A Comparison of Two Algorithms for Robot Learning from Demonstration

Halit Bener Suay and Sonia Chernova
Robotics Engineering Program
Worcester Polytechnic Institute,
Worcester, MA 01609, USA.
benersuay@wpi.edu, soniac@wpi.edu

***Abstract*—Robot learning from demonstration focuses on algorithms that enable a robot to learn a policy from demonstrations performed by a teacher, typically a human expert. This paper presents an experimental evaluation of two learning from demonstration algorithms, Interactive Reinforcement Learning and Behavior Networks. We evaluate the performance of these algorithms using a humanoid robot and discuss the relative advantages and drawbacks of these methods with respect to learning time, number of demonstrations, ease of implementation and other metrics. Our results show that Behavior Networks rely on a greater degree of domain knowledge and programmer expertise, requiring very precise definitions for behavior pre- and post-conditions. By contrast Interactive RL requires a relatively simple implementation based only on the robot's sensor data and actions. However, Behavior Networks leverage the pre-coded knowledge to effectively reduce learning time and the required number of human interactions to learn the task.**

***Index Terms*—Learning and Adaptive Systems, Personal Robots.**

I. Introduction

Robot *learning from demonstration (LfD)* is an established area of robotics research focusing on algorithms that enable a robot to learn a policy from demonstrations performed by a teacher, typically a human expert. LfD algorithms address two fundamental goals: 1) leveraging teacher input to improve policy learning, in terms of time and/or performance, over purely non-interactive, data-driven techniques, and 2) providing an intuitive interface for robot behavior customization for users who are not expert programmers.

Two recent survey papers [1], [2], cover the breadth of approaches in this research area. In [1], the authors present a categorization of existing algorithms, segmenting works by demonstration technique (i.e. how demonstrations are recorded) and policy derivation method (i.e. how the policy is learned from the demonstrations). Specifically, demonstration techniques are categorized as teleoperation, shadowing, sensors on the teacher or external observation based on who performs the task during demonstration (the human or robot) and what information about the demonstrators actions is available. Policy derivation methods are divided among mapping functions (e.g., classification [3] and regression [4] techniques), system model (e.g., reinforcement learning [5]) and planning [6] approaches.

Despite the existence of this diverse body of work in learning from demonstration, research in this area lacks comparative study. In fact, we are not aware of any two distinct LfD algorithms mentioned in the surveys whose performance has been compared side by side in the same domain. This fact is not due to any lack of academic rigor in these works, but instead highlights some of the challenges of this research area. The use of different robotic platforms, sensors, interfaces and demonstration techniques leads to different representations and characteristics of demonstration data and make comparison difficult across applications. The use of standard datasets, which is common in other research fields, is not possible due to the human interaction component and the diversity of interaction techniques. As a result, comparison requires full re-implementation of algorithms, a highly time consuming and challenging effort due to the absence of existing open source solutions.

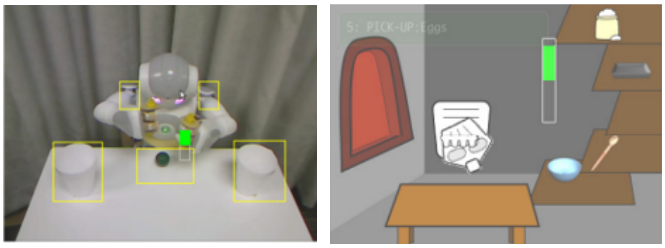
This paper presents an experimental evaluation of two learning from demonstration algorithms. We selected these approaches for the comparison because they are representatives of distinct classes of approaches for both teacher demonstration (reward/guidance versus teleoperation) and policy learning (system model versus planning). We evaluate these algorithms using a humanoid robot, then we discuss the advantages and drawbacks of each algorithm.

In the following section we present an overview of the Interactive Reinforcement Learning algorithm, followed by a summary of the Behavior Networks algorithm in Section III. In Section IV we describe the experimental domain and setup used in the comparison. In Section V we compare the algorithms and discuss their tradeoffs. We conclude the paper in Section VI.

II. Interactive Reinforcement Learning with Human Guidance

The Interactive Reinforcement Learning algorithm with human guidance, as described in [7], extends traditional Q-learning [8], a variant of Reinforcement Learning [9].

Interactive Reinforcement Learning extends traditional Q-learning with the addition of human *reward input* (replacing rewards received from the environment) and human *guidance input* to restrict action selection. When learning begins, the robot has a set of pre-programmed actions and no knowledge of the task. Programmer provides state variables, actions and guidance mapping (i.e. how each guidance message will restrict robot's choice of actions). As the robot explores the state-space and receives positive or negative reward from the human teacher, the Q-values are updated.The teacher is given



(a) (b)

Fig. 1: (a) Screenshot of our interface. Yellow rectangles highlight the interactable areas for guidance. The teacher can guide the robot with a right click on the object or area. (b) Screenshot of Sophie's Kitchen. A positive reward is given with a left click and drag upwards in both interfaces.

a short period of time following each action to provide positive or negative feedback. Additionally, during this period of time, the teacher can provide *guidance input*. Guidance is given with respect to an object or location (e.g., center of the table), with the effect of restricting the choice of immediately available actions to those related to the target object (e.g., look at table, put down object on table). This technique can be thought of as a means of directing the robot's attention to the target object or area.

We made our implementation of Interactive RL based on description provided in [7]; we set the learning rate $\alpha = 0.3$ and the discount factor $\gamma = 0.75$. Initial Q-values were set to 0.5 for all actions.

ε-greedy action selection method is used to select the next action both when a guidance signal is present and when it is absent.

A key contribution of the original Interactive RL paper [7] is the interactive reward interface (Fig. 1) that enables an online user to train a virtual robot to bake a cake in a domain called Sophie's Kitchen[1]. The teacher uses the left mouse button to bring up a reward bar (Fig. 1a) to provide a reward signal $r = [-1, 1]$ for the agent's previous action and the desirability of the current state. Right clicking anywhere in the image results in guidance input to that region; if the selected region has no associated guidance effects, no guidance is given.

As with the algorithm, we implemented the interface described in [7] in order to evaluate its use in real-world systems (Fig. 1b). Our interface shows a fixed view of the environment via a web-cam (Fig. 2a). Table I shows how guidance input restricts action selection. Each row refers to a yellow rectangle shown in Fig. 1a. Fig. 2b shows zone 1, 2 and 3.

III. Behavior Networks

A Behavior Network (BN) is a way of representing a high-level robot task as sequence of behaviors [10], [11]. The concept of Behavior Networks is very different than the Interactive Reinforcement Learning described in the previous section. In Behavior Networks, the robot is pre-programmed with a set of task specific high level behaviors (e.g. pick up the object) which are mapped to low level actuator commands (e.g. rotate motor *i*, 5 degrees). For instance, for a humanoid robot the behavior for picking up the object with the left hand could control all the actuators of the left arm, wrist and fingers of the robot. If the robot has a simple gripper, then *pick up the object* could simply control the low level actuator command that closes the gripper. In Behavior Networks, these high level behaviors are called Abstract Behaviors, and the low level actuator commands are called Primitive Behaviors.

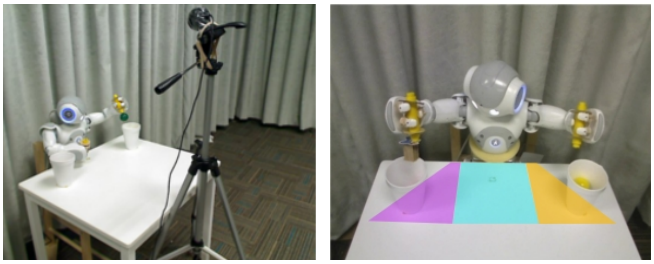
The tasks that humans want robots to accomplish may be (and usually are) broken into several simpler steps. In the context of Behavior Networks, we can think of each simple step of a task as an Abstract Behavior, and the task itself as a network consisting of those behaviors. Typically, each simple step has at least one sub-goal. In Behavior Networks, these sub-goals must be defined by the programmer in the form of post-conditions of a behavior. The post-conditions define the state in which the robot is expected to be after performing the associated Abstract Behavior.

In summary, Behavior Networks are made up of the following three components:

- **Primitive Behaviors (PB)** - a low level command that activates the actuators or queries the sensors of the robot (e.g. Rotate motor 2, 15 degrees).
- **Abstract Behaviors (AB)** - a high level behavior that wraps a Primitive Behavior (e.g. Turn right). Unlike Primitive Behaviors, Abstract Behaviors have hard coded pre-conditions and post-conditions. Pre-conditions of an AB are satisfied by the post-conditions of other ABs in the network. An AB has the ability to activate a PB when its pre-conditions are met.
- **Network Abstract Behaviors (NAB)** - a network of Abstract Behaviors. This is the highest level component that can be used to group ABs under one name to represent complex behaviors (e.g. Visit Targets).

NABs and ABs are connected to each other with three types of "links", depending on the relationship between the behaviors. A behavior that starts before and continues happening during the execution of second behavior becomes a **Permanent Precondition** to its follower; A behavior that starts before and ends right before or during the execution of another behavior becomes an **Enabling Precondition** of its follower; And a behavior that runs and ends before the execution of another behavior becomes an **Ordering Precondition** to its follower. These connections between behaviors are automatically identified by the algorithm based on the observed demonstrations, and are used to form the Behavior Network of a given task. Once the network is learned, the robot can use it to execute the learned task. The execution can be sequential, as in the case where the task segments contain temporal ordering constraints, or opportunistic, as in the case where order of execution does not matter. For complete details of the Behavior Network algorithm, please see [10], [11], [12].

[1]The original Sophie's Kitchen web application is available at http://www.cc.gatech.edu/∼athomaz/sophie



(a) (b)

Fig. 2: Experiment Setup. (a) Web camera captures the table and the robot for the user interface. (b) The table is divided in three zones. Orange, cyan blue and purple show $z_1, z_2,$ and z_3 respectively. The robot has an object at the tip of its right hand and is about to drop it in the right cup.

Guidance	ActionSet
Zone 1	$LLeft$, $LDrop$
Zone 2	$LPutDown$, $LPickUp$, $RPutDown$, $RPickUp$, $TakePicture$
Zone 3	$RRight$, $RDrop$
Left Shoulder	$LPickUp$, $LRight$, $LLeft$, $LPutDown$, $LDrop$
Right Shoulder	$RPickUp$, $RRight$, $RLeft$, $RPutDown$, $RDrop$

TABLE I: Guidance messages restrict action selection to the set of available actions as shown. The first letter *L* and *R* denotes either Left or Right hand depending on the hand with which the action is executed.

IV. Evaluation and Experimental Domain

Comparison of two algorithms with distinctly different properties is an inherently difficult task. Selecting a task that is suitable for both approaches is important in order to avoid introducing bias into the results. For our experiments we designed an object sorting domain using an Aldebaran Nao humanoid robot. We believe that a full comparison would require a comprehensive user study, however this type of evaluation is outside the scope of this paper and we leave it for future work. Instead we focus our evaluation on learning performance metrics, and discuss the advantages and disadvantages of each method in the light of our findings.

Our experimental setup consists of a stationary work table where the Nao is seated on a chair. Magnetic objects are placed in front of the robot one at a time. The robot must learn to use its on-board camera at the appropriate time to identify the characteristics of the object and then to pick up and place the object in one of two cups located on the table. In case of Interactive RL, the teacher is able to observe the robot directly and through a live camera feed from the web-cam mounted in front of the robot (see Fig. 2a). The web-cam feed is displayed on a computer monitor and is integrated into the interaction interface, reward and guidance input. In case of Behavior Networks the teacher observes the robot directly, and the interaction interface is a simple window with buttons displayed on the computer screen. The teacher clicks on the buttons to select the robot's actions.

The experimental domain $D = (Z, S, A, T)$ is defined by a finite set of zones Z, a finite number of world states S, a finite set of possible robot actions A and the transition function $T : S \times A \rightarrow S$ that determines transitions between states by way of actions. Zones Z represent the regions where the robot's arms or an object can be located. In our domain, $Z = \{z_1, z_2, z_3\}$ corresponds to the left, front and right side of the robot respectively (Fig. 2b).

The state of the world $S = \{S_r \cup S_o\}$ consists of the union of the state of the robot S_r and state of the object being classified S_o. The robot state is defined by $S_r = (z_{lh}, z_{rh}, hand_o)$, where $z_{lh}, z_{rh} \in Z$ correspond to the current zone of the left and right hands, respectively, and $hand_o \in \{left, right, none\}$ represents whether an object is located in either of the robot's hands, or not. For example, in Fig. 2b the robot's state is represented by $S_r = (z_1, z_3, right)$.

The object state is defined by $S_o = (I, o_z, o_p)$, where I specifies either the type of the object, or a finite set of image features used to characterize the object, as described below. $o_z \in Z$ is the current zone of the object, and $o_p \in P$ is the placement of the object such that $P = \{z_1, z_2, z_3, right\_cup, left\_cup\}$. For example, in Fig. 2b the object state is represented by $S_o = (pattern, z_3, z_3)$.

The robot has eleven possible actions, $TakePicture$, $nPickUp$, $nPutDown$, $nDrop$, $nLeft$, and $nRight$ where $n \in \{L, R\}$ determines the arm with which the action is performed. The $TakePicture$ action makes the robot take a snapshot of the table in front of it and extract features from the image to determine the characteristics of the object currently placed there (if any). $nRight$ and $nLeft$ actions move the hands between zones (e.g. $LLeft$ moves left hand left, that is, to Zone 1 and $RLeft$ moves right hand left, that is, to Zone 2. Note that the robot's left hand can operate only in $z_{1,2}$, and the right hand can operate only in $z_{2,3}$, which enables the robot to pick up an object with either hand but allows only one hand to reach each cup and $Drop$ the object into it. If the robot initially picks up the object with the wrong hand, it can switch hands by performing a $nPutDown$ action followed by a $nPickUp$.

To compare the Interactive RL and Behavior Networks algorithms we studied their application to two variants of the above domain, a small and a large state space representation. This comparison was performed in order to evaluate how each algorithm's behavior changed with respect to state space size. To vary the state space size we changed the number of features in S_o used to describe the object being sorted. Specifically we used the following two experimental conditions:

- **Small deterministic state space:** In the small state space condition, the object descriptor I of $S_o = (I, o_z, o_p)$ consists of a single variable with two possible values, $plain$ or $pattern$, representing the class of object. This mapping is determined based on the number of Speeded Up Robust Features (SURF) [13] detected in the image of the object and identifies the

object as either a solid colored ball or a character magnet, respectively. Specifically, objects for which over 50 SURF features were identified were characterized as *patterned*, and *plain* otherwise. This threshold value was determined empirically. In our tests we found that the number of SURF features deterministically separate these two object types. Combining this binary object representation with other state information, such as object location and robot state, results in 360 possible states.

- **Large, non-deterministic state space:** In the large state space condition, the object descriptor I consists of four elements with the following possible values: the number of SURF features $F \in [50; 100; 150]$, smoothness $R \in [0.05; 0.1]$, entropy $e \in [5.0; 10.0]$ and area of the bounding box of the object $B \in [15,000; 20,000; 25,000]$ (in $pixel^2$). We calculate smoothness and entropy as described in [14]. After image processing, the value for each descriptive feature is thresholded into the categories listed above. This representation was purposefully designed to reduce the size of the state space by providing a small set of possible values for each variable, while at the same time making sure that none of the descriptive elements alone is sufficient for distinguishing between the plain and patterned object types. For example, although they are the same type of object, the plain green and plain yellow magnets have different smoothness, perceived area and entropy states. To the robot, these objects therefore appear distinct and it must learn that they belong to the same group. Furthermore, this representation is non-deterministic and variations in object placement will result in slightly different state representations, allowing us to evaluate how each algorithm performs under these conditions. We believe this domain is closer to the common characteristics of many experimental robotics domains. In total, the large state space representation results in 6480 states.

During the experiments all processing was performed on a PC connected to the robot over the network. For image processing we used OpenCV [15]. We ran three trials for each experimental condition and all reported results are averaged over the three trials. All experiments were performed by the same teacher who is an expert roboticist, who followed a standard optimal teaching policy. We terminated learning once the robot was able to correctly sort three objects into each cup without guidance or reward. Objects were presented to the robot in random order.

V. Comparison of the Results and Discussion

In this section we first walk the reader through our experiment results, then draw attention to some important differences both in theory and implementation of these methods.

Fig. 3 presents a summary of learning results for the Interactive RL algorithm. Graphs plot the level of teacher involvement over time in terms of the percentage of actions for which the teacher provided reward and/or guidance. A value of 100% indicates that the teacher provided input to the robot following all of the actions executed in that minute.

Note that the degree to which guidance reduces the set of actions has a significant effect on the learning rate. If guidance

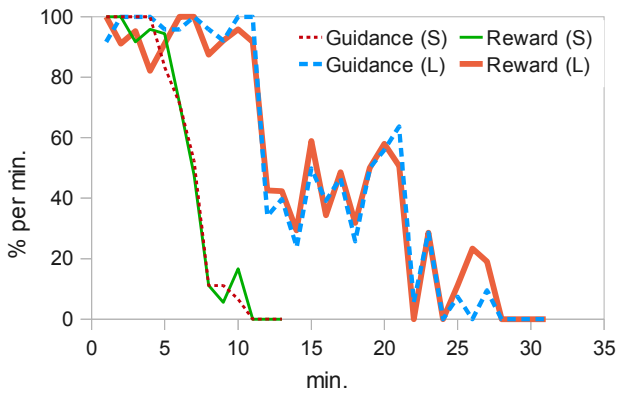


Fig. 3: Learning results for the small state space (*S*) with $\delta_\varepsilon = 0.002$ and large state space (*L*) with $\delta_\varepsilon = 0.001$.

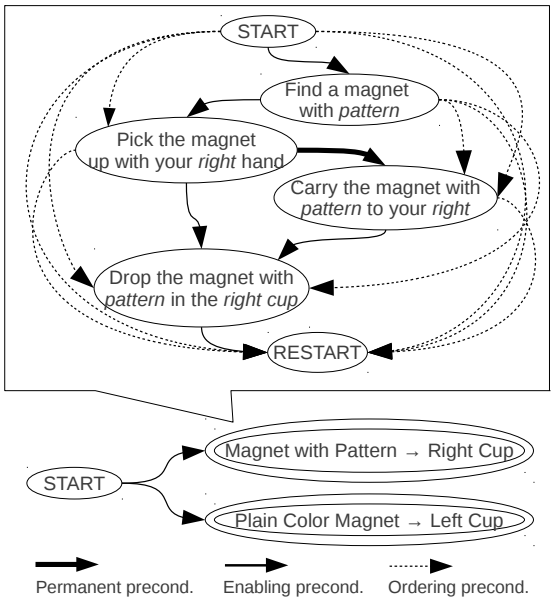


Fig. 4: Behavior Networks of the sorting task in Small State Space. Network Abstract Behaviors are shown with double lines.

does not significantly reduce the choice of available actions (see Table I), then the effect of this input method is reduced since the robot will have to choose randomly among them. In contrast, if the guidance message limits the robot to a single action, then all choice is taken away from the robot; which is not the purpose of this algorithm.

In our implementation we designed guidance as an effective teaching method that always restricts the list of available actions but never reduces action selection to a single choice.

Fig. 4 shows the NABs that the robot learns in the small state space condition as a result of teacher demonstrations. Since the object description is characterized by a binary condition (patterned or plain), just two demonstration sequences were enough demonstrate the entire task, one for each type of object.

In large state space, the teacher had to make one demonstration per object to sort. This is because each object puts the

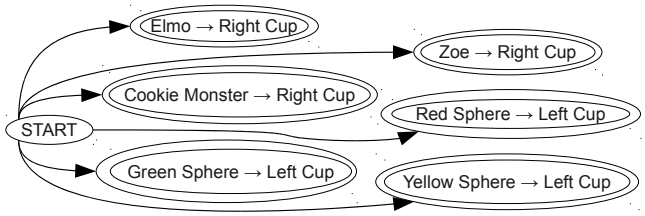


Fig. 5: One of the Network Abstract Behaviors: Magnet with Pattern → Right Cup

Abstract Behavior	PB	Post Condition of the AB
Find plain magnet	$TakePic$	$S_o = (plain, 2, 2)$
Find pattern	$TakePic$	$S_o = (pattern, 2, 2)$
Pick up plain left	$LPup$	$S_r = (2, (2or3), 1)$
Pick up pattern right	$RPup$	$S_r = ((1or2), 2, 2)$
Carry plain left	$LLeft$	$S_r = (1, (2or3), 1)$
Carry pattern right	$RRight$	$S_r = ((1or2), 3, 2)$
Drop plain left cup	$LDrop$	$S_r = (1, (2or3), 0)$
Drop pattern right cup	$RDrop$	$S_r = ((1or2), 3, 0)$

TABLE II: This table describes the Abstract Behaviors that our robot used for accomplishing the task. The second column shows the Primitive Behaviors (or actions) that the ABs activate and the third column shows the necessary post-conditions of the ABs.

robot in a different state. 6 demonstrations are made, followed by the execution of the task. Obtained network for the large state space is given in Fig. 5.

We find it necessary to list our choice of post-conditions of the Abstract Behaviors. Our selection is shown in Table II.

Table III summarizes the results of our empirical comparison. A comparison of learning time across approaches shows that Behavior Networks take approximately half the time to learn the small task and roughly a third of the time to learn the large task compared to Interactive RL. Comparison within approaches shows that the increase in state space size from small to large leads to a 115% increase in learning time for Interactive RL but only a 53% for Behavior Networks.

This result is worth mentioning because it suggests that learning using Behavior Networks will scale more efficiently to larger domains.

There are many differences in the ways the algorithms obtain information about the task. When using Interactive RL, the robot drives the action selection process, selecting actions that either exploit its current policy or explore the environment. The teacher's role is to respond to the robot's actions through reward, and to influence, but not fully control, the selection of future actions. In the case of Behavior Networks, the teacher guides the action selection process during learning and the robot executes only the actions it is told to perform, without exploration or reward.

Complementary Tradeoffs: Behavior Networks rely solely on the teacher's demonstrations for knowledge and aim to exactly reproduce the teacher's behavior. This means that any errors or suboptimalities in the human's actions are also captured and reproduced by the algorithm. Interactive RL, on the other hand, aims to optimize the reward function defined

	Small		Large	
	Int. RL	Beh. Net.	Int. RL	Beh. Net.
Time (min.)	13.0	5.7	28.0	8.7
# Actions	82.0	40.0	201.0	60.0
# Interactions	88.0	13.0	229.7	73.0
# Incorrect Actions	15.3	0.0	29.0	0.0
# Guidance Input	44.0	N/A	144.0	N/A

TABLE III: Comparison of the experiment results obtained in small and large state space. $\varepsilon = 0.1$ for all Interactive Reinforcement Learning experiments. $\delta_\varepsilon = 0.002$ for small state space and 0.001 for large state space.

by the person, but does not trust the teacher to perform action selection. Even in the presence of guidance, the importance of exploratory actions is maintained, allowing the algorithm to potentially improve upon the performance of the teacher by discovering action sequences that more effectively achieve the teacher-specified rewards. However, exploration can potentially lead the robot into unsafe situations, whereas direct teleoperation of the robot, as in the case of Behavior Networks, enables the teacher to fully control the states encountered by the robot.

How much background task knowledge is required of a programmer to develop the underlying learning framework? In the case of Interactive RL, all that must be defined by the programmer is the state and action sets of the robot, and the transitions between states via actions are learned implicitly during the learning process. In Behavior Networks, the programmer needs to know not only the state and actions, but also the state values that will result from the execution of different actions in order to form the pre- and post-conditions of behaviors. We found that coding the Behavior Networks required a significantly higher degree of knowledge of the domain, as well as a precise measurement of the sensor values that would be observed. Without this pre-coded information, the robot's state would not match the post-conditions of any behaviors and no behaviors would be activated and learned.

How much background task knowledge is required of a non-expert teacher to teach the robot? We found that in the case of Behavior Networks, the teacher had to know more about the domain. Specifically, the teacher had to know the effects of each of the robots actions in order to correctly select which one to perform next. This is in contrast to Interactive RL, in which the teacher only had to convey the desirability of the current state. In summary, we found that the Behavior Network algorithm requires a more in-depth understanding of the domain both from the programmer and from the user.

Potential Failure Cases: The two algorithms have very different potential failure cases with respect to their design. The post-conditions of the behaviors play a significant role in the construction of the Behavior Network. The need to accurately define the post-conditions of a behavior is crucial because without appropriately matching conditions the robot will be unable to construct the correct network and learn the task. In the case of Interactive RL, the critical decision lies in the choice of the limitation of guidance messages. If a given

Behavior Networks	Interactive Reinforcement Learning
+ Robot doesn't make any mistake during learning.	+ Simpler to implement and to teach.
+ Teaching is much faster.	+ Automatic action selection.
+ Modification and correction after teaching the task is easier.	+ Robot may perform better than what it learned (Exploration).
− Limited with what the teacher teaches (No exploration).	− Teaching time increases remarkably as the state space gets bigger.
− Requires lots of domain knowledge.	− Markovian; No state aliasing.

TABLE IV: A summary of the advantages (+) and disadvantages (−) of the two approaches compared .

guidance message does not restrict the available actions in a given state, then it will not have any effect on learning. On the other hand, a too restrictive guidance design would have the same effect as telling robot exactly what to do, which contradicts with the exploratory nature of the Reinforcement Learning. The aspects mentioned above are summarized in Table IV.

VI. Conclusions

In this paper we analyzed two different Learning from Demonstration methods on a common real world task using a humanoid robot. The methods we have chosen to analyze were Interactive Reinforcement Learning and Behavior Networks due to their distinct human-robot interaction, policy learning and representation approaches. We compared the algorithms in terms of learning time, number of demonstrations, ease of implementation and other metrics. Additionally, we explored how each technique scales by comparing two domain representations with different state space sizes. Our findings show that Behavior Networks rely on a greater degree of domain knowledge and programmer expertise, requiring very precise definitions for behavior pre- and post-conditions. By contrast Interactive RL requires a relatively simple implementation based only on the robot's sensor data and actions. However, Behavior Networks leverage the pre-coded knowledge to effectively reduce learning time and the required number of human interactions to learn the task. In summary, these approaches represent a tradeoff between learning time and programming effort. In future work, we aim to perform additional analysis in user studies and expand our comparison to other LfD algorithms.

References

[1] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robot. Auton. Syst.*, vol. 57, pp. 469–483, May 2009. [Online]. Available: http://portal.acm.org/citation.cfm?id=1523530.1524008

[2] E. Billing and T. Hellstrm, "A formalism for learning from demonstration," *Paladyn. Journal of Behavioral Robotics*, vol. 1, pp. 1–13, 2010, 10.2478/s13230-010-0001-5. [Online]. Available: http://dx.doi.org/10.2478/s13230-010-0001-5

[3] S. Chernova and M. Veloso, "Interactive policy learning through confidence-based autonomy," *J. Artificial Intelligence Research*, pp. 1–25, 2009.

[4] D. H. Grollman and O. C. Jenkins, "Dogged learning for robots," in *International Conference on Robotics and Automation*, Rome, Italy, Apr. 2007, pp. 2483 – 2488.

[5] S. S. Christopher. G. Atkeson, "Robot learning from demonstration," in *Machine Learning: Proceedings of the Fourteenth International Conference (ICML '97)*, J. Douglas H. Fisher, Ed. Morgan Kaufmann, San Francisco, CA, 1997, pp. 12–20.

[6] M. van Lent and J. E. Laird, "Learning procedural knowledge through observation," in *Proceedings of the 1st international conference on Knowledge capture*, ser. K-CAP '01. New York, NY, USA: ACM, 2001, pp. 179–186. [Online]. Available: http://doi.acm.org/10.1145/500737.500765

[7] A. L. Thomaz and C. Breazeal, "Adding guidance to interactive reinforcement learning," in *In Proceedings of the Twentieth Conference on Artificial Intelligence (AAAI)*, 2006.

[8] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[9] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, London, England: The MIT Press, 1998.

[10] M. Nicolescu and M. Mataric, "Learning and interacting in human-robot domains," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 31, no. 5, pp. 419 –430, Sept. 2001.

[11] M. N. Nicolescu and M. J. Matarić, "A hierarchical architecture for behavior-based robots," in *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, ser. AAMAS '02. New York, NY, USA: ACM, 2002, pp. 227–233. [Online]. Available: http://doi.acm.org/10.1145/544741.544798

[12] 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*, ser. AAMAS '03. New York, NY, USA: ACM, 2003, pp. 241–248. [Online]. Available: http://doi.acm.org/10.1145/860575.860614

[13] T. T. Herbert Bay, Andreas Ess and L. V. Gool, "Surf: Speeded up robust features," *Computer Vision and Image Understanding (CVIU)*, vol. 110, no. 3, pp. 346–359, 2008.

[14] R. E. W. Rafael C. Gonzalez and S. L. Eddins, *Digital Image Processing using Matlab*. Prentice Hall, 2003.

[15] G. Bradski and A. Kaehler, *Learning OpenCV*. O'Reilly Media, 2008.