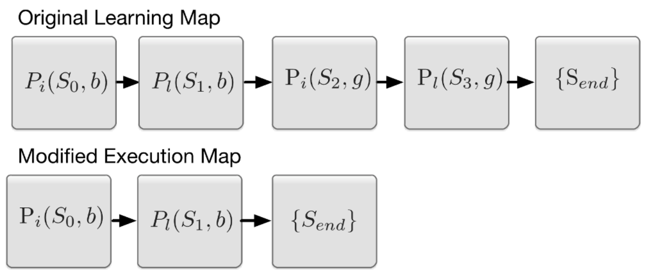

Fig. 7. Modification of the task map to obtain the execution for the considered perform phase. Pi and Pl are respectively *Pick* and *Place* primitives, g and b are *glass* and *bowl*. The primitives acting on the glass are removed.

Several other experiments have been conducted to test the task learning and planning with different setups and different numbers of objects. During the experiments the robot has been capable of learning, planning and executing the task without problems. Some problems can arise due to issues related to specific setup scenarios. In particular the faults mainly arise because of failure of vision and navigation systems. If the objects are placed too close the *object_recognition* pipeline fails in recovering exactly which objects are on the scene and where. When carrying objects from one location to the other, if the *navigation* module plans a trajectory not smooth enough, objects can be lost on the path. On the other hand faults due to bad manipulation are almost always perceived and corrected online by the planner. This is done by the planner policy to assess the effect of every primitive executed by the robot and compare its result to the desired one.

VI. Conclusions and further work

In this paper we presented a ROS integrated architecture for learning complex manipulation tasks from only one user demonstration. To tackle the task segmentation problem we introduced vocal cues, which guide the learning phase. A planner is included in the architecture to increase the robustness of the approach and deal with task generalization issues which cannot be inferred from one single task demonstration. The performance of the system have been tested in a kitchen-like environment with the tasks of setting up and clearing a table as workbenches. After the learning phase the system has been capable of autonomously executing the learned tasks, starting from different initial states. During the execution of the task some choices of the planner can lead to more or less time-consuming solutions. For that reason, future developments will regard the capability of the planner to take into accounts also variables like cost functions. Those variables will be taken into account during planning phase, and if more execution possibilities are available, chosen indexes will be minimized.

References

[1] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009.

[2] S. Calinon, "Robot programming by demonstration," in *Springer handbook of robotics*. Springer, 2008, pp. 1371–1394.

[3] 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, 2007.

[4] S. Calinon, F. D'halluin, E. L. Sauser, D. G. Caldwell, and A. G. Billard, "Learning and reproduction of gestures by imitation," *Robotics & Automation Magazine, IEEE*, vol. 17, no. 2, pp. 44–54, 2010.

[5] 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.

[6] G. E. Hovland, P. Sikka, and B. J. McCarragher, "Skill acquisition from human demonstration using a hidden markov model," in *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, vol. 3. Ieee, 1996, pp. 2706–2711.

[7] A. Ijspeert, J. Nakanishi, and S. Schaal, "Learning control policies for movement imitation and movement recognition," in *Neural information processing system*, vol. 15, 2003, pp. 1547–1554.

[8] S. Schaal and C. G. Atkeson, "Receptive field weighted regression," *ATR Human Information Processing Laboratories, Tech. Rep. TR-H-209*, 1997.

[9] M. Ito, K. Noda, Y. Hoshino, and J. Tani, "Dynamic and interactive generation of object handling behaviors by a small humanoid robot using a dynamic neural network model," *Neural Networks*, vol. 19, no. 3, pp. 323–337, 2006.

[10] M. N. Nicolescu and M. J. Mataric, "Natural methods for robot task learning: Instructive demonstrations, generalization and practice," in *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*. ACM, 2003, pp. 241–248.

[11] 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, 2007.

[12] 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*. IEEE, 2006, pp. 358–363.

[13] J. Saunders, C. L. Nehaniv, and K. Dautenhahn, "Teaching robots by moulding behavior and scaffolding the environment," in *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*. ACM, 2006, pp. 118–125.

[14] K. Ogawara, J. Takamatsu, H. Kimura, and K. Ikeuchi, "Generation of a task model by integrating multiple observations of human demonstrations," in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, vol. 2. IEEE, 2002, pp. 1545–1550.

[15] L. Peppoloni and A. Di Fava, "Programming of a mobile robotic manipulator through demonstration." in *ICINCO (2)*, 2012, pp. 468–471.

[16] R. Bischoff, U. Huggenberger, and E. Prassler, "Kuka youbot-a mobile manipulator for research and education," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–4.