Training Heterogeneous Teams of Robots

Keith Sullivan, Ermo Wei, Bill Squires, Drew Wicke, and Sean Luke

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

Abstract. Heterogeneous multi-robot teams are common solutions to complex tasks, especially those that are inherently cooperative. Training robots, rather than coding them, to work together in these teams is an attractive prospect, but is very difficult due to the extremely large state space and the inherent inverse problem which separates the agents' micro-level behaviors and the desired macro-level emergent phenomenon. We approach this problem with HiTAB, a learning from demonstration system which uses behavior decomposition to allow rapid training of teams with minimal samples. This paper presents and compares two approaches to training teams of heterogenous robots: first, forming a multiagent control hierarchy which scales to large numbers of robots but requires the training of additional virtual controller agents; and second, modifying each robot's feature space to include information about other robots' current behaviors or limited internal state.

1 Introduction

Learning from Demonstration (or LfD) is the training of a robot in real-time by a human who iteratively demonstrates a behavior, observes the learned result, and steps in to correct errors through additional demonstration. LfD may be done by teleoperation (as we do), or through the robot observing the human performing the behavior himself. LfD has natural application to a great many areas, particularly when humans wish to develop new robot behaviors rapidly, but lack the skills to program them. But while there is a rich literature on single-robot learning from demonstration, this is not the case for training *multiple* collaborative robots, and the literature on real-time training of *heterogeneous* teams is very rare.

This is in part because multi-robot learning from demonstration is nontrivial. At its core lie two challenges inherent in cooperative multiagent learning. First, there is the design space issue. Setting aside numerosity, agents may be heterogeneous, be broken into multiple classes, have complex interactions amongst themselves, and have complex internal actions and sensors. This results in a potentially high-dimensional space requiring large numbers of samples.

Second, there is the daunting inverse problem. Presume that we wished to teach the robots to storm a castle. Even if we knew how to quantify this

macro-level task, ultimately each robot must learn its own micro-level behavior. Even if we had a function f to translate micro-behaviors to emergent macro-phenomena (a multirobot simulator), we still wouldn't have the inverse function f^{-1} which yields the individual micro-behaviors necessary to produce the macro-phenomenon. The standard way to solve such inverse problems is optimization, and indeed in [13] we found that nearly all cooperative multiagent learning papers use such methods, primarily stochastic optimization or reinforcement learning. Optimization, also, tends to require many samples.

Unfortunately, in multiagent LfD samples are costly, as they arise from a human trainer working in real-time with physical robots. Thus the multiagent LfD literature is extremely sparse. Adding heterogeneity to the mix makes it even more so.

We suggest a different approach to multi-robot training which applies supervised learning rather than optimization, and can be used to train many robots in real time to perform nontrivial, heterogeneous, collaborative behaviors. Our method, Hierarchical Training of Agent Behavior (HiTAB), uses manual task decomposition in two different ways to simplify the problem dimensionality and to break the inverse function down into multiple simple and trivially solved ones.

We have previously used HiTAB to train single agents, and to train hundreds of homogeneous agents doing collaborative tasks [19]. We also used HiTAB to achieve a first at RoboCup 2014: we trained three homogeneous attacker robots how to play soccer, via demonstration at the venue, then entered them into the competition, where they won half their games [6]. In this paper we turn to the much more complex heterogeneous case: training different kinds of robots, in real time, to perform a collaborative task from a single human demonstrator. To our knowledge, this paper is the first such demonstration.

A major issue in training heterogeneous teams lies in their coordination architecture. We will examine two different methods. First, we put the robots under a hierarchy of virtual, trained *controller agents* to coordinate them. This approach scales to large numbers of robots, but requires more agents to be trained. Second, we modify each robot's feature space to include information about other robots' current behaviors.

2 Related Work

LfD uses input from a human demonstrator to learn proper behavior for a given situation [2, 23]. Since the agent is given the proper action for a given situation, LfD is broadly speaking a supervised problem, though much of the literature uses reinforcement learning to solve it. The LfD literature may be divided into two categories: learning *plans* or *behaviors* [1], and learning robot *trajectories* or *paths* [15]. In the first case there are usually very few samples to learn from, as the important samples usually only arrive on transition from one operation to another in the plan or behavior; this is not true for the second case, where every slight change in trajectory may be a useful sample. Our work falls in the first

category, and so much of it lies in approaches to effective learning of nontrivial behaviors despite an extreme paucity of samples.

Online *multi*-robot LfD presents unique difficulties, and as discussed earlier, research in this area is extremely sparse. Some work exists though: for example, [5] trained multiple robots to work together by having one request help from a demonstrator when it felt unsure of the proper action. A similar approach was used to train Sony AIBO robots to play soccer [8, 4]. Finally, Raza et al used human demonstrators to train multiple soccer agents within the RoboCup 3D Simulation League [18, 17]. Some work has dealt with the inverse problem by eliminating macrobehaviors entirely and issuing separate micro-level training directives to each individual agent [22, 11].

The multirobot examples cited above used homogeneous teams. Some work has used heterogeneous teams, notably the L-ALLIANCE architecture, used to learn hazardous waste clean up and box-pushing [14]. Blokzijl-Zanker and Demiris recently used multiple demonstrators to train a plan for a heterogenous robot team to open doors [3].

3 Training a Single Agent with HiTAB

HiTAB learns behaviors in the form of hierarchical finite state automata (HFA) represented as Moore machines [9, 21]. Each state in the HFA maps to a previously-learned HFA or hard-coded built-in behavior. The motivation behind HiTAB is to rapidly train nontrivial agent behaviors, and particularly multiagent behaviors, in environments with a paucity of samples. HiTAB uses manual behavior decomposition to reduce the size of the learning space and to simplify the micro- to macro-behavior gulf. This manual decomposition breaks complex automata and state transitions into simple, often simplistic, ones which can be learned from a small number of samples each.

Once the hierarchical structure is specified, the experimenter begins training bottom-up. Initially, the behavior library consists only of basic hard-coded behaviors, and these correspond to states in the initial automaton to be learned. The training process thus produces only the transition function of the automaton. This takes the form of one classifier per state, based on sensor features available to the robot. Before training an automaton the experimenter may choose which *sensor features* are available to the transition function, thus reducing the size of the learning space. Features within HiTAB are can be categorical, continuous, or toroidal, such as "bumper pressed", "angle to target", or "distance to wall". Though we do not do so in this paper, both features and behaviors can be *parameterized*. For example, instead of training the three behaviors *GoToWall*, *GotoTeammate*, and *GotoHomeBase*, we may instead train a single behavior *Goto(X)* where X is chosen from a set of available *targets* (the wall, the teammate, the home base) when the behavior is used later.

Training of a behavior iterates between a *training mode* and a *testing mode*. In the training mode, the agent performs behaviors as directed by the demonstrator, and each time a behavior (thus state) transition occurs, the agent records a

training example: 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 the behavior associated with state S_{t+1} is designed to be executed exactly once, then no additional samples are recorded. Otherwise, a *default sample* is also stored: $\langle S_{t+1}, \vec{f}_t, S_{t+1} \rangle$. This tells the agent to continue in the current state if the given feature vector does not change.

Once enough samples are collected, the demonstrator switches to *testing mode*, causing the agent to build the transition function. For a given state S_i, HiTAB takes all samples of the form $\langle S_i, \vec{f}_t, S_j \rangle (\forall\ t, j)$, reduces them to $\langle \vec{f}_t, S_j \rangle$, and feeds them to a classification algorithm (with $\vec{f}_t$ as a data point and S_j is its label). The resulting classifier forms the transition function for outgoing edges from S_i. Many classification algorithms can be used: here we use C4.5 [16].

After all the transition functions are built for a given behavior, the agent begins performing the learned behavior as described above. If the demonstrator observes the agent performing an incorrect behavior, he may switch to training mode to collect additional samples. New transition functions are then rebuilt with the additional data to create a new trained behavior. This turn-taking continues until the demonstrator is satisfied with the agent's behavior.

Once a behavior is trained satisfactorily, unused states and other information are trimmed from the automaton, and it is saved to the behavior library to become available as a state for future to-be-trained automata. This process repeats until the entire hierarchy of behaviors is trained.

4 Multiagent Heterogeneous HiTAB

Once we have trained agents with necessary individual behaviors, depending on the number of agents and the degree of agent interaction needed, we might train the agents independently with dummies standing in for the other agents; or train them concurrently (with multiple parallel trainers); or train them using *behavioral bootstrapping* [20]. Here, one robot is trained to perform a rudimentary version of a behavior in the presence of other (non-functioning) robots. This behavior is distributed to similar robots, then another robot is selected to learn a higher-level version of *its* current behavior in the presence of robots performing their present rudimentary behaviors, and so on.

Such approaches will work for very small groups of agents (or potentially larger swarms of simple homogeneous agents). To train large swarms of coordinated heterogeneous robots, we must rely on an abstraction mechanism to simplify the complexity of their coordination. One approach is to manually decompose the swarm into an *agent hierarchy*, a tree structure where leaf nodes are the individual robots performing tasks, and non-leaf nodes are virtual *controller agents*, each one in charge of coordinating its children in the tree. For example, we might decompose a soccer team into offensive and defensive groups, then decompose the defensive groups into defenders and the goalie, and the offensive group into attackers and midfielders. We then train the agent hierarchy bottom-up: we first train a small group of heterogeneous agents (the midfielders, say)

to perform various collaborative behaviors. We create a controller agent for the midfielders, whose basic behaviors correspond to the collaborative behaviors we had trained for them, and whose features correspond to statistics or signals from the midfielders necessary to develop behaviors to coordinate them: for example "a robot in my group has the ball" or "the average Y coordinate of robots in my group." We then train automata in this virtual agent using HiTAB. This continues up the hierarchy: the midfielder and attacker controller agents are grouped and trained to work together; then a controller agent is assigned to *them*, etc.

This approach has two advantages: first, controller agents provide a straightforward way of doing coordination among agent groups, and second, hierarchies scale to arbitrarily large swarms, because regardless of its size, at any position in a hierarchy an agent (and his trainer) must deal only with a fixed number of agents (his superior and immediate subordinates).

However for small swarms, a controller hierarchy may be cumbersome and overly complex: and it requires that the designer manually group joint behaviors together to form basic behaviors in the controller, and to manually define features to pass to the controller. An alternative is to simply let agents know the current top-level behaviors of their collaborators. We do this by changing the feature vector $\vec{f_t} = \langle f_1, ..., f_n \rangle$ to $\vec{f_t} = \langle f_1, ..., f_n, b_1, ..., b_m \rangle$ where each b_i is the current top-level behavior performed by agent i. For example, when an agent switches from *acquiring* a ball to *kicking* it, other agents might sense this and update their behaviors accordingly. A trainer can use this method to perform signaling as well: one agent might be trained to move *forward*, then switch to *forwardSpecial*, a behavior which simply does *forward* but, by virtue of its name, signals to other agents to change their behaviors as well.

We call this method *behavior-signaling*, which introduces a communication overhead and feature space growth that is unlikely to scale to large numbers of agents. But for small groups it might be a simpler coordination procedure.

5 Demonstration and Comparison

We want to do two things:

- Demonstrate successful real-time training of a collaborative, heterogeneous team of robots, albeit a small one, by a single demonstrator.
- Compare using a *controller hierarchy* to do this training against using *behavior signaling*.

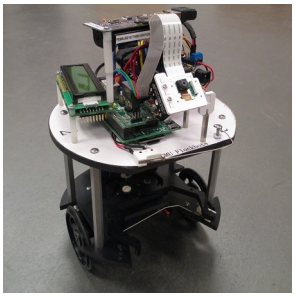

Fig. 1. Flockbot

Though each training session took only about an hour, it required experienced human trainers highly familiar with a complex piece of software: this made it extremely difficult to gather rigorous statistically significant comparison with many demonstrators. And so by the nature of the problem we are largely reduced

to anecdote. With this in mind, we will report the the number of samples required as a measure of technique complexity. We report this for two independent human demonstrators, each attempting both techniques.

For this demonstration and comparison, we set out to reproduce a well-known heterogeneous three-robot box-pushing experiment described in [12, 7]: the goal was to push a long box across the room using two robots, with a third robot in the back monitoring the box's angle. The critical difference being, of course, that in our work the robots would be trained rather than hand-coded.

The two robots pushing the box were Flockbots (Figure 1), roughly 7-inch-diameter differential robots of our design [10]. These robots are sensor poor and cannot monitor the box's angle. The third robot was a Pioneer 3DX (Figure 2), a much larger differential-drive robot equipped with a variety of sensors, including a scanning laser range finder, used to monitor the angle of the box.

This arrangement demonstrated two notions of heterogeneity. Clearly the Pioneer was different from the Flockbots in its capability. But while the Flockbots were homogeneous in *capability*, they were in fact different from one another in their necessary *coordinated behavior*.

Figure 5 shows the robot setup for the box-pushing experiment. To help replicability, we describe in detail exactly those automata trained and the features they relied on. The automata are summarized in Figure 4.

Fig. 2. Pioneer 3DX

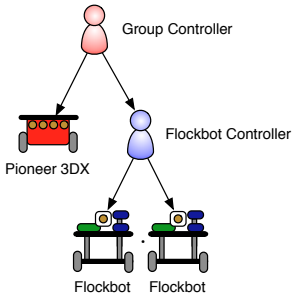

Fig. 3. Agent Hierarchy

5.1 Training with a Controller Hierarchy

We arranged the three robots into a two level agent hierarchy as shown in Figure 3: the three robots were the leaves, and a Flockbot Controller agent was responsible for coordinating the two Flockbots. The top-level controller agent (called the Group Controller) coordinated the Flockbot team and the Pioneer.

Each level of the hierarchy required different features:

- Each Flockbot used one sensor feature, **BumpPressed**, returning whether the Flockbot's bump sensor was pressed.
- The Pioneer used a single sensor feature **DistanceToBox** which measured the distance between the Pioneer and box with a laser range finder.
- The Flockbot Controller relied on the feature **AllDone**, which informed it that both Flockbots had completed initialization and reached *Done*.

– The Group Controller used two features: **AllDone** and **BoxAngle**. *AllDone* informed it that both the Pioneer and Flockbot Controller had reached *Done*. *BoxAngle* used data passed to the Group Controller by the Pioneer, used a split-merge algorithm to compute the angle of the longest line (box) from the laser range finder.

Similarly, each robot type had different basic behaviors. The Flockbot had four behaviors: **Done**, **Stop**, **ForwardFast** (0.4 meters per second), and **ForwardSlow** (0.1 meters per second). The Pioneer had three behaviors: **Done**, **Stop**, and **Forward** which moved the robot forward at 0.5 meters per second. **Done** simply raised a signal that it had been reached.

We first trained additional HFA for each of the robots:

– **FlockbotInit**: Used *BumpPressed*. Moved the Flockbot forward until it contacted the box.
– **GotoBox**: Used *DistanceToBox*. Kept the Pioneer within a pre-specified distance of the box.
– **PioneerInit**: Used *DistanceToBox*, *GotoBox*, and *Done*. Moved the Pioneer within a pre-specified distance of the box, and then signaled *Done* to its parent automaton. Although it appears that they could be merged, *PioneerInit* and *GotoBox* were in fact used at different times.

We then grouped the two Flockbots together, defining the following joint heterogeneous behaviors which later were mapped to basic behaviors in the Flockbot Controller:

– **FlockbotGroupInit**: both Flockbot ran *FlockbotInit*.
– **FlockbotForward**: both Flockbots ran *ForwardFast*.
– **CorrectLeft**: the left Flockbot ran *ForwardSlow* and the right Flockbot ran *ForwardFast*.
– **CorrectRight**: the left Flockbot ran *ForwardFast* and the right Flockbot ran *ForwardSlow*.

We then trained this Flockbot Controller behavior:

– **FlockbotControllerInit**: ran *FlockbotGroupInit* until *AllDone* was signaled.

Next, we grouped the Flockbot Controller with the Pioneer, defining the following joint heterogeneous behaviors which mapped to basic behaviors in the Group Controller:

– **GroupCorrectLeft**: the Pioneer ran *GotoBox* and the FlockbotController ran *CorrectLeft*.
– **GroupCorrectRight**: the Pioneer ran *GotoBox* and the FlockbotController ran *CorrectRight*.
– **GroupForward**: the Pioneer ran *GotoBox* while the FlockbotController ran *FlockbotFoward*.

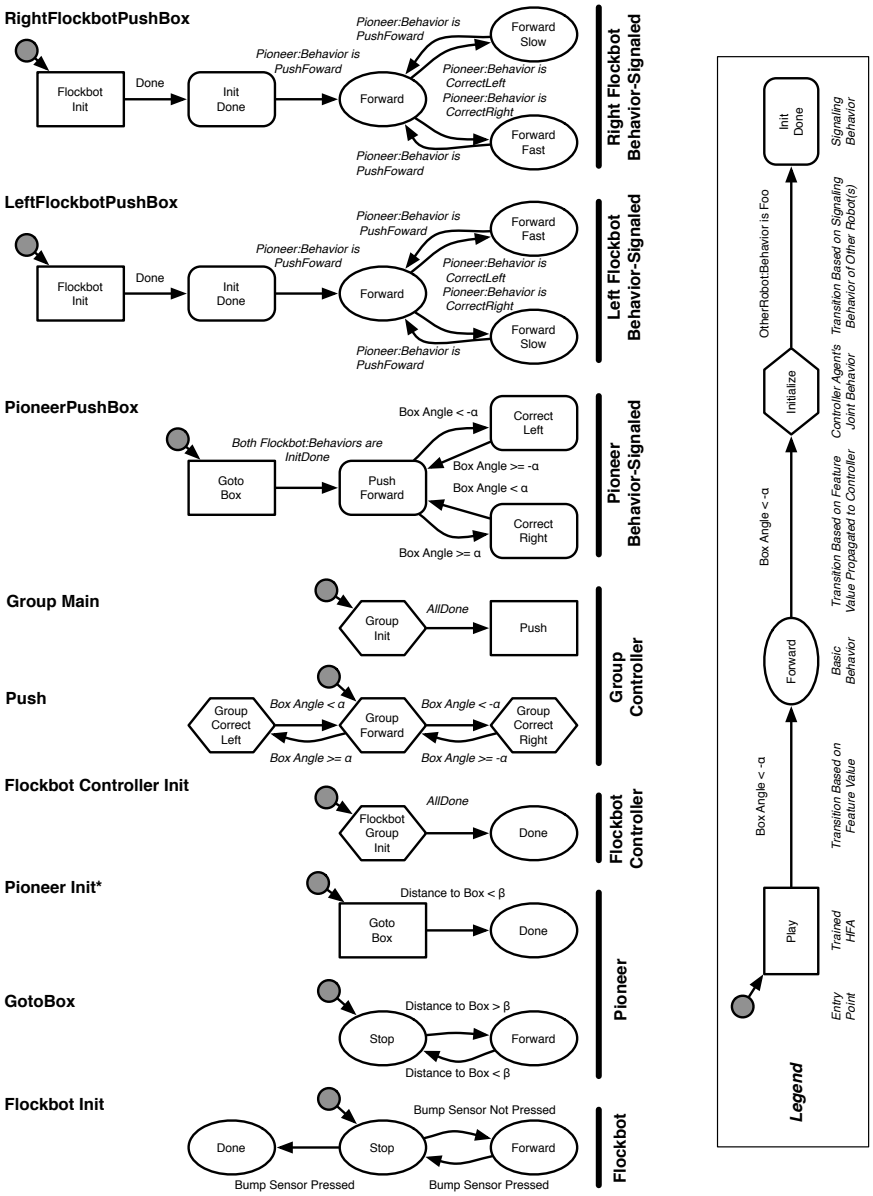

Fig. 4. Trained HFA for box pushing: both Controller and Behavior-signaled HFAs shown together for brevity. Both techniques relied on the *Pioneer* and *Flockbot* groups of HFA at the bottom of the figure. Additional Controller or Behavior-signaled HFA are shown above them. ⋆Note that *Pioneer Init* was only used by the Controller technique. The trivial *Init Done*, *Correct Left*, *Correct Right*, and *Push Forward* behaviors are not shown.

Finally, we trained the following Group Controller HFAs:

- **Push**: A servoing automaton which kept the box straight based on *BoxAngle*, by alternating between *GroupForward*, *GroupCorrectLeft*, and *GroupCorrectRight*. While it would seem that this behavior would naturally reside in the Flockbot Controller, because it relied on the box angle, it needed to be one level higher in our model.
- **GroupMain**: A top-level automaton which moved the robots to their initial positions, then began calling *Push*.

5.2 Training with Behavior-Signaling

With Behavior-Signaling, there was no hierarchy: instead the three robots coordinated via sensing the top-level behaviors of the other robots. Behavior-Signaling required slightly different features:

- Each Flockbot used the **BumpPressed** feature as before. It also used the signaled behavior of the Pioneer robot as the feature **Pioneer:Behavior**.
- The Pioneer used **DistanceToBox** as before. Unlike the Controller situation, where **BoxAngle** was only used by the Group Controller, in the Behavior-Signaled situation **BoxAngle** was used by the Pioneer behaviors directly. The Pioneer also used the signaled behaviors of both Flockbots **Flockbot:Behavior** for coordination.

To do Behavior-Signaling, we first trained the robots to do the **FlockbotInit** and **GotoBox** behaviors in the same way as in the Controller case. We also trained the Pioneer to do the **PioneerInit** behavior, except that *Done* was not signaled.

At this point the training diverged from the Controller case. We first trained three simple behaviors whose primary function was to signal to other agents:

- **InitDone** (Flockbot): The Flockbot stopped. This behavior was used to synchronize pushing among all robots.
- **PushForward** (Pioneer): The Pioneer did *GotoBox*. This behavior signaled to the Flockbots that the box was sufficiently straight and they could just move forward.
- **CorrectLeft** and **CorrectRight** (Pioneer): The Pioneer ran *GotoBox*. This behavior served to signal to the Flockbots that the box was tilted.

We then trained HFAs of each robot involve coordination:

- **PioneerPushBox**: Used *DistanceToBox*, *BoxAngle* and *Flockbot:Behavior* as features. The Pioneer did *GotoBox* to approach the box, and once both Flockbots signaled FlockbotInitDone, the Pioneer ran *PioneerPushForward*, correcting with *CorrectLeft* and *CorrectRight* as necessary when the box angle was too tilted.

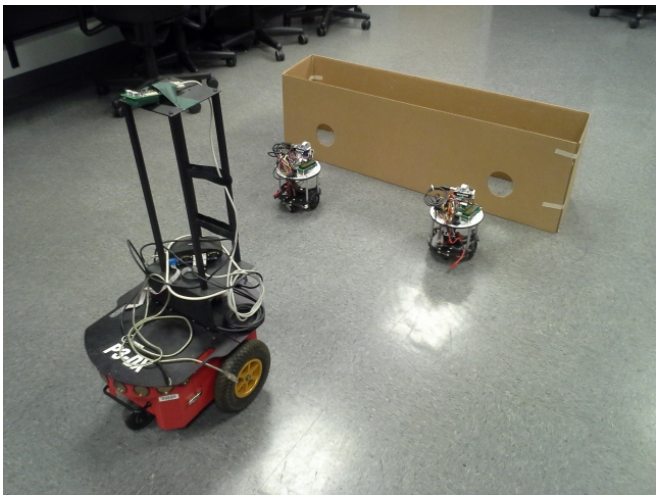

Fig. 5. Experimental Setup. The holes in the box are incidental.

– **LeftFlockbotPushBox** and **RightFlockbotPushBox**: These HFAs are symmetric, so only *LeftFlockbotPushBox* is described. The Flockbot ran *FlockbotInit* until *Done* was signaled, then transitioned to *FlockbotInitDone* behavior waiting for the Pioneer. Once the Pioneer started to run *Pioneer-PushForward* behavior, the Flockbot started to move forward, using the *ForwardFast* behavior. When the Pioneer performed the signal behavior *CorrectLeft*, the Flockbot would transition to *ForwardSlow*; and when the Pioneer performed the signal behavior *CorrectRight*, the Flockbot would stay at *ForwardFast*.

5.3 Experimental Details

Two experienced demonstrators each attempted to train the box-pushing experiment using both methods (Controllers and Behavior-Signaling). Training required teaching the Flockbots not to allow the box to deviate too far from a tilt angle of α and for the Pioneer to stay a given distance β away from the box. These values appear in transition edge labels in Figure 4. In all cases, the trainers endeavored to train with $\beta \approx 960$ mm and $\alpha \approx 11.5\,^{\circ}$.

6 Results

6.1 Demonstration

In all four cases (two experimenters, two methods), the robots were successfully trained to perform the proper box-pushing behavior in less than an hour each.

	Trainer 1	*Trainer 2*
Group Main	4	4
Push	10	10
Flockbot Controller Init	3	3
Pioneer Init	3	7
Goto Box	5	6
Flockbot Init	10	10
Total Samples	**35**	**40**

Table 1. Number of samples required to train each HFA using the Controller method.

	Trainer 1	*Trainer 2*
Correct Left (Right)	2 (2)	2 (2)
Push Forward	2	2
Goto Box	6	6
Pioneer Push Box	14	14
Flockbot Init	5	5
Init Done	1	2
Left (Right) Flockbot Push Box	14 (14)	14 (14)
Total Samples	**60**	**61**

Table 2. Number of samples required to train each HFA using the Behavior-Signaling method.

6.2 Comparison

Which method was simpler to train? The metric we used was the total number of user-provided demonstration samples required. Tables 1 and 2 break down these samples on a per-HFA and per-trainer basis.

The number of samples equates, more or less, to the amount of time required to train. As can be seen by the number of samples, certain complex behaviors were significantly harder to train than others. In general, the most complex behaviors took roughly between 10 and 15 minutes to train, while the simpler HFA took 2 minutes each. The trivial signaling HFA, *InitDone, CorrectRight, CorrectLeft,* and *PushForward*, took well less than 1 minute each.

We expected that the behavior Behavior-Signaling method would be simpler than the Controller method, as it required less complexity in terms of number of agents. But the experiments suggest that it actually required more total samples. This is partially because we had to train the Box-Pushing HFAs for *each* of the Flockbots separately. Additionally, due to the explicit coordination of the Behavior-Signaling method, the typical HFA was significantly more complex than in the Controller method. This could be another factor influencing Behavior-Signaling's scalability: the interaction among robots could make the HFAs too complex to easily decompose.

Still, this glosses over the fact that we had to hand-code domain-specific features for the Controller method (passing to controllers the Pioneer's *Box Angle*

and the fact that all robots are *Done*) and to manually specify joint behaviors (*Group Correct Right, Group Forward, Group Correct Left, Flockbot Group Init*). In the Behavior-Signaling method, none of this was necessary: robots simply needed to know what other behaviors other robots were doing, and custom signaled behaviors could be trained by the experimenter without any coding. In short: while the Controller method required less training, it required much more problem-specific coding.

The Controller is clearly superior to Behavior-Signaling method when there are a great many agents, since its hierarchy allows scalable automata with lower communication overhead. However, there is no reason the two techniques cannot be used simultaneously: we may try this in the future.

7 Is This Programming or Machine Learning?

The demonstration in this paper showed successful training of heterogenous multi-robot teams on a box-pushing task from the literature. Such training is rare, because the mixture of multiple agents, heterogeneity, and stateful behaviors yields a situation which often requires a large number of samples to successfully learn: but samples are very expensive.

Our approach enables a demonstrator to train a robot in a very short time, even in scenarios involving multiple heterogeneous robots and agents. We think this demonstration succeeded in showing this. But this was achieved by manually decomposing complex joint automata into very simple ones: indeed a great many are trivial and were able to be trained in a single presentation (thus ≤ 5 samples). The reason for this simplicity is that the decomposition process injects a great deal of user knowledge into the problem. For example, we *knew* that the Flockbots would have to be able to rotate the box, and so created joint behaviors which did just that.

Is this going too far? By using this level of task decomposition and joint behavior composition, we are in some sense *programming* the robots, only we are doing so not with a programming language but with the help of machine learning. We are trading domain knowledge (a kind of "programming") for the need to provide more samples. We believe that more general-purpose, knowledge-poor approaches have not proven successful in the multirobot training scenario: they are simply too costly in terms of samples. By injecting significant domain knowledge into the task we have been able to make headway in a quite difficult problem.

8 Conclusions and Future Work

We presented a multiagent supervised learning from demonstration system capable of training heterogeneous teams of agents in real time. By using a variety of tricks, we can simplify an ordinarily complex problem to the degree that supervised training becomes possible.

Using this, we have shown that heterogeneous teams can indeed be trained in real time by a single demonstrator to do a useful collaborative task. In previous work [19] we showed that HiTAB scales to large numbers of homogeneous agents. But we have not yet shown the combination of the two: training large heterogeneous robot teams with nontrivial stateful behaviors. Additionally, while we showed that training heterogeneous teams is *possible*, it still requires significant knowledge and (in the Controller case) coding. To make this method feasible, we need to address this. These issues will constitute much of our future work.

9 Acknowledgments

Our thanks to Kevin Andrea, Michael Bowen, and Ian Fleming.

References

1. Argall, B.D., Chernova, S., Veloso, M., Browning, B.: A survey of robot learning from demonstration. Robotics and Autonomous Systems 57(5), 469–483 (2009)
2. Atkeson, C.G., Schaal, S.: Robot learning from demonstration. In: Proc. ICML. pp. 12–20 (1997)
3. Blokzijl-Zanker, M., Demiris, Y.: Multi-robot learning by demonstration. In: Proc. AAMAS (2012)
4. Browning, B., Xu, L., Veloso, M.: Skill acquisition and use for a dynamically-balancing soccer robot. In: Proc. AAAI. pp. 599–604 (2004)
5. Chernova, S., Veloso, M.: Confidence-based multi-robot learning from demonstration. International Journal of Social Robotics 2(2), 195–215 (2010)
6. Freelan, D., Wicke, D., Sullivan, K., Luke, S.: Towards rapid multi-robot learning from demonstration at the robocup competition. In: Proc. RoboCup (2014)
7. Gerkey, B., Mataric, M.: Pusher-watcher: An approach to fault-tolerant tightly-coupled robot coordination. In: Proc. ICRA. pp. 464–469 (2002)
8. Grollman, D., Jenkins, O.: Dogged learning for robots. In: Proc. ICRA. pp. 2483–2488 (2007)
9. Luke, S., Ziparo, V.: Learn to behave! Rapid training of behavior automata. In: Proc. Adaptive and Learning Agents Workshop, AAMAS. pp. 61–68 (2010)
10. Luke *et al*, S.: The Flockbots. Open specification. Available at http://cs.gmu.edu/∼eclab/projects/robots/flockbots/ (2014)
11. Martins, M.F., Demiris, Y.: Learning multirobot joint action plans from simultaneous task execution demonstrations. In: Proc. AAMAS. pp. 931–938 (2010)
12. Mataric, M., Nilsson, M., Simsarin, K.: Cooperative multi-robot box-pushing. In: Proc. IROS. vol. 3, pp. 556–561 (Aug 1995)
13. Panait, L., Luke, S.: Cooperative multi-agent learning: The state of the art. Autonomous Agents and Multi-Agent Systems 11(3), 387–434 (November 2005)
14. Parker, L.E.: Heterogeneous Multi-Robot Cooperation. Ph.D. thesis, Massachusetts Institute of Technology (1994)
15. Pastor, P., Hoffmann, H., Asfour, T., Schaal, S.: Learning and generalization of motor skills by learning from demonstration. In: Proc. ICRA. pp. 763–768 (2009)
16. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, first edn. (1993)

17. Raza, S.: On teaching collaboration to a team of autonomous agents via imitation. In: Proc. IJCAI (2013)
18. Raza, S., Haider, S., Williams, M.A.: Teaching coordinated strategies to soccer robots via imitation. In: Proc. International Conference on Robotics and Biomimetics (2012)
19. Sullivan, K., Luke, S.: Learning from demonstration with swarm hierarchies. In: Proc. AAMAS (2012)
20. Sullivan, K., Luke, S.: Real-time training of team soccer behaviors. In: Proc. RoboCup (2012)
21. Sullivan, K., Luke, S., Ziparo, V.A.: Hierarchical learning from demonstration on humanoid robots. In: Proc. Humanoid Robots Learning from Human Interaction Workshop, HUMANOIDS (2010)
22. Takács, B., Demiris, Y.: Balancing spectral clustering for segmenting spatio-temporal observations of multi-agent systems. In: Proc. ICDM. pp. 580–587 (2008)
23. Yeasin, M., Chaudhuri, S.: Automatic robot programming by visual demonstration of task execution. In: Proc. ICAR (1997)