Department of Computer Science
George Mason University Technical Reports

4400 University Drive MS#4A5
Fairfax, VA 22030-4444 USA
http://cs.gmu.edu/ 703-993-1530

Hierarchical Multi-Robot Learning from Demonstration

Keith Sullivan
ksulliv2@cs.gmu.edu

Sean Luke
sean@cs.gmu.edu

Technical Report GMU-CS-TR-2011-5

Abstract

Developing robot behaviors is a tedious task requiring multiple coding, trial, and debugging cycles. This makes attractive the notion of learning from demonstration, whereby a robot learns behaviors in real time from the examples of a demonstrator. Learning from demonstration can be problematic, however, because of the number of trials necessary to gather sufficient samples to learn correctly. The problem is compounded in a multi-robot setting due to the potentially much larger design space arising from the number of and interactions between the robots. In this paper, we propose a learning from demonstration system capable of rapidly training multiple robots to perform a collaborative task. Our supervised learning method applies user domain knowledge to decompose complex behaviors into a hierarchy of simpler behaviors, which are easier to train and learn, and require many fewer samples to do so. The system further reduces the state space by only considering environmental features and actions pertinent to each decomposed simple behavior. Decomposition occurs not only within individual robot behaviors but also at the hierarchical group behavior level. Experiments using Pioneer robots in a patrol scenario illustrate our system.

1 Introduction

Learning from demonstration offers an attractive alternative to the express programming of robot behaviors: let the robots learn behaviors based on real-time examples provided by a demonstrator. Such behavioral learning is not just restricted to robotics. Developing game agent behaviors, simulated virtual agents, and character animation can benefit from nearly identical techniques as those found for robot training. To this end we have developed a novel learning from demonstration system which is capable of training various real robots and virtual agents in real time.

One fundamental challenge which learning from demonstration faces is that of gathering sufficient samples. Machine learning, particularly in high-dimensional or complex spaces, requires large numbers of samples to counter its so-called curse of dimensionality. But in robotics, a sample is expensive: it is often a data point from an experiment conducted in real time. This complexity is only increased when we consider the case of training multiple agents or robots to perform joint tasks.

Our goal is to perform rapid single- and multi-agent learning from demonstration, of potentially complex tasks, with a minimum of training samples. To this end we use domain knowledge to apply task decomposition, parameterized tasks, and per-task feature and behavioral selection to the problem, which has the effect of projecting the joint problem into much smaller and simpler subproblems, each of which can be easily learned with a very small number of samples. In short, the experimenter first decomposes the desired top-level behavior into a hierarchy of simpler behaviors, specifies the necessary states to learn and features to use for each of the simpler behaviors, then trains the robot bottom-up on each of these behaviors, using real-time experiments, until the top-level behavior is achieved.

Thus we position our work as straddling the middle-ground between providing examples (*learning*) and outright declaration (essentially *programming*). This is what we mean by our notion of "training": following an explicit pedagogy to train agents to perform sophisticated behaviors, starting at square one. In some sense one may view this middle-ground training as a kind of *assisted behavior programming by example*.

Our agents and robots learn behaviors in the form of hierarchical finite-state automata (HFAs): the states in the automata are themselves behaviors (either themselves learned HFAs, or pre-coded *basic behaviors*), and each transition function is learned using a classification algorithm on *features* gleaned from robot sensors, internal state and flag information, etc. Once learned, a

behavior joins the behavior library and can itself be used as a state in a higher-level learned HFA. This approach has a number of important advantages. First, it is supervised, and features may take any form allowed by the classifier (for example, continuous, toroidal, or categorical). Second, it allows agents to learn not just stateless policies but behaviors involving internal state. Third, the behavior hierarchy can potentially scale to very complex behaviors. Fourth, the learned HFAs can take the form of plan-like sequences or behaviors with rich transitions and recurrence. Fifth, the approach is formal and consistent: the same learning method is used at every level of the hierarchy.

We have developed a software toolkit which uses this approach, and with it we have trained many kinds of behaviors for simulated agents, everything from wall-following to scavenging for food to lining up in a triangle. We have also trained (not in simulation) a humanoid robot to find and acquire a ball, have tested the degree to which novice users can train the robot in this task, and have examined whether hierarchical servo behaviors are easier or more difficult to train than all-in-one behaviors. In this paper we begin with previous work and a description of the system, and then detail these experiments.

Following this, however, we discuss a new approach to training not a single robot but *teams* of homogeneous robots or agents, both independently and collectively under the direction of one or more *coordinator agents*, organized as a hierarchy, and which themselves may be trained with an HFA. We give a concrete demonstration example of a nontrivial HFA involving four robots and a coordinator agent in a patrolling exercise. Ultimately we are moving towards training behaviors not just for single agents but for entire teams or swarms of agents organized in hierarchies. With enough communication capacity, the approach is scalable to agent hierarchies of any size.

2 Related Work

Agent Hierarchies Hierarchies have long been employed to control a robot programmatically, from the traditional multi-tier planner/executive/control hierarchical frameworks, to *behavior hierarchies* establishing precedence among competing robot behaviors, of which an early example is the Subsumption architecture [5]. A significant body of literature has constructed groups of agents, with each agent employing its own internal hierarchical behavior mechanism [18, 8, 24]. Hierarchies *among* agents are less common, for example [9]. Some recent literature has focused on *hierarchies of control* among heterogeneous agents [10]. Hierarchies may also be constructed dynamically as a mechanism for task allocation [14].

Learning Policies and Plans The lion's share of learning from demonstration literature comes not from virtual or game agents but from autonomous robotics (for a survey, see [2]). Much of the learning from demonstration literature may be divided into systems which learn plans [1, 16, 19, 23] and those which learn (usually stateless) policies [3, 7, 12, 15] (for a stateful example, see [13]). In learning from demonstration, the proper action to perform in a given situation is usually directly provided to the agent: thus this is broadly speaking a supervised learning task. However a significant body of research in the topic in fact uses reinforcement learning, with the demonstrator's actions are converted into a signal from which the agent is expected to derive a policy [6, 22].

Hierarchical and Layered Learning Hierarchies are a natural way to achieve layered learning [21] via task decomposition. This is a common strategy to simplify the state space: see [11] for an example. Our HFA model bears some similarity to hierarchical learned behavior networks such as those for virtual agents [4] or physical robots [16], in which feed-forward plans are developed, then incorporated as subunits in larger and more complex plans. In this literature, the actual application of hierarchy to learning from demonstration has been unexpectedly limited. Hierarchy (albeit fixed) has been more extensively applied to multi-agent reinforcement learning, as in [22].

3 Description of Our System

We train robots from the bottom up by iteratively building a library of behaviors. The library initially consists of *basic 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.

3.1 The Model

The basic model is a hierarchy of finite-state automata in the form of 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 a time, designated S_t.
- $B = \{B_1, ..., B_k\}$ is the set of *basic behaviors*. Each state is associated with either an basic behavior or

another automaton from $\mathcal{H}$, with the stipulation that recursion is not permitted.

- $F = \{F_1, ..., F_m\}$ 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 time t is the environment's *feature vector* $\vec{f}_t = \langle f_1, ..., f_m \rangle$.
- $T = F_1 \times ... \times F_m \times S \rightarrow S$ 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}.
- We generalize the model with free variables (parameters) $G_1, \ldots, G_n$ for basic behaviors and features. We replace each behavior B_i with $B_i(G_1, \ldots, G_n)$ and feature F_i with $F_i(G_1, \ldots, G_n)$. The evaluation of the transition function and the execution of behaviors will both be based on ground instances (*targets*) of the free variables.

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. If the associated behavior is itself an automaton, this pulsing process recurses into the automaton.

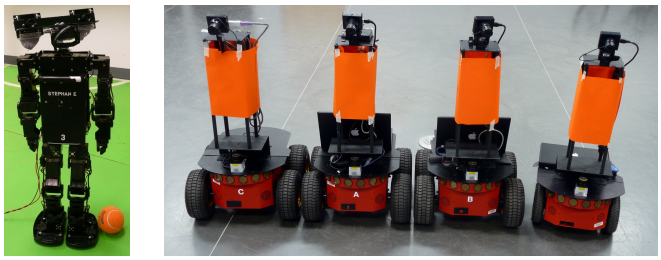
The purpose of a flag state is simply to raise a flag in the automaton to indicate that the automaton believes that some condition is now true. Two obvious conditions might be *done* and *failed*, but there could be many more. Flags in an automaton appear as optional features in its *parent* automaton. For example, the *done* flag may be used by the parent to transition away from the current automaton because the automaton believes it has completed its task.

Features may describe both internal and external (world) conditions, and may be toroidal (such as "angle to goal"), continuous ("distance to goal"), or categorical or boolean ("goal is visible").

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.

3.2 Training with the Model

Our system learns the transition function T of the automaton. We divide T into disjoint learned functions



(a) Humanoid (b) Three Pioneer ATs and one Pioneer DX

Figure 1: Robots used for the single and multi-robot experiments.

$T_S(\vec{f}_t) \rightarrow S'$, 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 for our classifiers.

The learning process works as follows. When the robot or agent 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} (again), and $\vec{f}_t$. When enough examples have been gathered, the demonstrator switches the robot to the *testing mode*, building the classifiers from the examples. For each state S_k, we build a classifier D_{S_k} based on all examples where S_k is the first element, that is, examples of the form $\langle S_k, \vec{f}, S_i \rangle$. Here, $\vec{f}$ and S_i form a data sample for the classifier: $\vec{f}$ is the input feature and S_i is the desired output class. If there are no examples at all (because the user never transitioned from S_k), the transition function is defined as staying at S_k.

The robot then begins to use the learned behavior. If the robot performs an incorrect behavior, the demonstrator may immediately return to training mode to correct the robot, adding further examples. When the demonstrator is satisfied with the robot's performance, he may then save the automaton to the behavior library and begin work on a new automaton (which can include the original automaton among its states). Note that the particular behaviors and features used may vary by automaton. This allows us to reduce the feature and state space 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 may simply make the learning space unduly complex. In these situations we may define each of the T_S to use the same classifier. This reduces the model to a stateless policy $\pi(\vec{f})$.

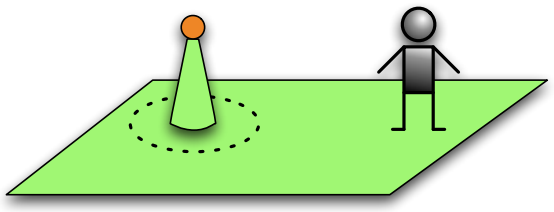


Figure 2: Experimental setup for the humanoid robot experiments. The orange ball rests on a green pillar on a green soccer field at eye level with the humanoid robot. The robot must approach to within a short distance of the pillar, as denoted by the dotted line.

4 Single Agent Experiments

We have applied our system to a single agent in simulation and on a real robot. In both cases, we were able to rapidly train the agent to perform complex tasks in a limited time. We begin by describing early experiment examples with virtual agents in simulation. We then move on to an experiment performed with a single humanoid robot. In the section after, we discuss an extension of the system to the homogeneous multiagent case.

4.1 Simulation

We have implemented a testbed for training agents using this approach in a simulator. A simulation agent can sense a variety of things: the relative locations of obstacles, other agents of different classes, certain predefined waypoints, food locations, etc. We have successfully trained many simple behaviors including: tracking and acquiring a target, wall-following, generic obstacle circumnavigation, and tracing paths (such as a figure eight path between two targets).

We have also trained an agent to perform a moderately complex foraging task, which we detail here: to harvest food from food sources and to bring it back to deposit at the agent's central station. Food can be located anywhere, as can the station. Food at a given location can be in any concentration, and depletes, eventually to zero, as it is harvested by the agent. The agent can only store so much food before it must return to the station to unload. There are various corner cases: for example, if the agent depletes food at a harvest location before it is full, it must continue harvesting at another location rather than return to the station.

Foraging tasks are of course old hat, and are not particularly difficult to code by hand. But *training* such a behavior is less trivial. We selected this task as an example because it illustrates a number of features special to our approach: our foraging behavior is in fact a three-layer HFA hierarchy; employs "done" states; involves real-valued, toroidal, and categorical (boolean) inputs; and requires one behavior with an unbound parameter used in two different ways.

The hierarchy relies on seven basic behaviors: *start*, *done*, *forward*, *rotate-left*, *rotate-right*, *load-food* (decrease the current location's food by 1, and add 1 to the agent's stored food), and *unload-food* (remove all the agent's stored food). It also requires several features: *distance-to(A)*, *angle-to(A)*, *food-below-me* (how much food is located here), *food-stored-in-me*, and *done*. Finally, it requires two targets to bind to A: the *station* and *nearest-food*.

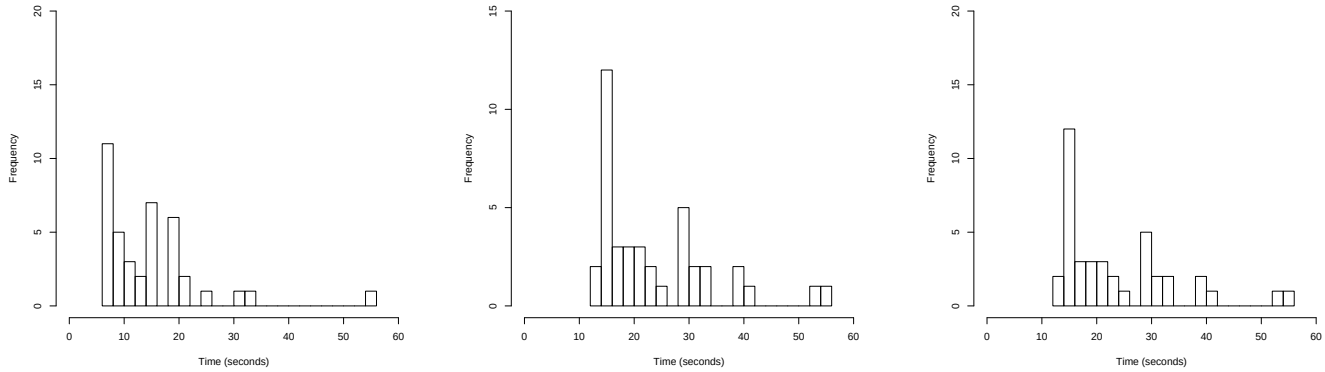
From this we decomposed the foraging task into a hierarchy of four HFA behaviors (*GoTo(A), Harvest, Deposit, Forage*), and trained each one in turn. All told, we were able to train all four behaviors, and demonstrate the agent properly foraging, in a manner of minutes.

4.2 Humanoid Robot

Taking RoboCup as motivation, we have taught one humanoid robot visual servoing (Figure 1(a)). 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 (see Figure 2).

This behavior is a 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 behaviors to compensate for this. We had two behaviors, *left* and *right*, which turned left and turned right respectively, but also had two identical behaviors, notionally called *forwardL* and *forwardR*, which both simply moved forward. A demonstrator could use these two behaviors 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



(a) Starting Position 1 (Ball in Left of Frame) (b) Starting Position 2 (Ball Not Visible) (c) Starting Position 3 (Ball in Right of Frame)

Figure 3: Histograms of times to reach the ball from each starting position of the four successful trials.

the robot to turn appropriately. Ultimately the robot may learn that if the ball disappeared while it was in the *forwardL* behavior, it should then transition to turning left; and likewise turn right if the ball disappeared while the robot was in the *forwardR* behavior.

First, we performed an experiment illustrating how the learning system can be used by novice users. We asked five computer science graduate students to train the robot for five minutes each. The students had minimal exposure to the humanoid robots and no experience with our training system. After some instruction on how to control the robot, and some suggestions on using *forwardL* and *forwardR*, they were allowed to move the robot into any position for data collection. The students were given sensor data in real-time and were allowed to visually observe the robot. After the five minutes elapsed, the robot built a HFA to perform visual servoing. Performance was determined by the time required to locate the ball and approach it to within 15 cm (determined visually), starting from three different known positions. Position 1 had the ball on the left of the frame, Position 2 the ball was not visible, and Position 3 had the ball on the right of the frame. We performed ten trails for each HFA at each position.

Four of the five students successfully trained the robot to approach the ball. The remaining trained behavior never successfully approached the ball, independent of the robot's starting location. Figure 3 shows 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 many more training samples because the joint training space (in terms of states and features) is higher.

5 Training Multi-Robot Team Hierarchies

Our ultimate goal is to apply the training technique not just to single agents but to supervised training of teams and swarms of arbitrary size.

We note that *supervised* cooperative multiagent training has a surprisingly small literature. From an extensive survey of cooperative multiagent learning [17], only a small number of papers dealt with supervised learning, and most of those were in the area of *agent modeling*, whereby agents learn about one another, rather than being trained by the experimenter. The lion's share of the remaining literature tends to fall into feedback-based methods such as reinforcement learning or stochastic optimization (genetic algorithms, etc.). For example, in one of the more celebrated examples of multiagent layered learning [20] (to which our work owes much), the supervised task ("pass evaluation") may be reasonably described as agent-modeling, while the full multiagent learning task ("pass selection") uses reinforcement learning. This is not unusual.

Why is this so? Supervised training, as opposed to agent modeling, generally requires that robots be told which micro-level behaviors to perform in various situa-

tions; but the experimenter often does not know this. He may only know the emergent macro-level phenomenon he wishes to achieve. This inverse problem poses a significant challenge to the application of supervised methods to this task. The standard response to inverse problems is to use a feedback-based technique. But there is an alternative: to decompose the problem into sub-problems, each of which is simple enough that the gulf between the micro- and macro-level behaviors is reduced to a manageable size. This is the technique which we have pursued.

Our approach is as follows. We organize the team of agents into an *agent hierarchy* (not to be confused with our HFA behavior hierarchies), with robots at leaf nodes of a tree, and coordinator agents as nonleaf nodes. This tree-structured organization fits in the middle ground between largely decentralized ("swarm"-style) multirobot systems and fully centralized systems. A tree structure has obvious advantages (and disadvantages) which we will not discuss here: we use it largely because of its clean integration with our task-decomposition focus.

Individual robots in the agent hierarchy may be trained as usual, producing behaviors in the form of hierarchical finite-state automata, with the caveat that all robots ultimately share the same behavior library. First-level coordinators are then trained to develop HFAs themselves: the "basic behaviors" at the bottom level of the HFAs are the behaviors from the robots' behavior library. Second-level coordinators are then trained to develop HFAs whose basic behaviors are the behaviors of their first-level children, and so on.

It is straightforward for coordinators to affect the behaviors of subsidiaries: but on what conditions should we base the transition functions of coordinator HFAs? Individual robots' transition functions are based on sensor information and flags: but coordinator agents have no sensors per se. We have opted to give coordinators "sensors" in the form of statistical information about their subsidiaries. Examples: whether a subsidiary has seen a bad guy, or whether all subsidiaries are *Done*, or the mean location of the coordinator's robot team members.

This organization of the team into hierarchies allows us to reduce the scope of each team learning task by reducing the number of agents and potential interactions as necessary. And the use of HFAs at the coordinator level allows us to decompose complex team behaviors into simpler ones which can be rapidly taught to agent teams because they have reduced the gulf between micro- and macro-level phenomena.

We ultimately plan to use this method to develop heterogeneous team behaviors: but for now we are concentrating on homogeneous behaviors. We note that this embedding of the HFA training into a robot team hierarchy suggests at least three different notions of "homogeneous" behaviors, as shown in Figure 4. First, all robots may simply perform the exact same HFA, but independent of one another. But we can go further than

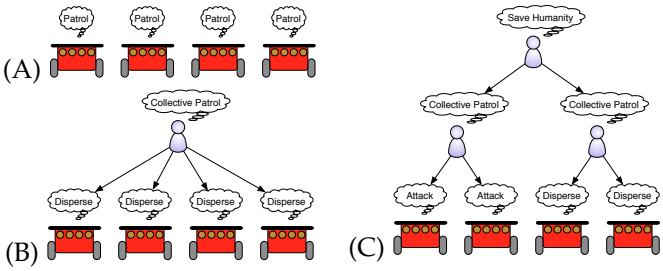


Figure 4: Three notions of homogeneity. (A) Each agent has the same top-level behavior, but acts independently. (B) The top-level behavior all agents is the same, but may all be switched according to a higher-level behavior under the control of a coordinator agent. (C) Squads in the team are directed by different coordinator agents, whose behaviors are the same but may all be switched by a higher-level coordinator agent (and so on).

this and still stay within the aegis of homogeneity: we may add a coordinator agent which controls *which* HFA the robots are performing. It does so by running its *own* HFA with those subsidiary HFA as basic behaviors. Coordination may continue further up the chain: second- or higher-level coordinator agents may also dictate their subsidiaries' choice of HFAs.

5.1 Demonstration

We have performed a demonstration which illustrates this approach of training *team hierarchies* of *hierarchical automata*. We trained a group of four Pioneer robots to perform a pursuit task while also deferring to and avoiding a "boss". Each robot had a color camera and sonar, and was marked with colored paper (see Figure 1(b)). The boss, intruders to pursue, and a home base were also marked with paper of different colors.

The task was as follows. Ordinarily all agents would *Disperse* in the environment, wandering randomly while avoiding obstacles (by sonar) and each other (by sonar or camera). Upon detecting an intruder in the environment, the robots would all *Attack* the intruder, servoing towards it in a stateful fashion, until one of them was close enough to "capture" the intruder and the intruder was eliminated. At this point the robots would all go to a home base (essentially *Attack* the base) until they were *all* within a certain distance of the base. Only then would they once again *Disperse*. At any time, if the boss entered the environment, each agent was to *RunAway* from the boss: turn to him, then back away from him slowly, stopping if it encountered an obstacle behind.

This task was designed to test and demonstrate every aspect of the hierarchical learning framework: it required the learning of hierarchies of individual agent behaviors, stateful automata, behaviors and features with targets, both continuous and categorical features, mul-

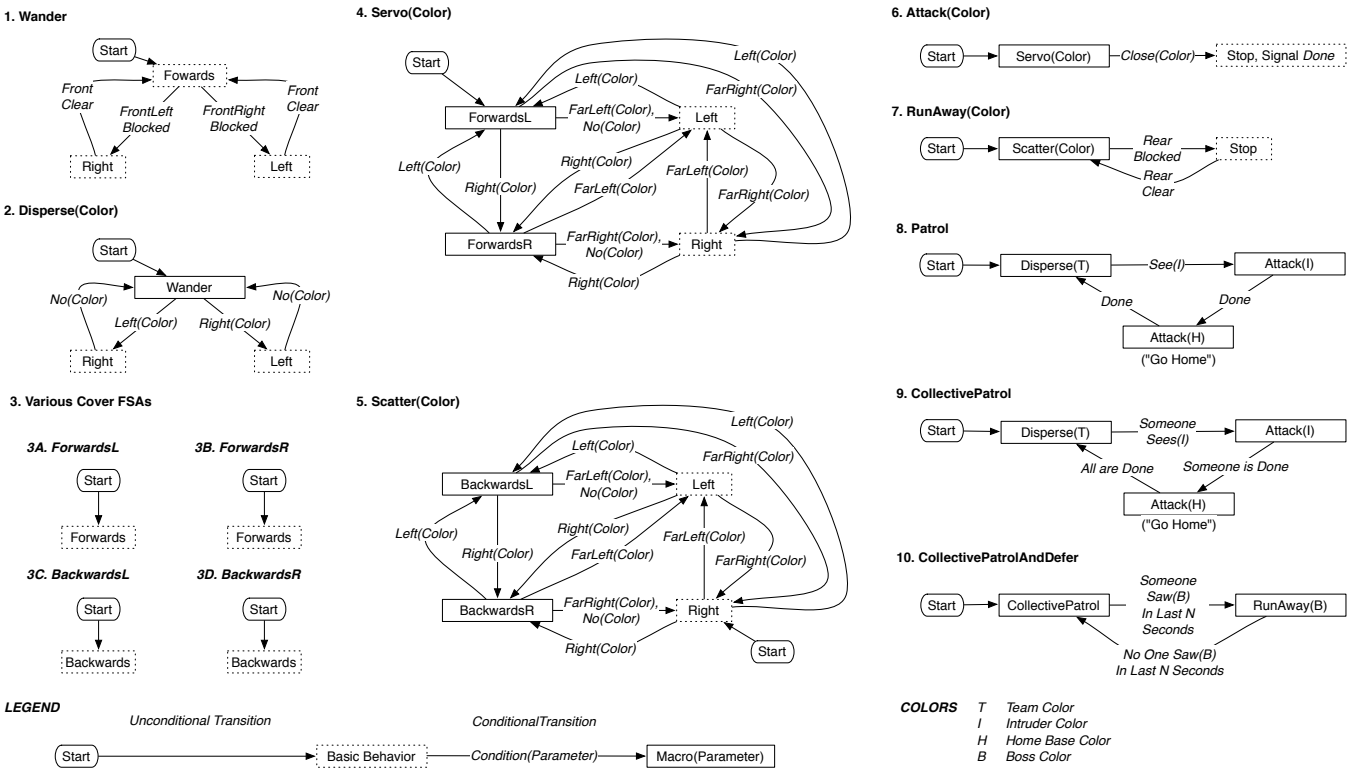


Figure 5: Decomposed hierarchical finite-state automaton learned in the demonstration. See discussion in the text on each subfigure. Most behaviors form a hierarchy within an individual robot, but *CollectivePatrol* and *CollectivePatrolAndDefer* form a separate hierarchy within the team controlling agent. Though the transition condition descriptions here are categorical sounding, most are in fact derived from continuous values: for example, the condition *Left(Color)* is trained based on various *X* coordinates of the color blob in the field of view.

tiple agents, and learned hierarchical behaviors for a coordinator agent.

Each agent was provided the following simple basic behaviors: to continuously go *Forwards* or *Backwards*, to continuously turn *Left* or *Right*, to *Stop*, and to *Stop* and raise the *Done* flag. Transitions in HFAs within individual agents were solely based on the following simple features: whether the current behavior had raised the *Done* flag; the minimum value of the *Front Left*, *Front Right*, or *Rear* sonars; and the *X Coordinate* or the *Size* of a blob of color in the environment (we provided four colors as targets to these two features, corresponding to *Teammates*, *Intruders*, the *Boss*, and the *Home Base*). Each robot was dressed in the *Teammate* color.

We began by training agents to learn various small parameterized HFAs, as detailed in Figure 5, Subfigures 1 through 7. Note that the *Servo* and *Scatter* HFAs are stateful: as was the case for the humanoid robot experiment, when the target disappeared, the robot had to discern which direction it had gone and turn appropriately. Since our system has only one behavior per state, we enabled multiple states with the same behavior by training the trivial HFA in subfigures 3A through 3D, just as in the humanoid experiment.

We then experimented with the "basic" homogeneous behavior approach as detailed in Figure 4(A): each agent simply performing the same top-level behavior but without any coordinator agent controlling them. This top-level behavior was *Patrol* (Figure 5, Subfigure 8), and iterated through the three previously described states: dispersing through the environment, attacking intruders, and returning to the home base. We did not bother to add deferral to the "boss" at this point.

Coordinated Homogeneity Simple homogeneous coordination like this was insufficient. In this simple configuration, when an agent found an intruder, it would attack the intruder until it had "captured" it, then go to the home base, then resume dispersing. But other agents would not join in unless they too had discovered the intruder (and typically they had not). Furthermore, if an agent captured an intruder and removed it from the environment, other agents presently attacking the intruder would not realize it had been captured, and would continue searching for the now missing intruder indefinitely!

These difficulties highlighted the value of one or more coordinator agents, and so we have also experimented

Figure 6: Learned multi-robot behavior in action. Demonstrator is holding a green target, signifying an intruder.

with placing all four robots under the control of a single coordinator that would choose the top-level behavior each robot would perform at a given time. The coordinator was trained to follow the *CollectivePatrol* behavior shown in Figure 5, Subfigure 9. This HFA was similar to the *Patrol* behavior, except that robots would attack when *any* robot saw an intruder, would all go to the Home Base when *any* robot had captured the intruder, and would all resume dispersing when *all* of the robots had reached the Home Base. This effectively solved the difficulties described earlier.

Note that in order to determine transitions for this HFA, the coordinator relied on certain features gleaned from statistical information on its team. We provided the coordinator with three simple features: whether any robot had seen the Intruder's color; whether any robot was *Done*, and whether all robots were *Done*.

Finally, we trained a simple hierarchical behavior on the coordinator agent as an example, called *CollectivePatrolAndDefer* (Subfigure 10). We first added a new statistical feature to the coordinator agent: whether anyone had seen the Boss color within the last $N - 10$ seconds. The coordinator agent would perform *CollectivePatrol* until someone had seen the Boss within the last 10 seconds, at which point the coordinator agent would switch to the *RunAway* behavior, causing all the agents to search for the Boss and back away from him. When no agent had seen the Boss for 10 seconds, the coordinator would resume the *CollectivePatrol* behavior (Figure 6).

Summary This is a reasonably comprehensive team behavior, with a very large non-decomposed finite-state automaton, spanning across four different robots acting in sync. We do not believe that we could train the agents to perform a behavior of this complexity without decomposition, and certainly not in real-time. There are too many states and aliased states, too many features (at least 12), and too many transition conditions. However decomposition is straightforward into simple, easily trained behaviors with small numbers of features and states, simple (indeed often trivial) and easily trained transition functions, and features and states which may vary from behavior to behavior.

Learning in the Small Our system is capable of doing classification over spaces of any level of complexity. But in order to reduce the necessary sample size and enable real-time on-the-fly training, our trained transition functions are usually based on a very small number of features (typically one or two, rarely three), and the resulting space is not complicated. In some cases the learned function requires a large decision tree: but more often then not, the tree has just a few nodes.

From a machine learning perspective, such learned behaviors are very simple. But this is exactly the point. Our goal is to enable rapid, assisted, agent and robot behavior development. From this perspective, decomposition to simple models allows even novices to build complex behaviors rapidly because the number of samples does not need be large. This puts a technique like ours at the very edge of what would be reasonably called *machine learning*: it is declaration by example.

6 Conclusion

We have presented a hierarchical learning from demonstration system capable of training multiple robots with minimal examples. By organizing a group of robots into a hierarchy, we provide a logical decomposition into simpler behaviors which may be trained quickly. The coordinator approach developed in this paper allows for arbitrary group decomposition where subsidiary groups are under different coordinator agents.

In the future, we would like to apply our system to heterogeneous behaviors where subgroups (of possibly one robot) are controlled via different HFAs. In fact, these different HFAs do not need to share a basic behavior library, which may be the case if the robots have different capabilities. In addition, we plan to explore more explicit coordination between robots and the challenges associated with training such coordination.

References

[1] Richard Angros, W. Lewis Johnson, Jeff Rickel, and Andrew Scholer. Learning domain knowledge for teaching procedural skills. In *The First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1372–1378. ACM, 2002.

[2] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57:469–483, 2009.

[3] Darrin C. Bentivegna, Christopher G. Atkeson, and Gordon Cheng. Learning tasks from observation and practice. *Robotics and Autonomous Systems*, 47 (2-3):163–169, 2004.

[4] Rama Bindiganavale, William Schuler, Jan M. Allbeck, Norman I. Badler, Aravind K. Joshi, and Martha Palmer. Dynamically altering agent behaviors using natural language instructions. In *Autonomous Agents*, pages 293–300. ACM Press, 2000.

[5] Rodney Brooks. A robust layered control system for a mobile robot. *IEEE Journal Of Robotics And Automation*, RA-2:14–23, April 1986.

[6] Adam Coates, Pieter Abbeel, and Andrew Y. Ng. Apprenticeship learning for helicopter control. *Communications of the ACM*, 52(7):97–105, 2009.

[7] Jonathan Dinerstein, Parris K. Egbert, and Dan Ventura. Learning policies for embodied virtual agents through demonstration. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1257–1252, 2007.

[8] E. H. Durfee and T. A. Montgomery. A hierarchical protocol for coordinating multiagent behaviors. In *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI-90)*, pages 86–93, Boston, MA, USA, 1990. AAAI Press.

[9] Dani Goldberg and Maja J. Mataric. Design and evaluation of robust behavior-based controllers. In Tucker Balch and Lynne E. Parker, editors, *Robot Teams: From Diversity to Polymorphism*, pages 315–344. A. K. Peters, 2002.

[10] R. Grabowski, L. E. Navarro-Serment, C. J. J. Paredis, and P. K. Khosla. Heterogeneous teams of modular robots for mapping and exploration. *Autonomous Robots*, July 2000.

[11] D.H. Grollman and O.C. Jenkins. Learning robot soccer skills from demonstration. In *IEEE 6th International Conference on Development and Learning (ICDL)*, pages 276–281, July 2007.

[12] Michael Kasper, Gernot Fricke, Katja Steuernagel, and Ewald von Puttkamer. A behavior-based mobile robot architecture for learning from demonstration. *Robotics and Autonomous Systems*, 34(2-3): 153–164, 2001.

[13] D. Kulic, Dongheui 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*, pages 326–332, Dec. 2008.

[14] James McLurkin and Daniel Yamins. Dynamic task assignment in robot swarms. In *Robotics: Science and Systems Conference*, 2005.

[15] Jun Nakanishi, Jun Morimoto, Gen Endo, Gordon Cheng, Stefan Schaal, and Mitsuo Kawato. Learning from demonstration and adaptation of biped locomotion. *Robotics and Autonomous Systems*, 47 (2-3):79–91, 2004.

[16] Monica N. Nicolescu and Maja J. Mataric. A hierarchical architecture for behavior-based robots. In *The First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 227–233. ACM, 2002.

[17] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[18] Lynne Parker. ALLIANCE: An architecture for fault tolerance multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2), 1998.

[19] Paul E. Rybski, Kevin Yoon, Jeremy Stolarz, and Manuela M. Veloso. Interactive robot task training through dialog and demonstration. In Cynthia Breazeal, Alan C. Schultz, Terry Fong, and Sara B. Kiesler, editors, *Proceedings of the Second ACM SIGCHI/SIGART Conference on Human-Robot Interaction (HRI)*, pages 49–56. ACM, 2007.

[20] Peter Stone and Manuela Veloso. Layered learning and flexible teamwork in robocup simulation agents. In Manuela Veloso, Enrico Pagello, and Hiroaki Kitano, editors, *RoboCup-99: Robot Soccer World Cup III*, volume 1856 of *Lecture Notes in Computer Science*, pages 65–72. Springer Berlin / Heidelberg, 2000.

[21] Peter Stone and Manuela M. Veloso. Layered learning. In Ramon López de Mántaras and Enric Plaza, editors, *11th European Conference on Machine Learning (ECML)*, pages 369–381. Springer, 2000.

[22] Yasutake Takahashi and Minoru Asada. Multi-layered learning system for real robot behavior acquisition. In Verdan Kordic, Aleksandar Lazinica, and Munir Merdan, editors, *Cutting Edge Robotics*. Pro Literatur, 2005.

[23] Harini Veeraraghavan and Manuela M. Veloso. Learning task specific plans through sound and visually interpretable demonstrations. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2599–2604. IEEE, 2008.

[24] Thuc Vu, Jared Go, Gal Kaminka, Manuela Veloso, and Brett Browning. MONAD: a flexible architecture for multi-agent control. In *AAMAS 2003*, pages 449–456, 2003.