Real-Time Training of Team Soccer Behaviors

Keith Sullivan and Sean Luke

Department of Computer Science, George Mason University
4400 University Drive MSN 4A5, Fairfax, VA 22030 USA
{ksulliv2,sean}@cs.gmu.edu

Abstract. Training robot or agent behaviors by example is an attractive alternative to directly coding them. However training complex behaviors can be challenging, particularly when it involves interactive behaviors involving multiple agents. We present a novel hierarchical learning from demonstration system which can be used to train both single-agent and scalable cooperative multiagent behaviors. The methodology applies manual task decomposition to break the complex training problem into simpler parts, then solves the problem by iteratively training each part. We discuss our application of this method to multiagent problems in the humanoid RoboCup competition, and apply the technique to the keepaway soccer problem in the RoboCup Soccer Simulator.

1 Introduction

In this paper we describe a Learning from Demonstration (LfD) system called *Hierarchical Training of Agent Behaviors*, or HiTAB, and its application to problems in RoboCup. In LfD, an agent learns a behavior in real-time based on provided examples from a human demonstrator, usually through teleoperation of the agent. The goal of HiTAB is to learn complex stateful behaviors in the form of hierarchical finite-state automata (HFA), in real time, based on a small number of samples provided by a demonstrator. HiTAB can be applied both to single-agent training and to command hierarchies of arbitrarily large swarms of agents. We have used HiTAB to train humanoid robots, a team of differential-drive robots, and a variety of virtual agents, up to thousands of agents at a time, on many different problems.

The distinguishing feature of (single-agent) HiTAB is its approach to learning behaviors based on a small number of samples, which in turn enables rapid training in areas, such as behavior-based robotics, where samples are sparse. HiTAB achieves this through manual task decomposition, breaking a complex joint finite-state automaton into a hierarchy of much smaller automata to be iteratively learned and composed. Though HiTAB uses standard classification techniques to learn these automata, the resulting learned automata are often very simple, indeed trivial. This is exactly the goal: simple automata in turn define a low-dimensional space which can be learned with a small number of samples.

All machine learning methods combine some degree of automated machine induction and human domain knowledge. At the very least, a human is choosing an appropriate representation and bias. HiTAB lies at the far end of the

X. Chen et al. (Eds.): RoboCup 2012, LNAI 7500, pp. 356–367, 2013.

induction/knowledge tradeoff. By manually decomposing the problem into a hierarchy of subproblems, the experimenter is defining the automaton's general architecture: HiTAB's machine learning is filling in the gaps. This puts HiTAB somewhere between machine learning and outright programming by demonstration.

In 2011 we applied HiTAB to train a humanoid kid-sized robot soccer behavior the night before the RoboCup competition, then fielded it in the competition alongside hardcoded robot behaviors. Our ultimate goal is to train *all* the top-level behaviors in our robot soccer team while at the competition.

In this paper we demonstrate another application of HiTAB to the Robocup domain: the keepaway problem in simulated soccer, using the RoboCup Soccer Simulator. In this problem, a group of *keepers* must collectively pass the ball amongst one another so as to prevent another team, the *takers*, from acquiring it. This problem requires the experimenter to train a homogeneous but interactive behavior among three agents.

The rest of this paper is organized as follows. We first discuss related work, then detail how HiTAB works in the single-agent case (for details on the multi-agent/swarm case, see [28]). We then discuss our prior and current attempts in the RoboCup Kid-Size Humanoid league. Then, we show how HiTAB may be applied to the keepaway problem in the RoboCup Soccer Simulator.

2 Related Work

Learning from Demonstration is a method to train agents by having a human demonstrator perform actions for the agent [1,2]. Since the agent is given the proper action to perform in a given situation, LfD is, broadly speaking, a supervised learning problem, though authors often use reinforcement learning, with a reward signal based on how closely a learned solution matches a trajectory shown by the demonstrator [7,19]. A variation of LfD, called *imitation learning*, attempts to mimic a demonstrator's actual actions (as a human) rather than observe the demonstrator teleoperate the robot [14,15].

The LfD literature may be divided into two categories: those which learn *plans* [22,31] and those which learn (usually stateless) *policies* [3,19] (for stateful examples see [8,13]). In most cases, the plan literature builds sparse machines describing occasional changes in behavior, whereas many, but not all, policy methods learn fine-grained changes in action, such as might be found in trajectory planning or control. The crucial difference between the two is that a plan learner may receive a new sample only when the user occasionally specifies a new behavior to perform; whereas trajectory policy learners may be inundated with samples with every slight modification or course correction. This in turn has an impact on the difficulty of learning: plan methods must often deal with an extreme sparsity in samples. Our work lies in the plan method category.

Like our own work discussed here, a number of other authors construct complex behaviors via scaffolding: breaking the task into smaller, easier to learn pieces and combining these smaller tasks to form complex behaviors [16,25,27].

Our approach requires manual decomposition and reassembly, but this is not the only approach. Instead of the demonstrator specifying how to combine simpler behaviors, the idea of *behavior fusion* has the agent learn how to automatically combine simple behaviors into more complex behaviors [20,21]. Closely related is the notion of automatic task decomposition which determines how to break complex behavior into simpler components [9,10].

Our work is distinguished in its application to both single- and multi-agent scenarios. Though in this paper we focus largely on single-agent learning, it is done in a collective environment. Multiagent learning from demonstration is a very difficult problem because of the gulf which exists between the desired emergent macrophenomena and the per-agent microbehaviors which give rise to them. This is particularly problematic for supervised learners, because in order to learn in a supervised fashion each agent must receive the *correct action* as a microbehavior: but the experimenter does not know what microbehaviors should be done to achieve the desired macrophenomenon, and parallel control of large numbers of agents is also difficult. As a result the vast majority of multiagent research has focused on reward-based techniques (reinforcement learning, evolutionary computation, etc.) rather than supervised learning [23]. Of those supervised learning methods used, most fall into the category of *agent modeling*, where agents learn about each other rather than a task given by a demonstrator. Still, there has been some work in multiagent LfD. Chernova et al. use confidence estimation to train multiple robots individually and rely on emergent multirobot behavior to accomplish the task [5,6]. A similar approach was used to train Sony AIBO robots to play soccer [4,11,12].

3 Hierarchical Training with a Single Agent

HiTAB's basic model consists of hierarchical finite-state automata. Each state in a HiTAB automaton corresponds to an agent behavior: and when in a given state, the agent performs the associated behavior. Behaviors may be either atomic behaviors hard-coded in the agent, or may themselves be other finite-state automata. Every automaton begins in its *Start* state, a blank state which immediately transitions to some other state. Automata may also have *flag states*, such as the *Done* state, which raises a flag indicating that the automaton believes it is done, then transitions to the *Start state*. Flag states allow parent automata to detect completion of sub-behaviors as if they were sensor features.

Transitions between states are controlled by *transition functions* which map the current state and feature vector to a new state. HiTAB's states are fixed (they are the current behaviors in its library), but it learns a transition function for every state in the automaton.

Learning the transition function is a classification task where the class labels are the individual states and attributes are the environmental features. While many classification algorithms are applicable, HiTAB at present uses a version of the C4.5 decision tree algorithm [24] with probabilistic leaf nodes. Decision trees nicely handle different types of data (e.g., continuous, toroidal, and categorical

data), and do not require scaling of features relative to one another. Additionally, many agent tasks can be approximated by rectangular partitions of the feature space, which makes them a good target for decision trees. Leaf nodes in decision trees traditionally deterministically compute the class using the plurality of examples which reach that leaf. HiTAB instead uses a probability distribution over the classes appearing at a leaf node.

The motivation behind HiTAB was to develop a LfD system which could rapidly train complex, stateful agent behaviors in real time. As mentioned before, training complex agent behaviors typically requires many samples. HiTAB employs task decomposition to reduce the number of samples necessary to produce a detailed behavior. It does this in various ways:

- Behaviors (which take the form of finite-state automata) are organized into a hierarchy, allowing the operator to decompose a large joint behavior into many simpler behaviors which are trained independently, then reused in different situations by higher-level trained behaviors.
- Each behavior may be trained solely in the context of features and lower-level behaviors relevant to it. In contrast, training a single large behavior would require the joint of all basic behaviors and features, resulting in a much higher dimensional learning space. This results in dramatic savings: typically decomposition allows the dimensionality, and corresponding need for samples, to decrease from exponential to polynomial sizes.
- Behaviors and sensor features are parameterizable. Thus an operator may train a behavior such as *go to X*, and later reuse it as *go to the ball* or *go to the nearest wall*, etc.
- Incorrectly trained behaviors may be retrained without having to retrain the entire top-level joint behavior.

Running HiTAB. An automaton starts in its *Start* state. 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 new state, and if $S_t \neq S_{t+1}$, stops performing S_t's behavior and starts performing S_{t+1}'s behavior. It then performs one pulse of the state's underlying behavior: if the behavior is an atomic behavior such as "go forward", this might result in a single step forward. If the behavior is itself an HFA, this results in recursively performing the aforementioned transition and pulsing procedure on the underlying automaton.

Training with HiTAB. To begin training an automaton, the operator first selects the features to be used as attributes for the automaton's transition classifiers. Training then iterates between a *training mode* and a *testing mode.*

In the training mode, the demonstrator is in control. Each time the demonstrator directs the agent to perform a new behavior, the agent begins performing it, and also records a tuple $\langle S_t, \vec{f_t}, S_{t+1} \rangle$ which stores the current feature vector, along with the previous and new states. If state S_{t+1} has a behavior designed to be executed exactly once, then no additional examples are recorded. Otherwise, a useful *default example* is stored of the form $\langle S_{t+1}, \vec{f_t}, S_{t+1} \rangle$. This helps

HiTAB's classifier realize that S_{t+1} should be continuously performed unless, as indicated by a further example, the situation changes again.[1]

Ultimately the demonstrator switches to the *testing mode*, which causes the transition functions to be built from the collected examples. For a given state S_i, HiTAB reduces all examples of the form $\langle S_i, \vec{f_t}, S_j \rangle$ to samples of the form $\langle \vec{f_t}, S_j \rangle$ which are input to the classifier ($\vec{f_t}$ are the features and S_j are the labels). The resulting classifier defines the transition function for outgoing edges from S_i.

The agent then starts following the learned behavior autonomously. If the demonstrator observes the agent performing an incorrect behavior, he may step in and switch the agent back to training mode to collect additional examples.

Ultimately the trained behavior is saved to the behavior library. To do this, HiTAB first trims unused states and features. In addition, any parameterized behaviors and features are bound to a target (e.g., "nearest obstacle"), or to a parameter of the automaton itself. After saving to the library, the behavior may be used as a state in a higher-level automaton to be learned at a later time.

Formal Model. The HFA is at the heart of HiTAB. An automaton is a tuple $\langle S, B, F, T, G \rangle \in \mathcal{H}$ defined as follows:

- $S = \{S_1, \ldots, S_n\}$ is the set of *states* in the automaton. Included is one special state, the *Start* state S_0, and zero or more *flag states* (such as *Done*). Exactly one state is active at a time, designated S_t. The purpose of a flag state is simply to raise a flag in the automaton to indicate that the automaton believes that some condition is now true. Flags in an automaton appear as optional features in its *parent* automaton.
- $B = \{B_1, \ldots, B_k\}$ is the set of *basic behaviors*. Each state is associated with either a basic behavior or *another automaton* from $\mathcal{H}$, though recursion is not permitted.
- $F = \{f_1, \ldots, f_m\}$ is the set of observable *features* in the environment. At any given time each feature has a numerical value. The collective values of F at time t is the environment's *feature vector* $\vec{f_t} = \langle f_1, ..., f_m \rangle$.
- $T = \vec{f_t} \times S \rightarrow S$ 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}.
- Optional free variables (parameters) $G = \{G_1, \ldots, G_n\}$ for basic behaviors and features generalize the model: each behavior B_i and feature f_i are replaced as $B_i(G_1, \ldots, G_n)$ and $f_i(G_1, \ldots, G_n)$. The evaluation of the transition function and the execution of behaviors are based on ground instances of the free variables. For example, 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

[1] Default examples are distinguished in HiTAB's decision tree mechanism: if the decision tree is choosing to place its pivot between a default example and a non-default example, the pivot is placed immediately adjacent to the non-default example. This differs from the normal case, where the pivot is placed exactly half-way between the two examples.

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 of the automaton itself. Thus HFAs may themselves be parameterized.

4 Training Teams of Agents

We have applied HiTAB in three ways to train teams or swarms of agents to perform group behaviors:

1. A single agent behavior is trained in isolation, then distributed to multiple agents. The behavior does not require agent interaction and can be essentially done in parallel.
2. A homogeneous behavior is trained to be used by multiple coordinated agents. For example, the agents learn to form ranks, or work together to capture a prey. Because the behavior must interact with other agents, this kind of training can be challenging. In lieu of training multiple agents simultaneously, we have taken a new approach, which we call *behavior bootstrapping*. Here, we train an agent to perform a rudimentary version of the desired behavior in the context of do-nothing teammates. We then distribute this rudimentary behavior to the teammates, then train the agent on a slightly more capable behavior in the context of teammates performing the rudimentary behavior. We then distribute the slightly more capable behavior to the teammates, and train an even more capable behavior, and so on, until the desired sophisticated behavior is achieved. This approach is only really effective with a relatively small number of agents.
3. A collection of coordinated homogeneous behaviors are trained among a swarm of a (potentially very large) number of agents. The way this is done is by organizing the swarm into a command hierarchy: small groups of agents are assigned a commander (a virtual agent); then small groups of commanders are assigned a commander, and so on until the whole swarm is structured as a tree. We use HiTAB to train commanders in essentially the same way as real (leaf node) agents are trained. A commander's atomic behaviors correspond to the learned top-level behaviors of its subordinate agents, and when it begins to perform an atomic behavior it directs its subordinates to all begin performing the equivalent top-level behavior. The resulting hierarchical command structure strikes a mid-ground between a fully distributed swarm and a fully centralized one.

Examples of the third approach may be found in [28]. Because the number of agents is small (three teammates), in this paper we concentrate on the first two approaches, and particularly on the novel use of behavior bootstrapping to train three agents in concert.

We note that these methods, or at least the last two, fall under the multiagent learning subcategory defined in [23] as *team learning*, whereby a single learner is

used at any particular time to train a team of agents. This is in contrast to *concurrent learning*, where multiple learners are simultaneously operating. Further, we note that the group behaviors described above are all homogeneous. However ultimately we aim to be able to train heterogeneous or mixed homogeneous and heterogeneous behaviors in large numbers of agents.

5 Team Robot Training of Humanoids at RoboCup

The goal of HiTAB is to allow real-time training of behaviors fast enough that it can be done in the field and on-the-fly by an operator. This has been demonstrated in previous work [18,29] for virtual agents, a single differential-drive robot, and a humanoid robot. But it had never been tested in a real-world challenge scenario. Thus as a proof of concept we fielded HiTAB-trained robots in RoboCup 2011.

Since 2009, we have competed in the RoboCup Kid-sized Humanoid League with the RoboPatriots [30]. Our humanoid robots have top-level behaviors in the form of hard-coded hierarchical finite-state automata. Such behaviors include locating the ball, servoing and approaching the ball, aligning with the goal, kicking and reattempting kicks, and so on.

On the soccer field the night before the 2011 competition, we deleted one of the hard-coded behaviors (servoing and approaching the ball) and trained a behavior in its place. We did this by directly teleoperating the humanoid on the field. We then saved out the trained behavior, and during the competition, the robots loaded this behavior from a file and used it in an interpreter along side the remaining hard-coded behaviors.

This behavior was not complex: it was meant as a proof of concept. However, the learned behavior worked perfectly. After discussions with colleagues at the competition, we have come to the conclusion that, to the best of our knowledge, this is the first time a team at RoboCup has used a behavior taught to the robots on the field at the competition itself.

For RoboCup 2012, our goal is to train most, if not all, of the top-level behaviors on the field at the competition. In essence, we will attempt to teach the team how to play soccer the night before the competition.

6 Team Robot Training of Keepaway Soccer

In preparation for the RoboCup 2012 humanoid goal, we applied HiTAB to the task of simulated soccer keepaway in the RoboCup Soccer Simulator. In the keepaway problem, a team of *keepers* tries to maintain possession for as long as possible from a team of *takers*. The two teams compete in a bounded area (in our case, a 20m $\times$ 20m box) within a regular soccer field in the RoboCup Soccer Simulator. In our version of keepaway, the agents have 360 degree and infinite view and cannot collide with the ball. We did not permit our keepers to dribble.

The keepaway problem presents several challenges. First, its limited inter-agent communication requires agents to learn independently, but the resulting

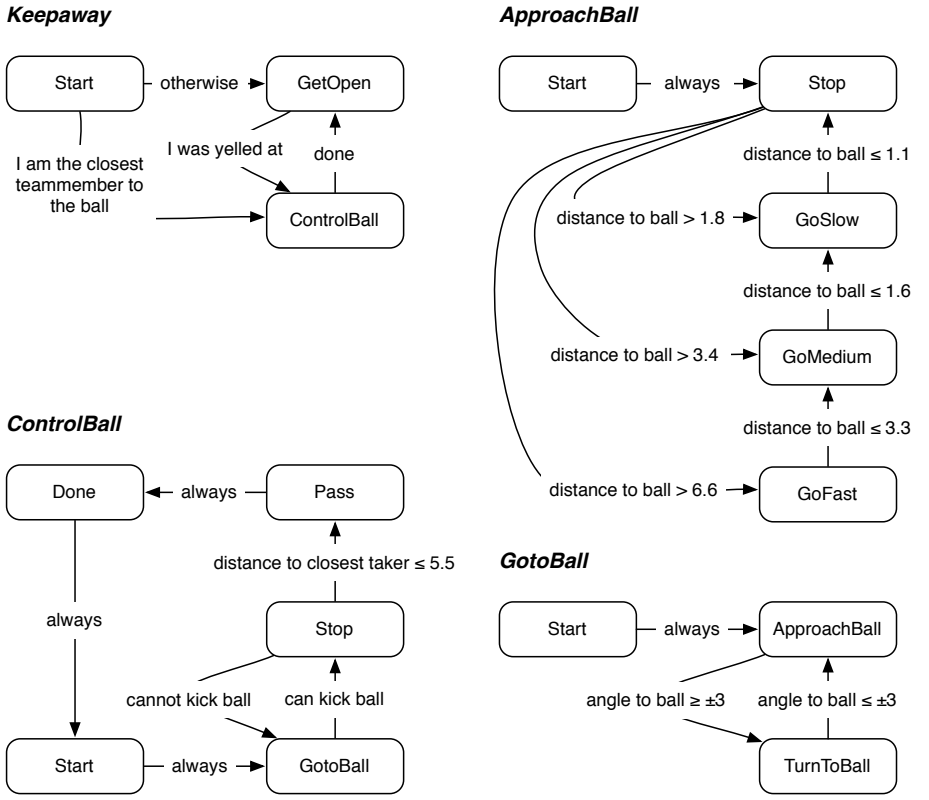

Fig. 1. Four automata trained for the Keepaway Problem. In each case, the automaton begins in *Start*. The *Done* behavior does nothing but raises a *done* flag in the automaton's parent, which is detected by the *done* feature (compare ControlBall with Keepaway). Real-valued numbers shown are the result of the training examples provided.

behaviors require coordination. Second, keepaway (and soccer in general) has a large state space. Third, the RoboCup Soccer Simulator injects random noise in agents' actions and sensors.

In this example we used HiTAB plus behavior bootstrapping to learn coordinated behaviors among the three agents. We first manually decomposed the keeper behaviors into a structure similar to Stone et al [26]. The keepers were provided with the following hard-coded behaviors: *GoFast, GoMedium, GoSlow, Stop, Pass, GetOpen,* and *TurnToBall.*

- The *GoFast, GoMedium,* and *GoSlow* functions moved the keeper straight ahead at velocities of 100, 90, and 75 respectively.
- The *Pass* function kicked the ball to the "most open" teammate. Openness was defined by determining the maximum angle subtended by the vector between the passer and receiver, and the vector between the passer and the closest taker. Kick strength accounted for friction and was proportional to

the distance between the passer and receiver[2]. The passer would then yell to the receiver to inform him of the incoming ball.
- *GetOpen* moved the keeper away from its teammates via a simple potential field, but constrained to be within 10 meters of the center of the box.
- *TurnToBall* rotated the keeper to directly face the ball.

The features we used were: *DistanceTo(X), DirectionTo(X), IWasYelledAt, BallIsKickable,* and *ATeammateIsCloserToBall*, where X could be set to either the ball, the closest keeper, or closest taker. The binary features *IWasYelledAt, BallIsKickable* and *ATeammateIsCloserToBall* were true when a yell message was received (from a passer), the ball was within kicking range, or another keeper was closer to the ball, respectively, and were false otherwise.

Given these basic behaviors and features, we trained four automata in the following order, as shown in Figure 1:

1. **ApproachBall:** A P-Controller in automaton form, based on *GoFast*, *GoMedium*, *GoSlow*, *Stop*, and *DistanceTo(ball)*. This automaton attempted to move the agent until it was within kicking distance of the ball location.
2. **GotoBall:** Iterated between *ApproachBall* and *TurnToBall*, using the angle to ball. This automaton attempted to servo on the ball location, and did not require state.
3. **ControlBall:** Used the *GotoBall*, *Stop*, and *Pass* behaviors, the optional *Done* state, and the *DistanceTo(Closest Taker)* and *BallIsKickable* features. This automaton servoed on the ball, waited until a taker was sufficiently close, then passed the ball, plus some error handling.
4. **Keepaway:** The top-level automaton, used *GetOpen* and *ControlBall*, plus three features: *ATeammateIsCloserToBall*, *IWasYelledAt*, and *Done*. This automaton would initially either get open or take control of the ball depending on whether the agent was initially closest to the ball. It then iterated between the *GetOpen* and *ControlBall* behaviors depending on whether the agent believed it was in control of the ball at any given time.

Keepaway was notable in that it was trained via a simple form of behavioral bootstrapping. We began by training a single agent to either go to *ControlBall* or *GetOpen* when started. We then distributed this behavior to all the agents. We next restarted the game, which caused one agent to go to *ControlBall* while the others went to *GetOpen*. We further trained the ball-controlling agent to pass the ball and then get open, and then copied that behavior to all agents. After restarting again, we trained a single open agent to transition to *ControlBall* when yelled at, and distributed the final version of the behavior.

7 Experiments

We ran our learned keepaway behaviors for 200 episodes. An episode ended when the takers gained possession of the ball, or when the ball was kicked out of the

[2] The exact kick strength computation followed the U Texas Austin code used in `http://www.cs.utexas.edu/~AustinVilla/sim/keepaway/`.

Table 1. Number of data points to train the final behaviors, and an approximation of the total time to train the final behaviors.

Behavior	Number of Examples	Time to Train (minutes)
ApproachBall	18	10
GotoBall	10	10
ControlBall	11	45
Keepaway	10	90

keepaway box. The takers were from [26], but running at one quarter the speed of the original. All experiments were conducted using the MASON multiagent simulation package [17] (running HiTAB) plus the RoboCup Soccer Simulator.

Our trained keepers maintained possession for an average of 14.6 ± 0.87 seconds, and completed an average of 3.8 passes per episode. We were able to train the behaviors to play successfully: but obviously they will require more tweaking to keep the ball away from the takers for a longer duration.

We also wanted to examine how quickly behaviors could be trained using HiTAB. Table 1 shows the length of time spent actually training the agents (including collecting the samples and constructing the HFA), and the number of examples collected for the final trained model. Typically, the demonstrator required several iterations to train the final behavior due to demonstrator error or experimentation with different ways of achieving the desired behavior (and thus different automata structures). *Keepaway* took longer to train due to the behavioral bootstrapping involved. In particular, the majority of the time was spent determining how to manage the system such that two agents were in the correct configuration to collect appropriate data: inability to manipulate the agents was largely a GUI issue which can be remedied in the future.

We believe the experiments show HiTAB's ability to train a complex multiagent behavior in a reasonable timeframe, and without requiring a significant amount of data. Based on these results, we think our goal to train the RoboPatriot soccer behavior in Mexico City is viable.

8 Conclusions and Future Work

This paper demonstrated a supervised learning from demonstration system capable of training complex behaviors in a multiagent problem domain in real time. Our system, HiTAB, achieves this through manual behavior decomposition, per-sub-behavior feature reduction, and machine learning through classification. HiTAB's purpose is to do learning on a very small number of samples. Its use of behavior decomposition places us on the far end of what may be reasonably called machine learning, and very close to explicit programing by example. Multiagent supervised training (as opposed to user modeling) is unusual, and HiTAB is nearly unique in tackling this problem.

The primary difficulties we encountered in adapting HiTAB for the soccer keepaway problem centered on representation: HiTAB employs classification

rather than regression, yet many of the behaviors in the robot soccer domain benefit from regression. For example, the *GetOpen* behavior computed the direction to go based on a potential field, which HiTAB could not easily do. Intelligent interception would also benefit from regression, as was originally demonstrated in [27]. It is reasonable to use HiTAB to train higher-level behaviors composed from lower-level behaviors which were either hard-coded or developed through another learning technique (such as a regression technique). Finally, our ultimate goal is to develop HiTAB towards heterogeneous multiagent behaviors. In the keepaway problem there is little need for heterogeneity: but in the Robocup Humanoid leagues it is plausible for all three robots to be heterogeneous, either by differences in capability (goalies) or simply behavior (a forward versus a midfielder).

References

1. Argall, B.D., Chernova, S., Veloso, M., Browning, B.: A survey of robot learning from demonstration. Robotics and Autonomous Systems 57, 469–483 (2009)
2. Atkeson, C.G., Schaal, S.: Robot learning from demonstration. In: Fisher, D.H. (ed.) Proceedings of International Conference on Machine Learning (ICML), pp. 12–20. Morgan Kaufmann (1997)
3. Bentivegna, D.C., Atkeson, C.G., Cheng, G.: Learning tasks from observation and practice. Robotics and Autonomous Systems 47(2-3), 163–169 (2004)
4. Browning, B., Xu, L., Veloso, M.: Skill acquisition and use for a dynamically-balancing soccer robot. In: Proceedings of the American Association of Artificial Intelligence (AAAI), pp. 599–604 (2004)
5. Chernova, S.: Confidence-based Robot Policy Learning from Demonstration. Ph.D. thesis, Carnegie Mellon University (2009)
6. Chernova, S., Veloso, M.: Confidence-based multi-robot learning from demonstration. International Journal of Social Robotics 2, 195–215 (2010)
7. Coates, A., Abbeel, P., Ng, A.Y.: Apprenticeship learning for helicopter control. Communications of the ACM 52(7), 97–105 (2009)
8. Dixon, K., Khosla, P.K.: Learning by observation with mobile robots: A computational approach. In: Proceedings of IEEE International Conference on Robotics and Automation (ICRA) (2004)
9. Eyharabide, V., Amandi, A.: Automatic task model generation for interface agent development. Inteligencia Artificial 9(26), 49–57 (2005)
10. Garland, A.: Learning hierarchical task models by demonstration. Tech. Rep. TR-2001-03, Mitsubishi Electric Research Laboratories (2001)
11. Grollman, D., Jenkins, O.: Dogged learning for robots. In: Proceedings of IEEE International Conference on Robotics and Automation (ICRA), pp. 2483–2488. IEEE (2007)
12. Grollman, D.H., Jenkins, O.C.: Can we learn finite state machine robot controllers from interactive demonstration? In: Sigaud, O., Peters, J. (eds.) From Motor Learning to Interaction Learning in Robots. SCI, vol. 264, pp. 407–430. Springer, Heidelberg (2010)
13. Hovland, G., Sikka, P., McCarragher, B.: Skill acquisition from human demonstration using a hidden markov model. In: Proceedings of IEEE International Conference on Robotics and Automation (ICRA), pp. 2706–2711. IEEE (1996)

14. Ijspeert, A.J., Nakanishi, J., Schaal, S.: Movement imitation with nonlinear dynamical systems in humanoid robots. In: Proceedings of IEEE International Conference on Robotics and Automation (ICRA), pp. 1398–1403 (2002)
15. Jenkins, O., Mataric, M., Weber, S.: Primitive-based movement classification for humanoid imitation. In: Proceedings of the IEEE-RAS International Conference on Humanoid Robotics (Humanoids) (2000)
16. Lockerd, A., Breazeal, C.: Tutelage and socially guided robot learning. In: Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS), vol. 4, pp. 3475–3480. IEEE (2004)
17. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., Balan, G.: MASON: A multi-agent simulation environment. Simulation 81(7), 517–527 (2005)
18. Luke, S., Ziparo, V.: Learn to behave! rapid training of behavior automata. In: Grześ, M., Taylor, M. (eds.) Proceedings of Adaptive and Learning Agents Workshop at AAMAS 2010, pp. 61–68 (2010)
19. Nakanishi, J., Morimoto, J., Endo, G., Cheng, G., Schaal, S., Kawato, M.: Learning from demonstration and adaptation of biped locomotion. Robotics and Autonomous Systems 47(2-3), 79–91 (2004)
20. Nicolescu, M., Jenkins, O., Olenderski, A.: Behavior fusion estimation for robot learning from demonstration. In: Proceedings of Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications. IEEE Computer Society (2006)
21. Nicolescu, M., Jenkins, O., Stanhope, A.: Fusing robot behaviors for human-level tasks. In: Proceedings of the International Conference on Development and Learning (ICDL), pp. 76–81. IEEE (2007)
22. Nicolescu, M.N.: A Framework for Learning from Demonstration, Generalization and Practice in Human-Robot Domains. Ph.D. thesis, University of Southern California (2003)
23. Panait, L., Luke, S.: Cooperative multi-agent learning: The state of the art. Autonomous Agents and Multi-Agent Systems 11(3), 387–434 (2005)
24. Quinlan, J.R.: C4.5: Programs for Machine Learning, 1st edn. Morgan Kaufmann Series in Machine Learning. Morgan Kaufmann (January 1993)
25. Saunders, J., Nehaniv, C., Dautenhahn, K.: Teaching robots by molding behavior and scaffolding the environment. In: Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction (HRI) (2006)
26. Stone, P., Sutton, R.S., Kuhlmann, G.: Reinforcement learning for RoboCup-soccer keepaway. Adaptive Behavior 13(3), 165–188 (2005)
27. Stone, P., Veloso, M.: Layered learning and flexible teamwork in robocup simulation agents. In: Veloso, M.M., Pagello, E., Kitano, H. (eds.) RoboCup 1999. LNCS (LNAI), vol. 1856, pp. 495–508. Springer, Heidelberg (2000)
28. Sullivan, K., Luke, S.: Learning from demonstration with swarm hierarchies. In: Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS) (2012)
29. Sullivan, K., Luke, S., Ziparo, V.A.: Hierarchical learning from demonstration on humanoid robots. In: Proceedings of the Humanoid Robots Learning from Interaction Workshop at Humanoids (2010)
30. Sullivan, K., Russell, K., Andrea, K., Stout, B., Luke, S.: RoboPatriots: George Mason University 2012 RoboCup team. In: Proceedings of the 2012 RoboCup Workshop (2012)
31. Veeraraghavan, H., Veloso, M.M.: Learning task specific plans through sound and visually interpretable demonstrations. In: Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS), pp. 2599–2604. IEEE (2008)