Hierarchical Learning from Demonstration on Humanoid Robots

Keith Sullivan, Sean Luke, and Vittorio Amos Ziparo

***Abstract*— Developing behaviors for humanoid robots is difficult due to the high complexity of programming these robots, which includes repeated trial and error cycles. We have recently developed a learning from demonstration system capable of training agent behaviors from a small number of training examples. Our system represents a complex behavior as a hierarchical finite automaton, permitting decomposition of the behavior into simple, easier-to-train sub-behaviors. The system was originally designed for swarms of virtual agents, but based on recent Robocup experience, we have ported the system to our humanoid robot platform. We discuss the system and the platform, and preliminary experiments involving both novice and expert users in a stateful visual servoing task.**

I. Introduction

The *RoboPatriots* [1] is a team of humanoid robots of our own construction for use in dynamic multirobot tasks such as those found in the RoboCup robot soccer competition [2]. Ordinarily we build behaviors for these robots by hand, often in simulation, then deploy them to the robots in the real environment. This task can be laborious, involving repeated cycles of offline coding and online testing, and so it is attractive to consider an alternative: training the robots to perform tasks through a series of demonstrations. This application of supervised learning to real-time robotics is broadly known as *learning from demonstration*.

We have developed a methodology for rapidly training virtual and game agents to perform complex and robust behaviors involving internal state [3]. This methodology allows us to train both individual agent behaviors and and swarm agent behaviors in a supervised fashion. We have recently ported this methodology to the real-world robot case, focusing (for now) on the single-robot scenario and its application to our humanoid robot platform.

Learning from demonstration on humanoid robots is difficult for a number of reasons. First, the potential number of degrees of freedom and features can, depending on the approach taken to learning, result in a learning task of high dimensionality. Second, such high dimensionality may in turn require a large number of examples, which in turn can translate to many presentations in real time. Third, in our experience training in real time is often fraught with demonstrator error and noise, and this requires approaches to revising training examples, and the learned model, on the fly.

Our goal is ultimately to train humanoids to perform relatively sophisticated behaviors potentially requiring significant internal state, many actions, and a variety of features gleaned from the environment. These all result in a learning space of high dimensionality: but we cannot afford the time necessary to sample such a space effectively. Our approach is to use a variety of methods to reduce the dimensionality and complexity of the learning space as much as possible in order to reconcile these conflicting needs. We do this in several ways:

- *Hierarchical Space Decomposition* Our primary trick is to decompose a behavior into a hierarchy of much simpler behaviors.
- *Per-behavior Feature, State, and Action Reduction* When training a behavior we reduce the learning space to include only those features, actions, and states necessary to learn the given behavior. This dramatically reduces the dimensionality of the space by essentially projecting it into subspaces of minimum dimensionality.
- *Parameterization* Our behaviors are parameterizable. For example, rather than train a behavior such as *go to home base*, we may train a behavior *go to A*, where *A* is a parameter to be bound to some target element at a later date. This allows us to reuse a behavior at several places in the hierarchy, reducing the total amount of training sessions required to learn a complex behavior.
- *Optional Statefulness* Our method learns stateful behaviors by default. However, many simple behaviors require no state at all, and in fact benefit from a lack of state because it simplifies the learning space. We can learn either stateful or stateless behaviors with the same technique.

Our learned model is a hierarchical finite-state automaton (HFA). In this formulation, each learned behavior is a deterministic finite-state automaton in the form of a Moore Machine. Each state in the automaton is itself a behavior. Some of these behaviors may be *basic behaviors*, which are hard-coded; other behaviors are themselves finite-state automata which had been trained at some earlier time. These lower-level automata may themselves be built on even lower-level automata, and so on. No recursion is permitted, so ultimately the leaf nodes in the hierarchy must be basic behaviors.

To learn a behavior, we start with an existing library of behaviors. Each behavior is represented by a state in the finite state automaton to be learned. The demonstrator specifies the features to use in the training task, then directs the robot to perform behaviors from this library, transitioning from state (behavior) to state as necessary. When enough examples have been gathered, the system learns a transition function

K. Sullivan and S. Luke are with the Department of Computer Science, George Mason University, Fairfax, VA (Washington, DC), USA.
{ksulliv2, sean}@cs.gmu.edu

V.A. Ziparo is with the Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Rome, Italy.
ziparo@dis.uniroma1.it

for each state in the automaton based on the examples. The learned automaton may be tested, then updated with additional examples. When training is complete and satisfactory, unused states and their associated behaviors are discarded, and then the automaton is added to the library as a new available behavior.

The demonstrator builds a hierarchical behavior from the bottom up. Initially the only behaviors available in the library are hard-coded basic behaviors. The demonstrator then trains one or more simple automata using these basic behaviors, then more abstract automata which rely on the simple automata (among other behaviors) as states, and so on, ultimately leading to the top-level automaton.

In this paper we first review related work in robotics and learning from demonstration. We then describe our humanoid robot architecture. From there, we describe the learning approach and demonstrate its use on the humanoid architecture. We conclude with a discussion of future work.

II. Related Work

Much of the learning from demonstration literature may be divided into to systems which learn plans and those which learn (usually stateless) policies. The first set of literature is closely related to our own work, and involves learning plans in the form of acyclic graphs of parameterizable behaviors [4], [5], [6], [7], [8]. These graph structures are usually induced from ordered sequences of actions intended to produce a desired goal. As a result, they generally have limited recurrence (simple loops if any iteration at all). In contrast, as we are interested in behaviors rather than goal direction, our model generally assumes a rich level of recurrence, and disregards specific ordering of action sequences.

Other literature describes the learning of policies (usually stateless functions of the form $\pi(\textit{situation}) \rightarrow \textit{action}$) from observing a demonstrator perform various actions when in various world situations. This is naturally cast as a supervised learning task [9], [10], [11], [12], [13], [14]. Some literature instead transforms the problem into a reinforcement learning problem by building a reinforcement signal based on the difference between the agent's actions and those of the demonstrator [15], [16]. Beyond our own approach, it's also possible to include internal state in a policy by learning stateful hidden Markov models (for example, [17]).

Hierarchies and Learning From Demonstration: Hierarchies are hardly new: they are a natural way to achieve layered learning [18] via task decomposition. But use of hierarchy in supervised learning from demonstration is surprisingly uncommon. Even methods which discuss hierarchical models and also discuss learning from demonstration [19] do not often combine the two. Hierarchies of one sort or another are more common in reinforcement learning from demonstration (for example, [20]).

III. Humanoid Architecture

The RoboPatriots are constructed from easily available commercially available hardware resulting in an inexpensive yet powerful platform. Because we are interested in the development of large teams of humanoids, our autonomous humanoid platform is distinguished in its attempt to provide as much functionality as possible for a minimum of cost: typically less than $2000 per robot. Figure 1 shows the hardware architecture and information flow between components. The robot base is a modified Kondo KHR-3HV. Each robot has 3 DOF per arm, 5 DOF per leg, and 2 DOF in the neck. The sixteen Kondo KRS-2555HV digital servos used in the arms and legs produce 14 kg-cm of torque at a speed of 0.14 sec / 60 degrees. The 2555HV servos communicate via a serial protocol and are controlled by the RCB-4 servo controller board. In addition, two KRG-3 single axis gyros connect to the RCB-4. The two Kondo KRS-788HV digital servos used our pan/tilt mount produce 10 kg-cm of torque at a speed of 0.14 sec / 60 degrees. These servos are controlled via PWM.

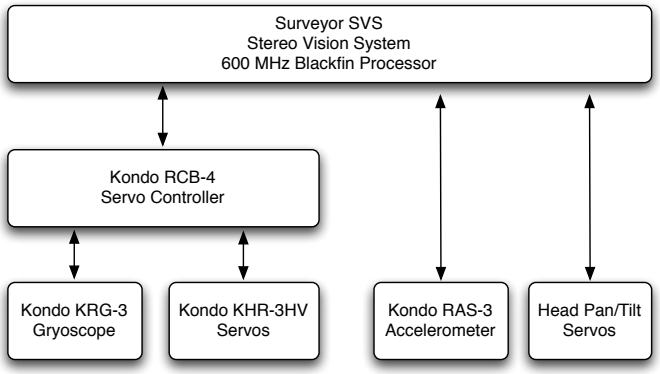


Fig. 1. The hardware architecture of the RoboPatriots, and the information flow between components.

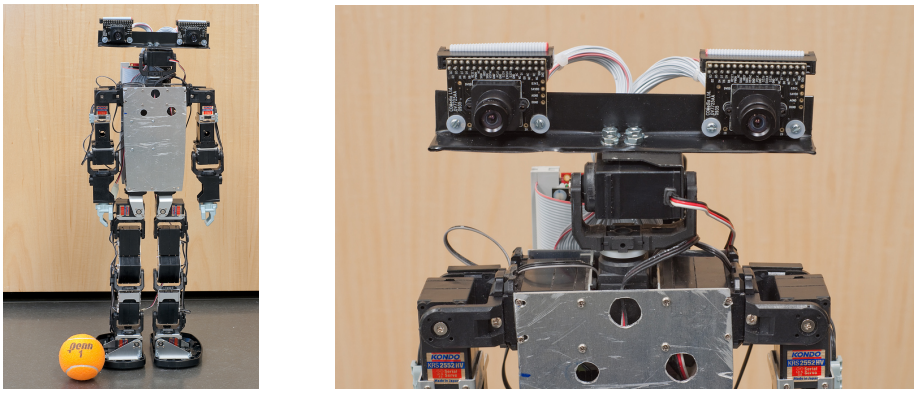
Our main sensor is the Surveyor SVS (Stereo Vision System) [21]. The SVS consists of two OmniVision OV 7725 camera modules connected to two independent 600 MHz Blackfin BF537 processors. The camera modules are mounted on a pan/tilt mount with 10.5 cm separation. See Figure 2(b). Each camera module operates at 640x480 resolution with a 90-degree field of view. The two processors are connected by a dedicated SPI bus, and the camera unit provides a Lantronix Matchport 802.11g wireless unit. Two single axis RAS-2 dual-axis accelerometers connect to the SVS. A custom inverter board allows serial communication at 115200 bps between the RCB-4 and the SVS.

The Blackfin cameras run a simple blob detection algorithm. If the master camera cannot see the ball (caused for, by example, occlusion from the shoulder), it asks the slave camera for its blob information. The Blackfins return the bounding box of the blob, size of the blob (in pixels), and which camera detected the blob.

At present, the learning system runs on a remote laptop which queries the robot for vision data. The laptop also issues commands to the RCB-4 for execution. The demonstrator uses a Nintendo WiiMote during training.

IV. The Training Approach

We train the humanoid robot from the bottom up by iteratively building a library of behaviors. The library initially



(a) Complete RoboPatriot robot (b) Close up of robot head

Fig. 2. 2010 RoboPatriot's robot and close up of stereo vision head and pan/tilt mount.

consists of *atomic behaviors:* hard-coded low-level behaviors such as "go forward" or "kick the ball". We then train a finite-state automaton whose states are associated with behaviors chosen from this library. After training, the automaton itself is saved to the library as a behavior. Thus we are able to first train simple automata, then more abstract automata which include those simple automata among their states, and so on, until we reach sufficiently powerful automata to perform the necessary task. A more detailed description of the system may be found in [3].

A. Model

Our basic model is a hierarchy of finite-state automata in the form of a Moore machines. An automaton is a tuple $\langle S, F, T, B, M \rangle \in \mathcal{H}$ defined as follows:

- $S = \{S_1, ..., S_n\}$ is the set of *states* in the automaton. Among other states, there is one special *start state* S_1, and zero or more *flag states*. Exactly one state is active at any time, designated S_t.
- $B = \{B_1, ..., B_k\}$ is the set of *atomic behaviors*. Each state is associated with either an atomic behavior or *another automaton* chosen from $\mathcal{H}$, with the stipulation that recursion is not permitted. That is, there is a one-to-one mapping $M : S \to \mathcal{H} \cup B$ such that if automaton $H \in \mathcal{H}$ contains another automaton $H' \in \mathcal{H}$ as a child behavior, H' may not in turn include H among its descendant behaviors.
- $F = \{F_1, ..., F_m\}$ is the set of observable *features* in the environment. At any given time each feature has a current *value*: a single number. The collective values of F at $\vec{f_t} = \langle f_1, ..., f_m \rangle$, is called the environment's current *feature vector*. A feature may be of one of three kinds: a *categorical* feature, whose values are chosen from a finite and unordered set; a *continuous* feature, whose values are chosen from a one-dimensional metric continuous space; or a *continuous toroidal* feature, whose values are chosen from a one-dimensional toroidal continuous space.
- $T = F_1 \times ... \times F_m \times S \to S$ is the *transition function* which maps the current state S_t and the current feature vector $\vec{f_t}$ to a new state S_{t+1}.

An automaton starts in its *start* state S_1, whose behavior simply idles. Each timestep, while in state S_t, the automaton first queries the transition function to determine the next state S_{t+1}, transitions to this state, and if $S_t \neq S_{t+1}$, stops performing S_t's behavior and starts performing S_{t+1}'s behavior. Finally, the S_{t+1}'s associated behavior is pulsed to progress it by an epsilon.

The underlying behavior associated with a given state may either be an atomic behavior, or it may itself be another automaton. When the automaton's behavior is "started", it resets its start state to S_1. When pulsed, the automaton itself performs one iteration as described in the previous paragraph.

The purpose of a flag state is only to raise a flag in the automaton to indicate that the automaton believes some condition is now true. We have two such conditions: *done* and *failed*. On transitioning to a flag state, its behavior raises this flag and transitions immediately to the start state. Flag states are useful for creating behavior sequences and are only a convenience, not a requirement.

Features may describe both internal and external (world) conditions, and may be toroidal (such as "angle to goal"), continuous (such as "distance to goal"), or categorical or boolean (such as "goal is visible"). The *done* and *failed* features are boolean features which are true if and only if the current state's associated behavior is itself an automaton, and that automaton has raised its done (or failed) flag.

Our behaviors and features may be optionally assigned one or more parameters: rather than have a behavior called *go to the ball*, we can create a behavior called *goTo(A)*, where *A* is left unspecified. Similarly, a feature might be defined not as *distance to the ball* but as *distanceTo(B)*. If such a behavior or feature is used in an automaton, either its parameter must be bound to a specific *target* (such as "the ball" or "the nearest obstacle"), or it must be bound to some higher-level parent *C* of the automaton itself. Thus finite-state automata may themselves be parameterized. Ultimately to use an automaton, all parameters must be bound to targets. In the study described in this paper, we have chosen not to use the parameterization feature.

B. Training with the Model

Our system learns the transition function T of the automaton. We divide T into disjoint learned functions $T_S(\vec{f_t}) \rightarrow S'$, one for each state S, which map the current feature vector to a new state S'. Each of these is a classifier. At the end of the learning process we have n such classifiers, one for each state $S_1 ... S_n$. At present we are using decision trees with probabilistic leaf nodes for our classifiers.

The learning process works as follows. When the robot is in the *training mode*, it performs the directives given it by the demonstrator. Each time the demonstrator directs the robot to perform a new behavior, the robot stores two example tuples: the first tuple consists of the current state S_t, the state S_{t+1} associated with this new behavior, and the current feature vector $\vec{f_t}$; the second tuple, which provides a default example, consists of S_{t+1}, S_{t+1}, and $\vec{f_t}$. After enough examples have been gathered, the demonstrator switches the robot to the *testing mode*, at which time the classifiers are built from the examples and the robot begins to follow the trained automaton. If at any time the robot performs an incorrect behavior, the demonstrator may switch back to the training mode and correct the robot, which adds further examples. When the demonstrator is satisfied with the performance of the robot, he may then save the automaton to the behavior library and begin working on a new higher-level automaton (which may include the original automaton among its states).

Note that the particular behaviors/states and features used by an automaton may vary from automaton to automaton. This allows us to reduce the feature and state space to be learned on a per-automaton basis.

Some simple learned behaviors do not require internal state and thus the full capacity of a finite-state automaton: and indeed the internal state of the automaton simply makes the learning space unduly complex. In these situations we can simply define each of the T_S to use the same classifier. This effectively reduces the model to a stateless policy $\pi(\vec{f})$. Finally, some learned behaviors may of course require more than one state to be associated with the same behavior, to handle aliased environmental states for example. It is straightforward to create multiple states with the same behavior B: the simplest approach is to quickly train a trivial automaton B' which only does behavior B: now both B' and B may appear as states in higher-level automata, but do exactly the same thing.

V. Example

Our port is very new but we have been able to perform certain preliminary experiments. First, we performed an experiment illustrating how the learning system can be used by novice users. Taking our RoboCup work as motivation, we aimed to teach the robot visual servoing. The goal was for the robot to search for the ball, orient towards the ball by turning the "correct" direction, and walk towards the ball. The robot uses two features from the camera: the x-coordinate of the ball within the frame, and the number of pixels in the ball's bounding box. Finally, the robot has three basic behaviors available to it: turn left, turn right, and walk forward. The robot's head remains fixed looking forward, and the ball does not move. To ensure the ball does not drop out of the bottom of the frame during the experiments, we raised the ball to the robot's eye level (Figure 3).

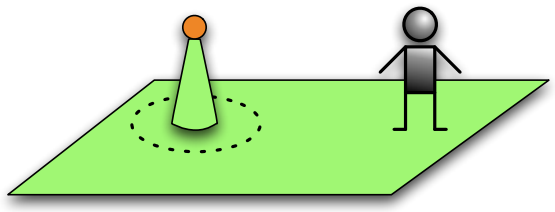
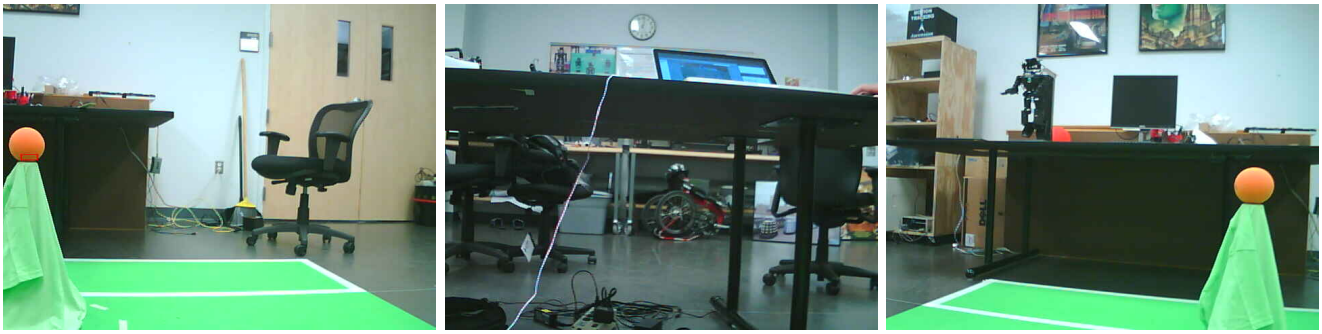


Fig. 3. Experimental setup. The orange ball rests on a green pillar on a green soccer field at eye level with the humanoid robot. The robot must approach to within a short distance of the pillar, as denoted by the dotted line.

Note that this is a very simple example of a behavior which may best be learned in a stateful fashion. When the ball disappears from the robot's field of view, which direction should the robot turn? This could be determined from the x-coordinate of the ball in the immediate *previous* frame, which suggests where the ball may have gone. But if the robot only follows a policy $\pi(\vec{f})$, it does not have this information, but simply knows that the ball has disappeared. Thus π would typically be reduced to just going forwards when the robot can see the ball, and turning (in one unique direction) when it cannot. Half the time the robot will turn the wrong direction, and as a result spin all the way around until it reacquires the ball. This can be quite slow.

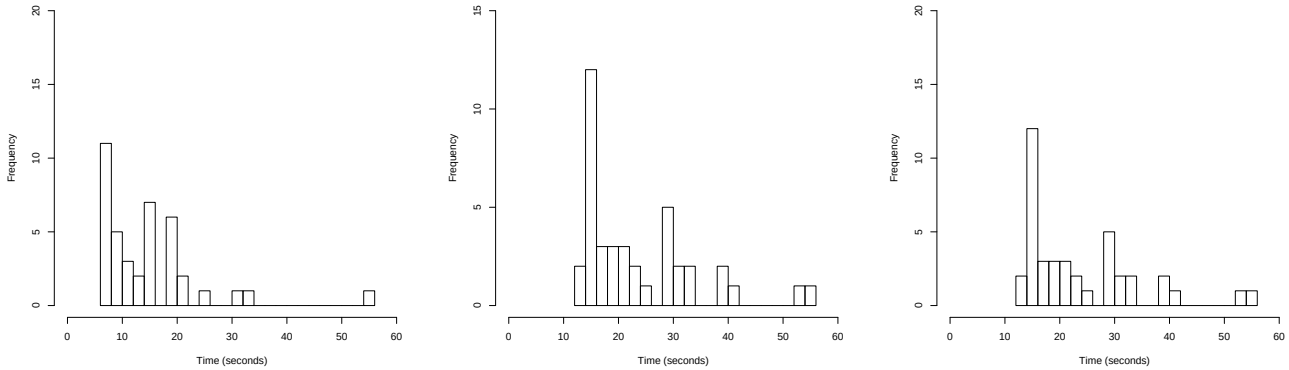
Our learning automaton setup had four states to compensate for this. We had two states, *left* and *right*, which turned left and turned right respectively, but also had two identical states, notionally called *forwardL* and *forwardR*, which both simply moved forward. A demonstrator could use these two states as follows: when the ball is in the left portion of the frame, he instructs the robot to go *forwardL.* When the ball is in the right portion of the frame, he instructs the robot to go *forwardR.* When the ball has disappeared, he instructs the robot to turn appropriately. Ultimately the robot may learn that if the ball disappeared while it was in the *forwardL* state, it should then transition to turning left; and likewise turn right if the ball disappeared while the robot was in the *forwardR* state.

We asked five students to train the robot for five minutes each. The students had no prior experience with the system nor the humanoid robot platform. We instructed the students on how to control the robot and what the features meant, and suggested to them how they might use the *forwardL* and *forwardR* states strategically. Otherwise, the students were were given no further guidance as to how the task should be performed. The students saw feature information from the camera in real time, and could also observe the robot directly. After the five minutes of training were up, we built a learned automaton behavior from the training examples. We then placed the robot at three starting locations with very different ball positions within the frame (see Figure 4).



(a) Starting Position 1 (b) Starting Position 2 (c) Starting Position 3

Fig. 4. Typical views from the three different starting positions. In Starting Position 2, the robot is facing away from the ball.



(a) Starting Position 1 (b) Starting Position 2 (c) Starting Position 3

Fig. 5. Histograms of times to reach the ball from each starting position (compare to Figure 4), of the four successful trials.

The locations were approximately 140 centimeters from the ball. For each trained behavior and each location, we ran 10 independent experiments, where the experiment stopped when the robot was within approximately 15 centimeters of the ball as determined by visual inspection. Our performance measure was the average time necessary for the robot to complete the task. All training and experiments were conducted on the same robot.

In this experiment, four of the five students successfully trained the robot to consistently and robustly approach the ball. The remaining trained behavior never successfully approached the ball, independent of the robot's starting location. Figure 5(a) and (b) show that in most cases the robot can quickly servo to the ball (even if the ball is lost). However, in several cases the robot takes significantly longer to successfully approach the ball, usually due to sensor noise and/or poor training.

We also tested the system's hierarchical ability. In this experiment, an expert (an author of this paper) trained the robot to approach the ball as before, but also to stop when the robot was close to the ball. This was done in two ways. First, the expert attempted to train the robot to do all these tasks in the same automaton. Second, the expert first trained the robot to approach the ball using only the ball position within the frame, and then using this saved approach behavior, trained a simple higher-level automaton in which the robot would approach the ball until it was large enough, then stop. Anecdotal results suggest that the hierarchical approach is much easier to do rapidly than the monolithic approach. Learning the monolithic behavior requires significantly more training exemplars because the joint training space (in terms of states and features) is higher.

VI. Observations and Future Directions

Our ultimate goal is to extend the hierarchical learning procedure to the multiple agent case, by training hierarchies of groups of agents as well as hierarchies of behaviors. This would enable us, for example, to train set plays and other coordinated behaviors in soccer problems. Multiagent coordination would also (in theory) allow the extension of decomposition further to subparts of the agent (arms, legs, etc.), or even to individual servos and effectors.

For now, however, in the real-robot case we must focus on the single-agent situation. The primary challenge to learning in this scenario is the low number of examples that can be collected due to their high cost.

Unlearning: One consequence of the low number of examples is that it is very hard for the learning method to distinguish between an outlier due to a demonstration error

and a crucially important correct outlier in the model. We can compensate for this only to some degree using generalization methods (such as automated tree pruning, in the decision tree case). At present we only add new examples to the model: but ultimately we will need a method to remove or modify invalid examples after they have added to the model.

We see two ways to go about this. First, we might add an undo stack: when the demonstrator performs one or more "typos", so to speak, he could undo these recent incorrect state transitions, reverting the robot to an earlier state, and continue from there. Perhaps more interesting would be observing and correcting incorrect behavior during testing of the behavior. Ideally the system would not only add the corrected results but identify which earlier examples were most responsible for the model choosing an incorrect behavior at that point in time, and remove them.

Demonstration vs. Declaration: The very low number of examples also place techniques like ours in the nebulous region between learning and explicit programming. The more we explicitly add domain knowledge into the problem, through express decomposition and per-behavior feature reduction, the closer we get to essentially programming the robots. Furthermore if the number of examples and dimensionality are sufficiently low, a demonstrator may be (perhaps fairly) accused of largely delimiting the boundaries between behaviors with examples as sentinels. We are not troubled by such an accusation: ultimately the goal is to create a robust behavior through demonstration, and is still reasonable if it contains a mixture of machine inference and explicit domain knowledge and simplification.

VII. Conclusions

We have presented a learning from demonstration system designed for low numbers of examples and its application to a humanoid robot platform. The system takes advantage of a hierarchical decomposition of behaviors, which allows us to use a limited number of training examples to learn complex behaviors. Using a limited number of examples is particularly appealing for learning from demonstration on robots (rather than simulation) where the cost and time associated with extensive data gathering are prohibitive. Novice users were able to train our RoboPatriots humanoids to visually servo towards a ball, and approach the ball with only a few minutes of training.

References

[1] K. Sullivan, C. Vo, S. Luke, and J.-M. Lien, "RoboPatriots: George Mason University 2010 RoboCup team," in *Proceedings of the 2010 RoboCup Workshop*, 2010.

[2] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa, "RoboCup: The robot world cup initiative," in *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, W. L. Johnson and B. Hayes-Roth, Eds. New York: ACM Press, 5–8, 1997, pp. 340–347.

[3] S. Luke and V. Ziparo, "Learn to behave! rapid training of behavior automata," in *Proceedings of Adaptive and Learning Agents Workshop at AAMAS 2010*, M. Grześ and M. Taylor, Eds., 2010, pp. 61 – 68.

[4] R. Angros, W. L. Johnson, J. Rickel, and A. Scholer, "Learning domain knowledge for teaching procedural skills," in *The First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. ACM, 2002, pp. 1372–1378.

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

[6] ——, "Natural methods for robot task learning: instructive demonstrations, generalization and practice," in *The Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. ACM, 2003, pp. 241–248.

[7] P. E. Rybski, K. Yoon, J. Stolarz, and M. M. Veloso, "Interactive robot task training through dialog and demonstration," in *Proceedings of the Second ACM SIGCHI/SIGART Conference on Human-Robot Interaction (HRI)*, C. Breazeal, A. C. Schultz, T. Fong, and S. B. Kiesler, Eds. ACM, 2007, pp. 49–56.

[8] H. Veeraraghavan and M. M. Veloso, "Learning task specific plans through sound and visually interpretable demonstrations," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2008, pp. 2599–2604.

[9] M. Bain and C. Sammut, "A framework for behavioural cloning," in *Machine Intelligence 15*. Oxford University Press, 1996, pp. 103–129.

[10] D. C. Bentivegna, C. G. Atkeson, and G. Cheng, "Learning tasks from observation and practice," *Robotics and Autonomous Systems*, vol. 47, no. 2-3, pp. 163–169, 2004.

[11] S. Calinon and A. Billard, "Incremental learning of gestures by imitation in a humanoid robot," in *Proceedings of the Second ACM SIGCHI/SIGART Conference on Human-Robot Interaction (HRI)*, C. Breazeal, A. C. Schultz, T. Fong, and S. B. Kiesler, Eds. ACM, 2007, pp. 255–262.

[12] J. Dinerstein, P. K. Egbert, and D. Ventura, "Learning policies for embodied virtual agents through demonstration," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2007, pp. 1257–1252.

[13] M. Kasper, G. Fricke, K. Steuernagel, and E. von Puttkamer, "A behavior-based mobile robot architecture for learning from demonstration," *Robotics and Autonomous Systems*, vol. 34, no. 2-3, pp. 153–164, 2001.

[14] J. Nakanishi, J. Morimoto, G. Endo, G. Cheng, S. Schaal, and M. Kawato, "Learning from demonstration and adaptation of biped locomotion," *Robotics and Autonomous Systems*, vol. 47, no. 2-3, pp. 79–91, 2004.

[15] A. Coates, P. Abbeel, and A. Y. Ng, "Apprenticeship learning for helicopter control," *Communications of the ACM*, vol. 52, no. 7, pp. 97–105, 2009.

[16] Y. Takahashi, Y. Tamura, and M. Asada, "Mutual development of behavior acquisition and recognition based on value system," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2008, pp. 386–392.

[17] D. Kulic, D. Lee, C. Ott, and Y. Nakamura, "Incremental learning of full body motion primitives for humanoid robots," in *8th IEEE-RAS International Conference on Humanoid Robots*, Dec. 2008, pp. 326–332.

[18] P. Stone and M. M. Veloso, "Layered learning," in *11th European Conference on Machine Learning (ECML)*, R. L. de Mántaras and E. Plaza, Eds. Springer, 2000, pp. 369–381.

[19] M. N. Nicolescu and M. J. Mataric, "A hierarchical architecture for behavior-based robots," in *The First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. ACM, 2002, pp. 227–233.

[20] Y. Takahashi and M. Asada, "Multi-layered learning system for real robot behavior acquisition," in *Cutting Edge Robotics*, V. Kordic, A. Lazinica, and M. Merdan, Eds. Pro Literatur, 2005.

[21] Surveyor Corporation, "Surveyor stereo vision system (SVS)," http://www.surveyor.com/stereo/stereo_info.html, 2010.