HIERARCHICAL MULTIAGENT LEARNING FROM DEMONSTRATION

by

Keith M. Sullivan
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computer Science

Committee:

_____	Dr. Sean Luke, Dissertation Director
_____	Dr. Zoran Duric, Committee Member
_____	Dr. Huzefa Rangwala, Committee Member
_____	Dr. Glen Henshaw, Committee Member
_____	Dr. Claudio Cioffi-Revilla, Committee Member
_____	Dr. Sanjeev Setia, Department Chair
_____	Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering
Date: _____	Spring Semester 2015 George Mason University Fairfax, VA

Hierarchical Multiagent Learning from Demonstration

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Keith M. Sullivan
Master of Science,
George Mason University, 2005
Bachelor of Science
Bachelor of Arts
Indiana University, 1998

Director: Dr. Sean Luke, Professor
Department of Computer Science

Spring Semester 2015
George Mason University
Fairfax, VA

Table of Contents

List of Tables

List of Figures

Abstract

HIERARCHICAL MULTIAGENT LEARNING FROM DEMONSTRATION

Keith M. Sullivan, PhD

George Mason University, 2015

Dissertation Director: Dr. Sean Luke

Developing agent behaviors is often a tedious, time-consuming task consisting of repeated code, test, and debug cycles. Despite the difficulties, complex agent behaviors have been developed, but they required significant programming ability. An alternative approach is to have a human train the agents, a process called learning from demonstration. This thesis develops a learning from demonstration system called Hierarchical Training of Agent Behaviors (HiTAB) which allows rapid training of complex agent behaviors. HiTAB manually decomposes complex behaviors into small, easier to train pieces, and then reassembles the pieces in a hierarchy to form the final complex behavior. This decomposition shrinks the learning space, allowing rapid training. I used the HiTAB system to train George Mason University's humanoid robot soccer team at the competition which marked the first time a team used machine learning techniques at the competition venue. Based on this initial work, we created several algorithms to automatically correct demonstrator error.

The primary thrust of the thesis is to apply HiTAB to the multiagent case, which presents two primary difficulties. First, multiagent systems typically have large, high-dimensional learning spaces requiring significant examples to sufficiently cover. Second, a daunting inverse problem exists: the demonstrator knows which macro-level behaviors the system should perform, but not the micro-behaviors each agent should perform to accomplish the desired macro-behavior. We developed two techniques to reduce the gap between micro- and macro-level behaviors: agent hierarchies and

behavior bootstrapping. Both techniques reduce the gap through simplifying the problem by reducing the number of agents being trained and reducing the number of agents directly interacting with each other. We applied agent hierarchies to a homogeneous swarm of hundreds of agents (the largest multiagent learning from demonstration system to date) and a group of robots performing a patrolling task. Additionally, we trained a heterogeneous group of robots to perform a box pushing task using agent hierarchies.

Chapter 1: Introduction

In the Humanoid League of RoboCup, teams of autonomous robots with human-like bodies and human-like sensors play soccer against each other. Almost all teams focus on developing robust, hand-coded behaviors for competition with the goal of winning the competition. The RoboPatriots, GMU's Humanoid soccer team (I am the primary developer.), initially followed this pattern by manually developing behaviors, but in 2011, 2012, and 2014, the RoboPatriots took a different approach, based on work in this thesis: the evening prior to competition, we deleted our hand-coded behaviors, and *trained* new behaviors by directly teleoperating the robot(s) on the field. Robots using these trained behaviors then preceded to win multiple games against hard-coded opponents. This was the first time *any* team at RoboCup used behaviors trained on the competition field.

This dissertation concerns HiTAB (Hierarchical Training of Agent Behavior), a method for training potentially large number of agents or robots to do complex, stateful behaviors in real time. HiTAB is the method which made possible our RoboCup first-ever successes. HiTAB is particularly useful in scenarios such as robotics, game development, computer animation, and agent-based simulation, where researchers or experimenters must specify complex behaviors for large numbers of agents. Developing agent behaviors requires considerable expertise and programming knowledge, in part due to developing a solid world model based on mathematics and dynamics. In addition, programming agent behaviors is a tedious, time-consuming task involving repeated code, test, and debug cycles. Despite the difficulties, complex agent behaviors have been developed, including foraging [1, 2], civilization modeling [3, 4], traffic control [5], and epidemiological models [6]. Creating these models required significant programming ability and debugging time which makes *training* the agents, rather than programming them, attractive.

Training is a natural way to train agents since it closely mimics how humans teach each other. In many domains, people show someone a task, then correct the behavior as the trainee performs the

task. Examples include sports, music, and physical therapy. As such, training offers an easy way for consumers, artists, and other non-technical users to reprogram their agent to perform the new task. Why would a consumer want to reprogram an agent? Consider that today's agent might perform a specific task, but in the future (measured in hours, days, months, years), that same agent might be called upon to perform a completely different task that the original designers did not anticipate.

One training approach is *Learning from Demonstration (LfD)*, where the agent learns a behavior in real-time based on provided examples from a human demonstrator. LfD teaches an agent a *policy* which maps environment features to agent action(s). The policy is learned from a database of examples (state/action pairs) provided by a demonstrator, and is constructed interactively: initially, the agent is in "training mode", where the demonstrator controls the agent. Every time the demonstrator changes the agent's behavior, the agent saves an example to the database. When the demonstrator has finished collecting examples, the agent learns the policy, and enters "testing mode" where the agent operates autonomously. Based on observing the agent in action, the demonstrator may then offer corrections to the agent. These corrections add further examples to the database, and the policy is then re-learned. This incremental learning approach allows the system to rapidly learn behaviors from minimal examples.

In general, LfD is applicable at different control levels: low-level control actions, basic behaviors (sometime referred to as action primitives), and high-level complex behaviors (e.g., Forage, PlaySoccer) [7]. This thesis focuses on applying LfD at the behavior level. I assume hard-coded modules handle low-level control (such as movement and sensor reading). As a result, the number of samples is low since data is only collected when the agent's behavior changes, and not continuously as the agent traverses the environment. Consider a robot arm learning to move through space. Samples (e.g., joint angles) are collected multiple times per second, resulting in a database containing thousands of points. In contrast, when using LfD to learn behaviors, the database only contains examples when the behavior was changed by the user.

This thesis develops a LfD system called Hierarchical Training of Agent Behavior (HiTAB), which learns behaviors as a hierarchical finite state automaton (HFA) represented as a Moore machine: individual states correspond to agent behaviors, or the states may themselves be another HFA. An

HFA is constructed iteratively: staring with a behavior library consisting solely of atomic behaviors (e.g., turn, go forward), the demonstrator trains a slightly more complicated behavior, which is then saved to the behavior library. The now expanded behavior library is then used to train an even more complex behavior which is then saved to the library. This process continues until the desired behavior is trained.

A challenge with training is the tradeoff between the real-time nature of training and the large number of examples required by highly complex problems (to sufficiently sample the learning space). It may be infeasible to collect hundreds or thousands of examples since each datapoint corresponds to part of an experiment conducted in real-time, which can be expensive and time-consuming, especially on real robots rather than simulated agents. All algorithms meant to train behaviors face this issue. To overcome this, HiTAB use a *behavior hierarchy* to reduce the size of the learning space. Complex behaviors are manually decomposed into much smaller, easier to learn behaviors, and the final behavior is iteratively composed from these simpler behaviors. For example, the behavior hierarchy allowed us to train a competitive humanoid soccer team with less than 200 examples in a matter of hours (see Chapter 6).

LfD has shown promise in training a single agent [8–11], but training multiple agents simultaneously (especially as the number of agents increases) is difficult due to unforeseen emergent phenomena caused by their interactions and the wide gulf between micro-level and macro-level behavior. An inverse problem results from when the experimenter knows the desired macro-behavior (e.g., storm the castle), but does not know the individual micro-behaviors (e.g., move forward, throw spear) each agent must perform to achieve the macro-behavior. As a result, the number of required samples increases dramatically as does the size of the learning space. Thus most work on cooperative teams has focused on weaker, non-supervised optimization techniques such as reinforcement learning and genetic algorithms. Furthermore, rapid training of multiple agents is rarity in the supervised learning community due to the inverse problem discussed above.

In this thesis, I extend HiTAB to the multiagent case by organizing the agents in a hierarchy: agents are arranged in a tree where leaves represent regular agents, and non-leaf nodes are *controller agents.* Controller agents control a small group of subordinates (other regular or controller agents),

thus reducing the number of interactions which then reduces the size of the learning space. The combination of a *behavior hierarchy* with an *agent hierarchy* allows rapid training of multiple agents to perform a complex task, via supervised learning. HiTAB's agent hierarchy reduces the gulf between the micro- and macro-level behaviors by decomposing the problem into small enough pieces where the inverse problem is manageable. As a result, from a machine learning perspective, the decomposed behaviors are simple, which is the point. The simple behaviors require a limited number of samples which allows rapid training, and places HiTAB between explicit programming and unfettered learning.

Further complicating training multiple agents is the situation where agents must cooperate directly with one another to accomplish a task. To this end, I developed a technique called *behavior bootstrapping*, which offers a way to rapidly train coordination and cooperation between agents by only training a single agent rather than multiple agents simultaneously.

HiTAB's rapid training can introduce errors, such as the trainer providing the incorrect behavior for a given world situation. Obviously, one can immediately delete the errant example and provide a correct one (a sort of UnDo operation). More commonly, the trainer will observe incorrect behavior after the HFA is constructed. Then the question is how to identify the errant example(s) which cased the incorrect behavior. Simply adding more examples to reduce the statistical impact of the errant examples is not feasible since each new example requires a new experiment. To solve this problem, I developed two *unlearning* methods to identify and remove errant examples. These unlearning techniques conservatively identify possible examples for deletion, and then remove them only after additional tests are passed.

1.1 Thesis Contributions

There is very little existing work in multiagent learning from demonstration. This thesis aims to help fill the void by developing new approaches to multiagent LfD, particularly how to train multiple agents and how to train cooperation. The thesis has the following contributions:

- Based on work by Luke and Ziparo [12], I developed the only really effective large-scale

real-time multiagent learning from demonstration system.

- I introduced *behavior bootstrapping* as a technique to train coordination and cooperation in flat swarms of both homogeneous agents.
- I developed *agent hierarchies* which allow training of unlimited sized homogeneous and heterogeneous swarms.
- I introduced the idea of *unlearning*, a technique to correct demonstrator error while preserving undersampled regions of the learning space.
- I developed multiple humanoid robot architectures for the Humanoid League at RoboCup, and, successfully used learning from demonstration on these robots at the competition, the first time LfD has been used in the Humanoid league. Notably, the trained robots performed better than hard-coded ones!

1.2 Thesis Layout

This thesis is organized as follows: Chapter 2 reviews LfD literature and associated topics in multiagent system related to this thesis while Chapter 3 discusses how my prior work led to the development of HiTAB. Chapter 4 introduces the formal HiTAB model and presents single robot experiments illustrating how HiTAB can be used by novices and HiTAB's applicability to RoboCup. Chapter 5 presents several algorithms to correct demonstrator error within HiTAB. Chapters 6 and 7 apply HiTAB to the homogeneous multiagent case: Chapter 6 presents behavior bootstrapping as a method to train coordination, while Chapter 7 presents agent hierarchies and controller agents. Chapter 8 extends the agent hierarchies to the heterogeneous case. Finally, Chapter 9 summaries the thesis work and offers suggestions for future research.

Chapter 2: Background and Related Work

This thesis is concerned with the application of supervised machine learning to multiagent systems. Before describing the intellectual work, this chapter introduces background material and related work. After introducing finite-state machines and classification, I present background information about multiagent systems before reviewing related research.

2.1 Finite-State Machines

HiTAB represents behaviors as finite-state machines (FSM)[1] which are abstract models of computation that can be in one of a finite number of *states*. Machines can only be in one state at a time, and periodically change from one state to another state based on some condition (called a *transition*) [13]. Formally a FSM is a tuple (S, I, O, T, s_0) [14] where

- S is a non-empty, finite set of states
- I is a finite set of input values (sometimes called the input alphabet)
- O is a finite set of output values (sometimes called the output alphabet)
- T is a transition function: $S \times I \rightarrow S \times O$. Conventionally, T is a partial function: it does not define a mapping for all possible combinations of $s \in S$ and $i \in I$.
- $s_o \in S$ is the initial (start) state.

At every step of computation the FSM maps the current state $s_t \in S$ and current input $i_t \in I$ to the next state and output value via the transition function: $T(s_t, i_t) = (s_{t+1}, o_{t+1})$. Given a sequence of input signals and an initial state, an FSM will produce a sequence of states and output values. All sequences are potentially infinite. This thesis focuses on deterministic machines where there is only

[1] Also called finite state automata (FSA). I use the terms machine and automata interchangeably throughout the thesis.

one possible output for a given state/input pair, as opposed to non-deterministic finite state machines which have multiple outputs for a given state/input pair [15]. FSMs have limited computation power due to their finite memory as compared to Turning machines, which have infinite memory and, thus, are capable of solving much more complex problems [16]. Even with their simplicity and limited memory, FSMs have been successfully used to model a wide variety of problems including gesture recognition [17], natural langue processing [18], compilers [19], digital event reconstruction [20], bipedal robot gaits [21], and behaviors for video game characters [22].

HiTAB uses a particular type of FSM called a *Moore machine* [23]: a finite-state automata whose outputs are solely determined by its current state.[2] Moore machines have been used for multiagent behavior modeling [25], controlling behavior for Sony AIBOs in Robocup [26], generating behavior networks for mobile robots [27], and opponent modeling in multiagent learning [28].

FSMs, which are flat and sequential, cannot model highly complex systems due to the large number of states and transitions involved. Additionally, FSMs cannot capture commonalities between similar states. Developed by Harel [29], the idea of a *hierarchal finite automata (HFA)* gets around these problems. In an HFA, a state many be an additional HFA and automata may be nested as deep as necessary, but recursion is not permitted. HFAs capture a way to reuse common behaviors across many states which allows rapid creation of new behaviors [30]. HFAs have been used for digital control [31], control systems for space robotics [32], and behavior control in multiagent systems [33]. HiTAB's behavior hierarchy depends heavily on HFAs.

2.2 Classification

At its heart, HiTAB uses a *classifier* to learn the mapping between states and actions (the transition function of a Moore machine). Classification (or applying a classifier) is the process of assigning a class to an observation (typically represented as a vector) on the basis of known training data. In unsupervised classification (also called clustering), no information about the class-labels is known, so the goal is to group unlabeled data into clusters such that data within each cluster share some

[2]The other common type of FSM is a *Mealy* machine [24] whose outputs are determined its current state and the value of all its inputs.

common property. Unseen examples are classified based on which cluster they are most similar to. Common algorithms include k-means clustering [34] and mixture modeling [35]. In supervised classification, we are given a set of training examples $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n) \in X \times Y$ and try to learn a function $y = f(x)$ that predicts the classification y of unseen examples x with minimal error. Supervised classification is apropos for HiTAB due to its training nature: every example is associated with a human-provided (assumed correct) behavior. Supervised classification algorithms include support vector machines (SVMs), decision trees, K-nearest neighbor (K-NN), and neural networks [36]. While HiTAB works with any supervised classification method, this thesis focuses on decision trees, K-NN, and support vector machines.

Related to supervised classification is the idea of boosting [37], which is based on the notion that the combination of many simple classifiers can perform better than a single complex classifier. Boosting starts by determining a classification rule based on a subset of the training data, and continues learning different rules based on different subsets of the training data. After several iterations, the individual rules are then combined into a single rule that, hopefully, is significantly more accurate then the single classification rules. While boosting has improved classification accuracy in many problems [38–42], HiTAB does not use boosting due to a lack of training data.

2.2.1 Decision Trees

The primary classifier used within HiTAB is a *decision tree*, which approximates discrete-valued functions as a decision tree where internal (non-leaf) nodes are a test on a single attribute (each branch represents the outcome of the test) and leaf nodes are class labels. Decision trees can also be represented as more human readable if-then rules.

Decision tree classification is well suited for problems with the following characteristics [43]

- Instances are represented as attribute-value pairs.
- The target function has discrete output values.
- Disjunctive descriptions may be required.

- Training data might contain errors in either class label or attribute values.
- Training data might contain missing values.

Additionally, decision trees can handle different attribute types such as discrete, continuous, torodial (e.g., angle), and categorical. Decision trees are a popular inductive learning method which have shown success on a broad range of problems including lane detection for autonomous driving [44], data mining [45], decision making in robotic soccer [46], language games [47], and image segmentation [48].

Decision trees classify a new instance by traversing the tree starting at the root and moving towards the leaves: testing the appropriate attribute at each level, and moving down the branch corresponding to the attribute's value. The process recurses at the subtree rooted at the new node. When a the subtree consists of a single leaf, the class label associated with the leaf is returned [36].

A central choice when constructing a decision tree is which statistical test to use when splitting the training examples. The most common is *information gain* which is based on *entropy* that characterizes the purity of an collection of examples [49]. Given a collection of examples S with c possible classes, entropy is defined as

$$Entropy(S) = -\sum_{i=0}^{c} \log_2 p_i \tag{2.1}$$

where p_i is the proportion of examples in S with class i. With this definition of entropy, information gain is simply the expected reduction in entropy from splitting the examples with a given attribute. Formally, information gain for a collection S of examples on attribute A is

$$Gain(S,A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v) \tag{2.2}$$

where $Values(A)$ is the set of possible values of A, and S_v is the subset of S for which attribute A has value v. The first term of Equation 2.2 is the entropy of the entire collection while the second term is the expected value of the entropy after S is partitioned using A. Thus, $Gain(S,A)$ is the expected

reduction in entropy when partitioning S by A.

Using information gain, a decision is tree is constructed top-down in a greedy fashion which never backtracks to re-consider past decisions. Starting at the root, the instance attributes are evaluated using a information gain to determine how well they classify the training examples [43]. The best attribute is selected, and descendant nodes are created for each possible attribute value. The training examples are sorted to the appropriate descendant node, and the process continues from each descendant node. An individual attribute might be used multiple times during construction.

Since each node uses a single attribute, decision trees induce hard, axis-aligned splits of the attribute space. Each node splits the data according to a hyperplane, and classification is a series of tests against hyperplanes to determine which rectangular region of attribute space the example lies in. Oblique decision trees relax the axis-parallel hyperplanes by using a linear combination of attributes at each node [50]:

$$a_{d+1} + \sum_{i=i}^{d} a_i x_i \geq 0$$

where $a_1, a_2, \ldots, a_{d+1}$ are real valued coefficients. Training an oblique decision tree has exponential computational complexity, and, in fact, training an oblique decision tree is NP-Hard [51]. Due to the sparsity of samples, HiTAB uses an axis-parallel split in its decision trees.

Typically decision trees are *pruned* after construction: the greedy top-down construction procedure can produce overly complex trees that overfit the training data. Pruning seeks to simplify the resulting tree while improving accuracy by removing sections of the tree which provide little power to classify. Reduced error pruning [52], error pruning [52], cost-complexity pruning [53], critical value pruning [54] are common pruning algorithms which require a separate pruning set (typically a subset of the training data, but it could be a third independent set). Pessimistic Error Pruning relaxes the requirement of separate pruning set [55]. HiTAB does not perform pruning since the decision trees resulting from training are typically not very large (in fact, one could call them decision stumps rather than trees), and, thus, are not amenable to pruning. Even if pruning was desired, training does not produce enough examples to construct adequate pruning sets.

C4.5

Several decision tree algorithms have been developed over the years including ID3 [55], CART [53], CHAID [56], and MARS [56]. HiTAB uses an algorithm called C4.5 [52, 57] which is successor of ID3. C4.5 extends the basic decision tree algorithm described above (which is what ID3 uses) by allowing both continuous and discrete attributes: continuous values are handled by creating a threshold t and splitting the examples into those which are above the threshold and those below the threshold. To compute the threshold for attribute A, the examples are first sorted into increasing order of A giving a list of distinct values $v_1, v_2, \ldots, v_n$ which produces a list of potential thresholds $t = (v_i + v_{i+1})/2$. The threshold that yields the best information gain is then used.

2.2.2 Support Vector Machines

The Support Vector Machine (SVM) is a kernel-based classification learning technique which attempts to abstract out the issue of learning bias [58]. SVMs construct a hyperplane which divides examples such that examples of one class are all on one side of the hyperplane, and examples of the other class are all on the other side. However, interesting data is rarely linearly separable. Thus, the SVM first uses a function Φ to transform the data to a higher dimensional space where it is linearly separable, and then applies the hyperplane. Because the transformation Φ may be to many, even infinite, dimensions, the SVM does not actually perform the transformation. Instead, noting that the dot product is all that is necessary to compute the optimal hyperplane, the SVM composes the transformation Φ and the dot product in the higher dimensional space into a single kernel function k, which computes the dot product of two vectors when they are transformed into the higher space. This notion, dubbed the *kernel trick*, allows easy application of Φ transformation for purposes of classification to large dimensional spaces [59].

Consider input data of the form (x_i, y_i), where the vectors x_i are in a dot product space $\mathscr{H}$ (such as a real-valued multi-dimensional space), and y_i are the class labels. Formally, any hyperplane in $\mathscr{H}$ is defined as

$$\{x \in \mathscr{H} \mid \langle \mathbf{w}, x \rangle + b = 0\} \quad \mathbf{w} \in \mathscr{H}, b \in \mathbb{R}$$

where $\mathbf{w}$ is a vector orthogonal to the hyperplane and $\langle \cdot, \cdot \rangle$ represents the dot product. In a SVM, the idea is to find the hyperplane that maximizes the minimum distance (called the *margin*) [36] from any training data point. The following constraint problem describes the optimal hyperplane:

$$\min_{w \in \mathscr{H}, b \in \mathbb{R}} \quad \tau(\mathbf{w}) = \frac{1}{2}||\mathbf{w}||^2 \tag{2.3}$$

$$\text{subject to} \quad y_i\left(\langle \mathbf{x_i}, \mathbf{w} \rangle + b\right) \geq 1 \tag{2.4}$$

for $i = 1, 2, \ldots m$ where m is the number of training examples. This optimal hyperplane minimizes training error while maximizing the margin, which is believed to translate to maximizing generalization ability.

Using the kernel to transform the dot product to a higher dimensional space (i.e., "kernelize" the hyperplane) results in the following dual form of the optimization problem:

$$\max_{\alpha \in \mathbb{R}} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \tag{2.5}$$

$$\text{subject to} \quad \alpha_i \geq 0, \tag{2.6}$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0. \tag{2.7}$$

Any vector x_i which lies on the margin, i.e.,

$$\alpha_i[y_i(\langle w, x_i \rangle + b)] = 1$$

is called a *support vector*. The optimal hyperplane is thus determined by data examples which form this set of support vectors.

At this point, it is unknown if a separating hyperplane is the best solution, or even if one exists. Allowing a certain fraction of outliers can alleviate this problem by making the formulation less

brittle. Called *soft margin hyperplanes*, the idea is based on slack variables $\zeta_i \geq 0$ which allow the presence of outliers without influencing the hyperplane, which results in the hyperplane

$$y_i(\langle \mathbf{w}, x_i \rangle + b) \geq 1 - \zeta_i$$

Thus, a classifier that generalizes well is found by controlling the sum of the slack variables. A term is added to the objective functions to prevent the trivial solution where all ζ_i take on large values. The simplest solution with slack variables, called C-SVC (support vector classifier), tries to determine the tradeoff between minimizing training error and maximizing the margin [59]. However, there is no intuitive a priori manner to determine this tradeoff [60]. An alternative method, called ν-SVC, seeks to control the number of margin errors and support vectors via the following optimization problem [61]:

$$\max_{\mathbf{w} \in \mathscr{H}, \zeta_i \in \mathbb{R}^m, \rho, b \in \mathbb{R}} \quad \frac{1}{2}||\mathbf{w}||^2 - \nu\rho + \frac{1}{m}\sum_{i=1}^{m} \zeta_i \tag{2.8}$$

$$\text{subject to} \quad y_i(\langle \mathbf{w}, x_i \rangle + b) \geq \rho - \zeta_i \tag{2.9}$$

$$\zeta_i \geq 0, \ \rho \geq 0 \tag{2.10}$$

ν is simultaneously an upper bound on the fraction of margin errors and a lower bound on the fraction of support vectors, and so it provides the tuning parameter for how soft the hyperplane is.

2.2.3 K-Nearest Neighbor

K-nearest neighbor (K-NN) classification is an instance-based technique to approximate discrete-valued target functions under the assumption that all examples correspond to points in $\mathbb{R}^n$ [62]. There is no concept of training a K-NN classifier: training is simply storing the data. To classify a new instance, K-NN determines the K closest neighbors (using standard Euclidean distance) from the stored training data. The new instance is then assigned the class of the plurality of the K

closest neighbors; thus, K-NN is sensitive to the local structure of the data.[3] One disadvantage of this approach is that classifying new instances has high computational overheard since all the computation occurs during classification rather than in training [63]. Therefore, the main implementation issue is quickly determining the K neighbors, especially for large datasets. The most common solution is KD-trees [64] which allow fast $O(\ln N)$ determination of the K neighbors. However, if the number of datapoints is close to the number of dimensions, then a simple linear scan of the data is equally fast with less memory.

The inductive bias of K-NN corresponds to the assumption that examples with the same classification are nearby in Euclidean distance. However, relying on *all* attributes rather than a subset (like decision trees) can cause practical problems: the Euclidean distance will be based on irrelevant attributes, which could cause misleading classifications [43]. Since HiTAB uses a small number of features for each behavior, HiTAB's K-NN assumes all features are relevant (since the user chooses the features a priori), thus, it does not suffer this problem.

2.3 Multiagent Systems

The thrust of this thesis is extending the basic HiTAB algorithm to the multiagent case. In this section, I introduce some multiagent systems concepts which guided how I extended HiTAB to the multiagent setting. I consider an agent an autonomous computational entity which can sense the environment, make a decision based on its sensor information, and then act based on its decision. A multiagent system is then a collection of N agents interacting in some manner with various constraints (e.g. communication, heterogeneous vs. homogeneous agents, available information) [65]. I focus on *cooperative multiagent systems*, where agents actively assist each other, rather than *competitive multiagent systems* where agents have an adversarial relationship.

Multiagent systems have several advantages over single agent systems: redundancy, ability to parallelize tasks, they can solve inherently distributed tasks, and they can solve problems requiring cooperation [66]. Still, these advantages have costs: multiagent systems have radically increased

[3] In fact, K-NN constructs a Vorononi diagram on the hypothesis space.

learning space (curse of dimensionality) caused by agent interactions and potential emergent phenomena; inter-agent communication; and, in the case of multirobot systems, maintenance.

2.3.1 Control Structures and Architectures

One of the first decisions when constructing a multiagent system is how to structure the agent's decision making process. In other words, how does the agent process sensor information and alter its behavior based on the processed information. Four broad control methods exist, and the proper choice depends on the current environment and problem [67]: deliberative, reactive, hybrid and behavior-based control. *Deliberative systems* amass all available information and internal knowledge to make a decision. Usually the architecture is decomposed into function modules such as sensor processing, world modeling, planning and execution [68]. This decomposition allows complex behaviors but implies a strict interdependency structure among the modules [69]. Deliberative architectures use a planning system for reasoning, and are the primary computational load. However, planning requires an accurate world model which limits deliberative architectures to static environments. *Reactive* systems involve minimal sensor processing and no reasoning to make decisions [70]. Similar to stimulus-response systems, reactive systems work well in dynamic, unstructured environments where a world-model is not required [71]. Reactive architectures are typically based on rule systems which limit their ability to store information or have memory. However, reactive systems work quite well for certain classes of problems. *Hybrid systems* combine deliberative and reactive architectures together to get fast responding systems with rational and optimal decision making. The challenge is coordinating the deliberative and reactive pieces since they typically operate at different time scales [72]. Usually, hybrid systems use a third component to manage coordination, thus, hybrid systems are sometimes called three-layer architectures [73]. *Behavior-based architectures* use modular, distributed behaviors to control agent behavior. Behaviors are arranged in a network where individual behaviors receive input from sensors or other behaviors [74]. There is no centralized world model; individual and networked behaviors maintain state and world models.

Normally a multiagent system is split across multiple control strategies based on the tradeoff between response time, modularity, and the ability to look ahead [67]. HiTAB depends on multiple

control strategies: low-level behaviors are typically reactive or deliberative, while higher-level behaviors (e.g., trained behaviors) rely on behavior-based control and hybrid structures.

A multiagent control method is implemented within a multiagent control architecture, which must satisfy the same control issues as single agent architectures in addition to handling agent interactions and emergent behaviors. While there are infinite ways to program MAS, all methods fall into four general categories [75]: *centralized*, *decentralized*, *hybrid*, and *hierarchical*. Single agent HiTAB uses a centralized architecture, while multiagent agent HiTAB uses a combination of all four architectures.

Centralized architectures control the entire system from a single control point. The central control point receives sensor information from all the agents, processes the joint information, and the broadcasts control commands to all the agents [76]. Centralized systems have practical limitations due to the single point of failure and communication bandwidth, and they tend to have exponential computational complexity [77]. However, centralized systems work well when the controller has a clear vantage point and can easily broadcast control commands (e.g., Micro-Robot World Cup Soccer Tournament [78] and the small size league in Robocup [79]).

Decentralized control architectures are very common and robust against failure due to the lack of a central point of failure [80]. The agents make decisions based on local information. However, communicating information to the entire group may be difficult (e.g., when communication range is short compared to the environment size) [81]. All the robot teams discussed below use a decentralized architecture.

Hybrid architectures combine local control with higher level controls capable of influencing every agent [82]. A common approach is to extend the traditional three-layer single agent architecture, either by allowing individual layers to directly communicate with each other [83], or by adding an additional global layer which handles communication and coordination [84].

Hierarchical architectures organize the agents into a tree such that agents oversee small groups of agents and are a common method to control an agent. Hierarchies are a natural way to achieve layered learning via task decomposition [85, 86]. Extending this approach to policies, Saunders et al trains sequences iteratively built on previous ones [87]. Plans can also be iteratively constructed

in a similar fashion [88]. Hierarchies have been extensively applied to multi-agent reinforcement learning, as in [89].

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 [90] (e.g., Real-time Control System [91, 92], and the Subsumption architecture [69]). Hierarchies *among* agents are less common, for example [93, 94]. Some recent literature has focused on *hierarchies of control* among heterogeneous agents [95]. Hierarchies may also be constructed dynamically as a mechanism for task allocation [96].

A large body of work has constructed groups of agents with each agent employing its own internal hierarchical behavior mechanism. The ALLIANCE framework embeds individual behavior hierarchies in distributed agents [97] while MONAD uses a general purpose behavior hierarchy [98]. The Millibots project focused on hierarchies of control among heterogeneous agents [99].

2.4 Multiagent Learning

Multiagent systems are typically grounded in large, unpredictable, dynamic environments which make it extremely difficult to program agent behaviors. Successful programming would require knowing, a priori, all possible environmental conditions and corresponding agent states. Machine learning offers a reasoned way to give agents the ability to individually change their behavior to improve system performance. While the application of machine learning to a single agent has a long, rich history, the application of machine learning to multiagent systems has exploded in the past several decades, with many dedicated conferences and journals [100]. Multiagent learning is the application of machine learning to multiagent systems, and falls into two broad approaches [66]: *centralized learning* where a single learner discovers a set of behaviors for a team of agents; and *decentralized learning* where each agent has its own, independent learning algorithm (which introduces game-theoretic issues not addressed in this thesis). Using a single learner side-steps game theory, but does introduce issues related to joint behaviors, especially the state space size. Consider two agents each of which can be in 100 states. Then, the joint behavior can be one of 10,000 states.

This exponential explosion can cause problems for learning algorithms which explore state utilities, such as reinforcement learning [101].

When each agent has its own, independent learning algorithm, the large state space is projected into smaller spaces. If the problem is decomposable, then each piece may be learned in parallel. However, individual agents do not know other agent's behavior which means agents are adapting their own behavior in the context of other agents. Thus, as agents continually adapt to each other they (potentially) invalidate other agent's behavior by invalidating many assumptions of traditional machine learning techniques [102].

Both decentralized and centralized learning require a method to assess performance. In *supervised learning*, a teacher provides the correct action, while in *unsupervised learning*, no explicit feedback is provided, and in *reward-based learning* a critic provides a quality assessment (the "reward") [103]. The complexity and large feature space of multiagent systems typically preclude supervised learning approaches due to the reliance on a teacher to provide the correct behavior for all situations. As a result, the vast majority of multiagent learning is reward-based [104].

The majority of supervised multiagent learning research covers agent modeling, where agents learn about each other rather than attempting to solve a task [105]. For example, in [85], the supervised task ("pass evaluation") is reasonably described as agent modeling, while the full multiagent learning task ("pass selection") uses reinforcement learning. Supervised training, as opposed to agent modeling, requires agents be told which micro-level behaviors to perform in all situations, but the experimenter does not know this: they only know the higher level macro-behaviors the system should perform. The standard solution to this inverse problem is reward-based techniques (e.g., reinforcement learning [106]). However, HiTAB is trying to learn a mapping between states and actions, inherently a supervised problem, which makes HiTAB highly unusual. To reduce the gap between micro- and macro-level behaviors, HiTAB takes the approach of decomposing the problem into smaller pieces, thus, reducing the gap between micro- and macro-level behaviors.

2.4.1 Problem Decomposition

Central to HiTAB is the notion of problem decomposition, which splits a problem into smaller, easier to handle pieces and has been used in machine learning for decades (e.g., [87, 107–110]). Decomposition also allows knowledge sharing and reuse through sharing of previously learning simple concepts. *Behavior fusion* addresses the inverse problem: how to automatically combine simple behaviors into more complex behaviors [111, 112]

Manual problem decomposition depends on a human to provide an appropriate decomposition, which relies on human knowledge of the problem at hand [108]. However, when decomposing a problem, the experimenter must be aware how much human knowledge is encoded within the decomposition: injecting too much human knowledge might over constrain the hypothesis space preventing adequate learning, but human knowledge might make learning easier and allow learning more complex tasks [113].

Circumventing this issue, automatic problem decomposition eliminates the human by decomposing the problem based on data analysis [114–119]. For example, using large training sets and neural networks, Jacobs learned the appropriate decomposition for a vision classification task [120]. Due to the lack of data, HiTAB uses a manual problem decomposition.

2.4.2 Layered Learning

HiTAB is similar to the notion of layered learning, which is a machine learning technique for multiple agents to learn complex coordinated behaviors. Layered learning decomposes complex behaviors into a hierarchy, and applies machine learning algorithms at each layer [121]. Starting at the bottom, the inputs to a layer's machine learning algorithm is the output's from the previous layer's algorithm (see Figure 2.1). The first layer uses input from the environment. In Stone's original formulation, once a layer was done learning, it was frozen. Later work relaxed this assumption and allowed lower layers to continue learning, in a processes called concurrent layered learning [122]. Gustafson and Hsu used layered learning and genetic programming to play simulated soccer [123]. Utgoff and Stracuzzi developed a similar on-line layered learning approach, called many layered learning, which incrementally constructs building blocks for use in multi-layered neural networks [124].

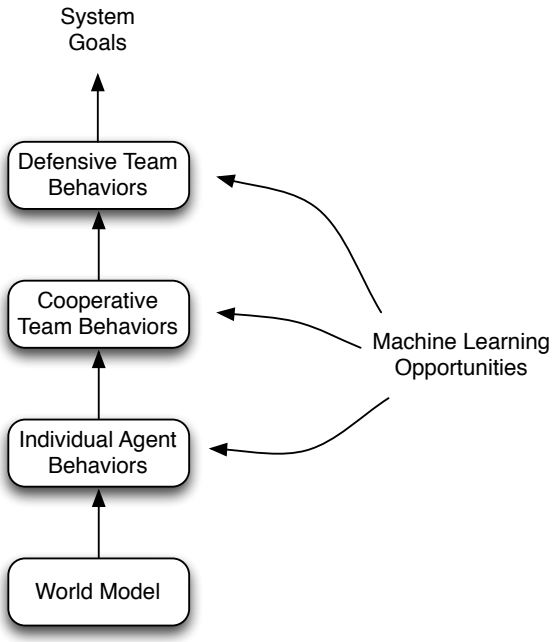


Figure 2.1: Layered learning example

2.5 Learning from Demonstration

With the previous sections as background information, this section reviews current learning from demonstration literature [7, 125, 126], which HiTAB builds upon. The LfD literature may be divided into two broad categories: those which learn *plans* [88, 127] and those which learn (usually stateless) *policies* [128, 129] (for a stateful example see [130, 131]). 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 while policy methods normally have a surfeit of data. This thesis lies in the plan method category.

Two problems present themselves for systems that learn plans or policies. First, proprioceptive feedback experienced by the demonstrator is not communicated to the agent [132], and, second, there may be a physical mismatch between the demonstrator and the agent – the correspondence problem [133]. One way to avoid these two issues is for the demonstrator to tele-operate the

agent. For example, [134] collected data from a flight simulator to train an agent to fly, and Young et al developed a broomstick control interface to drive a differential drive robot, thus, providing demonstration examples which accounted for the robot's kinematics [135].

For systems which learn polices, the system gathers a set of $(state, action)$ pairs based on observation of a human demonstrator, and then builds a policy function i.e., a mapping between states and actions. Many authors approach this problem as a reinforcement learning problem based on how closely the learned policy matches the $(state, action)$ trajectory provided by the demonstrator [89, 129, 136–139]. Called *imitation learning*, this approach has shown promise in getting humanoid robots to mimic a human's behavior [136, 137, 140–151], and to train agents to play the real-time strategy video game WARGUS [152].

Imitation learning, initially called *Programming by Demonstration (PfD)*, attracted attention in the 1980s since it appeared as a promising way to program industrial robots. Symbolic reasoning was among the first approaches to PbD through methods such as teach-in, guiding or play-back. In these methods, the demonstrator teleoperated the end effector to the desired position. The system records forces and positions, and then splits the trajectory into discrete subgoals represented as state-action-state transitions [153, 154]. Dufay and Latombe considered variability of human demonstrators by converting state-action-state sequences into if-then rules [155].

These early works highlighted the need for demonstration modalities the robot can understand and providing enough examples to achieve the desired generality [156]. Thus, traditional ways of guiding the robot were replaced by more user-friendly interfaces such as vision [157–159], data gloves [160], lasers [161] or kinesthetic teaching [162, 163]. As the field starting moving away from simply copying demonstrated motions, machine learning became prominent as a way to generalize behaviors [164–168].

2.5.1 Human Robot Interaction and Learning from Demonstration

The field of human-robot interaction (HRI) offers insights how to better train agents since LfD has primary two pieces: learning the task and learning the intent of the demonstrator. Social cues offer a way to speed up learning through the introduction of statistical priors, and by offering insights

to the demonstrator's intent [169]. Pointing and gesturing are common ways to communicate the demonstrator's intent [110, 148, 170–175] as is speech [176–179]. However, demonstrations might be ambiguous in intent, necessitating the agent building a cognitive model of the demonstrator [180, 181]. Still, knowing the demonstrator's intent is not sufficient. In particular, different morphologies might require different solutions [182], or an optimal solution chosen from a set [183, 184].

2.5.2 Multiagent Learning from Demonstration

Despite the long history of LfD, there is remarkably little work involving multiple agents (probably due to the large learning space and gulf between micro- and macro-behaviors). Still, there has been some work in multiagent LfD. The majority of which has focused on teaching multiple independent agents by a single teacher.

Chernova et al use confidence estimation to train multiple robots individually and rely on emergent multirobot behavior to accomplish the task [185, 186]. They present three methods for multi-robot coordination based on Confidence-Based Autonomy (CBA). In CBA, a robot requests a demonstration when it is unsure about which action to perform. The first coordination method, implicit coordination, uses no communication between robots and defaults to all the robots performing independent actions with coordination occurring due to environmental changes. Coordination through active communication, the second technique, has the teacher demonstrate when to communicate. The final technique, coordination through shared state, has the robots continually communicate. Based on experiments using two humanoid robots performing a ball sorting task, the authors conclude that communicating shared state is the most general approach and requires the fewest demonstrations.

Grollman et al used a similar approach to train Sony AIBO robots to play soccer [187–189]. Their LfD system uses Guassian Process Regression to learn the mapping between states and actions. Similar to CBA above, the robot bases its action selection on a confidence threshold: if confidence of the current action is below a threshold, the robots asks for additional demonstrations. Experiments trained the robot to approach and acquire the ball in a soccer task.

Several papers relaxed the assumption of using a single human with multiple agents. All papers rely on the interaction between the humans demonstrators for coordination, rather than teaching the

robots how to coordinate. First, Blokziil-Zanker trained multiple robots using multiple demonstrators to train a plan for a heterogenous robot team to open doors [190]. Second, Raza et al used human demonstrators to train multiple soccer agents within the RoboCup 3D Simulation League [191, 192]. An alternative approach has multiple humans each perform single robot learning from demonstration, and then all demonstrations are combined to learn a group behavior policy [193, 194]. While using multiple humans works well for small teams (say, less than 3), it is unlikely to scale since one human is required for each robot. Still, a hybrid approach could be used: using techniques in this thesis a single demonstrator could train a swarm, and then multiple human controlled swarms could work together to solve a problem.

2.5.3 Metrics in Learning from Demonstration

As the previous sections show, learning from demonstration has wide applicability and many algorithms have been presented in the literature. However, to date, there is only one paper comparing multiple LfD algorithms [195]. This lack of comparison is likely due to the relative immaturity of the field, and that different researchers use different platforms, interfaces, and demonstration techniques which lead to different representations that make direct comparisons difficult. Compounding the issue, the LfD lacks common datasets due to the human interaction component and diversity of interaction techniques. Thus, comparison requires reimplementation of multiple techniques, a time-consuming and error-prone task.

Suay, Torris, and Chernova are the first paper to compare multiple LfD algorithms in a systematic manner [195]. They implemented three common LfD algorithms and had novice users train a humanoid robot to remove small toys from a table-top. Independent of the LfD algorithm, quantitative results showed that users need a natural interface with the robot and a solid understanding of the robot's internal state. Qualitative results showed the superiority of one algorithm – Confidence Based Autonomy – performed best and was well received by the study participants.

Af LfD matures, common evaluation metrics will develop, driving the development of general-purpose LfD techniques. In the meantime, researchers are starting to develop metrics for specific areas of LfD such as evaluation of robot imitation [196] and demonstrator quality [197, 198]. Additionally,

the human-robot interaction community might contribute to developing LfD metrics [199].

Developing new evaluation metrics for LfD is outside the scope of this thesis. Thus, throughout the thesis, I simply use the number of training examples (and associated wall-clock time to gather the examples) as the primary metric. Another possible metric is comparing how well the LfD technique solves the problem to a hand-coded solution. However, typically, LfD researchers are not looking to advance the state-of-the-art solution, but are examining different aspects of the problem, such as human robot interaction issues, and only need an adequate solution rather than the optimal one.

Evaluation of LfD techniques suffer from another problem: typically robotic researchers conduct a minimal number of experiments and, as such, do not perform statistical analysis of their proposed techniques. The primary reason for this, is that conducting multiple, repeatable experimental runs with robotic hardware is difficult, time-consuming, and prone to hardware problems. Additionally, robotic researchers tend to focus on a single problem domain, thus, they do not test their technique across multiple problem classes.

2.6 Applications in Robotics and Multiagent Systems

As a multiagent training tool, HiTAB has potential applicability of a wide variety of problems. The following paragraphs illustrate widely differing application areas, and is not meant to be an exhaustive enumeration of possible application areas for HiTAB.

Multirobot Teams With the falling cost of robots, it is now feasible to field several (ones to thousands) of robots. One of the first large robot teams was the Millibots, a heterogeneous robot team which used group dynamics and robot specialization to solve tasks typically assigned to larger robots [99]. Another heterogeneous project, SWARMS, aimed to to link robots and swarm-intelligence inspired algorithms using heterogeneous robot teams for various military tasks [200]. Due to the cost and complexity, all other robot teams are homogenous. The Nerd Herd was an early robot team which composed simple behaviors into more complex group behaviors [201]. The open-source FlockBots are designed for embodied multiagent systems research [202] and the Swarm-Bot [203] project developed small, simple robots to study embodied swarm intelligence. Looking at larger

teams, the Centibots, used a team of 100 Amigabots for distributed mapping and surveillance [204], and the NASA Ants project aims to use thousands of tetrahedral-shaped robots for asteroid and planetary exploration [205].

Multiagent and Multirobot Contests With the development of inexpensive robot teams, researchers needed a way to compare different algorithms in a standardized manner. The RoboCup competition uses soccer to compare teams and encourage multirobot research [79]. In the past several years, RoboCup has expanded to include domestic robots and search and rescue. Seeking a more real-world application, the Multi Autonomous Ground-robotic International Challenge (MAGIC) competition required competitors to deploy a multi-robot team capable of autonomous exploration and patrolling of a dynamic urban environment [206]. The winning team used 14 autonomous robots. Increasing air traffic encouraged the Federal Aviation Administration to develop a new air traffic control system called NextGen which autonomously ensures adequate separation and 4D trajectory assurance for both manned and unmanned aircraft [207]. The NASA UAS Challenge seeks to implement and test different approaches for solving aircraft detection and collision avoidance in a noisy environment [208]. Post 9/11, surveillance has become a hot topic. The BORDERS challenge [209] and UAVForge [210] are two aerial surveillance contests where competitors must monitor an area and track suspicious targets. Entering the simulation arena, the Multiagent Programming Contest pits different multiagent system frameworks against each other to learn when different frameworks are apropos [211, 212]. The current project requires simulated agents to explore Mars while defending themselves against raids by rivals.

Target Tracking Cooperative target observation problems are interesting testbeds for studying multiagent coordination, control, and planning. These problems are good examples of dynamic multi-agent interaction and they have real world applications [213–218]. One practical application is air traffic control [207, 219–221]. Similar to CTO problems, patrol problems also have direct application to real-world scenarios while providing dynamic testbeds for multiagent coordination [222–224].

Object manipulation Object manipulation focuses on multiple robots cooperatively carrying, pushing or manipulating an object [225–228] . Common examples include foraging [1, 229, 230], collective sorting [231, 232], and harvesting [233]. Two dynamic examples are herding and predator/prey. In herding a group of agents tries to force other noncooperative agents to a specific location [234–236]. Predator/prey problems are very common multiagent systems environments, where a number of agents (predators) cooperatively chase prey agents [237].

Group Behaviors Group behavior problems are well-studied and include formation keeping [238–242] and path planning [243, 244]. Early work addressed the composition of basic behaviors to achieve complex behaviors [201, 245]. Probably the most well-known group behavior is flocking [246, 247], a common creature control algorithm in movies, e.g., the orcs in Lord of the Rings [248]. Turpin et al. extended 2D path planning to 3D using quadcopters [249]. The same group later used multiple quadcopters for simple construction tasks [250, 251]. Finally, several groups studied autonomous multirobot construction [252–254].

Traffic Control Here the task is to maximize the number of cars passing through a grid of intersections. Each intersection has a traffic light controlled by an agent, and the agents must coordinate to deal with traffic fluctuations [255]. Possible solutions include reservations [256] and tokens [257]. The Monitorix system uses video to dynamically adapt traffic management to local conditions [258].

Localization, Mapping and Exploration The simultaneous localization and mapping (SLAM) problem is a fundamental problem in mobile robotics, and is a major stepping stone towards fully autonomous robots [259]. The goal of SLAM is to construct an accurate map of the environment and the robot's path within that environment. While the single robot SLAM problem is well studied [260–270], the multi-robot SLAM has received comparably little attention. Most work has focused on multi-robot localization [259, 271–280] and exploration [281–284]. However, some work is starting to address multi-robot SLAM specific issues such as what information to exchange [285, 286] and when to exchange the information [287]. Forcing robots to rendezvous at certain meeting points simplifies the data association problem [287], as does the prescence of known, global landmarks

[288]. Still, multirobot SLAM research centers on a centralized approach [289] where the SLAM algorithm uses data from all the robots.

Human/Agent Interaction All the previous examples were autonomous; however, there are several domains where human/agent interaction is important. In search and rescue, a human works with a remote robot to detect and locate survivors in a disaster scenario [290, 291]. Video games are another domain requiring human/agent interaction. For example, in the Sims, humans must control many virtual, semi-autonomous characters [292]. Crowd simulations are used for urban simulations [293], search and rescue [294], and video games [295].

Chapter 3: Prior Work in Multiagent Systems

Multiagent systems are traditionally hard to develop due to the interactions between agents in an unpredictable, dynamic environment. Successfully programming a multiagent system would require knowing, a priori, all possible combinations of environment and agent states. Increasing the challenge of multiagent systems are emergent behaviors: behaviors that result from the unforeseen interactions between agents (i.e., the group's behavior is not just the superposition of the individual agent's behaviors). Hence, machine learning offers a potential solution to these issues. Still, applying machine learning to multiagent systems introduces additional challenges including the (possible) presence of multiple learners which introduces game-theoretic issues which are not fully understood.

This chapter presents my prior work on three multiagent learning systems which illustrate these challenges. Initially, I studied *tuneably decentralized algorithms* for multiagent control in a cooperative target observation problem in which we could control the level of centralization with a single parameter. This work illustrated the complexity involved in developing a distributed solution to a multiagent problem. The next learning multiagent system used *favors* as a cooperation mechanism between agents. By bounding the loss associated with helping other agents, we could induce cooperation in a variety of multiagent problems. The final multiagent learning system used the notion of *lenience* to encourage cooperation: agents should permit their teammates to make mistakes early in the learning process.

My work with these three multiagent learning systems (and my work with the FlockBots and RoboPatriots; see Appendix A) led me to two observations: first, all these algorithms required significant code, test, debug cycles, and, second, user implementation would require marked programming ability. The remainder of the thesis describes the learning from demonstration system, HiTAB, that resulted from investigating easier, broadly accessible methods to induce coordination and cooperation between agents.

3.1 Cooperative Target Observation

Cooperative target observation (CTO) problems are important problems in multiagent systems since they illustrate the need for dynamic multiagent interaction and they are good examples of emergent behavior. Additionally, there are many applications of CTO: unmanned vehicle control; tracking items in a warehouse or factory; tracking people in search and rescue; and keeping tissue in continuous view during medical procedures.

Typically, CTO solutions are centralized: all information is gathered at a central location where a single process computes the best solution and commands are sent to individual agents. In certain situations a centralized approach might not be the best option. Local sensors, limited communication, and availability of distributed computational resources make a decentralized approach appealing. In fully decentralized techniques, agents individually make decisions, while a partially decentralized approach groups agents into autonomous squads where each squad is controlled by a single process.

In a set of papers, we used CTO for comparing the performance of centralized, partially centralized, and fully decentralized algorithms under different levels of dynamism and sensor capabilities [216, 217, 296]. In our CTO problem, a group of mobile agents (called *observers*) collectively attempts to stay within an "observation range" of as many targets as possible. The observers have global information: they know the location of all other observers and targets. We formulated the problem this way to lift sensing constraints which bias the problem toward distributed solutions.

We developed two algorithms for controlling the observers based on K-means clustering and hill-climbing. We also considered the two in combination: K-means clustering followed by hill climbing. The three algorithms are tunably decentralized by adjusting a parameter which determines how many subsets the observers are divided into: all observers within a given subset participate in a centralized decision-making process. Thus, one subset is equivalent to a centralized system while subsets containing only one agent correspond to a fully decentralized system. For this work, we created squads independent of location and kept squad membership static.

In our CTO formulation, the observers move faster than the targets, and the environment is a nontorodial continuous 2D field free of obstacles. The model has N observers and M targets with $N < M$. Both observers and targets can move in any direction, and the observers have an *observation*

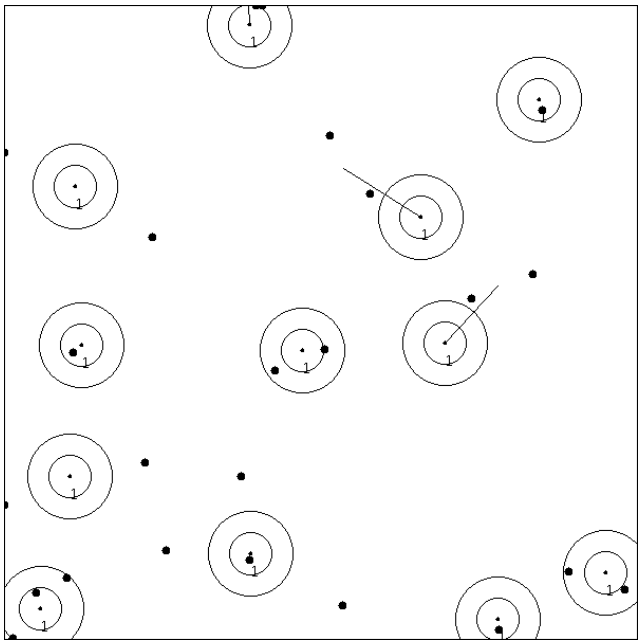


Figure 3.1: Screenshot of the model. Small doubly-circled dots are observers. Outer circles are their observation ranges. Large dots are targets. Straight lines connect observers with newly-chosen desired destinations. The number 1 on each observer is the subset the observer belongs to. In this case, all observers are in a single subset.

range R: any target within a circle centered on the observer of radius *R* is considered observed. Figure 3.1 shows a snapshot of the model in action.

The targets move randomly throughout the space and do not try to avoid the observers. Movement of both the targets and observers is done by setting a destination point and then having the agent move towards this point. The targets travel toward their destination for 100 time steps: if they reach the destination before 100 time steps, a new destination is immediately computed. If the targets do not reach their destination within 100 time steps, a new destination is computed. To help prevent targets from clustering near the center of the environment, targets' destination points are randomly chosen from a rectangle (one quarter of the environment height and width) centered on the target.

The observers compute a new destination point every α time steps. If an observer reaches its destination prior to α time steps, then it waits until a new destination point is computed. The destination point is computed using of of three cooperative target observation algorithms: hill

climbing with subsets, K-means clustering with subsets, and K-means clustering with subsets followed by hill climbing with subsets.

Tunable decentralization of the algorithms works follows. The N observers are divided into C disjoint subsets, and a separate instance of the current algorithm is run for each subset. Each algorithm instance only adjusts the destination points for the observers within its subset, and assumes all other observers and targets are fixed in their current positions. Subset size C/N ranges from 1 to N: a subset size of N corresponds to a centralized system, while a smaller subset size yields higher degrees of algorithm decentralization with a subset size of 1 being a fully decentralized system.

Hill-climbing with Subsets Hill-climbing iteratively improves candidate observer destinations by first copying the current locations into the initial candidate destinations, then performing 1000 iterations of the following operation:

1. Copy the current candidate observer positions and mutate the copy by randomizing the position of one observer member of the subset. The observer position is picked at random from the intersection of the field and a box centered at the observer with an initial width and height equal to one-half the environment width and height. The box decreases in width and height by 1% each iteration, but not below one tenth of the environment width and height.

2. If the mutated child is "better" than its parent, replace the parent with the child, else discard the child.

3. Goto 1

A candidates' quality is assessed by testing how it affects the *entire observer team*, not just the subset. We use a lexicographic ordering of various useful quality measures. Specifically, we accept the child and replace the parent with it using the following test:

1. Accept the child if it observes more targets.

2. Else if the child and parent observe the same number of targets, compute

$$H = \sum_{o \in O} \min_{t \in T} f(o,t)$$

where

$$f(o,t) = \begin{cases} \min(R, \text{DISTANCE}(o,t)) & \text{if } R/2 \leq \text{DISTANCE}(o,t) \\ 0 & \text{if } R/2 > \text{DISTANCE}(o,t) \end{cases}$$

where $\text{DISTANCE}(o,t)$ is the Euclidean distance between the observer $o \in O$ and target $t \in T$. Accept the child solution if $H_{child} < H_{parent}$. This encourages the observers to center themselves among the targets they are observing and to not share targets.

3. Else if $H_{child} = H_{parent}$, and the child and parent observe the same number of targets, then compute

$$G = \sum_{o \in O'} \min_{t \in T'} \text{DISTANCE}(o,t)$$

where O' is the set of observers that observe no targets, and T' is the set of targets not observed by any observers. Accept the child solution if $G_{child} < G_{parent}$. This encourages the observers to move towards unobserved targets.

4. Else reject the child.

After the hill-climber has iterated 1000 times, we determine which observer in the subset will go to which of the new candidate destinations. For this we used a simple greedy coloring algorithm.

1. Pick the candidate destination and the observer in the subset that are closest to one another.
2. Assign the observer to that destination.
3. Remove the observer and the destination from consideration.
4. Goto 1.

K-means Clustering with Subsets K-means is a widely used clustering algorithm which assumes that the number of clusters is known *a priori*. Suppose that N points need to be grouped into K clusters, with the requirement that the mean distance from the points to the centers of the clusters is minimized. The algorithm works as follows

1. Select K initial centers for the clusters.

2. Assign each of the N points according to which of the cluster centers is closest to the point.

3. Reposition each cluster center slightly closer to the mean location of all the points assigned to that center. More specifically, let K_i be the i^{th} cluster center, let m_i be the mean of the all points assigned to the i^{th} center, and let α be the step size. Then each center is repositioned by

$$K_i \leftarrow (1-\alpha)K_i + \alpha m_i$$

4. Go to 2.

This process is repeated until no more significant progress is made, or until time is exhausted.

In order to apply the K-means algorithm to our cooperative target observation problem domain, we consider the targets to be the points to be clustered, and observer destinations as the candidate cluster centers. We set the initial positions of the cluster centers to the current positions of the observers, and assign each to the observer at that position. We modify the clustering algorithm so that the only candidate centers which are allowed to move (in step 3) are those assigned to observers which are members of the current subset.

3.1.1 CTO Experiments

Our experiments examined how different levels of decentralization were affected by various parameters. We expected the performance of both hill climbing and K-means clustering to degrade as the targets got fasters, as the rate of algorithmic update slowed, and as the sensing radii decreased. Our results consistently bore this out, however, the algorithms degraded differently, which was unexpected since both algorithms are tunably decentralized in the same manner.

All experiments were conducted using the MASON simulation environment [297]. Our performance metric was the mean over all time steps of the number of targets under observation. If a target was observed by multiple observers, it was counted only once. Statistical significance was established using Welch's two-sample tests with a Bonferroni correction to compensate for the large number of tests performed. As a consequence, each two sample test was performed at a confidence level of 99.995%.

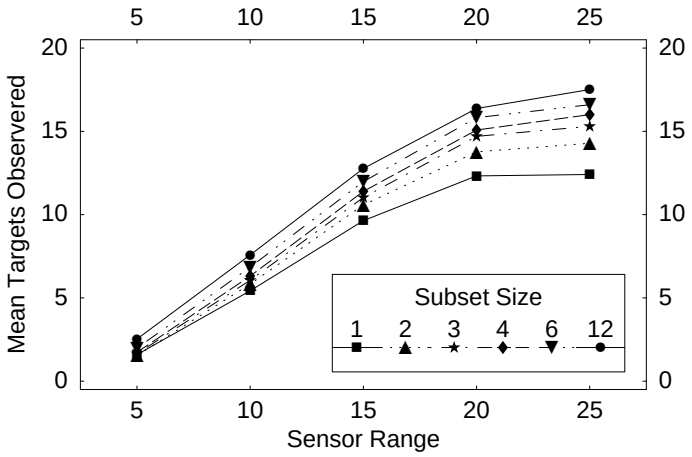


Figure 3.2: Performance of the hill-climbing algorithm using different subset sizes when varying the sensor range.

Our initial experiments compared K-means clustering and hill climbing against random and stationary behaviors. We found that K-means and hill climbing are statistically better than the random and stationary behaviors across all combinations of target speed, level of decentralization, sensor range, and rate of updates. Both hill climbing and K-means perform best at large ranges, small target speeds, and small update rates. Curiously, hill climbing was sensitive to subset size while K-means was not.

Hill-Climbing Figures 3.2, 3.3, and 3.4 show the performance of the hill-climbing algorithm as the range, target speed, and update rates vary, respectively. If not being varied, the sensor range is set to 15, target speed to 0.5, and the update rate to 10. The graphs verify that performance decreases when either range decreases, target speed increases, or update rate decreases. More importantly, the graphs show that increased the degree of decentralization leads to a decrease in performance. Notably though, the difference due to decentralization is almost invariant over any change in environment parameter.

K-Means The same experiments with K-means produced very different results. Like hill-climbing, K-means performed best at large ranges, small target speeds, and small update rates. However,

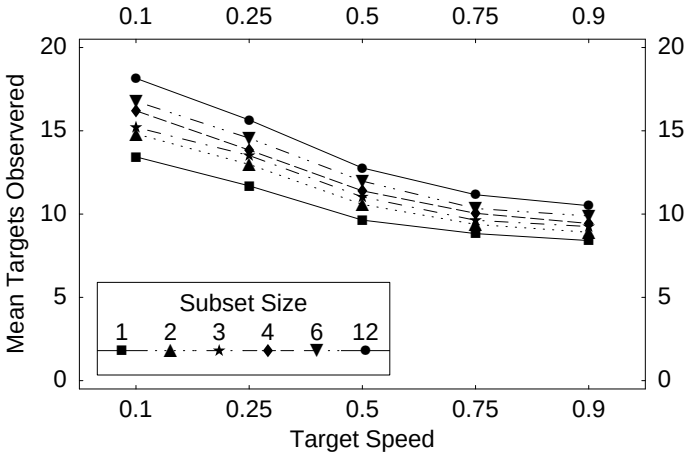


Figure 3.3: Performance of the hill-climbing algorithm using different subset sizes when varying the target speed.

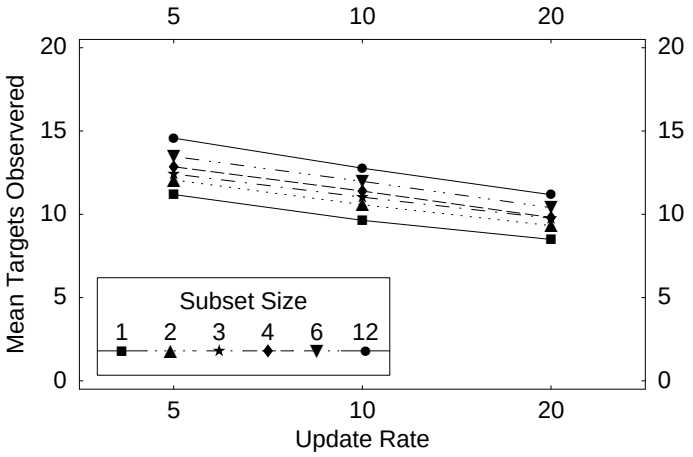


Figure 3.4: Performance of the hill-climbing algorithm using different subset sizes when varying the update rate.

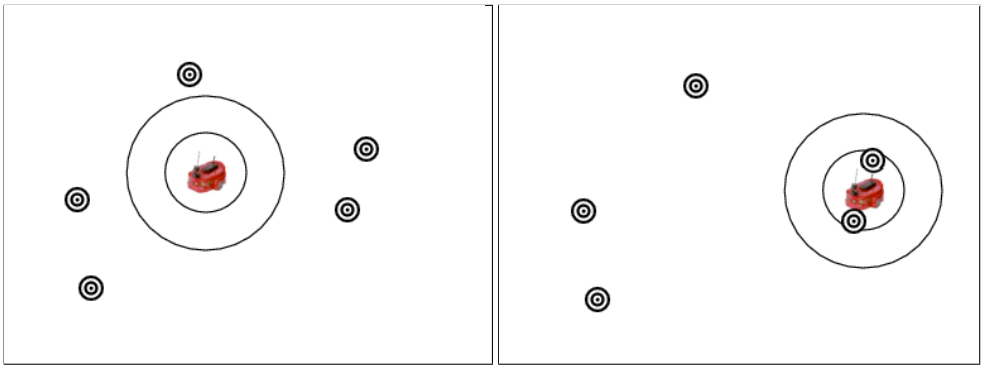


Figure 3.5: K-means clustering disadvantages. K-means clustering (left) centers in the region of its cluster, whereas hill-climbing (right) attempts to maximize coverage inside the sensor range.

K-means showed *almost identical performance* regardless of subset size. Decentralization appears to have had no effect on the results at all; thus, no figures are presented.

Comparison Given that hill climbing and K-means performed differently over different subset sizes, we directly compared the two algorithms across all environment parameters. We chose to use a subset of size 12 as it yielded the best results for hill-climbing. Table 3.1 shows that hill climbing performs better with slower-moving targets while K-means performs better with faster moving targets.

This seems intuitive: with slow moving targets or a small sensor range, hill climbing discovers better solutions since K-means centers observers in the middle of targets without considering how far apart the targets are (see Figure 3.5). With faster targets, hill climbing cannot reach its optimal destination in time, and essentially chases the targets. However, the targets tend to form clusters since they are moving towards each other (and hence toward the cluster mean). By moving observers towards the cluster mean, K-means positions observers in the path of fast moving targets, giving it an advantage over hill climbing in faster environments.

In Combination The final experiments compared the two algorithms in combination: K-means followed by hill climbing compared to K-means alone, and also against hill climbing alone. The combination of the two does quite well as a middle-ground. In Table 3.2, the combination is compared

Table 3.1: K-means versus hill-climbing, subset size of 12. $\gg$ means that K-means is statistically better, $\ll$ means that hill-climbing is statistically better and $\simeq$ indicates no statistically significant difference.

Update Rate	Sensor Range	Target Speed				
		0.1	0.25	0.5	0.75	0.9
5	5	$\ll$	$\ll$	$\gg$	$\gg$	$\gg$
	10	$\ll$	$\ll$	$\gg$	$\gg$	$\gg$
	15	$\ll$	$\ll$	$\gg$	$\gg$	$\gg$
	20	$\ll$	$\ll$	$\simeq$	$\gg$	$\gg$
	25	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$
10	5	$\ll$	$\gg$	$\gg$	$\gg$	$\gg$
	10	$\ll$	$\simeq$	$\gg$	$\gg$	$\gg$
	15	$\ll$	$\simeq$	$\gg$	$\gg$	$\gg$
	20	$\ll$	$\simeq$	$\gg$	$\gg$	$\gg$
	25	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$
20	5	$\simeq$	$\gg$	$\gg$	$\gg$	$\gg$
	10	$\simeq$	$\gg$	$\gg$	$\gg$	$\gg$
	15	$\simeq$	$\gg$	$\gg$	$\gg$	$\gg$
	20	$\simeq$	$\gg$	$\gg$	$\gg$	$\gg$
	25	$\ll$	$\ll$	$\simeq$	$\simeq$	$\simeq$

Table 3.2: Hill-climbing versus hill-climbing and K-means in combination, subset size of 12. $\gg$ means that hill-climbing is statistically better, $\ll$ means that the combination is statistically better, and $\simeq$ indicates no statistically significant difference.

Update Rate	Sensor Range	Target Speed				
		0.1	0.25	0.5	0.75	0.9
5	5	$\gg$	$\simeq$	$\ll$	$\ll$	$\ll$
	10	$\simeq$	$\simeq$	$\ll$	$\ll$	$\ll$
	15	$\simeq$	$\ll$	$\ll$	$\ll$	$\ll$
	20	$\simeq$	$\simeq$	$\ll$	$\ll$	$\ll$
	25	$\gg$	$\gg$	$\simeq$	$\simeq$	$\simeq$
10	5	$\gg$	$\ll$	$\ll$	$\ll$	$\ll$
	10	$\simeq$	$\ll$	$\ll$	$\ll$	$\ll$
	15	$\simeq$	$\ll$	$\ll$	$\ll$	$\ll$
	20	$\simeq$	$\ll$	$\ll$	$\ll$	$\ll$
	25	$\simeq$	$\simeq$	$\simeq$	$\simeq$	$\simeq$
20	5	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$
	10	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$
	15	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$
	20	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$
	25	$\simeq$	$\simeq$	$\simeq$	$\simeq$	$\simeq$

Table 3.3: K-means versus hill-climbing and K-means in combination, subset size of 12. $\gg$ means that K-means is statistically better, $\ll$ means that the combination is statistically better, and $\simeq$ indicates no statistically significant difference.

Update Rate	Sensor Range	Target Speed				
		0.1	0.25	0.5	0.75	0.9
5	5	$\ll$	$\ll$	$\simeq$	$\gg$	$\gg$
	10	$\ll$	$\ll$	$\ll$	$\simeq$	$\gg$
	15	$\ll$	$\ll$	$\ll$	$\simeq$	$\gg$
	20	$\ll$	$\ll$	$\ll$	$\ll$	$\simeq$
	25	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$
10	5	$\ll$	$\ll$	$\simeq$	$\gg$	$\gg$
	10	$\ll$	$\ll$	$\simeq$	$\simeq$	$\gg$
	15	$\ll$	$\ll$	$\simeq$	$\simeq$	$\gg$
	20	$\ll$	$\ll$	$\ll$	$\simeq$	$\simeq$
	25	$\ll$	$\ll$	$\ll$	$\ll$	$\ll$
20	5	$\ll$	$\ll$	$\simeq$	$\gg$	$\gg$
	10	$\ll$	$\simeq$	$\simeq$	$\gg$	$\gg$
	15	$\ll$	$\simeq$	$\simeq$	$\gg$	$\gg$
	20	$\ll$	$\ll$	$\simeq$	$\gg$	$\gg$
	25	$\ll$	$\ll$	$\simeq$	$\simeq$	$\simeq$

against hill-climbing: here it performs better than or equal to hill-climbing almost everywhere, except for very small radii or very low target speeds. Table 3.3 shows that it also performs well against K-means, though K-means is still superior at very high target speeds. Like K-means, the combination showed almost identical performance across all subset sizes.

3.2 Favors

In multiagent problems, the agent's typically have limited resources that restrict the problems they can solve. If the agent's can coordinate, then the system is more effective, particularly if the agents coordinate both locally and globally. One approach to facilitate coordination among agents is having the agents provide *favors* to other agents. Most work on favors follows some version of *reciprocal altruism*: an agent is willing to incur a cost *now* to provide value to another agent with the expectation of receiving value in the future. While most work in reciprocal altruism focuses on rational agents,

trust, social laws, and reputation, we examined the issue from the perspective of *amortized risk*: the risk an agent is willing to incur is restricted such that if the second agent never reciprocates again the first agent's loss, amortized over all previous interactions with the second agent, is bounded.

Why is this important? In large multiagent systems, most agents are acting towards the common good and have reasonable expectations of similar behavior from fellow agents. However, initially the agents have no information about the nature of the other agents, which could be dishonest, but could also have poor perception of their own abilities; or they may be swamped with work; or there could be noise in the system. Large scale real-world applications (e.g., swarm foraging and observation, distribution of tasks across heterogeneous computers) all require an optimistic view one's neighbors.

Bounded amortized risk yields a simple decision rule, which we called *flexible reciprocal altruism* [298]. In flexible reciprocal altruism, an agent grants favors by considering the degree to which it has had favorable interactions with the grantee agent in the past. If the grantee agent has provided many significant favors in the past, then grantor is willing to offer the grantee agent more unmatched favors in the future. Consider that agent i is considering granting agent j a favor. Let v_{ij} be the sum total value to agent i of the favors agent j has granted until now. Let c_{ij} be the sum total cost to agent i of the favors which agent i has granted agent j until now. Then, agent i will grant the proposed favor if:

$$c_{ij} - v_{ji} + \text{cost}(favor) \leq \alpha v_{ji} + \beta.$$

The idea is that the degree which agent i will go out on a limb for agent j is no more than some constant α time how well agent j has treated agent i in the past. α is essentially a measure of risk tolerance: more altruistic agents will have higher values of α while risk-averse agents will have lower values of α. An initialization constant β determines the initial level of altruism and allows agents to grant favors to grantees with which they have no history (i.e., $v_{ij} = 0$). Thus, our decision equation becomes:

$$\beta + (1+\alpha)v_{ji} \geq c_{ij} + \text{cost}(favor). \tag{3.1}$$

Our decision function is based on amortized altruism rather than rationality: a favor will be granted provided it does not increase c_{ij} to greater than $\beta + (1+\alpha)v_{ij}$. This allows one agent to cheat and

walk away provided the amortized cost to benefit ratio is bounded. Clearly, $c_{ij} = O(v_{ij})$, so the augmented cost to benefit ratio is:

$$\frac{c_{ij} + \text{cost}(favor) - \beta}{v_{ji}} \leq 1 + \alpha. \tag{3.2}$$

As the agents build a transaction history and c_{ij} and v_{ij} grow, β becomes inconsequential and the cost to benefit ratio approaches $1 + \alpha$.

Free Riders Our approach is susceptible to free riders ("leeches") in two ways. First, when agent A meets agent B for the first time, A will always agree to grant a small initial favor (less than β) to B. Were there an infinite number of agents, B could wander about forever getting one favor from each agent without have to grant a favor. Second, an agent B could build a transaction history with agent A, then ask for a large number of favors before walking away.

In a finite group, free-riders of the first kind are capped: they run out of other agents to cheat. While free-riders of the second kind have the potential for unlimited cheating of the system, the ratio of loss with respect to their provided value is bounded. Importantly, this ratio is set by the altruistic agents: by changing the value of α, the altruistic agents can either bound their loss to at most β, or lower α even more to grant only a single favor before ignoring future unmatched favors.

Favor Brokers While flexible reciprocal altruism allows agents to build a transactions history relatively quickly, it also allows extensions which allow even faster ways to build transaction history. In particular, we studied *favor brokers* which act as a bridge between two agents with little transaction history. Consider two agents A and C who have a limited transaction history. Agent A needs a significant favor, but C will not grant it due to the limited history between the two agents. However, favor broker B has a significant history with both agents A and C, so B is willing to broker A's favor by asking C to perform the favor. C grants the favor and incurs the cost of the favor, while A receives the value of the favor. Importantly, A believes B granted the favor and C believes that B asked for the favor, so agent A and C do not change their transaction history with each other. Favor brokers offer a way for populations to offer favors despite little prior history, or for two alien populations to offer favors to one another.

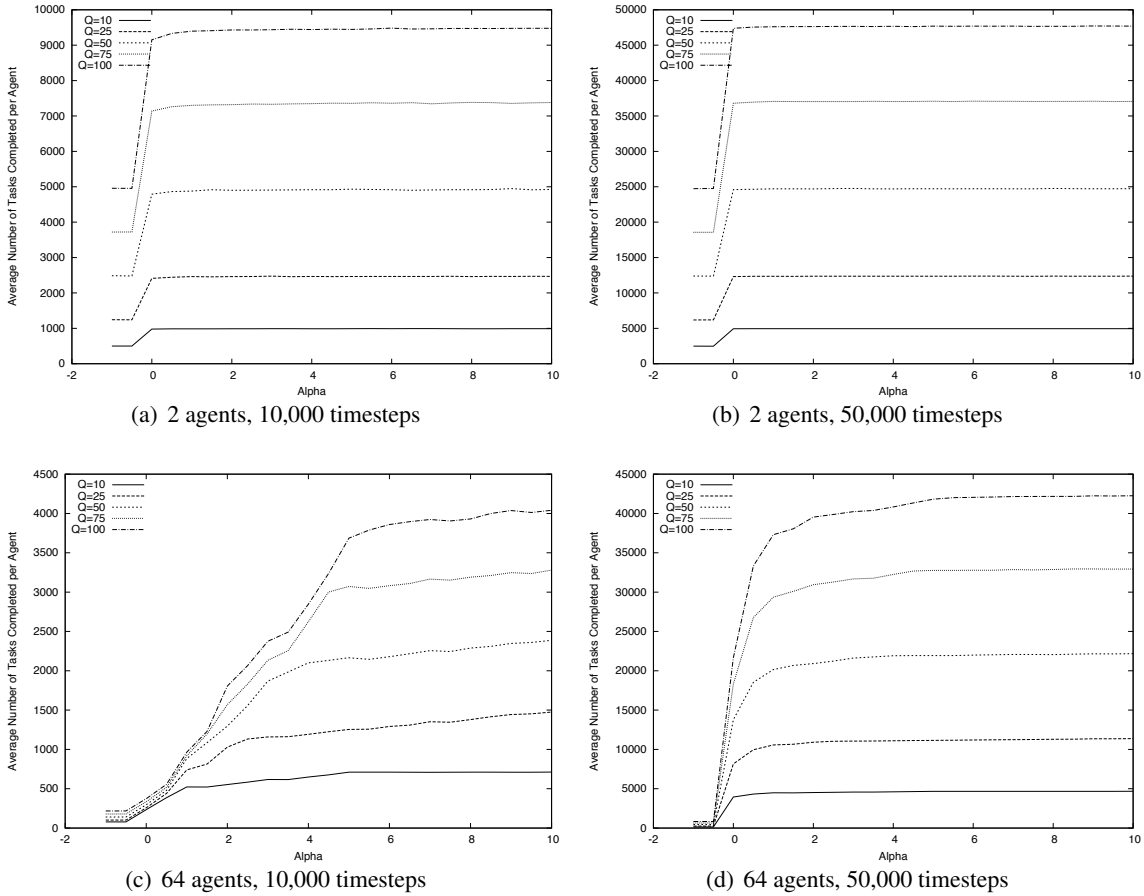
3.2.1 Favor Experiments

Our experiments focused on an abstraction of common factory-floor models. Each agent has a queue of certain length (how many tasks it can hold) and each time step removes a task from the queue and performs it. New tasks arrive stochastically, and if the agent's queue is full when tasks arrive, the agent asks other agents for a favor: to put the task in their queues instead. If an agent cannot curry favor, the task is lost. Costs are incurred when an agent places a task into its queue, and the task's owner receives the value. Granting a favor costs the grantor two ways: first, the grantor incurs the cost of the task, and, second, the grantor runs the risk of filling its own queue with tasks belonging to someone else.

Tasks arrived following a Poisson distribution with a mean of 100 timesteps, and normally round robin selection determined which agent received the task. When asking for a favor, agent A asks random agent B for a favor, and if B does not grant the favor, then A chooses another random agent to ask, then another, and so on. If no agent grants A a favor, then the task is lost. All experiments were done in the MASON simulator [297] and the results were averaged over 50 independent runs. The task value was set to 3 and task cost to 2.

Experiment 1: Task Load, Number of Agents, and Degree of Altruism The first experiments sought to examine the dynamics of the system: what happens under increasing load (shorter queues), more agents, and varying degrees of altruism (α). Figure 3.6 shows the results. The left column shows the results for 10000 timesteps, which emphasizes the warm-up period until the agents offer large favors to each other, and the right column shows 50000 timesteps, which is closer to the steady state of the system. As is obvious, shorter queue lengths results in fewer tasks being completed since more tasks are dropped per agent.

Figure 3.6 also shows that more altruistic agents (larger values of α) warm up to one another faster. Agents with $\alpha = -1$ (greedy agents who only provide just one favor for the entire duration of the run) cannot improve. Agents with $\alpha \geq 0$ are able to warm up to each other since their decisions are based on the ratio of value to cost rather than the specific number of favors granted. The rate of warm up increases as α increases.



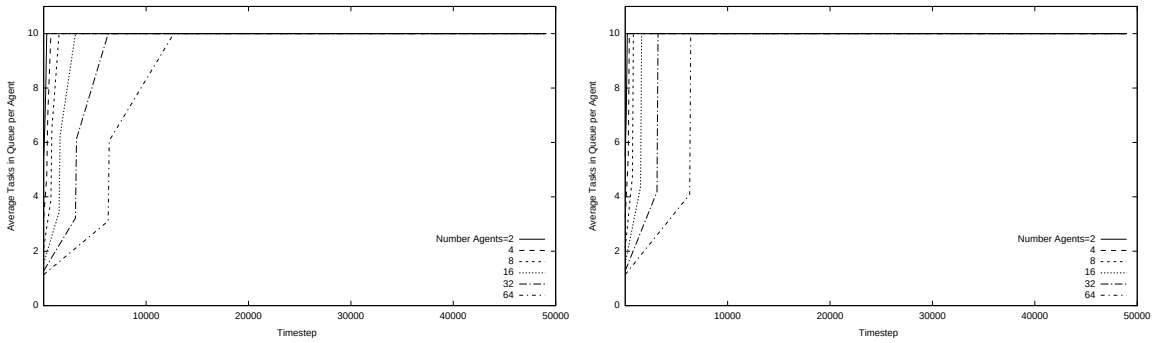
(a) 2 agents, 10,000 timesteps

(b) 2 agents, 50,000 timesteps

(c) 64 agents, 10,000 timesteps

(d) 64 agents, 50,000 timesteps

Figure 3.6: Average number of completed tasks for 2 and 64 agents and various task queue lengths (Q).



(a) $\alpha = 0.0$, 50,000 timesteps (b) $\alpha = 1.0$, 50,000 timesteps

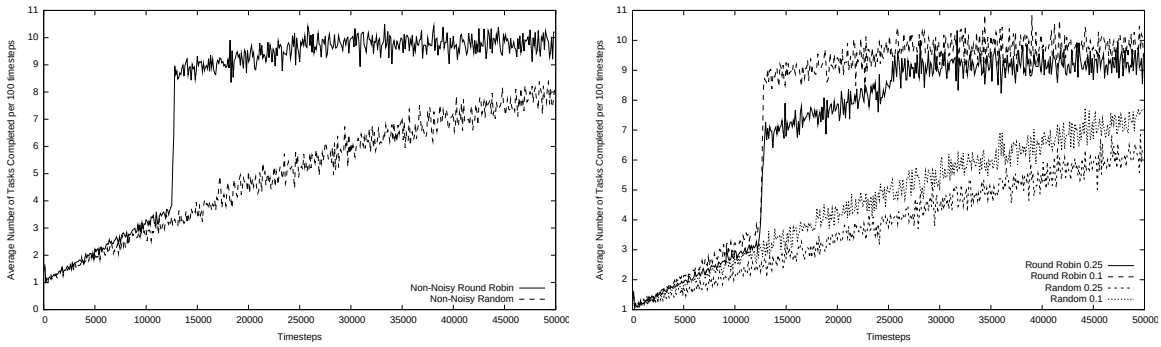
Figure 3.7: Average queue length per agent. Maximum queue length was 10.

Last, but most importantly, intuition suggests that increasing the number of agents would allow for more favor opportunities, and this should, in turn, increase the average number of tasks completed. But, as the number of agents increases, an individual agent takes longer to warm up to any given grantor. As a result the total number of tasks is reduced until the agents have adequately warmed up to each other. Ultimately, the steady state converges to the same values regardless of the number of agents.

Experiment 2: Queue Saturation To verify that agents look longer to warm up to each other as the number of agents increases, the next set of experiments examined how full agents' queues were during the task. Figure 3.7 shows the average number of tasks waiting in a queue, per agent. The maximum queue length was 10. Again, we see that as the number of agents increases, the agents are idle longer (shorter queue lengths), but, in the steady state, all queues are filled.

Experiment 3: Task Allocation and Noise Next, we examined how incoming tasks are assigned: the first method used simple round robin selection, and the second method was random selection. In this set of experiments, we also added noise to the system by increasing the probability that the grantor does not receive any value (e.g., the grantee will think they granted a favor while the grantor will think the favor was not granted).

We set the number of agents to 125, $\alpha = 1.0$, $\beta = 2$, and the queue length to 10. For the noisy experiments, we set the noise level at 0.1 and 0.25 (a noise level of 0.1 meant that 10% of the time,



(a) No Noise (b) Different Noise Levels

Figure 3.8: Average number of tasks completed per agent per 100 timesteps for round-robin and random task distribution, $\alpha = 1.0, \beta = 2.0$, queue length = 10, 125 agents

the grantor would not receive any value). Figure 3.8(a) shows that changing how incoming tasks are assigned alters performance. In the round robin case, after all agents have received the tasks once (around time step 12500) all agents will then grant one another a favor; hence the spike in the graph. When tasks are assigned randomly, it takes longer to establish a relationship between all the agents, thus, resulting in poorer performance. Figure 3.8(b) shows how varying the level of noise effects system performance. As expected, as noise increased, system performance decreased; however, our model formulation overcomes the noise, but takes longer to do so.

Experiment 4: Initially Altruistic Societies This experiment studied one possible method to decrease the warm up period. In flexible reciprocal altruism, β controls how *initially* altruistic the society is: as β increases, the agents are willing to grant more initial favors before expecting the other agent to reciprocate.

Figure 3.9 shows how the warm up period changes with β, given $\alpha = 1$, maximum queue length of 10, and 125 agents. The figure is truncated to 20000 timesteps to emphasize the warm up period; after 20000 timesteps the curves are statistically the same. Note that for a highly altruistic society (high β) the average throughput is maximal from the beginning.

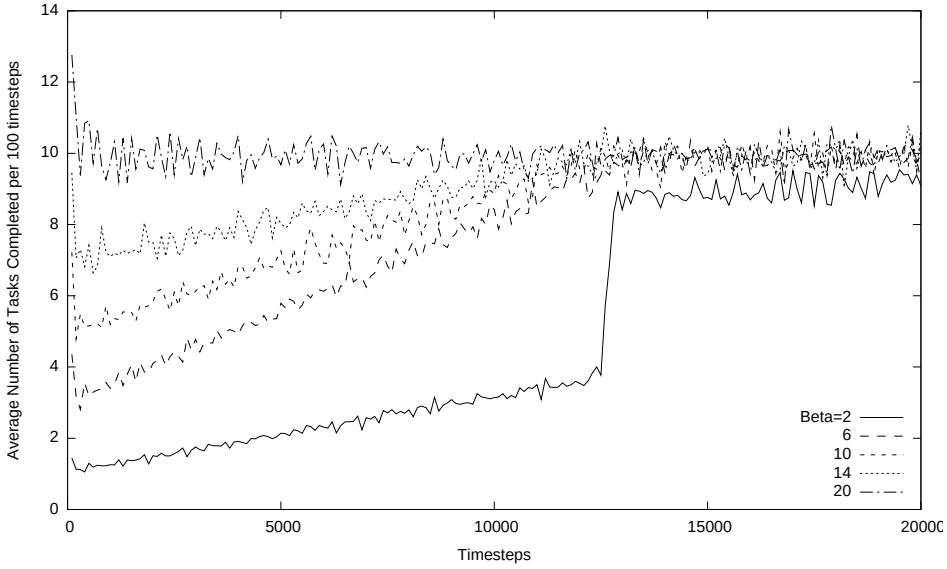


Figure 3.9: Average number of tasks completed for different levels of initial altruism. $\alpha = 1.0$, queue length = 10, and 125 agents.

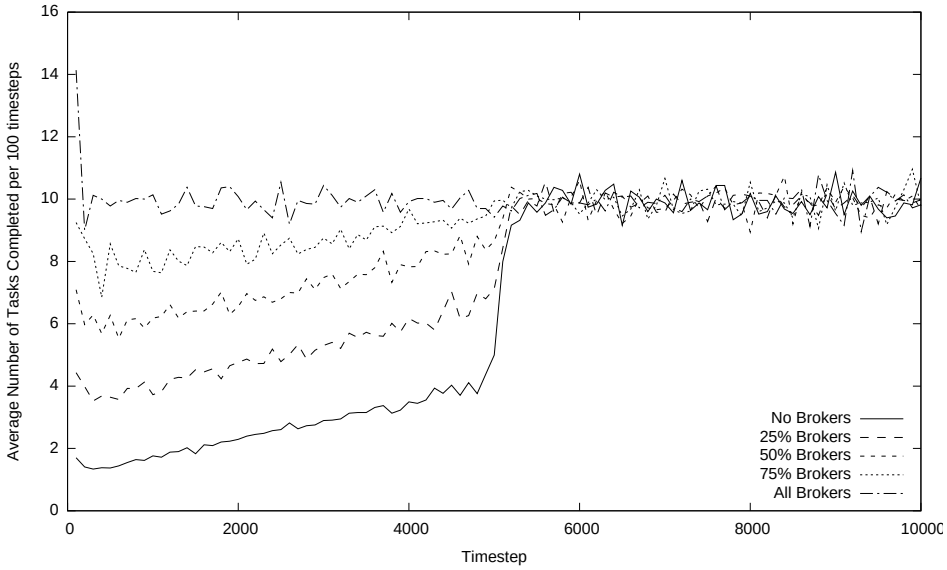


Figure 3.10: Average number of tasks completed per 100 timesteps for varying percentage of brokers in the population. $\alpha = 1, \beta = 2$, queue length = 10, and 50 agents.

Experiment 5: Favor Brokers The final experiment studied how favor brokers influence performance. In societies with low altruism, favor brokers offer a potential way to increase interactions between agents, and thus decrease the warm up period. Figure 3.10 shows the average number of tasks completed per 100 timesteps per agent for $\alpha = 0.1$, $\beta = 2.0$, maximum queue length of 10, and 50 agents. Increasing the percentage of brokers in the population increases the number of interactions between agents, thus causing quicker convergence even in a society that is not very altruistic. Changing α and the number of agents does not significantly alter the results.

3.3 Lenience

Favors work well when two agents want to coordinated through direct interaction. However, there are situations when agents need to coordinate without direct communication, and (possibly) by running different learning algorithms. In concurrent multiagent learning, each agent can only view part of the search space, cannot control its teammate's actions, and the team only receives feedback through a joint reward. Typical multiagent learning approaches to concurrent learning directly applied machine learning assuming that each agent behaves rationally and seeks to improve its own behavior. Recent work debates the use of rationality and Nash equilibria [299, 300], and suggest concepts such as reputation and mutual trust to improve learning [301,302]. We continued the trend of Nash equalibria, reputation, and mutual trust by introducing the notion of *lenience*: each agent is lenient with its teammates in the early stages of learning when each agent is exploring is action space [303, 304]. Lenience may be reduced as learning progresses and agents start converging on a solution.

We applied the notion of lenience to both evolutionary computation and reinforcement learning, and the combination of both where agents are using entirely different learning algorithms rather than simply a variation of the same learning method. Additionally, we showed that, for certain problem domains, all agents must be "smart" enough to obtain good results [305].

3.3.1 Single Agent Perspective on Search Space

Consider a simple scenario where two agents learn to cooperate. If X is the set of actions for the first agent, and Y is the set of actions for the second agent, then the task is for each agent to learn the

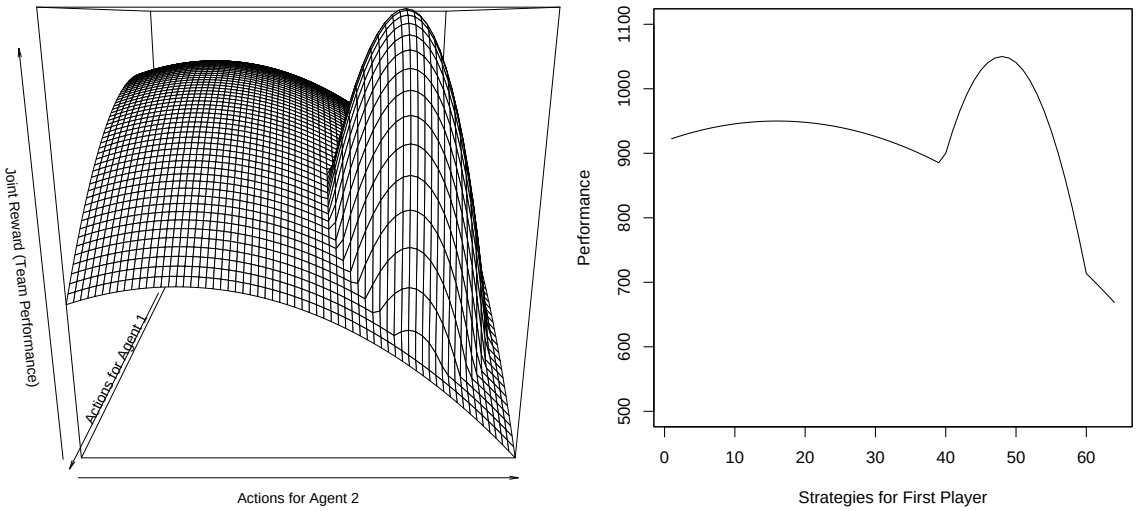


Figure 3.11: (a) A bimodal search space for the possible rewards received by a two-agent team. Wider peaks may attract many search trajectories, even though such peaks may be globally suboptimal. (b) A desirable projection of solution quality for the first population provides enough information about the location of the optimal peak.

appropriate action (say x and y) to maximize the joint reward $f(x,y)$. Figure 3.11 shows an example joint reward function for a simple coordination game.

The two peaks – the broad locally optimal peak with wide coverage and the narrow globally optimal narrow peak – in the figure both represent Nash equalibria: once an agent discovers a peak, there is no rational incentive to change actions. In concurrent learning, each agent only sees part of the search space. Specifically, each agent only detects the difference in reward it received for past actions. For the joint reward space of Figure 3.11(a), an ideal search space is illustrated in Figure 3.11(b): for each action x we plotted $g(x) = \max_{y \in Y} f(x,y)$.

A more realistic projection of the search space is shown in Figure 3.12. Here, the projection at point x is the average and the maximum of the joint reward when the first agent chooses action x and the second agent chooses 5 or 20 random actions. Taking the maximum results in a better projections, especially for larger number of possible rewards. However, agents typically do not select their actions uniformly: initially uniform selection is apropos since their is no a priori information about the space. As learning progresses, the agent tends to choose actions which resulted in a higher

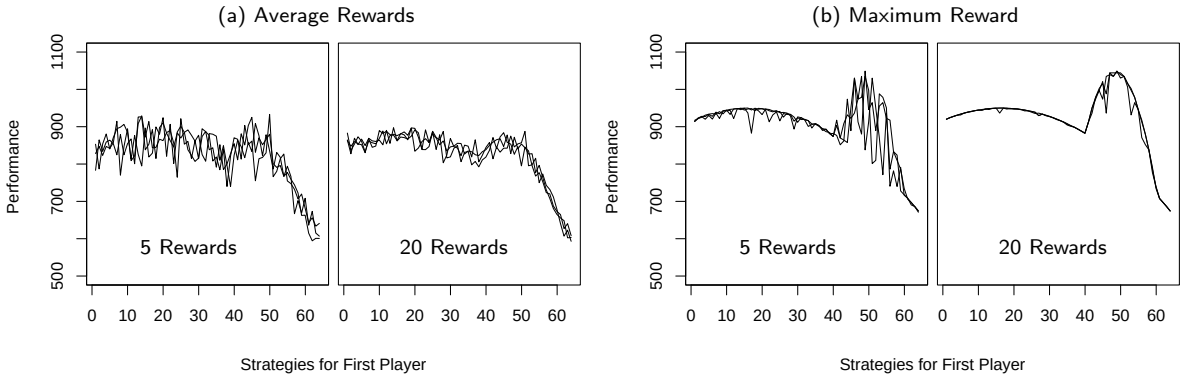


Figure 3.12: Projected search space for the first agent in the two-peaks domain, assuming the second agent chooses his action randomly with uniform probability. Due to noise, the process is repeated three times — there are three curves on each graph. (a) The projection at point x is computed by averaging 5 and respectively 20 joint rewards $f(x, y_i)$ for different actions y_i. (b) The projection at point x is computed as the maximum of 5 and respectively 20 joint rewards (the lower 4 and respectively 19 rewards are ignored).

reward in the past. Figure 3.13 shows the projected search space when the second agent chooses actions according to a normal distribution.

Figures 3.12 and 3.13 show taking the maximum reward provides the agents information about the search space which suggests that agents should be lenient with their teammates during early stages of learning: ignore many of the low joint rewards and only consider the higher rewards. Additionally, the figures suggest the quality of the projections decreases with more rewards as the learners start to converge.

3.3.2 Lenient Cooperative Coevolution

Cooperative coevolution is a variant of evolutionary computation which apply multiple parallel learners to work jointly on different parts of the problem [306], which make them a good fit for multiagent learning where the joint problem is decomposed into several individual agent problems. A standard approach to applying cooperative coevolutionary algorithms (CCEAs) to multiagent learning assigns each agent its own population of actions. CCEAs determine the fitness of an individual agent by testing it within the context of a complete team (called *collaborators*), where teammates are sampled from the other populations (either randomly or based on past performances). The received

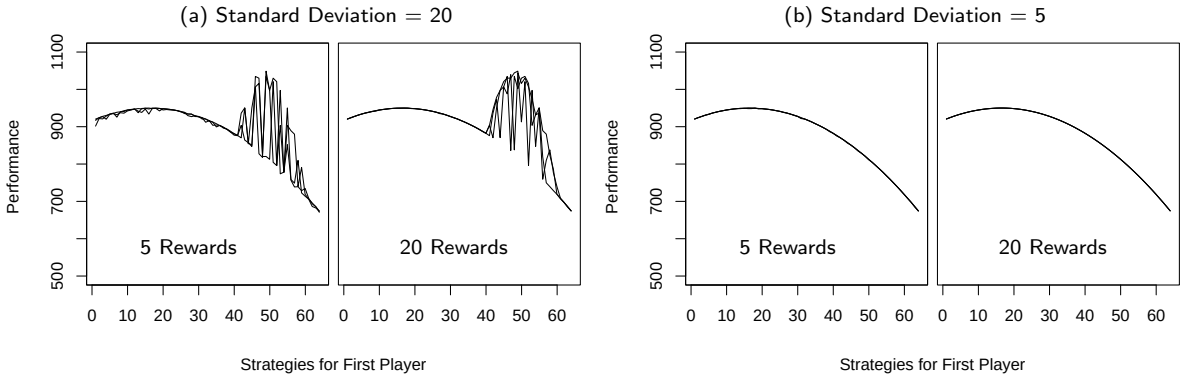


Figure 3.13: Projected search space for the first agent in the two-peaks domain, assuming the second agent chooses his actions according to a normal distribution centered on the suboptimal peak. The projection at point x is computed as the maximum of 5 and respectively 20 joint rewards $f(x, y_i)$ (the lower 4 and respectively 19 rewards are ignored). The y_i are randomly generated from normal distributions with mean 16 (centered on the suboptimal peak) and standard deviations 20 and 5, respectively. The process is repeated three times.

reward is the joint reward for the entire team, and the fitness for an individual agent is computed by aggregating multiple such rewards (e.g., maximum, average). The only interaction between agents is this collaborative evaluation; otherwise, the agent follows parallel, independent evolutionary processes.

For example, consider we are learning the joint reward for a team of three agents. If the agents choose actions x, y, and z, the joint reward is $f(x,y,z)$. A possible way to apply CCEAs to this problem is to assign one population of the actions of x, one population to the actions of y, and one population to the actions of z. Each population is evolved separately with the only interaction occurring during computation of the joint reward. An action's fitness is computed as the aggregate of joint evaluations with various collaborators: for examples, the fitness of x could be computed as $\max(f_1(x,y_1,z_1), f_2(x,y_2,z_2))$.

Recent research studies [307, 308] suggest that computing the fitness as the maximum of multiple joint actions performs better than computing the fitness as the minimum or the average. Also, while increasing the number of collaborators improves performance in complex domains involving non-linear interactions, it also increases the computational complexity. Taking the maximum fitness value implies that all but one joint reward is discarded and corresponds to a constant level of lenience

for each agent towards its teammates. Based on the above discussion, we argue agents should be more lenient early in the evolutionary process. In CCEAs, this translates as an agent having more collaborators early in the evolutionary process and fewer collaborators as the evolutionary process progresses. While there are many possible approaches to decreasing the number of collaborators, we take a simple approach: start with 10 collaborators for the first 5 evaluations, followed by using only 2 collaborators until exhausting computational resources.

Lenient CCEA Experiments

Our experiments used several well-known benchmark problems from the literature. Each problem domain was discretized into 1024×1024 intervals: each agent had 1024 actions to choose from. Each agent maintained a population of size 32 and kept the best action discovered to date. The CCEA had a budget of 17600 evaluations of joint rewards. We felt this number of evaluations struck a balance between allowing the agents to sufficiently search the learning space without wasting evaluations.

We compared our method of using a variable number of collaborators against using a fixed number of collaborators: we tested 10 different numbers of collaborators (1 to 10). A single collaborator allowed more generations while more collaborators provided a more accurate fitness assessment. Experiments used the ECJ[1] library and involved 250 runs per method. Statistical significance was computed at the 95% confidence level.

There are a few problems with our experiments. First, the same collaborators are used to evaluate the entire population at each generation, as opposed to drawing random collaborators for each action. This reduces evaluation noise, but also limits the exploration of the search space. Second, the collaborators are chosen based on tournament selection of size 2 rather than randomly. This might diminish the benefit of multiple collaborators later on in the search: in our defense, the results of the selection process might be similar if using random selection from a less diverse population, or using tournament selection with size 2 from a more diverse population.

The first experiment compares the results of the settings described above on the Two-Peaks

[1] `http://cs.gmu.edu/~eclab/projects/ecj/`

Table 3.4: Performance of different collaboration schemes in the discretized two-peak problem domain. 10*5+2*rest significantly outperforms all other settings.

Number Collaborators	Mean Performance	St Dev Performance
1	960.308064	30.622387
2	989.164076	48.826596
3	1004.111361	49.314857
4	1020.230865	44.603223
5	1031.049270	37.868720
6	1035.819193	32.235383
7	1036.992860	30.671499
8	1042.224061	22.536767
9	1042.303012	22.076877
10	1041.991019	20.991886
10*5+2*rest	**1047.110667**	**16.520011**

problem illustrated in Figure 3.11. Specifically, the joint reward for a team with two agents, where the first agent chooses action x and the second agent chooses action y, is computed as

$$f(x,y) = \max \begin{cases} 950 - 500 * \left(\left(\frac{x-16}{64}\right)^2 + \left(\frac{y-16}{64}\right)^2 \right) \\ 1050 - 9600 * \left(\left(\frac{x-48}{64}\right)^2 + \left(\frac{y-48}{64}\right)^2 \right) \end{cases}$$

where x and y range from 0 to 64, discretized into 1024 intervals. The results are summarized in Table 3.4. The results for our decreasingly lenient setting, 10*5+2*rest, are significantly better than the ones for all other settings.

The second experiment tested the methods using a Rosenbrock-like function

$$f(x,y) = 1000 - \left(2 * \left(x^2 - y\right)^2 + (1-x)^2\right)$$

where x and y range between -5.12 and 5.12 discretized into 1024 intervals. The results are presented in Table 3.5. The results for 10*5+2*rest are again significantly better than the ones for all fixed

Table 3.5: Performance of different collaboration schemes in the discretized Rosenbrock-like problem domain. 10*5+2*rest significantly outperforms all other settings.

Number Collaborators	Mean Performance	St Dev Performance
1	999.805368	0.702603
2	999.931824	0.165259
3	999.948679	0.151703
4	999.946774	0.133769
5	999.939639	0.142567
6	999.947729	0.110188
7	999.951085	0.082815
8	999.943306	0.107280
9	999.940341	0.117084
10	999.945124	0.088891
10*5+2*rest	**999.986614**	**0.054527**

collaboration schemes.

A third experiment used the Griewangk function

$$f(x,y) = 1000 - \left(1 + \frac{x^2}{4000} + \frac{y^2}{4000} - \cos(x) * \cos\left(\frac{y}{\sqrt{2}}\right)\right)$$

with x and y between -5.12 and 5.12, discretized again into 1024 intervals. This modified function has several suboptimal peaks, and an optimum of value 1000 at (0,0). The results of different collaboration methods on this problem domain are presented in Table 3.6. The results indicate that 10*5+2*rest is significantly better than all other settings except for 8 collaborators.

The fourth experiment used the Booth problem domain

$$f(x,y) = 1000 - \left((x + 2*y - 7)^2 + (2*x + y - 5)^2\right)$$

(transformed to a maximization problem), with x and y between -5.12 and 5.12, discretized into 1024 intervals. This is a challenging problem for coevolutionary search because of non-linearities among the variables; the optimum of the function is at (1,3) and it has a value of 1000. The results of the

Table 3.6: Performance of different collaboration schemes in the discretized Griewangk problem domain. 10*5+2*rest significantly outperforms all other settings, except for 8 collaborators.

Number Collaborators	Mean Performance	St Dev Performance
1	999.974879	0.087959
2	999.992303	0.017698
3	999.992725	0.014395
4	999.994459	0.003547
5	999.994457	0.003247
6	999.994630	0.003522
7	999.994800	0.003434
8	**999.995458**	**0.003786**
9	999.994743	0.004014
10	999.994870	0.004457
10*5+2*rest	**999.995700**	**0.003646**

methods in this domain are presented in Table 3.7. The 10*5+2*rest method has a significantly better performance than all fixed settings.

3.3.3 Lenient Reinforcement Learning

Reinforcement learning (RL) methods update the utility estimates of performing actions in various environmental states, or for being in those states. These utilities are used for both exploration of the space as well as exploitation of the agent's knowledge of the space. Due to exponential memory growth of traditional RL techniques, multiagent reinforcement learning breaks the joint utility tables into simpler utility tables, one per agent. This projection of the joint utility tables into per-agent utility tables removes the direct coordination between agents.

Typical multiagent RL research performs utility updates based on every reward received, i.e., without lenience. Our method to introduce lenience into multiagent RL is based on the following idea: in the early stages of learning, the agent takes the maximum reward received when choosing action a_i. As both agents become more selective in action selection, it is expected that the agents will primarily select a single action (the one with highest reward) after some time. At this point, we

Table 3.7: Performance of different collaboration schemes in the discretized Booth problem domain. 10*5+2*rest significantly outperforms all other settings.

Number Collaborators	Mean Performance	St Dev Performance
1	999.944621	0.258064
2	999.945253	0.278818
3	999.896547	0.624545
4	999.927854	0.182885
5	999.940833	0.168899
6	999.911407	0.215897
7	999.912132	0.194326
8	999.893961	0.227951
9	999.903006	0.208813
10	999.877023	0.315515
10*5+2*rest	**999.996837**	**0.022781**

want the agent to perform utility updates with every received reward, which will lower the estimated utility until it is equal to the mean reward for that joint action.

The algorithm is implemented as follows. We always update the utility of an action if the current reward exceeds the utility of the action. Otherwise, we update probabilistically: if the agent has not sufficiently explored an action, it should show lenience to its teammates and not update the utility; but if the agent has sufficiently explored the action, it should be more critical and use the reward to lower the utility of that action. The algorithm associates a temperature with each action, rather than a single temperature associated with the learning process. The temperature associated with an action is decreased slightly every time the action is selected, thus, the agent is more lenient with actions with a high temperature, and more critical with actions with a lower temperature. While this might cause overoptimistic evaluations initially (and cause the agent to temporarily choose suboptimal actions), the utilities of such actions will decrease in time and the agent will likely choose the optimal action. There is also a small (0.01) probability of ignoring small rewards at all times: our experiments showed this allows the agent to have non-zero probability of selecting an action at each time step.

Other than the above modifications, our lenient reinforcement learning algorithms follows a traditional RL approach (See Algorithm 1) : the action selection uses a Boltzman distribution (Lines

Algorithm 1 Lenient Multiagent Reinforcement Learning

Parameter: $MaxTemp$: maximum temperature
Parameter: α: temperature multiplication coefficient
Parameter: β: exponent coefficient
Parameter: δ: temperature decay coefficient
Parameter: λ: learning rate
Parameter: N: number of actions

1: **for all** actions i **do**
2: $U_i =$ random value between 0 and 0.001
3: $Temp_i = MaxTemp$
4: **loop**
5: $MinTemp = 10^{-6} + \min_{i=1}^{N} Temp_i$
6: $W_i = \frac{e^{\frac{U_i}{MinTemp}}}{\sum_{j=1}^{N} e^{\frac{U_j}{MinTemp}}}$
7: Use probability distribution W_1, ..., W_N to select action i
8: $Temp_i = Temp_i * \delta$
9: Perform action i and observe reward r
10: $RandVal =$ random value between 0 and 1
11: **if** $(U_i \leq r)$ or $\left(RandVal < 10^{-2} + \beta^{-\alpha * Temp_i}\right)$ **then**
12: $U_i = \lambda * U_i + (1 - \lambda) * r$

5 – 8) and the utilities are updated based in part on the reward currently received for an action (Lines 9 – 12).

Lenient RL Experiments

We conducted experiments with learning in repeated coordination games where agents repeatedly and independently choose actions and both receive the joint reward based on the joint action they selected. The goal is to maximize their reward (or maximize their average reward in stochastic games). Table 3.8 shows the four games from the literature we used. Climb is difficult due to the large miscoordination penalties while Penalty is challenging due to the presence of two optimal joint actions. Both games also have Nash equalibria. The stochastic versions of the Climb game introduces noise into the reward function.

We experimented with lenient multiagent RL in all four problem domains. Based on a preliminary sensitivity study of the parameters, we set $MaxTemp = 500$, $\alpha = 2$, $\beta = 2.5$, $\delta = 0.995$, and $\lambda = 0.95$. The agents learned over 7500 moves and our results are averaged over 10 trials of 1000 runs each.

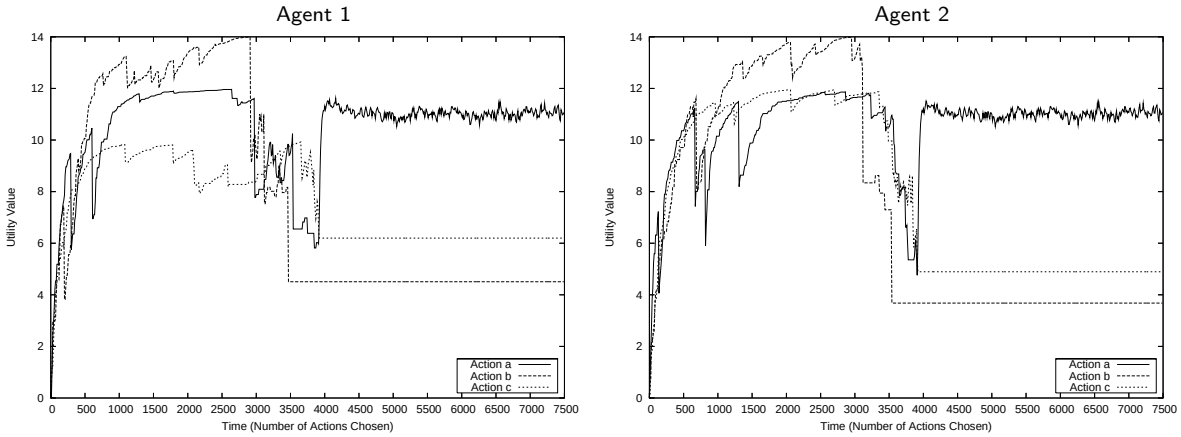


Figure 3.14: Utility of each agent's three actions.

Table 3.9 reports the average number of runs that converged to each of the nine joint actions. The lenient multiagent RL algorithm consistently converged to the global optimum in the Climb, Penalty, and Partially-Stochastic games. Unlike other algorithms from the literature, lenience also helped the agents learn the global joint action for the Fully-Stochastic Climb domain in more than 93.5% of the runs.

For additionally clarity, Figure 3.14 illustrates the utilities for each action during a concurrent learning process in the Fully-Stochastic Climb domain. Both agents start with low utilities for all actions, and they slowly start to believe that action b has utility of 14, higher than the other actions. As both agents start to choose b, they decrease the associated temperature and they start to incorporate lower rewards (the joint actions (b,b) has average reward of 7). Thus, the utility of action b decreases and the agents start to explore other actions, which introduces additional miscoordination penalties which lower the utility estimates for all actions. Given the high temperature of action a, its utility is affected by high rewards, and both agents start to prefer it over other actions. This decreases the miscoordination penalties, and both agents end up with precise utilities associated with that action.

Lenient RL Experiments

We conducted experiments with learning in repeated coordination games where agents repeatedly and independently choose actions and both receive the joint reward based on the joint action they

Table 3.8: Joint reward matrixes for the (a) Climb, (b) Penalty, the (c) Partially-Stochastic and (d) the Fully-Stochastic (bottom right) versions of the Climb domain. In the stochastic games, the first reward is returned with probability p, and the second reward is returned with probability $1-p$.

Agent 1 \ Agent 2	a	b	c
a	11	-30	0
b	-30	7	6
c	0	0	5

(a)

Agent 1 \ Agent 2	a	b	c
a	10	0	k
b	0	2	0
c	k	0	10

(b)

Agent 1 \ Agent 2	a	b	c
a	11	-30	0
b	-30	14/0	6
c	0	0	5

(c)

Agent 1 \ Agent 2	a	b	c
a	10/12	5/-65	8/-8
b	5/-65	14/0	12/0
c	5/-5	5/-5	10/0

(d)

selected. The goal is to maximize their reward (or maximize their average reward in stochastic games). Table 3.8 shows the four games from the literature we used. Climb is difficult due to the large miscoordination penalties while Penalty is challenging due to the presence of two optimal joint actions. Both games also have Nash equalibria. The stochastic versions of the Climb game introduces noise into the reward function.

We experimented with lenient multiagent RL in all four problem domains. Based on a preliminary sensitivity study of the parameters, we set $MaxTemp = 500$, $\alpha = 2$, $\beta = 2.5$, $\delta = 0.995$, and $\lambda = 0.95$. The agents learned over 7500 moves and our results are averaged over 10 trials of 1000 runs each. Table 3.9 reports the average number of runs that converged to each of the nine joint actions. The lenient multiagent RL algorithm consistently converged to the global optimum in the Climb, Penalty, and Partially-Stochastic games. Unlike other algorithms from the literature, lenience also helped the agents learn the global joint action for the Fully-Stochastic Climb domain in more than 93.5% of the runs.

Table 3.9: Average number of runs (out of 1000) that converged to each of the joint actions for the four coordination games.

Climb domain

	a	b	c
a	992.4	0	0
b	0	7.6	0
c	0	0	0

Penalty domain

	a	b	c
a	494.6	0	0
b	0	0	0
c	0	0	505.4

Partially-Stochastic Climb

	a	b	c
a	999.0	0	0
b	0	0.0	0.6
c	0	0	0.4

Fully-Stochastic Climb

	a	b	c
a	935.2	0	0.001
b	0	20.2	20.4
c	0	0	24.1

3.3.4 Combination of Lenient Learning with Other Algorithms

Concurrent learning raises a question in multiagent learning: what if the agents use entirely different learning algorithms? The previous sections assumed that all the agents used exactly the same algorithm, but this is not always true: consider a web-based agent which would have no control of the other learning agents. While other researchers showed a "smart" learner can compensate for a less smart learner in certain environments, our work showed that in difficult domains, both learners must be "smart" in order to converge to the global optima, and that poor learners may force "smart" learners to suboptimal solutions [305].

We experimented with four learning algorithms: traditional reinforcement learning, FMQ (a RL-variant from the literature) [309], and lenient RL (LMRL) and lenient CCEA (EA) discussed above. We ran each combination of learning algorithms 1000 times and computed the number of times the optimal joint action was discovered. Results were averaged over 30 runs. For more experimental details see [305].

Tables 3.10 show the results for the Climbing and the Penalty domains, as well as for the partially and fully stochastic versions of the Climbing domain. The tables are symmetric — the values under the main diagonal are left blank for clarity.

Table 3.10: Average number of iterations (out of 1000) that converged to the joint action for the Climbing, Penalty, Partially Stochastic, and Fully Stochastic games.

	RL	FMQ	LMRL	EA
RL	462.7	999.8	468.9	913.5
FMQ	–	999.9	999.9	910.2
LMRL	–	–	991.6	913.4
EA	–	–	–	837.3

(a) Climbing Game

	RL	FMQ	LMRL	EA
RL	999.8	1000.0	999.5	998.2
FMQ	–	1000.0	1000.0	998.3
LMRL	–	–	1000.0	997.4
EA	–	–	–	985.3

(b) Penalty Game

	RL	FMQ	LMRL	EA
RL	468.5	998.4	0.8	218.5
FMQ	–	999.8	998.1	419.9
LMRL	–	–	998.0	214.7
EA	–	–	–	303.8

(c) Partially Stochastic Game

	RL	FMQ	LMRL	EA
RL	464.4	3.3	2.6	230.9
FMQ	–	141.5	4.3	108.0
LMRL	–	–	928.5	235.1
EA	–	–	–	197.0

(d) Fully Stochastic Game

In the Climbing game (Table 3.10(a)), all but three learning teams discovered the optimal action more than 91% of the time. Overall, FMQ and LMRL were the best learners in the Climbing game and they also performed well when combined with each other. FMQ had a slight advantage over LMRL as it also performed very well when paired with RL.

All teams discovered the optimal strategy more than 98% of time in the Penalty game (Table 3.10(b)). This improvement in performance was partly due to the fact that the Penalty game has twice the number of optimal solutions to which learning may converge. Results in the partially stochastic Climbing game (Table 3.10(c)) were similar to the deterministic Climbing game. FMQ and LMRL again were the "smarter" algorithms: they found the optimal joint action in almost all the runs when paired with themselves or each other. FMQ also helped RL converge, but LMRL achieved that in almost every run. The EA algorithm deteriorated significantly across the board.

So far FMQ has done well when paired with other algorithms. However, Table 3.10(d) shows a different result in the fully stochastic Climb game. Only one pair (LMRL-LMRL) efficiently solved the problem. But, importantly, every method but EA did best by far when paired with itself. In general, heterogeneous pairings performed poorly. Notably, combinations that performed well in other games performed poorly in the fully stochastic Climb game.

Chapter 4: Single Agent Learning from Demonstration

Programming agent behaviors is a tedious task requiring multiple code, test, debug cycles, each of which requires expert knowledge to accomplish. Additionally, successfully programming a multiagent system requires knowing all possible states the system can reach. A further challenge arises from the emergent, unforeseen interactions between agents (which increases the size of the learning space). Thus, *training* the agent becomes attractive. However, training has its own issues, particularly the trade-off between real-time nature of training and the large number of examples to learn complex domains. The learning system developed in this thesis, HiTAB, uses several methods to reduce the complexity of the learning space:

- *Space Decomposition:* HiTAB's primary trick is decomposing behaviors into a hierarchy of simpler behaviors.
- *Per-Behavior Feature, State, and Action Reduction:* During training, HiTAB only uses those features, actions, and states necessary to learn the given behavior, thus reducing the dimensionally of the learning space by projecting it into smaller subspaces.
- *Parameterization:* Behaviors and features are parameterizable. For example, rather than training a behavior GOTOHOME, the user can train a general GOTO(X) behavior where *X* is specified at runtime[1] . Features are parameterized in a similar manner. Thus, behaviors and features can be reused within the hierarchy, reducing the total number of training sessions.

The rest of this chapter describes how HiTAB implements these tricks for dimensionality reduction in the single agent case. After presenting the formal HiTAB model, I then present a series of experiments using a single humanoid robot which demonstrates the wide applicability of HiTAB to rapidly training complex, single agent behaviors.

[1]Throughout the thesis, basic behaviors are typeset in bold, features are italicized, and trained HFAs are in small caps.

4.1 Hierarchical Training of Agent Behaviors

HiTAB (Hierarchical Training of Agent Behaviors) is a learning from demonstration (LfD) system which learns a hierarchical finite state automaton (HFA) represented as a Moore machine: individual states correspond to hard-coded agent behaviors, or the states may themselves be another automaton. An HFA is constructed iteratively: starting with a behavior library consisting solely of atomic behaviors (e.g., turn, go forward), the demonstrator trains a slightly more complicated behavior, which is then saved to the behavior library. The now expanded behavior library is then used to train an even more complex behavior which is then saved to the library. This process continues until the desired behavior is trained. An early version of HiTAB was developed by Luke and Ziparo [12].

The motivation behind HiTAB was to develop a LfD system which could rapidly train complex agent behaviors. HiTAB uses manual behavior decomposition as its primary method to reduce the size of the learning space, thus allowing rapid training in areas such as behavior based robotics, where samples are sparse and expensive to collect. HiTAB's behavior decomposition breaks complex joint finite-state automata into a hierarchy of simpler automata which are iteratively learned and then composed via scaffolding into more complex behaviors. This manual decomposition requires the experimenter to choose the automaton's general structure: HiTAB then determines the appropriate transitions. Thus, HiTAB is somewhere between machine learning and programming by demonstration.

Once the hierarchical behavior structure is chosen, the experimenter begins training from the bottom up. Initially, the behavior library consists only of basic behaviors and the experimenter trains one or more automata using these basic behaviors, which are saved to the behavior library. More abstract automata are trained using these simple automata as states, and the process continues until the entire hierarchical behavior is trained. Note that recursion is not permitted, so, the leaf nodes in the behavior hierarchy are basic-behaviors. When a trained behavior is saved for later inclusion in the behavior library, unused states and features are trimmed before saving. Automatons at all levels contain a special *start* state which simply idles until transitioning to another state. Some automatons might also have *flag* states, which are used to signal some event to the parent automaton such as completing a task or success of some action.

When training an hierarchical behavior, the experimenter can choose which features are available, thus, further reducing the size of the learning space. Features within HiTAB 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"). Categorical features are also used to detect flags raised from subsidiary automatons.

HiTAB learns transitions functions which map the current state and feature vector to a new state. Rather than learn a single monolithic transition function, HiTAB learns a set of disjoint functions, one for each state in the automaton. Thus, each automaton has a set of transition functions, each represented as a classifier. If a given state from the behavior library is not used during training, then a corresponding transition function is not learned for that state.

Figure 4.1 shows a simple example of an HFA within HiTAB taken from soccer keepaway, where a group of keepers tries to maintain possession from a group of takers for as long as possible. The figure shows the behavior used to determine when to hold or pass the ball. Figure 4.2 shows an HFA using the HOLDORPASS behavior to gain possession of the ball. It uses two flags from the HOLLDORPASS behavior to determine its sub-behaviors, and uses its own flag state to signal completion to its parent.

HiTAB's automata map a single behavior to a state. However, when using HiTAB, the user typically needs to map a single behavior to multiple states. To circumvent this problem, HiTAB manually trains "wrapper" behaviors: an HFA is trained with a single behavior, then saved to the behavior library with a unique name. In this way, HiTAB can train stateful HFAs.

Both features and states are *parameterizable*: they can be bound to runtime targets. For example, instead of training states called GOTOBALL, GOTOTEAMMATE, and GOTONEARESTPOINT, the experimenter uses parameterization to train a single behavior called GOTO(X) where X is set at runtime. Similarly, features can be parameterized: instead of *DistanceToBall*, HiTAB can use *DistanceTo(Y)*. Parameterization requires that all parameters are bound to some *target* such as "the ball" or "the nearest agent", or they must be bound to another higher-level automaton. In this way, HFAs may also be parameterized. Parameterization reduces the number of behaviors to train, hence allowing faster training of complex behaviors which use similar states and/or features.

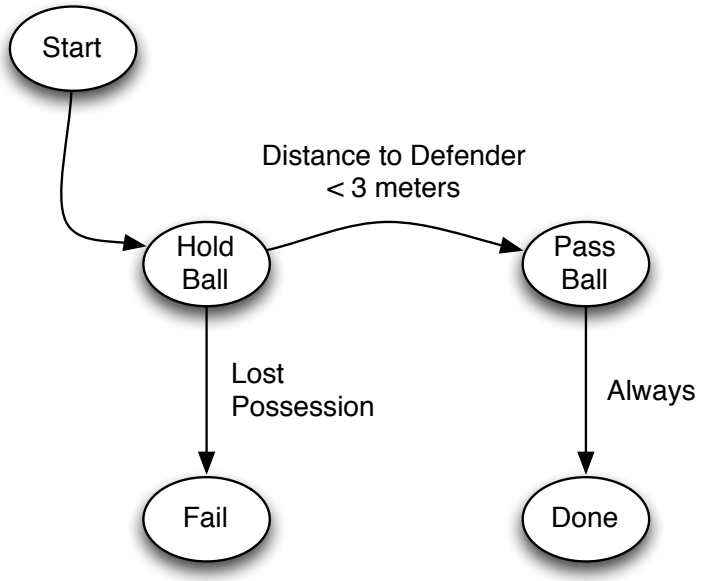


Figure 4.1: A simple behavior for choosing to pass or hold a ball in soccer keepaway. All conditions not shown are assumed to indicate that the agent remains in its current state.

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 state, and if $S_t \neq S_{t+1}$, stops performing S_t's behavior and starts performing S_{t+1}'s behavior. If S_{t+1} is an automaton, this process recurses until a basic behavior is reached.

Training with HiTAB Training is an iterative process of a *training mode* and a *testing mode.* In training mode, the agent performs exactly those behaviors directed by the demonstrator, and each time a 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 old and new states. If the behavior associated with state S_{t+1} is designed to be executed exactly once, then no additional examples are recorded. Otherwise, a default example 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 is observed again.

Once enough examples are collected, the demonstrator switches to *testing mode*, which causes the agent to learn the transition functions within the finite-state automaton. For a given state S_i,

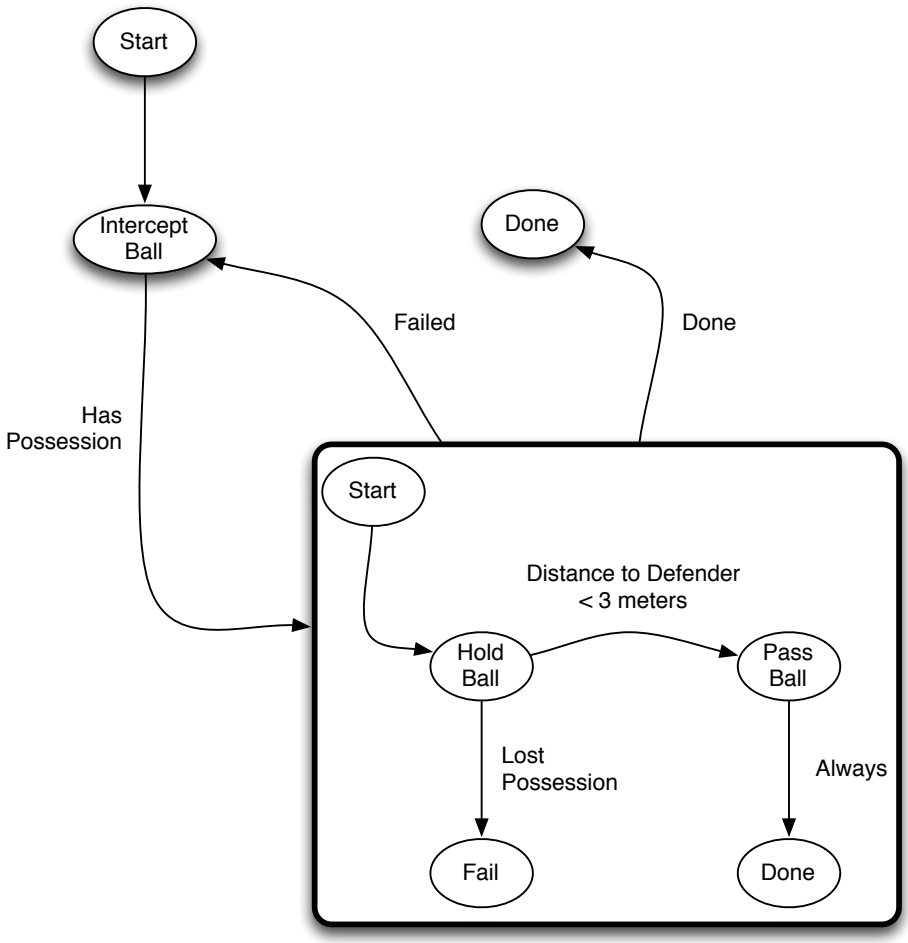


Figure 4.2: A simple HFA for getting a ball in soccer keepaway. All conditions not shown are assumed to indicate that the agent remains in its current state.

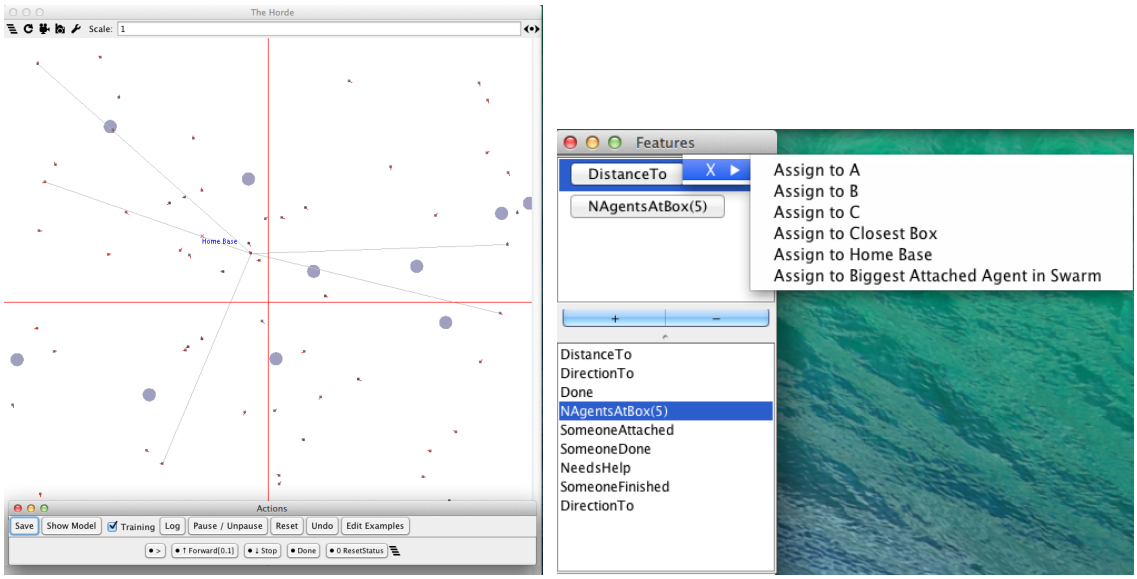
HiTAB takes all examples of the form $\langle S_i, f_t, S_j \rangle$ and reduces them to $\langle f_t, S_j \rangle$ which are fed into the classifier (f_t features and S_j are the labels). The resulting classifier defines the transition function for all outgoing edges from S_i.

While many classification algorithms are applicable, HiTAB uses a version of the C4.5 decision tree algorithm [57] with probabilistic leaf nodes. Decision trees nicely handle different types of data (e.g., continuous, toroidal, and categorical), and do not require scaling the input data. Also, many agent tasks can be approximated by rectangular partitions of the feature space, which makes them a good target for decision trees. Leaf nodes traditionally deterministically compute the class using a plurality of examples which reach the leaf. HiTAB instead computes the class by sampling from a probability distribution over the classes appearing at a leaf node. This approach allows for performing actions some *percentage* of the time. For example, HiTAB can train a random walk by having turn left, turn right, and move forward behaviors all map to the same leaf. The degree of randomness is controlled by the relative number of turn left, turn right, and move forward examples.

After all the transition functions are built, 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 examples. 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.

Implementation HiTAB is implemented atop MASON [297], an open-source multiagent simulation toolkit. MASON provides a discrete-event simulation loop along with (optional) graphics for displaying environments (continuous or discrete) and agents. Developed in Java, MASON executes quickly due to stringent coding practices. Due to the probabilistic leaf nodes, the C4.5 decision tree is directly coded rather than relying on a third-party library.

Constructing a HiTAB model within MASON starts by listing the relevant behaviors and features for the problem at hand. Next, the experimenter stages the environment including static and dynamic obstacles, and any other non-training agents (Figure 4.3(a)). To train an HFA within MASON, the demonstrator first selects relevant features, then grounds targets for behaviors and features (Figure



(a) (b)

Figure 4.3: Screen shots of HiTAB implemented in MASON. (a) An example of a foraging model with multiple agents, food sources, and a single collection point. The window at the bottom shows the current behaviors available to the training agent. (b) Shows how features are bound to specific targets within the problem. All the features listed in the lower panel are possible: only features in the upper panel are used in the trained model.

4.3(b)). Agents are then directed around the environment using either buttons or keystrokes, where examples are collected upon any change in the agent's behavior. Finally, the learned HFA is added to the behavior library for future use.

Formal Model The HFA is at the heart of HiTAB. The class of hierarchical finite state automata $\mathscr{H}$ is defined as a set of tuples $\langle S, B, F, T \rangle$ where

- $S = \{S_1, \ldots, S_n\}$ is the set of *states* in the automaton. Included is one special state, the *start state* S_0, and zero or more *flag states*. Exactly one state is active at a time, designated S_t.

  The purpose of a flag state is simply to raise a flag 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.

Two additional special states are *Increment Counter* and *Reset Counter*. Like the flag states, counters appear as optional features in their parent automata, thus allowing the parent to transition based on the counter value.

- $B = \{B_1, \ldots, B_k\}$ is the set of *basic behaviors*. Each state is associated with either a basic behavior or *another automaton* from $\mathscr{H}$, with the proviso that 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) $G_1, \ldots, G_n$ 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. 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.

4.2 Humanoid Robot Experiments

HiTAB's ability to learn complex behaviors from a small number of examples makes it an ideal LfD system for training humanoid robots. To that end, I conducted a series of experiments using the RoboPatriots (see Appendix A), GMU's humanoid robot soccer team, of which I am the primary developer. The RoboPartriots compete in the international RoboCup [79] competition which offers a complex, dynamic environment ideal for demonstrating single robot behaviors.

Learning from demonstration on humanoid robots is difficult for several reasons. First, the high number of degrees of freedom and number of features can produce a learning task of very high dimension. Second, this high dimensional learning space requires a large number of samples,

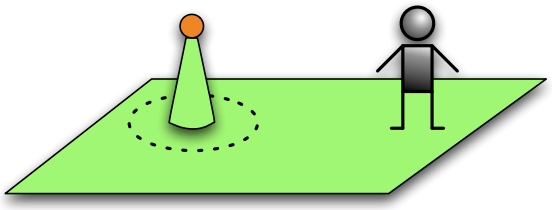


Figure 4.4: Experimental setup. The orange ball rests on a green pillar on a green soccer field at eye level with the humanoid robot. The robot must approach to within a short distance of the pillar, as denoted by the dotted line.

each of which requires a real-time experiment. Third, humanoid demonstrations are fraught with demonstrator error, which require relearning the model on the fly.

The first experiment had novice users train a humanoid robot to perform visual servoing, a fundamental behavior for RoboCup. Additionally, this first experiment validated a claim of LfD: novice users can quickly train complex agent (robot) behaviors. The second set of experiments trained a robot to play soccer while *at* RoboCup (a first in the competition's history), and then used this trained behavior in a penalty kick scenario.

4.2.1 Novice Training

Proponents claim that LfD allows novice users to rapidly train complex agent behaviors without detailed programming knowledge. Taking RoboCup as motivation, I conducted experiments to verify this claim by having novice users train a single humanoid robot visual servoing [310]. The goal was for the robot to search for the ball, orient towards the ball by turning the "correct" direction, and walk towards the ball. The robot uses two features from the camera: the x-coordinate of the ball within the frame, and the number of pixels in the ball's bounding box. Finally, the robot has three basic behaviors available to it: turn left, turn right, and walk forward. The robot's head remains fixed looking forward, and the ball does not move. To ensure the ball does not drop out of the bottom of the frame during the experiments, we raised the ball to the robot's eye level (Figure 4.4).

Note that this is a very simple example of a behavior which may best be learned in a stateful fashion. When the ball disappears from the robot's field of view, which direction should the robot

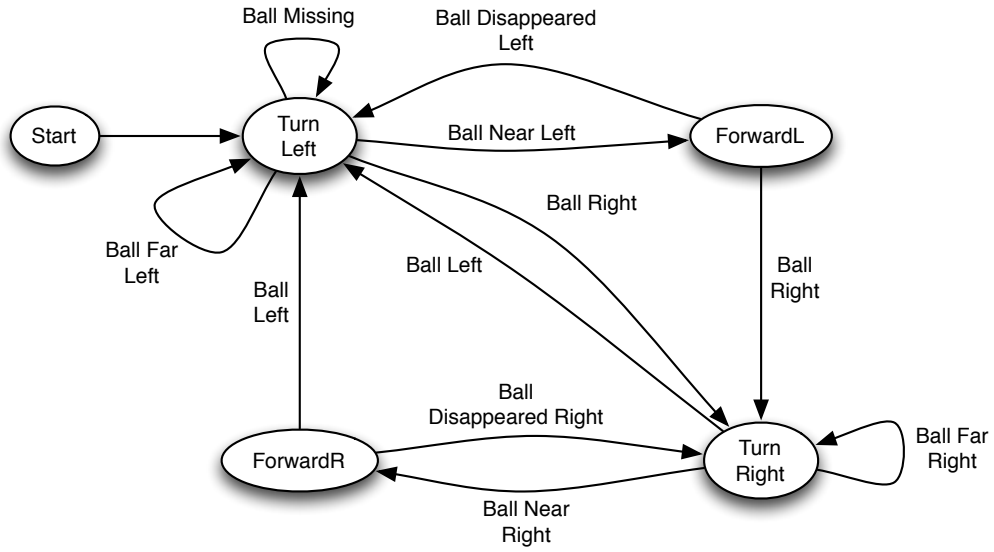
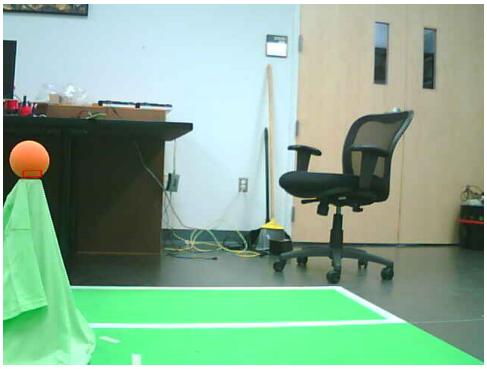
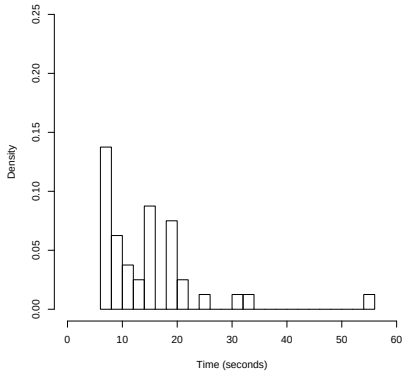


Figure 4.5: The "stateful" HFA to acquire the ball.

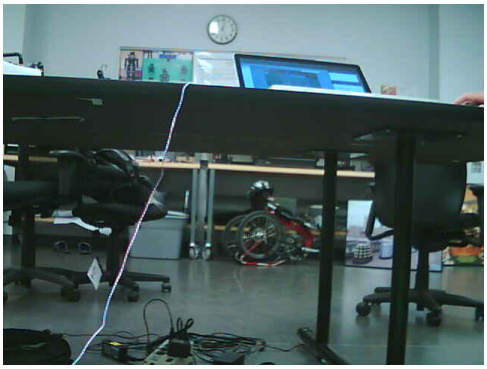
turn? This could be determined from the x-coordinate of the ball in the immediate *previous* frame, which suggests where the ball may have gone. But if the robot only follows a policy $\pi(\vec{f})$, it does not have this information, but simply knows that the ball has disappeared. Thus π would typically 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.

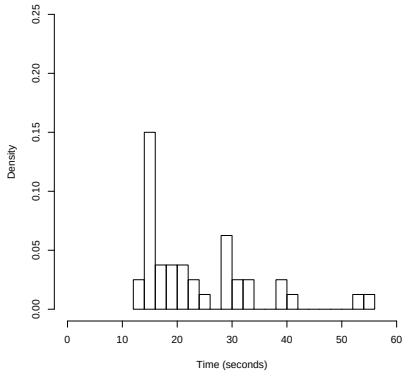
The learning automaton setup had four states to compensate for this (see Figure 4.5): two states, **left** and **right**, which turned left and turned right respectively, but also had two identical states, notionally called **forwardL** and **forwardR**, which both simply moved forward. A demonstrator could use these two states as follows: when the ball is in the left portion of the frame, he instructs the robot to go **forwardL.** When the ball is in the right portion of the frame, he instructs the robot to go **forwardR.** When the ball has disappeared, he instructs the robot to turn appropriately. Ultimately the robot may learn that if the ball disappeared while it was in the **forwardL** state, it should then transition to turning left; and likewise turn right if the ball disappeared while the robot was in the **forwardR** state.



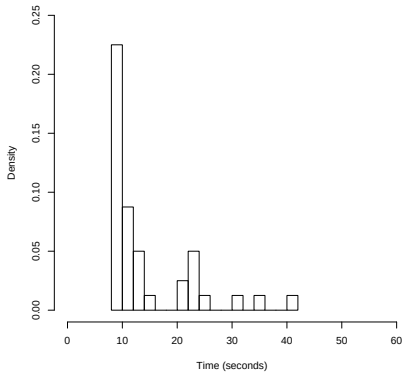


Starting Position 1





Starting Position 2



Starting Position 3

Figure 4.6: The first column shows typical views from the three different starting positions. In Starting Position 2, the robot is facing away from the ball. The second column shows histograms of times to reach the ball from each starting position for the four successful trials.

Five students from a graduate computer science robotics class were asked to train the robot for five minutes each. The students had no prior experience with the system nor the humanoid robot platform. The students were given instructions on how to control the robot, what features meant, and some suggestions on how to use **forwardL** and **forwardR**. Otherwise, the students were given no further guidance as to how the task should be performed. The students saw feature information from the camera in real time, and could also observe the robot directly. The students could place the robot and ball in any configuration, and were allowed to alternate between training and testing mode. After the five minutes of training were up, a final learned automaton was constructed using C4.5. HiTAB ran on a laptop machine with a direct tether to the robot which transferred sensor information from the robot and motion commands to the robot. Performance was tested by placing the robot in three different starting positions with very different ball positions within the frame (see Figure 4.6). The locations were approximately 140 centimeters form the ball. For each trained behavior and each location, I ran 10 independent experiments where the experiment stopped when the robot was within approximately 15 centimeters of the ball as determined by visual inspection. The performance metric was time to complete the task, and all experiments and training were conducted on the same robot.

Four of the five students successfully trained the robot to consistently and robustly approach the ball. The remaining trained behavior never successfully approached the ball, independent of the robot's starting location, due to a training error: the student instructed the robot to perform random behaviors when the ball was lost. Thus, since the starting locations ensure the ball is lost, the robot would randomly wander off the field. The right column of Figure 4.6 shows that in most cases the robot can quickly servo to the ball. However, in several cases the robot takes significantly longer to successfully approach the ball, usually due to sensor noise and/or poor training. Since each run was independent (i.e., essentially starting over), we can consider each run as a Bernoulli distribution, thus, the overall shape of the histograms.

A second experiment tested HiTAB's hierarchical ability: an expert trained the robot to approach the ball as before, but also to stop when the robot was close to the ball. This was done in two ways. First, the expert attempted to train the robot to do all these tasks in the same automaton. Second, the expert first trained the robot to approach the ball using only the ball position within the frame,

and then using this saved approach behavior, trained a simple higher-level automaton in which the robot would approach the ball until it was large enough, then stop. Anecdotal results suggest that the hierarchical approach is much easier to do rapidly than the monolithic approach. Learning the monolithic behavior requires significantly more training exemplars because the joint training space (in terms of states and features) is higher.

4.2.2 Humanoid RoboCup

RoboCup is an annual international competition designed to promote robot and artificial intelligence research. Initially, the competition was only soccer, but has expanded to include search and rescue and domestic assistance competitions. RoboCup's stated goal is fielding a fully autonomous humanoid robot soccer team capable of defeating the World Cup champions (humans) by 2050. RoboCup soccer consists of several leagues, and GMU competes in the difficult and prestigious Humanoid League. Robots in the Humanoid League have human-like proportions and sensors, meaning cameras are the primary way to gather environmental information. Additionally, the robots are fully autonomous. Humanoid RoboCup has many challenges, primarily stability and vision processing, which must be solved in a dynamic, noisy environment. Due to these challenges, teams typically hand-code behaviors rather than apply machine learning techniques.

The RoboPatriots are GMU's entry into the RoboCup Humanoid League [311–315], and I am their primary developer and team leader. (See Appendix A for more details about the RoboPatriots.) Initially, the RoboPatriots focused on issues related to robot design, dynamic stability and vision processing; thus, we exclusively used hand-coded behaviors. Our version of soccer was a simple HFA ("kiddie" soccer): look for the ball, approach the ball, align towards the goal, align for kicking, kick the ball, and repeat. However, as computer scientists, we wanted to apply artificial intelligence and machine learning to the RoboPatriots.

At RoboCup 2011, we fielded HiTAB-trained robots as a proof-of-concept. On the soccer field the night before the 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 via a Nintendo WiiMote. We then saved out the trained behavior, and during the competition,

Table 4.1: Basic behaviors in Robocup Experiments

Continuously turn left	Continuously turn right
Continuously walk forward	Walk forward one step
Sidestep one step left	Sidestep one step right
Stop	Re-calibrate gyros
Pivot left	Pivot right
Left kick	Right kick
Failed	Done
Increment Counter	Reset counter
Wait for next feature packet from camera	

Table 4.2: Features in the Robocup Experiments

Is the ball visible?	X and Y coordinates of the ball on the floor
Bearing to the ball	Bearing to the attack goal
Is an HFA done?	Is an HFA failed
Counter value	

the robots loaded this behavior from a file and used it in an interpreter along side the remaining hard-coded behaviors. The trained behavior was simple and meant as a proof of concept. However, the learned behavior worked perfectly and based on discussions with colleagues at RoboCup, this was the first time LfD had been used at the competition

For 2012, we had a more ambitious goal: train the entire library of behaviors two days prior to the competition. The trained robot used a decomposition of 17 HFAs, shown in Figures 4.7(a)-(d) and 4.8(a)-(b). Admittedly, this is a simple version of soccer ("kiddie" soccer): search for the ball, approach the ball, align to the goal, align for kicking, kick, and repeat. Table 4.1 and Table 4.2 show the basic behaviors and features used during training: these are essentially the same behaviors and features used in the hard-coded version. Not all features were used for each HFA. The **Wait for Camera** behavior ensured we had current vision information before transitioning (the speed difference between vision processing and the main control loop necessitated this coordination). The final HFA was saved to disk and run through an interpreter during game play.

The top-level HFA MAIN (See Figure 4.8(b)) performed "kiddie" soccer using the following

sub-HFAs. MAIN used *Done* and *Failed* flags to control the overall behavior: if any sub-HFA failed, a **Stop** is performed to recalibrate the gyros, while a transition occurs when any sub-HFA is *Done*.

- SEARCH FOR BALL: Using the bearing to the ball, the robot performs visual servoing on the ball, with the additional constraint of performing a rotation if the ball is missing for several frames. If the robot rotates several times, then it walks forward before resuming searching. At the end of this HFA the robot is pointed at the ball. See Figure 4.7(a).

- APPROACH BALL: Using the bearing to the ball and distance to the ball, the robot moves towards the ball while performing course corrections en route. The HFA fails if the ball is missing for several frames. At the end of this HFA, the ball is near robot's feet. See Figure 4.7(b).

- ALIGN TO GOAL: Using the bearing to the goal, the robot orients toward the goal, while maintaining the ball near its feet. The robot pivots around the ball if it cannot see the goal. The HFA fails if the robot pivots several times without seeing the goal. The robot is pointed towards the goal with the ball near its feet at the end of this HFA. See Figure 4.7(c).

- ALIGN FOR KICK: Using the X and Y position of the ball, the robot takes small steps to get the ball near the feet so a kick can be performed. If the ball is outside a box centered at the robots feet or if the ball is missing for several frames, this HFA fails. At the end of this HFA, the ball is very close to the robot's feet. See Figure 4.7(d).

- KICK BALL: The robot performs a kick based on X position of the ball. If after a kick the ball is still there, then the robot kicks with the other foot. If the ball is *still* there, the robot takes a step forward and tries to kick with the original foot. If the ball is *still* there, then the robot kicks a final time with the opposite foot. The HFA fails if the ball is still present. See Figure 4.8(a).

The higher level HFA related to the game state was not trained. Also, if the robot fell over, hand-coded logic righted the robot and reset the trained HFA to SEARCH FOR BALL.

It took approximately 8 hours to the train the HFAs. Poor network reliability combined with HiTAB GUI issues were the primary reasons training took so long; fixing these issues would result in

far less training time in the future. Table 4.3 shows the number of samples collected for all 17 trained HFAs. The first column includes automatically included default samples while the second column is only the directly provided samples. Given the problem complexity (and associated learning space size), the number of samples was remarkably low.

As an illustration of HiTAB's ability to rapidly train behaviors, after the first day of competition we trained an additional HFA. During our early matches, we observed an error in the AIM FOR KICK behavior: we assumed the ball would be located near the robot's feet. However, due to sensor noise, we entered ALIGN TO GOAL when the ball was far away, so upon entry into AIM FOR KICK the ball was distant, resulting in the robot taking many, many small steps to the ball. To fix this problem, we trained a new version AIM FOR KICK WITH BALL AHEAD to include a failure for when the ball is outside a box centered at the robot's feet. The new HFA was included during our later matches.

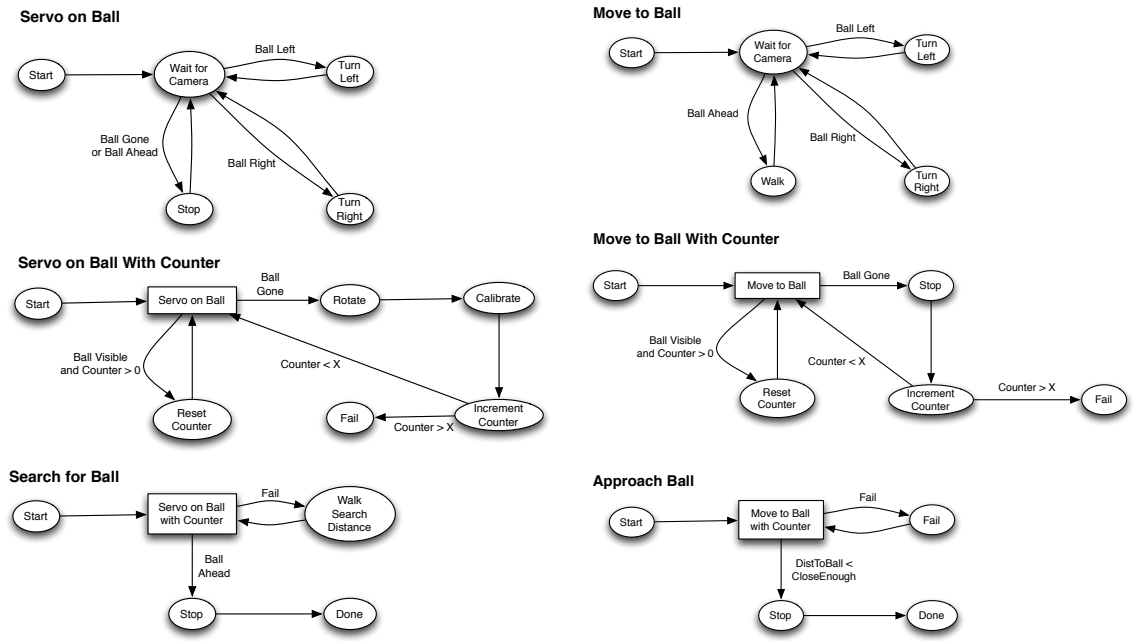
During our second match versus Team JEAP from Osaka, Japan, the trained robot scored a goal which proved to be the winning difference. After discussions with colleagues at the competition, we concluded that, to the best of our knowledge, this is the first time a team at RoboCup has used a behavior taught to the robots on the field at the competition itself.

4.2.3 Penalty Kick Experiments

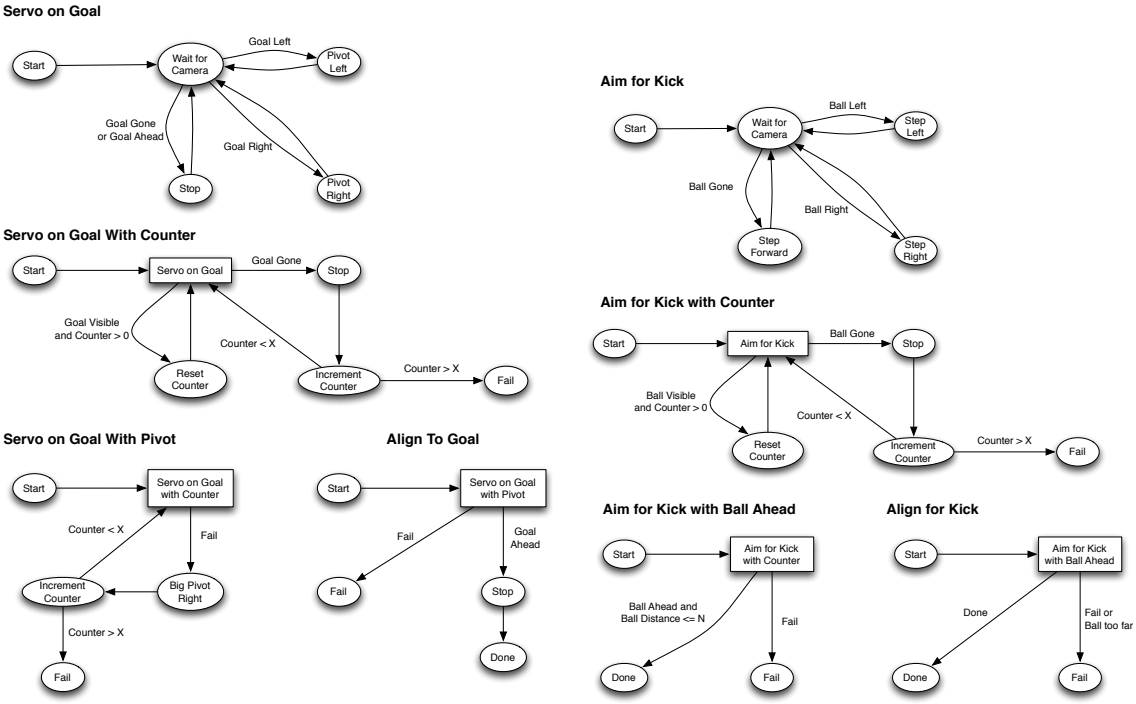
One claimed benefit of LfD is that trained behaviors perform similarly to hand-coded behaviors. After RoboCup 2012, I conducted experiments to validate this claim by comparing the above trained soccer behavior with the hand-coded behavior deployed on the other attacker. The task was penalty kicks, similar to those used during RoboCup competition.

The robot was placed 40 cm away from the penalty kick mark with a neutral head position and facing the goal. The ball was randomly placed within a 20 cm diameter circle centered on the penalty kick mark (see Figure 4.9(a)). Initially, the robot could see the goal, but not the ball (see Figure 4.9(b)). The metric was time to kick the ball, independent of whether a goal was scored. Both behaviors were run 30 times. Since each behavior run was independent of the others, we can consider each run as a Bernoulli variable.

Figures 4.10(a)-(b) show histograms for the hardcoded and trained behaviors. For both behaviors, sensor noise caused one trail to take significantly longer than the rest. The trained behavior had

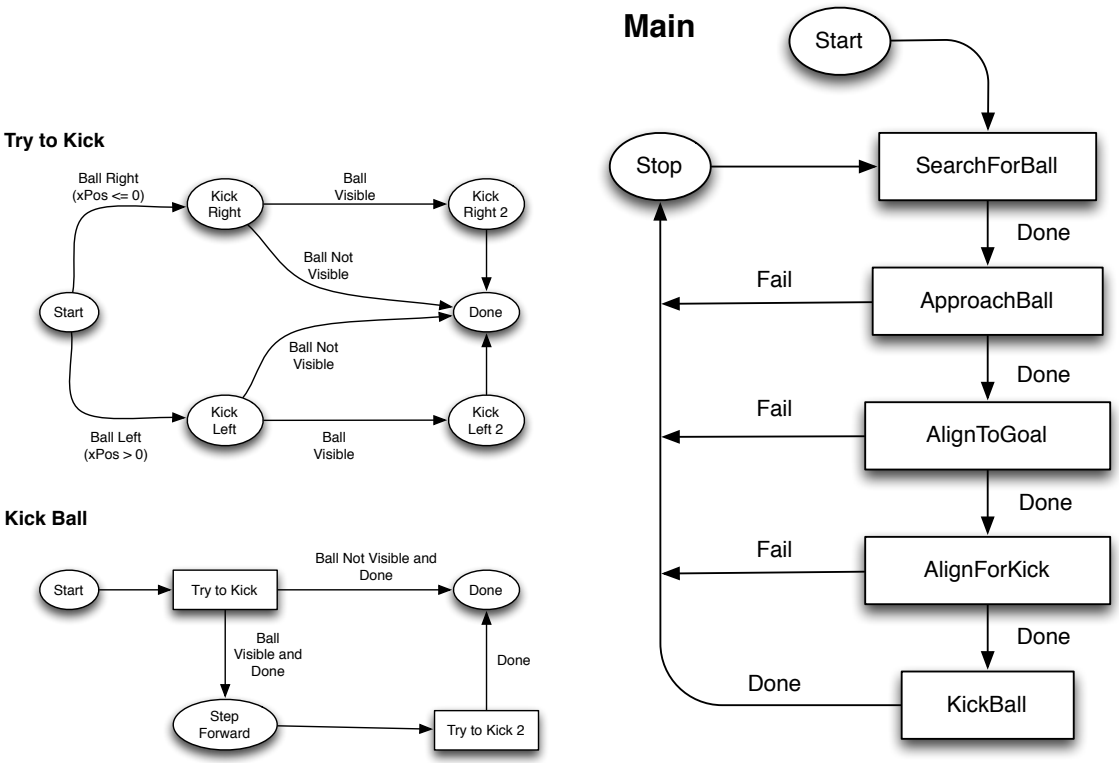


(a) Search (b) Approach



(c) Align to Goal (d) Align for Kick

Figure 4.7: The first four trained HFAs for Robocup. Unlabeled transitions are always executed.



(a) Kick ball

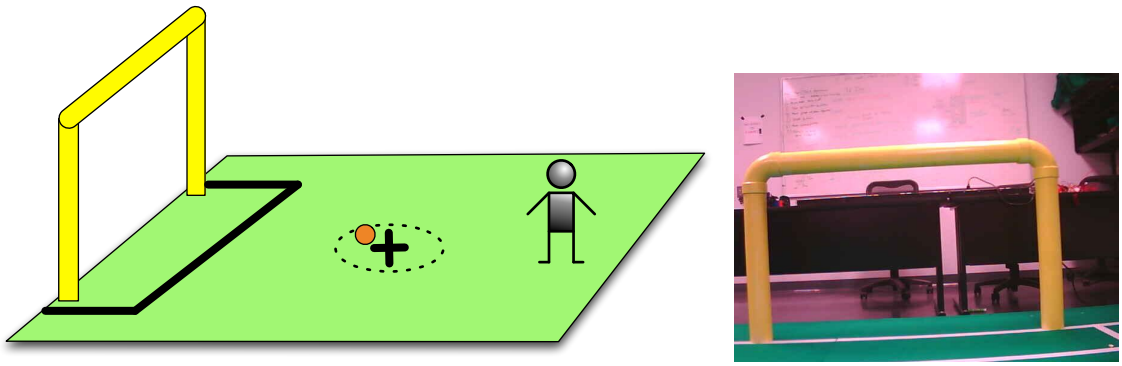
(b) Main

Figure 4.8: The final HFAs and the main HFA for Robocup. Unlabeled transitions are always executed.

Table 4.3: Number of data samples for each HFA trained at RoboCup 2012. The data for *Servo On Ball With Counter* was not saved, so the estimate is based on other HFAs which used a counter.

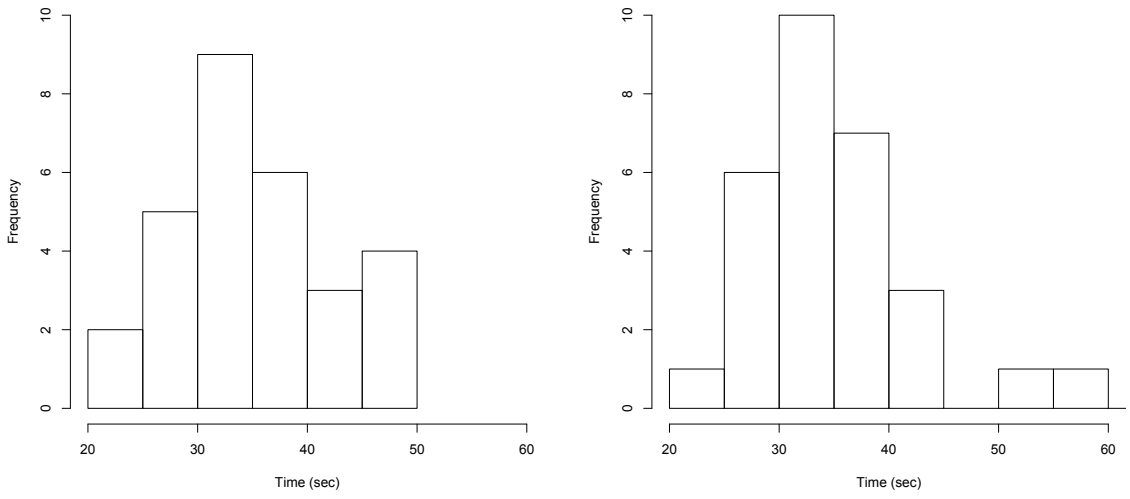
Behavior	**Number of Samples**	**Number of Provided Samples**
Servo On Ball	11	11
Servo On Ball With Counter	(estimate) 10	(estimate) 10
Search For Ball	10	8
Move To Ball	9	9
Move To Ball With Counter	10	9
Approach Ball	15	11
Servo On Goal	9	9
Servo On Goal With Counter	12	11
Servo On Goal With Pivot	9	7
Align To Goal	12	9
Aim For Kick	9	9
Aim For Kick With Counter	10	9
Aim For Kick With Ball Ahead	22	14
Align For Kick	42	35
Try To Kick	10	10
Kick Ball	9	6
Main	34	19
Total	243	196

a mean execution time of 37.47 $\pm$ 5.51 seconds (95% confidence interval) while the hardcoded behavior had a mean of 35.85 $\pm$ 3.08. The means were statistically the same. The results verify a claim of LfD: a trained behavior performs as well as a hard-coded behavior.



(a) Penalty kick experimental layout (b) Initial conditions

Figure 4.9: Penalty kick experimental layout. The unusual white-balance is on purpose to ensure better color segmentation in the lab. Also, note that the robot cannot initially see the ball.



(a) Hardcoded results (b) Trained results

Figure 4.10: Penalty kick results. Note that in both experiments, one run took longer than 60 seconds.

Chapter 5: Unlearning

HiTAB's real-time nature can cause demonstrator error in the form of *bad* examples, which cause the agent to learn an incorrect HFA. For example, during training the demonstrator might tell the agent to turn left, when, in reality, the agent should have turned right. These bad examples could cause HiTAB to learn an incorrect model which leads to erroneous behavior that is only noticeable during testing. Additionally, due to HiTAB's paucity of examples, incorrect examples can have a profound impact on the trained behavior. Ordinarily a machine learning experimenter would simply add additional examples until the learned model can determine which examples are noise and which are not. However, adding many additional examples to HiTAB could be prohibitively costly since each example is added in real-time. I take a different approach that exploits the training aspect of HiTAB: the user provides a set of corrective examples (assumed to be accurate) which HiTAB uses to detect and potentially remove incorrect examples. Referred to as *unlearning*, this approach allows HiTAB to iteratively correct erroneous classifiers with a limited set of data.

The unlearning algorithms presented in this chapter work as follows: after the trainer has observed an incorrect behavior, he provides a set of corrective examples. For each corrective example, the agent queries the classifier to determine if this example is misclassified. If it is misclassified, then the agent determines a set of *misclassifying examples* which are candidates for possible deletion. The agent then deletes selected examples from this set which it determines are likely noise. The corrective example is then added to the database and the classifier is rebuilt.

One challenge is identifying examples that are truly noisy or are simply overgeneralizing the space. Misclassifications may come from undersampled regions where a correct example is inappropriately responsible for classifying a large region. This is particularly problematic for HiTAB since examples are scarce. Thus, the unlearning algorithms presented in this chapter are highly conservative about deleting examples since each example may contain critical information about how the agent should behave. I present two methods for identifying noisy examples (versus correct but overgeneralizing

ones) and unlearning them, and I also present methods for gathering candidates for unlearning. All these algorithms rely on the specific classifier being used: I show methods for decision trees, support vector machines, and K-nearest neighbor.

5.1 Related Work

Minimal work has addressed the problem of correcting a learned behavior [316, 317]. The bulk of research has focused on dealing with noisy data in classification problems [318–320]. However, such research has largely focused on determining the nature of the noise rather than correcting the data directly. More closely related to this work is the idea of correcting a sample bias where the training and testing data are drawn from different probability distributions [321]. Majority voting among multiple classifiers has been used to identify errantly labeled data from physical experiments [322], and classifiers have been used to fill in missing data values [323, 324]. These techniques are closely related to the notion of *boosting* (most famously [325]), where successive classifiers are trained on data weighted such that misclassified samples are given more consideration. Unlike this work, these approaches do not assume the presence of a corrective dataset.

Another related area is *incremental learning*, where a classifier is built incrementally from currently available data or from a subset of large dataset. Incremental learning allows the construction of dynamic classifiers capable of adapting to changing data. Incremental learning addresses situations where not all data fits in memory, or where new data arrives as a stream [326–328]. Note that this new data is not meant to correct old data per se: it simply is newer information which may require an incremental restructuring of the model.

Researchers have implemented incremental decision trees [329], Bayesian inductive learning [330], and neural networks [331]. Additionally, [332, 333] developed an on-line incremental and decremental support vector machine. As training vectors are encountered, an exact solution is computed using all the previously seen data combined with the new training vector. This incremental approach continually partitions the data into three sets, one of which is the support vectors. Ma et al extended this approach to support vector regression [334].

5.2 Unlearning Method

I developed two unlearning methods, one which assumes a distance metric in feature space and one which does not. More formally, the unlearning algorithms operate as follows. Let **C** be the current errant classifier, and let G be the data set from which **C** was constructed. The goal is to identify and remove noisy examples from G. The demonstrator supplies a set E of *corrective examples* which is used to identify noisy examples and provide additional examples to prevent overgenerlaization. For each e in E, if **C** misclassifies e, both methods call a *candidate-gathering algorithm* which constructs a set M of *candidate examples* which were responsible for the misclassification. The unlearning methods determine which, if any, examples from M should be removed from G, remove them, and then add e to G before rebuilding **C**. When using K-NN, the candidate-gathering algorithm is more conservative, using a technique to reduce M to only those examples which are not likely to be on the "border" delimiting two different classes.

Both unlearning methods use two heuristics to distinguish between noisy examples and overgeneralizing but correct examples. First, the *similarity heuristic* declares a misclassifying example m noisy if it is highly similar to a corrective example e. On the other hand, if m is misclassifying from afar, then m may be overgeneralizing. Second, if multiple misclassifying candidates are clustered together, then e is likely a special case. This is the *strength in numbers heuristic*.

The first unlearning method, *metric unlearning* assumes the presence of an appropriate distance metric, and uses distance when applying both heuristics. Obviously, a reasonable distance metric is not always available. The second unlearning method, *non-metric unlearning*, removes the assumption of distance, and simply lowers the "score" of each misclassifying examples, ultimately removing examples whose score has fallen below zero.

5.2.1 Metric Unlearning

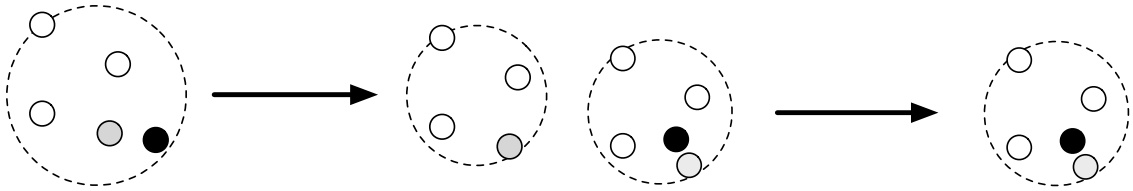
The metric unlearning method first scales the corrective example e and all the candidate examples $m \in M$ to [0,1] based on user-defined minimum and maximum values (see Algorithm 2). Next, the candidate $m^* \in M$ most similar to e is identified. I use cosine similarity, but any similarity metric is possible (e.g., distance, Person's similarity). Then, the algorithm computes the bounding hypersphere

Algorithm 2 METRIC-UNLEARNING $(\mathbf{C}, E, G, \gamma, \beta)$

for all $e \in E$ **do**
  $e' \leftarrow$ Scale e to $[0,1]$
  $M \leftarrow$ GATHER-EXAMPLES$(\mathbf{C}, e, ...)$
  $M' \leftarrow$ Scale all $m \in M$ to $[0,1]$
  $m^\star \leftarrow \text{argmax}_{m' \in M'}$ SIMILARITY(m', e')
  c $\leftarrow$ centroid of all points $m' \in M'$
  r $\leftarrow \max_{m' \in M'}$ DISTANCE(m', c)
  if $r = 0$ and SIMILARITY$(m^\star, e') > \gamma$ **then**
    Delete $m^\star$ from G
  else if DISTANCE$(e', c) < r$ **then**
    $M'' \leftarrow M' - \{m^\star\}$
    c $\leftarrow$ centroid of all points $m'' \in M''$
    r $\leftarrow \max_{m'' \in M''}$ DISTANCE(m'', c)
    if $r = 0$ and SIMILARITY$(m'', e') < \beta$ **then**
      Delete m^* from G
    else if DISTANCE$(e', c) > r$ **then**
      Delete m^* from G
  Add e to G
  Recompute $\mathbf{C}$ from G

H of the examples in M where the centroid is the numerical average of the feature vector and the radius is the maximal distance between the centroid and all the examples in M. There are two possible situations. If there is only a single misclassifying candidate (thus, the radius of the bounding hypershpere is zero), then the candidate m^* is deleted from G if it is sufficiently similar to e. The threshold γ controls how much noise in the feature vectors is acceptable to call two examples similar.

If there is more than one misclassifying example, then the strength in numbers heuristic is applicable. The goal is to determine if m^* is distinct from other points in M, or if all of M are acting as a group to misclassify e. The test considers the relationship between H and e: if e is outside H, then M is considered a group and no examples are removed from G. If e is within H, then a second hypersphere H' is constructed consisting of all candidates except for m^*. If there are multiple candidates inside this second hypersphere (i.e., H' has non-zero radius), then m^* is deleted if e is outside H' based on the belief that e and m^* are significantly separated from other points in M. See Figure 5.1 for a visual explanation. If there is only one candidate m'' (i.e., H' has zero radius), then m^* is deleted if e and m'' are *not* so similar that e and m'' can be considered clustered together. The threshold β determines this level of similarity. Finally, e is added to G and the classifier $\mathbf{C}$ is rebuilt.



(a) Removal (b) Non-removal

Figure 5.1: The black circle represents a corrective example e, and the white and gray circles represent candidate examples M. The gray circle m^* is most similar to e. Figure (a) shows the black example being removed based on the second hypersphere. Figure (b) shows the black example being retained since it still falls within the second hypersphere.

Similarity Note that this algorithm uses two different measures of similarity. Computing hyperspheres is not-scale free, so the algorithm uses metric distance, DISTANCE, for comparison of hypersphere centers. However, because the scaling is user-defined, a more conservative scale-free measure is used for SIMILARITY. Cosine similarity uses the angle between two vectors to determine their similarity:

$$\text{CosSim}(A,B) = \cos(\theta) = \frac{A \cdot B}{\| A \| \| B \|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Cosine similarity ranges from -1 to 1. A value of -1 means the vectors are exactly the opposite, a value of 0 means they are orthogonal, and a value of 1 means they are identical. Cosine similarity is invariant to rotation and distance, but not to translation. The experiments presented below use cosine similarity.

Another option is Person's similarity which measures the linear correlation between two vectors, and is also invariant to rotation and distance. However, since the vectors are shifted, it is also invariant to translation. The Pearson's similarity is defined as:

$$\text{PersonSim}(A,B) = \frac{\sum_{i=1}^{n} (A_i - \bar{A})(B_i - \bar{B})}{\sqrt{\sum_{i=1}^{n} (A_i - \bar{A})^2}\sqrt{\sum_{i=1}^{n} (B_i - \bar{B})^2}} = \text{CosSim}(A - \bar{A}, B - \bar{B}).$$

Algorithm 3 NON-METRIC UNLEARNING $(\mathbf{C}, E, G, \alpha)$

```
1: for all e ∈ E do
2:     M ← GATHER-EXAMPLES(C, e, ...)
3:     for all m ∈ M do
4:         m_score ← m_score − α/|M|
5:         if m_score < 0 then
6:             Delete m from G
7:     Add e to G
8:     Recompute C from G
```

Pearson's similarity also ranges from -1 to 1. A value of 1 implies a perfect linear relationship between the vectors where all the points lie on a line for which B increases as A increases. A value of -1 also implies a perfect linear relationship where as B decreases so does A. A value of 0 means there is no linear correlation between the vectors.

5.2.2 Non-metric Unlearning

The non-metric unlearning algorithm simply punishes examples when they contribute to an incorrect classification (see Algorithm 3). Every time an example contributes to an incorrect classification, its score (initially 1.0) is reduced. If the score falls below 0.0, then the example is removed from G. This approach allows examples to misclassify a few times before being removed.

Following the strength in numbers heuristic, examples are punished less when there are multiple other examples also misclassifying the corrective example. Accordingly, the score is reduced by the number of misclassifying examples. The parameter $0 < \alpha \leq 1$ is the amount of score reduction when there is just a single misclassifying example: it should be set small enough that a misclassifying example is given a few chances before outright removal. The similarity heuristic is applied indirectly. After adding several corrective points, if a misclassifying example m is distant from other points, it is not likely to effect nearby regions. However, if m is close by, then it might have more impact on classification.

5.2.3 Classifiers

Each of the previous two methods calls a function GATHER-EXAMPLES($\mathbf{C}$, e, ...) whose purpose is to return all examples $m \in M$ responsible for causing the classifier $\mathbf{C}$ to misclassify the corrective

Algorithm 4 GATHER-K-NN-EXAMPLES $(\mathbf{C}, e, \mu)$

```
1: M ← {}
2: class ← K-NN-CLASSIFY(C, e)
3: if class is incorrect then
4:     D ← Plurality of K-NN-CLASSIFY(C, e)
5:     for all d ∈ D do
6:         Q ← Plurality of (K+1)NN-CLASSIFY(C, d)
7:         if ≤ μ members of Q have class class then
8:             M ← M ∪ {d}
9: return M
```

example e (i.e., GATHER-EXAMPLES($\mathbf{C}$, e, ...) constructs M). How this function is implemented varies from classifier to classifier, and this thesis focuses on three different classifiers: decision trees (C4.5), K-Nearest Neighbor (K-NN), and support vector machines (SVMs).

All three implementations work in largely the same way. First, the algorithm determines if e is misclassified by $\mathbf{C}$. Then, if e is indeed misclassified, the algorithm returns, as M, those examples responsible for misclassifying e which were used during the construction of $\mathbf{C}$.

K-Nearest Neighbor In K-Nearest-Neighbor, a point's class is determined by voting among the K nearest examples. From these K points, the algorithm determines D, the plurality of examples which voted for the incorrect class (see Algorithm 4). If e lies near the boundary between two classes, it may be misclassified simply because of sparsity of the space, and the resulting plurality will all be valid examples properly lying on their side of the boundary. Thus, an additional application of K-NN attempts to identify these boundary examples. Specifically for each $d \in D$ we perform a K-Nearest-Neighbor classification on the point represented by d, using a value of K one larger than normally performed in the classifier. If a sufficient number μ of neighbors support the original class of d, then d is likely a border example rather than an isolated and potentially noisy example. All members of D which fail this test are returned.

Decision Trees I modified the standard C4.5 decision tree algorithm to store all the examples used to construct each leaf. The construction of M starts by determining the leaf node L which misclassified e (see Algorithm 5). Since the decision tree might be poorly pruned, L might contain

Algorithm 5 GATHER-DECISION-TREE-EXAMPLES $(\mathbf{C}, e)$

```
M ← {}
class ← DECISION-TREE-CLASSIFY(C, e)
if class is incorrect then
    L ← leaf node in C used to classify e
    if L is the root node then
        P ← L
    else
        P ← parent of L
    D ← examples used to form subtree rooted by P
    for all d ∈ D do
        if class of d matches class then
            M ← M ∪ {d}
return M
```

very few misclassifying examples to form M, so I add all misclassifying examples from the subtree rooted at L's parent.

Support Vector Machines The misclassifying examples in a support vector machine are one or more of the support vectors. However, in a sparse problem there might not be very many support vectors, and, furthermore, there may be other misclassifying examples just outside the SVM's margin which would also misclassify e if certain support vectors were removed. Therefore, I increase the margin by τ (default value of 2) and collect all misclassifying examples within this larger margin (see Algorithm 6). In a sense, this is similar to choosing the parent of the leaf in the previous decision tree algorithm.

Algorithm 6 GATHER-SVM-EXAMPLES $(\mathbf{C}, e, \tau)$

```
M ← {}
class ← SVM-CLASSIFY(C, e)
if class is incorrect then
    w ← width of margin in C
    C' ← C with margin whose width is w × τ
    D ← examples within margin of C'
    for all d ∈ D do
        if class of d matches class then
            M ← M ∪ {d}
return M
```

5.3 Experiments

We conducted experiments to compare the metric and non-metric unlearning methods against simply leaving noisy data in the dataset and relying on the learning algorithm to compensate for it [335]. Additionally, we compared against perfect noise reduction (just eliminate the noisy data). We expected the unlearning methods to perform better than leaving the noisy data in, but worse than perfect noise reduction.

The experimental procedure was as follows. Data was randomly shuffled and split into three pieces: a set of *unchanged* data (70% of the original data), a set of *corrective* data (20% of the original data), and a set of *testing* data (10 % of the original data). Using the corrective data, we generated an equal sized set of *error* examples. Examples in the error set were the same as the corrective set but with the addition of Gaussian noise to the feature vector and a random relabeling to an incorrect label. A classifier was trained using the unchanged and error datasets, and then unlearning was performed using the corrective data before testing for generality using the testing data. This splitting, correction, and testing process was repeated 1000 times for each experiment.

The goal of the experiments was to compare unlearning methods using simple datasets and a variety of classification algorithms, similar to those found in learning from demonstration.

- *Datasets.* Experiments used the Iris, Glass, and Wine datasets from the UCI repository [336].

- *Classification Algorithms.* Experiments used decision trees (C4.5), K-Nearest Neighbor (K-NN), and support vector machines (SVMs). For K-NN, we choose K=1 and K=3. The SVM used a radial basis function kernel with an exponent of 0.5. For decision trees, we examined both unpruned and pruned trees using Pessimistic Error Pruning [52] as the pruning algorithm. While PEP does not have as good performance as many other pruning algorithms in the literature, it has the distinct advantage of not requiring a separate "pruning set" of data, which is important to HiTAB due to the lack of examples (i.e., there are not enough examples to create training, testing, and pruning sets typically required by pruning algorithms).

- *Degree of Noise.* While the *amount* of noise (and corrective data) was fixed to 20% of the original data, the *degree* of noise varied, in terms of the variance of the Gaussian curve. The

variance was $\eta(max - min)$, where *min* and *max* are the minimum and maximum legal values of the feature and η was set to 1/5, 1/20, and 1/100 (other in-between values of η behaved with similar expected results).

- *Data Sparsity.* As already discussed, this thesis focuses learning from demonstration on small datasets. So, while the Wine, Glass, and Iris datasets are already fairly small (between 100 and 250 data points), we experimented with artificially shrinking the datasets by removing random points. In particular, we examined the full ($\omega = 100\%$) data set, an $\omega = 50\%$ sized set, and a $\omega = 25\%$ sized set.

Before conducting the formal experiments, we performed some informal tuning to see how sensitive the unlearning methods are to parameter values. In the non-metric unlearning algorithms, α was varied from 0 to 1 in steps of 0.1, while in the metric unlearning algorithm, γ was varied over 0.5, 0.75, and 1.0 while β ranged from 0 to 1 in steps of 0.1. Additionally, μ was varied from 0 to 5 in steps of 1. In general, γ had little effect on classification accuracy. However, for unpruned decision trees on the Wine dataset, increasing γ resulted in decreased accuracy. Additionally, β appeared to have minimal effect, but this was probably due to the infrequency of the second hypersphere containing a single point. In the non-metric algorithms, accuracy either stayed constant or increased as α increased from 0 to 1. For K-NN, setting $\mu = 2$ resulted in the best performance with no statistically significant impact as μ increased to 5.

Based on these trends, for metric unlearning, we fixed $\gamma = 0.5$ and $\beta = 0.5$, while for non-metric unlearning, we set $\alpha = 0.9$. For K-NN, I set $\mu = 2$, and, as mentioned earlier, for SVMs we set $\tau = 2$. Table 5.1 shows the results. For each algorithm and dataset, we compared against simply adding a point (called the "naive" approach) and running the unlearning algorithms. Bold numbers indicate a statistically significant increases over the naive approach, while underlined numbers indicate the statistically higher performance between the metric and non-metric unlearning algorithms. All statistical tests were at the 95% confidence level with the appropriate Bonferroni correction. In general, the unlearning algorithms performed better than the naive approach, with metric unlearning slightly outperforming non-metric algorithms. However, in all cases, the unlearning algorithms failed to completely remove the error points.

Table 5.1: Results for $\omega = 100\%$. Bold numbers indicate statistically significant difference between the naive approach (U+C+E) and unlearning, while underlined numbers indicate a statistically significant difference between metric and non-metric unlearning. The column U+C represents a perfect dataset and serves as an upper bound on unlearn performance.

	Noise = 1/5				*Noise = 1/20*				*Noise = 1/100*			
Dataset	*U+C*	*U+C+E*	*Metric*	*Non-Metric*	*U+C*	*U+C+E*	*Metric*	*Non-Metric*	*U+C*	*U+C+E*	*Metric*	*Non-Metric*
						1-NN						
Iris	0.9553	0.9131	**0.9307**	**0.9255**	0.9553	0.8002	**0.8901**	**0.8601**	0.9553	0.7519	**0.9461**	**0.8490**
Glass	0.6921	0.6707	0.6810	**0.6822**	0.6921	0.6441	**0.6816**	**0.6705**	0.6921	0.5653	**0.6887**	**0.6421**
Wine	0.9533	0.9370	**0.9464**	0.9442	0.9533	0.7998	**0.9506**	**0.8722**	0.9533	0.7566	**0.9520**	**0.8488**
						3-NN						
Iris	0.9537	0.9409	0.9468	0.9492	0.9537	0.8887	**0.9361**	**0.9295**	0.9537	0.8539	**0.9370**	**0.9331**
Glass	0.7008	0.6734	**0.6895**	**0.6980**	0.7008	0.6615	**0.6927**	**0.6971**	0.7008	0.6193	**0.6866**	**0.6828**
Wine	0.9615	0.9524	**0.9607**	0.9594	0.9615	0.8895	**0.9511**	**0.9472**	0.9615	0.8548	**0.9462**	**0.9408**
						Decision Tree (Unpruned)						
Iris	0.9459	0.8705	**0.8915**	**0.8877**	0.9459	0.8029	**0.8497**	**0.8535**	0.9459	0.8014	**0.8765**	**0.8616**
Glass	0.6701	0.6379	**0.6577**	**0.6572**	0.6701	0.6355	**0.6544**	**0.6514**	0.6701	0.6306	**0.6591**	**0.6492**
Wine	0.9332	0.8321	**0.8638**	**0.8636**	0.9332	0.7375	**0.8103**	**0.7956**	0.9332	0.7206	**0.8365**	**0.8079**
						Decision Tree (Pruned)						
Iris	0.9427	0.9135	0.9213	**0.9226**	0.9427	0.8761	**0.9081**	**0.9094**	0.9427	0.8799	**0.9250**	**0.9213**
Glass	0.6711	0.6330	**0.6520**	**0.6529**	0.6711	0.6274	**0.6460**	**0.6426**	0.6711	0.6301	**0.6501**	**0.6496**
Wine	0.9340	0.8591	**0.8811**	**0.8846**	0.9340	0.8185	**0.8749**	**0.8715**	0.9340	0.8093	**0.8892**	**0.8844**
						Support Vector Machine						
Iris	0.9102	0.3886	0.4280	**0.9070**	0.9102	0.7389	**0.8649**	**0.8705**	0.9102	0.7374	**0.8695**	**0.8668**
Glass	0.3346	0.3311	0.3163	0.3393	0.3346	0.3329	0.3313	0.3284	0.3346	0.3249	0.3259	0.3350
Wine	0.9329	0.3906	0.3991	**0.9350**	0.9329	0.6400	**0.8828**	**0.8861**	0.9329	0.6544	**0.8834**	**0.8867**

The next experiment investigated how these trends held up with smaller datasets by changing the size of the dataset: at the start of each iteration, the dataset was randomly shuffled, and only the top $\omega\%$ was used. Table 5.2 shows the results for $\omega = 50\%$ and Table 5.3 shows the results for $\omega = 25\%$. We saw the same trends as before: unlearning performed better than the naive approach, with metric unlearning performing slightly better than non-metric unlearning. However, as the dataset shrunk, unlearning algorithms started performing closer to the naive approach due to the fewer available points (and associated information) for the unlearning algorithms.

Table 5.2: Results for $\omega = 50\%$. Bold numbers indicate statistically significant difference between the naive approach (U+C+E) and unlearning, while underlined numbers indicate a statistically significant difference between metric and non-metric unlearning. The column U+C represents a perfect dataset and serves as an upper bound on unlearn performance.

	Noise = 1/5				*Noise = 1/20*				*Noise = 1/100*			
Dataset	*U+C*	*U+C+E*	*Metric*	*Non-Metric*	*U+C*	*U+C+E*	*Metric*	*Non-Metric*	*U+C*	*U+C+E*	*Metric*	*Non-Metric*
						1-NN						
Iris	0.9471	0.8982	**0.9176**	**0.9194**	0.9471	0.7839	**0.8969**	**0.8534**	0.9471	0.7458	**0.9445**	**0.8430**
Glass	0.6532	0.6257	0.6437	0.6502	0.6532	0.5911	**0.6345**	**0.6346**	0.6532	0.5383	**0.6543**	**0.6004**
Wine	0.9471	0.9190	**0.9351**	0.9343	0.9471	0.7871	**0.9482**	**0.8724**	0.9471	0.7556	**0.9462**	**0.8546**
						3-NN						
Iris	0.9533	0.9356	0.9470	0.9474	0.9533	0.8744	**0.9323**	**0.9304**	0.9533	0.8435	**0.9313**	**0.9280**
Glass	0.6418	0.6177	0.6374	0.6412	0.6418	0.6052	**0.6412**	**0.6338**	0.6418	0.5721	**0.6358**	**0.6236**
Wine	0.9514	0.9437	0.9509	0.9465	0.9514	0.8814	**0.9377**	**0.9343**	0.9514	0.8530	**0.9373**	**0.9305**
						Decision Tree (Unpruned)						
Iris	0.9396	0.8531	**0.8776**	**0.8765**	0.9396	0.7714	**0.8404**	**0.8337**	0.9396	0.7681	**0.8591**	**0.8370**
Glass	0.6241	0.5875	0.6075	0.6003	0.6241	0.5756	0.5966	**0.5988**	0.6241	0.5695	**0.6034**	**0.5938**
Wine	0.8896	0.8036	**0.8420**	**0.8348**	0.8896	0.7275	**0.7987**	**0.7769**	0.8896	0.7045	**0.8191**	**0.7810**
						Decision Tree (Pruned)						
Iris	0.9421	0.9024	0.9175	**0.9145**	0.9421	0.8634	**0.9096**	**0.9058**	0.9421	0.8512	**0.9190**	**0.9204**
Glass	0.6311	0.5905	0.6094	**0.6093**	0.6311	0.5777	**0.6009**	**0.6040**	0.6311	0.5634	**0.5987**	**0.6006**
Wine	0.8873	0.8324	**0.8621**	**0.8543**	0.8873	0.7906	**0.8480**	**0.8446**	0.8873	0.7722	**0.8483**	**0.8514**
						Support Vector Machine						
Iris	0.6733	0.5751	**0.6700**	**0.6733**	0.6733	0.5235	**0.6350**	**0.6401**	0.6733	0.5321	**0.6424**	**0.6304**
Glass	0.3411	0.3000	0.2833	0.3431	0.3411	0.3056	0.3611	0.3333	0.3411	0.3372	0.3330	0.3280
Wine	0.4557	0.3865	**0.4505**	**0.4973**	0.4557	0.3923	0.4462	**0.4209**	0.4557	0.3873	0.4273	**0.4106**

Table 5.3: Results for $\omega = 25\%$. Bold numbers indicate statistically significant difference between the naive approach (U+C+E) and unlearning, while underlined numbers indicate a statistically significant difference between metric and non-metric unlearning. The column U+C represents a perfect dataset and serves as an upper bound on unlearn performance.

	Noise = 1/5				*Noise = 1/20*				*Noise = 1/100*			
Dataset	*U+C*	*U+C+E*	*Metric*	*Non-Metric*	*U+C*	*U+C+E*	*Metric*	*Non-Metric*	*U+C*	*U+C+E*	*Metric*	*Non-Metric*
						1-NN						
Iris	0.9336	0.8792	**0.9134**	0.9052	0.9336	0.7882	**<u>0.9072</u>**	**0.8438**	0.9336	0.7256	**<u>0.9318</u>**	**0.8294**
Glass	0.6038	0.5892	0.6097	0.5977	0.6038	0.5442	**0.5945**	**0.5997**	0.6038	0.5040	**<u>0.6092</u>**	**0.5650**
Wine	0.9345	0.9045	**0.9227**	0.9222	0.9345	0.7735	**<u>0.9407</u>**	**0.8548**	0.9345	0.7382	**<u>0.9293</u>**	**0.8393**
						3-NN						
Iris	0.9378	0.9202	0.9354	0.9314	0.9378	0.8474	**0.9174**	**0.9238**	0.9378	0.8328	**0.9182**	**0.9232**
Glass	0.5897	0.5755	0.5950	0.5927	0.5897	0.5577	**0.5953**	**0.5953**	0.5897	0.5172	**0.5772**	**0.5768**
Wine	0.9465	0.9273	0.9330	0.9417	0.9465	0.8560	**0.9240**	**0.9270**	0.9465	0.8462	**0.9228**	**0.9203**
						Decision Tree (Unpruned)						
Iris	0.9350	0.8410	**0.8758**	0.8602	0.9350	0.7552	**0.8354**	**0.8180**	0.9350	0.7296	**0.8392**	**0.8250**
Glass	0.5947	0.5177	0.5400	0.5662	0.5947	0.5083	**0.5483**	0.5302	0.5947	0.4865	**0.5360**	0.5297
Wine	0.8503	0.7687	**0.8068**	**0.8025**	0.8503	0.6875	**0.7755**	**0.7565**	0.8503	0.6845	**<u>0.7900</u>**	**0.7578**
						Decision Tree (Pruned)						
Iris	0.9326	0.8700	**0.9016**	**0.8958**	0.9326	0.8064	**0.8790**	**0.8838**	0.9326	0.8144	**0.9040**	**0.8972**
Glass	0.5637	0.5268	0.5552	0.5548	0.5637	0.5182	0.5488	**0.5488**	0.5637	0.5013	**0.5438**	0.5440
Wine	0.8415	0.7880	**0.8215**	**0.8248**	0.8415	0.7292	**0.8043**	**0.7958**	0.8415	0.7362	**0.8140**	**0.8087**
						Support Vector Machine						
Iris	0.4682	0.3853	**0.4389**	**0.4556**	0.4682	0.3772	0.4124	**0.4220**	0.4682	0.3750	**0.4164**	**0.4226**
Glass	0.3422	0.3375	0.3403	0.3447	0.3422	0.4500	0.4000	0.3278	0.3422	0.3366	0.3399	0.3397
Wine	0.3905	0.3618	0.3776	0.3968	0.3905	0.3945	0.3923	0.3888	0.3905	0.3832	0.3904	0.3823

Chapter 6: Flat Multi-Agent HiTAB

Multiagent learning from demonstration is difficult for two primary reasons inherent to cooperative multiagent learning. The first difficulty relates to design: a multiagent system consists of many agents, complex inter-agent interactions (including unforeseen emergent phenomena), and complex internal behaviors and features. Additionally, when one agent is trained, its behaviors and features *depend* on other agents within the system. This results in a high-dimensional space requiring many samples to adequately learn, thus, making rapid training difficult.

The second difficulty is a daunting inverse problem. In supervised training, the trainer only knows the high level macro-behaviors the group should perform. But, the individual agents need to know which micro-level behaviors to perform to accomplish the macro-level task[1]. While we might have a function $f(\ldots)$ which translates micro-behaviors to macro-behaviors (e.g., a robot simulator), we do not have the inverse function $f^{-1}(\ldots)$ telling us which micro-behaviors are necessary to produce the desired macro-behavior. The gulf between the micro and macro is just too great. Thus, standard solutions rely on optimization of $f(\ldots)$ via reinforcement learning and other stochastic optimization methods, which, again, require a significant number of examples.

This chapter and the next chapter present two methods for reducing the gulf between macro- and micro-level behaviors. Following the classification scheme developed by Panait and Luke [104], the first technique, *behavior bootstrapping*, is considered *concurrent learning* since multiple classifiers are learning to work together in a coordinated fashion. The second technique, *agent hierarchies* is a *team learner* since a single classifier (at the controller agent) is learning to control all the subsidiary agents.

[1] Micro-level behaviors are those performed by a single agent (e.g., go forward, turn), while macro-level behaviors are those desired by the entire system (e.g., forage, play soccer).

6.1 Training Multiple Agents

I see three different ways to train multiple agents using HiTAB:

- A single agent is trained in isolation, and the trained behavior is the distributed to other agents in the system. The behavior does not require agent interaction, and can essentially be done in parallel.

- A homogeneous behavior is trained to be used by multiple coordinated agents. As discussed above, training this type of behavior is difficult. Instead of training multiple agents simultaneously, I developed a new approach called *behavior bootstrapping* which iteratively trains one agent in the context of the others, then transfers the trained behavior to other agents. To wit, an agent is initially trained to perform a rudimentary version of the desired behavior in the context of do-nothing teammates. The rudimentary behavior is then distributed to all other agents, and another randomly chosen agent is trained on a slightly more capable behavior in the context of the teammates performing the rudimentary behavior. The slightly more capable behavior is then distributed to the other agents, and the process continues until the final desired behavior is distributed to all the agents.

- A collection of coordinated behaviors is training among a group of agents. The agents are organized into a command hierarchy: small groups of agents are collected beneath a controller agent; then small groups of controllers are organized under another controller, and so on until the all agents are arranged into a tree-structure. Chapter 7 discusses this approach is more detail.

This chapter focuses on behavior bootstrapping to train a small group of agents, but other possibilities exist to train multiple agents without using a hierarchy. One approach could use the notion of a *dummy* agent: one agent is trained as normal, while the other agent(s) are tele-operated. While a simple approach, this approach requires multiple humans to tele-operate the other agent(s), and intuitively does not feel like an appropriate match for multiagent training. Another second approach would be to use one HiTAB training session per agent, training all the agents simultaneously, thus,

relying on the demonstrators for coordination (see discussion on page 24). A final approach would let each agent know the current top-level behavior of all other agents within the system. To achieve this, each agents feature vector changes from $\vec{f}_t = \langle f_1, ..., f_m \rangle$ to $\vec{f}_t = \langle f_1, ..., f_m, b_1, b_2, \ldots, b_n \rangle$ where each b_i is the top-level behavior currently being performed by agent i [337].

Training with Behavior Bootstrapping Training multiple agents with behavior bootstrapping is similar to training a single agent (see "Training with HiTAB" on page 64). The primary difference is waiting for the non-training agents to be in a relevant state before collecting a new example: when the tuple $\langle S_t, \vec{f}_t, S_{t+1} \rangle$ is stored, the other agents must be in an appropriate state so that the feature vector $\vec{f}_t$ contains information about the coordination between the training agent and other agent(s) in the system. Like the single agent case, if the behaviors associated with S_{t+1} is designed to be executed more than once, an additional tuple $\langle S_{t+1}, \vec{f}_t, S_{t+1} \rangle$ is saved.

6.2 Behavior Bootstrapping Experiments

During preparation for RoboCup 2012 (see previous chapter), I applied HiTAB to the task of soccer keepaway in the RoboCup Soccer Simulator. In the keepaway problem, a team of *keepers* tries to maintain possession of the ball for as long as possible from a team of *takers*. The two teams compete in a bounded area (a 20m $\times$ 20m box) within a regular soccer field in the RoboCup Soccer Simulator. For these experiments, the agents have 360 degree and infinite view, cannot collide with the ball, and the keepers do not dribble.

The keepaway problem presents several challenges. First, its limited inter-agent communication requires agents to learn independently, but the resulting behaviors require coordination. Second, keepaway (and soccer in general) has a large state space usually requiring a large number of samples to learn. Third, the RoboCup Soccer Simulator injects random noise in agents' actions and sensors.

I used HiTAB plus behavior bootstrapping to learn coordinated behaviors for a 3v2 keepaway problem: three keepers versus two takers [338]. The takers behavior was hard-coded to simply go towards the ball as fast as possible, while the keeper's behavior was manually decomposed into a

structure similar to Stone et al [339]. The keepers were provided with the following hard-coded basic behaviors: **GoFast, GoMedium, GoSlow, Stop, Pass, GetOpen,** and **TurnToBall.**

- The **GoFast, GoMedium,** and **GoSlow** behaviors 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 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.

Four automata were trained based on these basic behaviors and features (Figure 6.1):

- APPROACHBALL: A P-Controller in automaton form, based on **GoFast, GoMedium, GoSlow** and **Stop**, along with *DistanceTo(ball)*. This automaton attempted to move the agent until it was within kicking distance of the ball location.

- GOTOBALL: Iterated between APPROACHBALL and **TurnToBall**, using *DirectionTo(ball)*. This automaton attempted to servo on the ball location, and did not require state (i.e., the keeper did not attempt to turn the "correct" direction if it momentary lost sight of the ball).

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

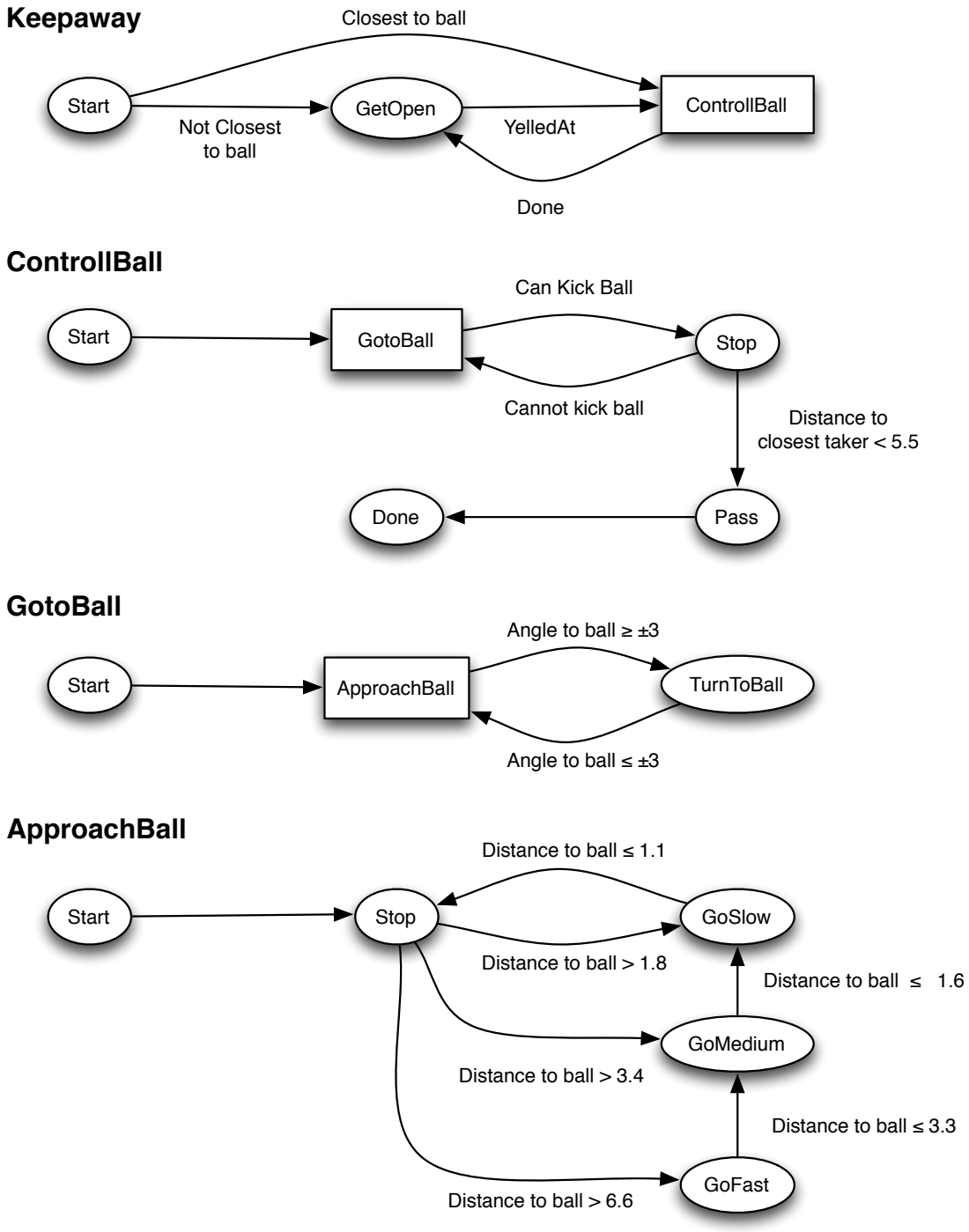


Figure 6.1: Four automata trained for the Keepaway Problem. In each case, the automaton begins in *Start*. The *Done* behavior does nothing but raises a *done* flag in the automaton's parent, which is detected by the *done* feature (compare ControlBall with Keepaway). Real-valued numbers shown are the result of the training examples provided.

- CONTROLBALL: Used the GOTOBALL, **Stop**, and **Pass** behaviors, the **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.

- 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.

Training began by training a single agent to either CONTROLLBALL or **GetOpen** when started. This behavior was then distributed to the other keepers. Upon restarting the game, which caused one agent to CONTROLLBALL while the other two keepers performed **GetOpen**, the ball-controlling agent was trained to pass the ball and then get open. After distributing this behavior to all three agents and restarting the game, one of the open agents was trained to transition to CONTROLLBALL when yelled at. This final trained behavior was distributed to all agents.

The experiments ran KEEPAWAY for 200 episodes, where an episode started at time 0 and ended when either the takers gained possession or the ball was kick out of the keepaway box. The takers were from [339], but running at one quarter the speed of the original. All experiments were conducted using the MASON multiagent simulation package [297] (running HiTAB) plus the RoboCup Soccer Simulator.

The trained keepers maintained possession for an average of 14.6 ± 0.87 seconds, and completed an average of 3.8 passes per episode. While the keepers played successfully, limiting the speed of the takers is an obvious weakness. The takers were slowed to ease the collection of training examples (the takers were too quick to setup the necessary situations for behavior bootstrapping); the next set of experiments addressed this weakness.

Table 6.1 shows the length of time spent actually training the agents (including collecting the samples and constructing the HFAs), 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

Table 6.1: Number of data points to train the automata, and an approximation of the total time to train the automata.

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

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.

The above experiments did not perform well compared to hand-coded solutions. By training behaviors such as APPROACHBALL and GOTOBALL, cycles in the soccer simulator were wasted. Rather than using a single cycle to gather information, and make an intelligent decision about which action to take, these behaviors required several cycles to act resulting in slow moving behaviors (hence, why the takers were slowed).

To rectify this, I re-did the experiments, but using the same basic behaviors as the University of Texas code. These behaviors do not waste simulator cycles since they gather information and make a decision in a single cycle. Also, these behaviors use intelligent prediction of future events to make more accurate behaviors. I used the following basic behaviors:

- **GetOpen** Move to a position within the keepaway box that is free from defenders and open to receive a pass from the ball's current location. This behavior is based on Strategic Positioning with Attraction and Repulsion (SPAR) [340] which uses potential fields to maximize the repulsion from defenders and maximize the attraction to the ball.

- **Freeze** Stops a moving ball by kicking it with a velocity equal to the incoming velocity and in the opposite direction of the ball's direction.

- **HoldBall** Holds the ball close to the body. If defenders are close, then the ball is moved to the most difficult location for the defender to tackle.

- **PassAndYell** This behavior consists of three steps. First, the receiving player is determined by assigning an "openness" score to each teammate. Each teammate's score is based on the distance to the nearest defender, the angle between the teammate and defender closest to the passing line [339]. Second, the agent "yells" at the receiver by sending a communication message. Finally, the agent passes the ball to the receiver with enough velocity for the ball to stop near the receiver.

- **Receive** Stops the agent, and turns towards the ball.

- **Intercept** Moves the agent such that the ball in within kickable range as soon as possible.

The features changed to *YelledAt, ClosestToBall, DistanceToBall, DefenderClose,* and *PossessBall*. Similar to the previous experiments, the binary features *YelledAt, ClosestToBall, DefenderClose,* and *PossessBall* were true when the keeper was yelled at, the keeper was closest to the ball, a taker was within 4 meters of the keeper[3], and the keeper possessed the ball.

Using these basic behaviors and features, I trained the following automata in order (Figure 6.2):

- FREEZEANDHOLD: Using the behaviors **Freeze, HoldBall, Done** and **Fail** the keeper first freezes the ball, and then if *DefenderClose* is true, transitions to **Done**. Otherwise, the keeper performs **HoldBall** until *DefenderClose* is true. If the keeper looses possession of the ball, then it transitions into **Fail**.

- HOLDANDPASS: Combining the basic behavior **PassAndYell** with the previously trained FREEZEANDHOLD, this behavior either passes the ball if *DefenderClose* is true, or performs FREEZEANDHOLD if *DefenderClose* is false. The keeper transitions to **Fail** it if loses possession, and transitions to **Done** after passing the ball.

- GETBALL: This behavior causes the keeper to move towards the ball and then maintain possession. If the keeper is not the closest to the ball, it transitions to **Done**. Otherwise,

[3]Four meters was chosen to match [339] and the University of Texas code-base.

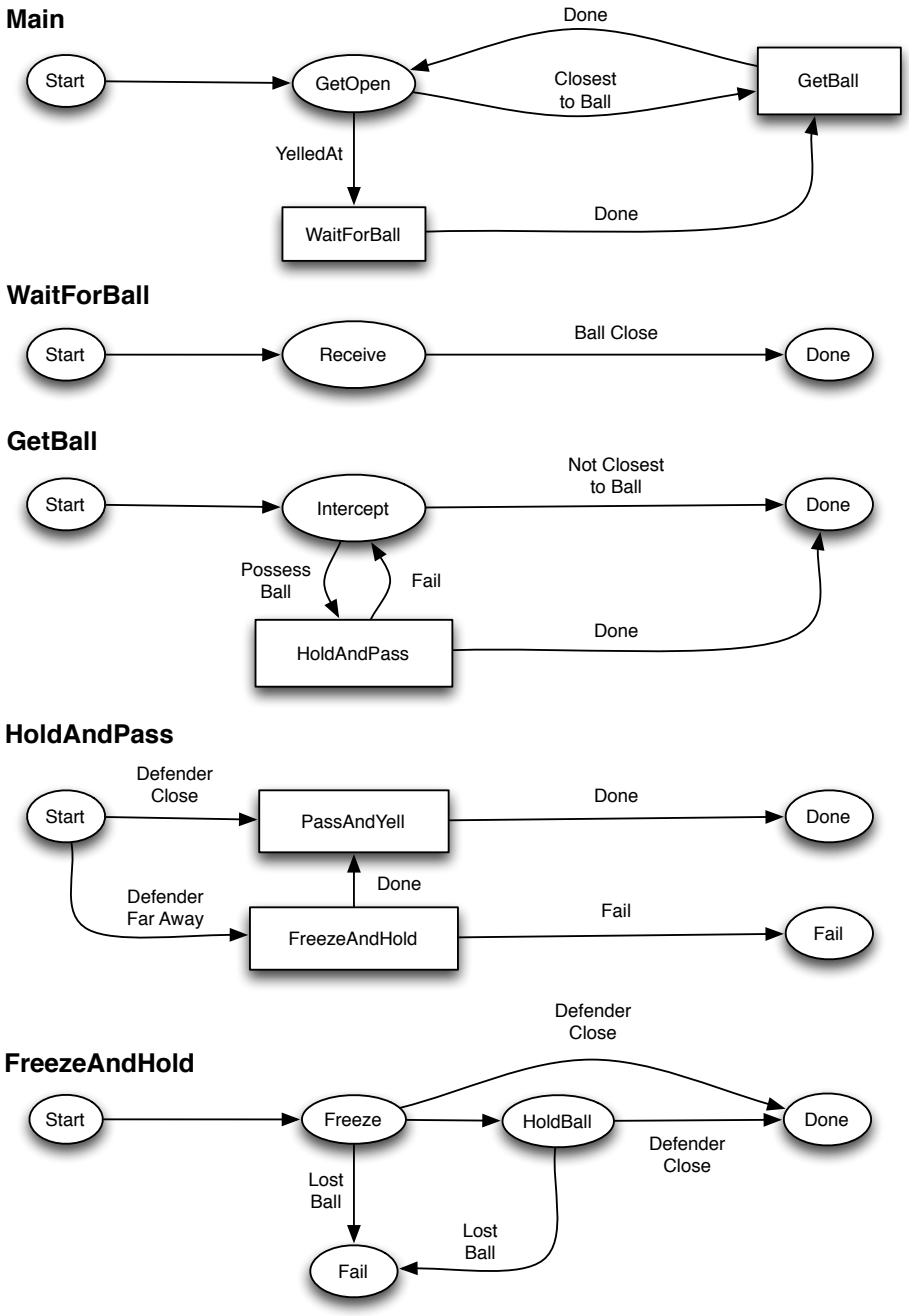


Figure 6.2: Advanced keepaway HFAs

it perfumes the basic behavior **Intercept** to move towards the ball, and once in possession, executes the previously trained HOLDANDPASS to maintain possession.

- WAITFORBALL: This behavior is a wrapper behavior used to prepare a keeper to receive a ball: the keeper performs the basic behavior **Receive** until *DistanceToBall* was less than 6 meters.

- KEEPAWAY: The high-level keepaway behavior causes the keeper to either **GetOpen**, or if it is closest to the ball, perform GETBALL. Additionally, if the keeper was *YelledAt* by a teammate, perform WAITFORBALL.

Similar to the earlier experiments, the top-level automaton, KEEPWAY was trained using behavior bootstrapping. After training a single agent to transition between **GetOpen** and GETBALL, this behavior was copied to the other keepers, and after restarting the game, one open agent was trained to perform WAITFORBALL when *YelledAt*. The final trained behavior was distributed to all keepers.

Table 6.2 shows the number of examples necessary to train each HFA which composed the KEEPAWAY automaton. Like before, the number of examples required to train the complex KEEPAWAY behavior is small. Each HFA typically required several iterations, and the final KEEPAWAY automaton took the longest to train due to the behavior bootstrapping.

Both experiments required approximately the same number of examples, but the second experiment was clearly superior in perforce due to the inclusion of more expert knowledge. Again, we see that HiTAB is closer to programming by demonstration rather than machine learning, but this is what allows HiTAB to make progress on supervised multiagent training.

Table 6.2: Number of data points to train the final behaviors for the new keepaway behaviors.

Behaviors	Number of Examples
FreezeAndHold	11
HoldAndPass	9
GetBall	11
WaitForBall	3
Keepaway	8

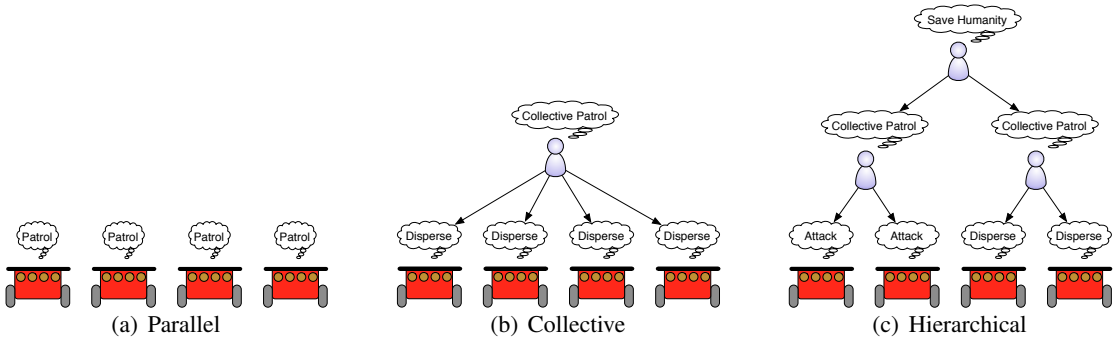
Chapter 7: Hierarchical Multi-Agent HiTAB

As mentioned in the previous chapter, training multiagent systems is challenging due to the large number of samples required to bridge the gulf between micro- and macro-level behaviors. This chapter presents *agent hierarchies* as a structured way to reduce the number of training examples in a multiagent problem: agent hierarchies arrange the agents in a tree structure to reduce the number of agent-agent interactions. Hierarchies are a natural fit for organizing heterogeneous agent swarms, but counterintuitively they're also useful for swarms of agents with homogeneous behaviors too: I train hierarchies of behaviors to control *homogeneous* agents, and demonstrate trained behaviors which are superior to those found in flat ("swarm"-style) structures.

7.1 Agent Hierarchies and Controller Agents

Multiagent systems are characterized by explicit coordination between agents. To rapidly train coordination, we organized the agents into an *agent hierarchy*[1]: a tree structure where leaf nodes are the individual agents or robots performing tasks, and non-leaf nodes are (possibly virtual) *controller agents* [341]. This tree-structured organization fits between fully decentralized ("swarm"-style) multiagent systems and fully centralized systems. While the tree structure has obvious disadvantages (e.g., it is not robust to agent failure), it has one overriding scaling advantage: regardless of its size, at any position in the structure an agent must deal only with a fixed number of agents (his superior and immediate subordinates). I take advantage of this to make the multiagent training task feasible regardless the number of agents. After training the leaf nodes, at all times training consists of only a controller agent and its immediate subordinates: the micro-to-macro gulf is much smaller and simpler with, say, five or fewer agents than it it is with hundreds or thousands of agents. Furthermore, using HFAs at the controller level permits decomposition of complex team behaviors into simpler

[1] Not to be confused with the agent's behavior hierarchy.



(a) Parallel (b) Collective (c) Hierarchical

Figure 7.1: 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 controller agent. (C) Squads in the team are directed by different controller agents, whose behaviors are the same but may all be switched by a higher-level controller agent (and so on).

behaviors in much the same way that behaviors were decomposed in the single agent HiTAB case. Thus, HiTAB can rapidly train complex team behaviors.

Agent hierarchies suggest three different notions of "homogeneous" behaviors as shown in Figure 7.1. First, all agents perform the same HFA independent of each other, thus, relying on random interactions for multiagent behaviors. Second, via its own HFA, a single controller agent controls *which* HFA its subordinate agents are performing. Coordination continues up through the tree: higher level controller agents dictate their subordinates choice of HFAs.

The subsidiary agents all have the same behaviors in their libraries; but the controller agent has its own separate library of behaviors, both basic behaviors and learned automata. In the homogeneous case, a controller agent's basic behaviors generally do not manipulate the controller, but instead correspond to a unique behavior in the libraries of the subsidiaries.[2] When a coordinator agent transitions to a new behavior, it causes all its subsidiary agents to immediately change behavior, independent of the subsidiary agent's local information (see Figure 7.2). The ability to change the behavior of the subsidiary agents facilities coordination since the controller agent's behavior is based on global information with minimal communication overhead.

While the basic behaviors for a controller agent are straightforward, what is a controller agent's

[2]There is no restriction that the controller agent cannot have basic behaviors which do manipulate the controller agent ala the basic agents.

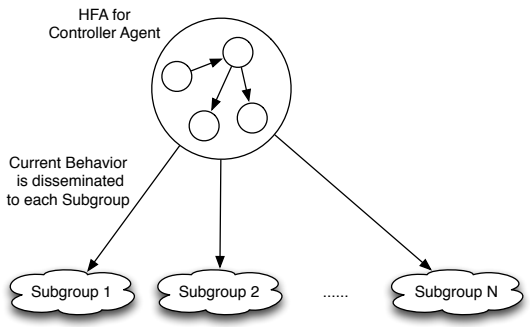


Figure 7.2: Homogeneous multiagent HiTAB model.

set of features? I presume that, unlike a basic agent, a controller agent isn't embodied: its features are derived from statistical results from its subsidiaries: for example "a basic agent in my group is stuck (or isn't)", or "all my immediate subsidiaries are 'done' (or not)", or "the average Y position of basic agents in my group." Typically a controller agent only accesses its immediate subsidiary agents, but there are no restrictions as to how deep in the hierarchy the controller agent can gather information. Like an agent's basic features, the choice of features available to a controller agent are domain-specific. In general, the controller agent could be embodied and posses features which are local to the controller agent.

A homogeneous agent hierarchy is constructed bottom-up: basic agents are gathered into subgroups, each headed by a Level-1 controller agent. Then, multiple Level-1 controller agents are grouped together under Level-2 controller agents, and so on up to Level-M controller agents forming one or more roots. Just as the basic agents all have the same behaviors, controller agents at a given level have the same behaviors. The agent hierarchy structure (number of levels, number of agents per controller, etc.) is currently set a priori by the experimenter and remains static throughout the experiment (i.e., agents cannot move between controller agents).

The agents within the hierarchy are trained starting at the leaf nodes, and working towards the root node one level at a time. A basic agent is trained in the usual fashion, and the learned automata is copied to all other basic agents. Then, a single Level-1 controller agent is trained, and the learned automata is copied to all other Level-1 controller agents. Training continues until Level-M controller

agents are trained. This training process constructs a homogeneous agent hierarchy: all agents at a given level run the same HFA, but may be in different states based on local information.

7.2 Formal Model

The formal model for agent hierarchies is similar to the single agent formal model on page 67, but the behaviors and features are different for the controller agents. The set of states S, transition function T, and the optional free variables G are the same as the in the single agent case.

- $A = \{a_1, a_2 \ldots a_n\}$ is the set of the controller agent's subsidiary agents which could be either other controller agents, or basic agents.

- $B = \{B(A)_1, B(A)_2 \ldots B(A)_k, B_{k+1}, B_{k+2}, \ldots B_l\}$ is the set of of basic behaviors for the controller agent. The first k behaviors are wrappers around the subsidiary agent's behaviors, i.e., $B(A)_i$ causes all subsidiary agents to perform the same behavior (say, "Turn left"). The remaining $l-k$ behaviors are the controller agent's basic behaviors which operate exactly the same as a basic agent's basic behaviors. There is no requirement that the controller agent has its own basic behaviors; thus $l = k$ is permitted.

- $F = \{F(A)_1, F(A)_2, \ldots F(A)_m, F_{m+1}, F_{m+2}, \ldots, F_j\}$ is the set of features for the controller agent. The first m features depend on the set of subsidiary agents, while the remaining $j-m$ features are local features to the controller agent. Like the behaviors, there is no requirement that the controller agent has its own features; thus $j = m$ is permitted.

7.3 Robot Demonstration

I used agent hierarchies to train a simple hierarchy of four Pioneer robots under the control of a single controller agent [341]: the group performed a pursuit task while deferring to and avoiding a "boss." Each robot had a color camera and sonar, and was marked with colored paper. The boss, intruders to pursue, and a home base were also marked with paper of different colors. See Figure 7.3.

Figure 7.3: Learned multi-robot behavior. Demonstrator is holding a green ("intruder") target.

The pursuit task was as follows. Ordinarily all robots 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 SCATTER 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, multiple agents, and learned hierarchical behaviors for a controller agent.

Each robot 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 *Left(C), FarLeft(C), Right(C),* and *FarRight(C)* which correspond to if a color *C* is to the left, far left, right, or far right based on the X-coordinate of the color blob; and the

Size(C) of a blob of color *C* in the environment (we provided four colors as targets to these these features, corresponding to *Teammates*, *Intruders*, the *Boss*, and the *Home Base*). Each robot was dressed in the *Teammate* color.

Initial training produced various small parameterized HFAs, as detailed in Figure 7.4. Note that the SERVO and SCATTER HFAs are stateful: when the target disappeared, the robot had to discern which direction it had gone and turn appropriately. Since HiTAB has only one behavior per state, the trivial HFAs (subfigures 3A through 3D) allow multiple states with the same behavior. Training these behaviors required approximately 30 minutes.

The first experiment used the "basic" homogeneous behavior approach as detailed in Figure 7.1(A): each robot simply performed the same top-level behavior but without any controller agent. This top-level behavior was PATROL (Figure 7.5, Subfigure 8), and iterated through the three previously described states: dispersing through the environment, attacking intruders, and returning to the home base. Deferral to the "boss" was not included at this point.

Simple homogeneous coordination like this was insufficient. In this simple configuration, when a robot 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 a robot 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 controller agents, and so the next experiment placed the four robots under a single virtual controller agent that would choose the top-level behavior each robot would perform at a given time. The controller was trained to follow the COLLECTIVEPATROL behavior shown in Figure 7.5. 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.

The controller agent had three simple features: whether any robot had seen the Intruder's color; whether any robot was *Done*, and whether all robots were *Done*. A simple hierarchical behavior on

the controller agent, called COLLECTIVEPATROLANDDEFER (Subfigure 10), handled deferring to the "boss". This required a new statistical feature for the controller agent: whether anyone had seen the Boss color within the last 10 seconds. The controller agent would perform COLLECTIVEPATROL until someone had seen the Boss within the last 10 seconds, at which point the controller agent would switch to the SCATTER 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 controller would resume the COLLECTIVEPATROL behavior (Figure 7.3).

This is a reasonably comprehensive team behavior, with a large non-decomposed finite-state automaton, spanning across four different robots acting in sync. The agent hierarchy and behavior decomposition allowed training this behavior in real-time: there are simply too many states and aliased states, too many features, and too many transition conditions. The decomposition into simple, easily trained behaviors with a small numbers of features and states allowed rapid training. Additionally, the agent hierarchy allowed training of explicit coordination and the sharing of information among the four robots.

7.4 Box Foraging Experiments

The above robot demonstration showed the benefit of controller agents but did not quantify the benefit. To quantify the benefit of controller agents, I devised a simulated box foraging problem which requires both a behavior hierarchy and agent hierarchy to maximize performance [341]. Homogeneous agents were trained to hunt for boxes and pull them to a known deposit location. The boxes are randomly distributed throughout the environment, and after collection at the deposit location, the box disappears and a new box is placed randomly in the environment. The environment is obstacle-free and contains various circular "boxes" of different sizes, which likewise require different numbers of agents (5, 25, 125) to pull them. Figure 7.6 shows the model in action.

The experiments followed the notions of homogeneity in Figure 7.1: *swarms* of independent agents, *groups* of agents under a single controller agent, and groups of agents under multiple layers of controllers (Level-1, Level-2 and so on). Performing these experiments required training three kinds of behaviors. First, basic agents were trained as usual, then Level-1 controller agents were

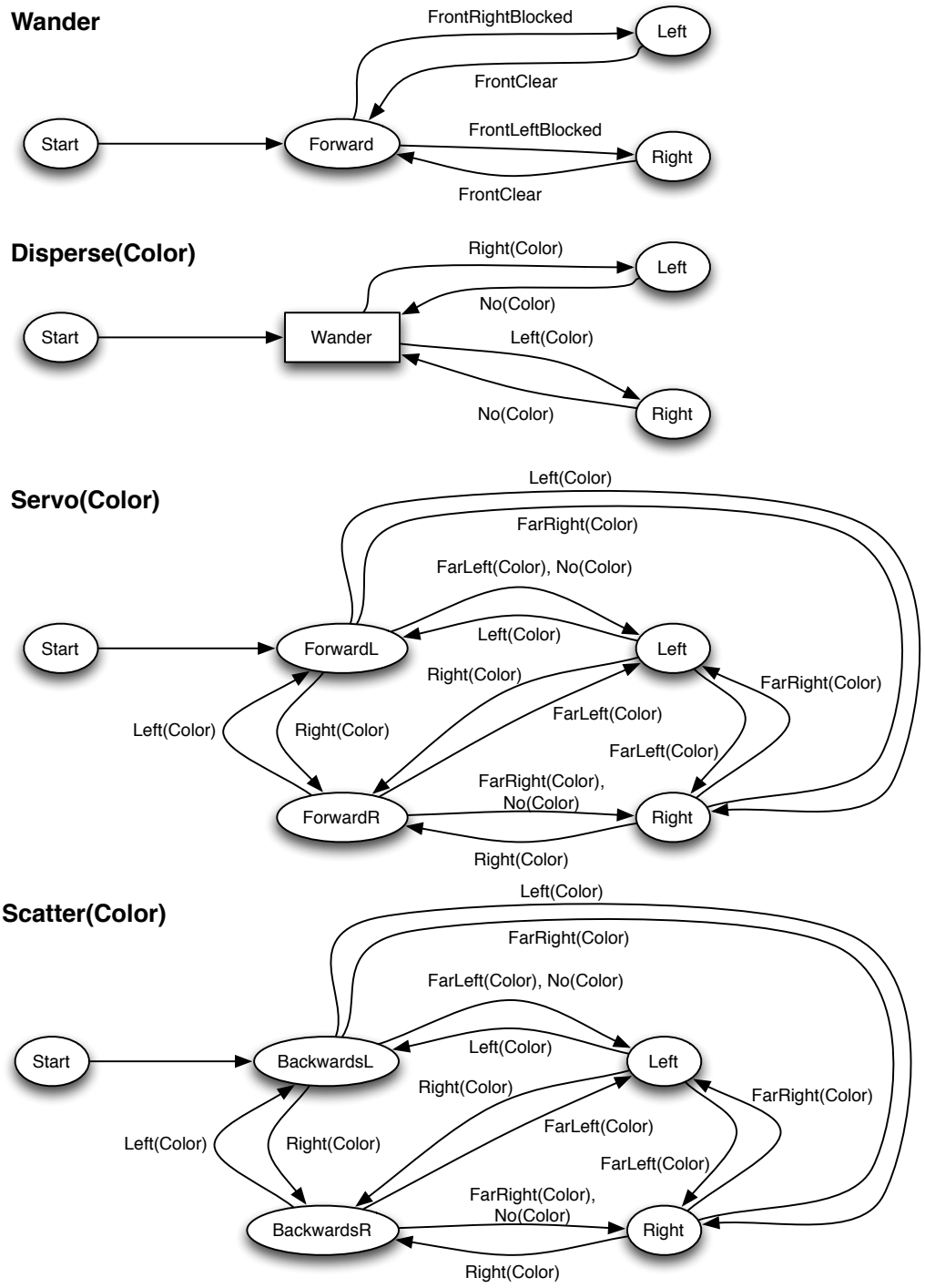


Figure 7.4: Decomposed hierarchical finite-state automaton learned in the robot demonstration (Part I). These behaviors are within an individual robot.

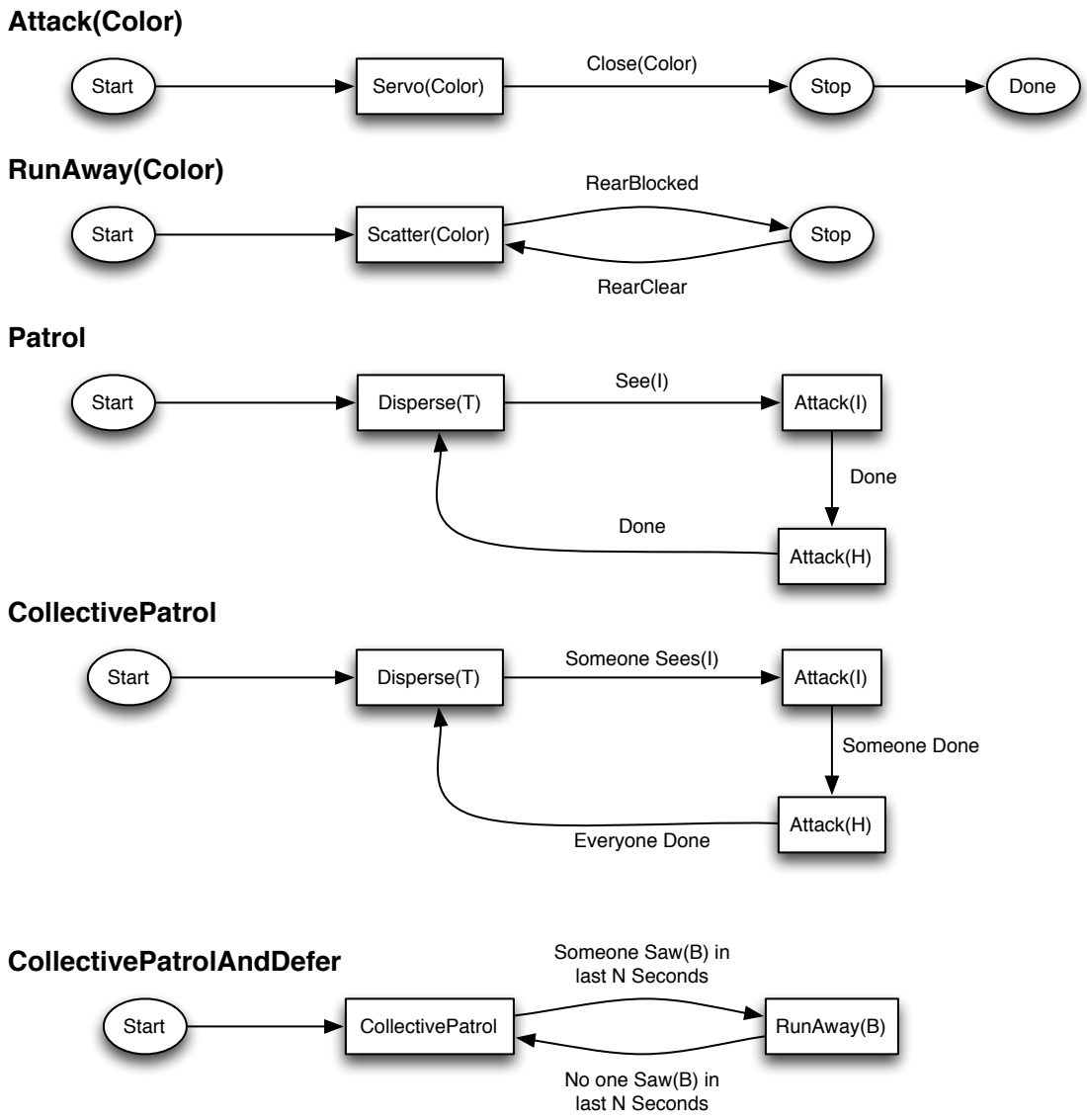


Figure 7.5: Decomposed hierarchical finite-state automaton learned in the robot demonstration (Part II). Most behaviors form a hierarchy within an individual robot, but *CollectivePatrol* and *CollectivePatrolAndDefer* form a separate hierarchy within the controller agent.

trained, and, finally, Level-$N \geq 2$ were trained. This set of behaviors was sufficient to scale to any number of levels.

The basic agents behaviors were **Forward, RotateLeft, RotateRight, GrabBox, ReleaseBox, ReleaseBoxAndFinish,** and **Done**. Both **ReleaseBox** and **Done** would raise the *done* flag and (as normal) immediately transfer to the start state. **ReleaseBoxAndFinish** would as well, except that it would also raise a *finished* flag in the agent which could be detected by controllers as a feature. Boxes could only be grabbed if they were sufficiently close (5 units). The basic agents features were *DistanceTo(X), DirectionTo(X), ICanSeeABox, IAmAttachedToABox,* and *Done*. The first two features were parameterizable to either visible boxes or to the deposit location. The last feature was true when the *done* flag had been raised. Boxes could only be seen if they were within 10 units. With these basic behaviors and features, I trained the following HFAs:

- WANDER uses **Forward, RotateLeft**, and **RotateRight** to wander randomly. The randomness exploits the probabilistic leaf nodes within the decision tree: the decision trees consists of a single leaf node with a distribution of the three basic behaviors, with a bias to **Forward**.

- GOTO(X) servos towards a given target using **Forward, RotateLeft,** and **RotateRight**, plus the *DistanceTo(X)* and *DirectionTo(X)* features.

- RETURNWITHBOX uses GOTO(X), **GrabBox**, **ReleaseBoxAndFinish**, and *DistanceTo(X)* to pull a box back to the deposit location and released when the agent was close enough to home.

- The top-level automaton, FORAGE uses WANDER and RETURNWITHBOX to forage for boxes and bring them back to the deposit location.

If agents were acting on their own (they had no controller), their top-level behavior would be simply FORAGE. When acting under a Level-1 controller, the current behavior of the agent would be determined by the controller. Training the agent's behavior required approximately 30 – 40 minutes.

After training the basic agents automaton, I next trained Level-1 controller agents using the features *SomeoneIsFinished* and *SomeoneIsAttachedToABox*. The latter feature was true if any subsidiary agent had raised its *finished* flag. A controller also had access to an additional target:

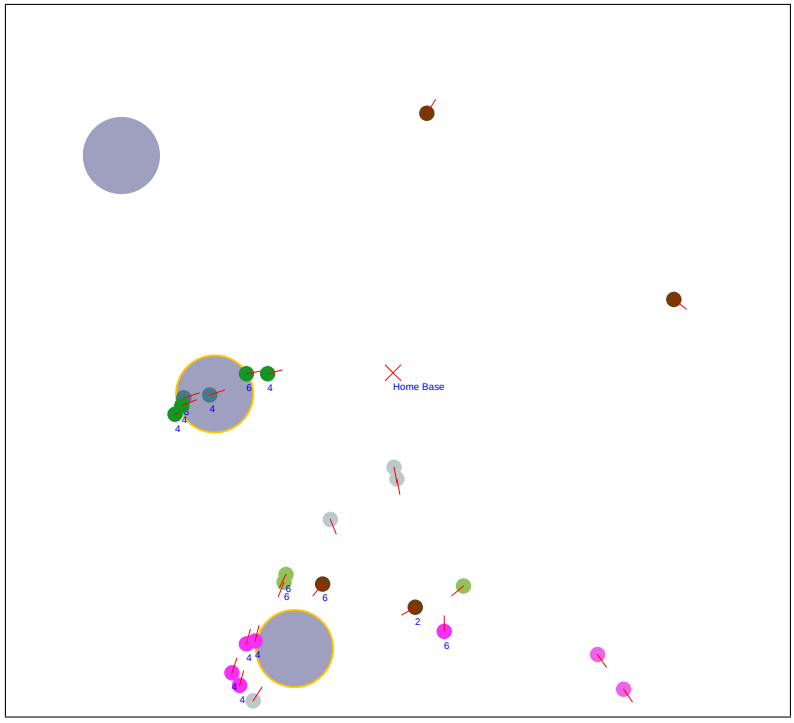


Figure 7.6: A screenshot of the box foraging simulation in action (showing part of the environment). The large grey circles are the boxes, and the X in the middle is the collection location. Note that while the agents pulling the box on left are all from the same subgroup, the box in the bottom is being pulled by agents from different subgroups.

closest-attached-agent, which pointed to the subsidiary agent which had grabbed the box (if any). The controller agent's basic behaviors correspond to full set of behaviors of its subsidiary agents: **Forward, RotateLeft, RotateRight, GrabBox, ReleaseBoxAndDone, Done, Wander, Goto(X), ReturnWithBox,** and **Forage**.

With these basic behaviors and features, I trained the Level-1 controller agents top-level automaton CONTROLFORAGE: using **Goto(closest-attached-agent)**, **ReleaseBox**, **ReturnWithBox**, **Forage**, *SomeoneIsAttachedToABox*, and *SomeoneIsFinished*, CONTROLFORAGE subsidiary agent to FORAGE until an agent finds a box. Then, the controller directs the subsidiary agents to *Goto(closest-attached-agent)*; once agents where close to the attached agent, they would grab the box and begin pulling it towards the deposit location. Once one agent finished pulling the box, the controller would direct the other agents to **ReleaseBox**, and to resume FORAGE.

If there are no Level-2 controller agents, the Level-1 controller agents used CONTROLFORAGE as their top-level automaton. When acting under a Level-2 controller, the current behavior of the Level-1 controllers would be determined by their Level-2 controllers. Training Level-1 controller agents required a few minutes.

All controller agents at levels ≥ 2 used exactly the same behavior hierarchy, which was:

- A Level N$\geq$2 controller agent's basic features were *SomeoneIsFinished* and *SomeoneNeedsHelp* which is true when a subsidiary agent knows of a box which requires more agents to push it than are available to the controller agent. A Level N$\geq$2 controller also had an additional target: *biggest-attached-agent*, which is the agent attached to the largest box that the N$\geq$2 controller "knows about". The controller would learn of such boxes from its superior, or from subsidiary controllers unable to manage the box themselves.

- A Level N$\geq$2 controller agent's basic behaviors corresponded to behaviors from its subsidiary controllers: *ControlForage*, *ReturnWithBox*, *Goto(X)*.

- Level N$\geq$2 CONTROLFORAGE was similar to the Level-1 CONTROLFORAGE behavior. The difference is that the Level N$\geq$2 behavior directs agents to **Goto(biggest-attached-agent)** when a lower level controller requires help. Again, the trivially trained RETURNWITHBOX and GOTO(X) behaviors simply called their corresponding basic behaviors. Training Level N $\geq$ 2 controller agents required a few minutes for each level.

A Level N$\geq$2 controller's top behavior was *ControlForage*.

The first two experiments considered three hierarchical structures: (1) 50 independent agents (2) ten independent Level-1 controller agents, each heading a five-agent subgroup (3) two independent Level 2 controller agents, each heading five Level-1 controller agents, each heading a five-agent subgroup. The first experiment sought to demonstrate that a simple hierarchy can out-perform a group of independent agents; and additionally, that the trained behaviors perform similarly to fine-tuned hand-coded behaviors. The second experiment sought to demonstrate that a two-layer hierarchy outperforms a one-layer hierarchy. In these experiments, the environment was 200×200 units and

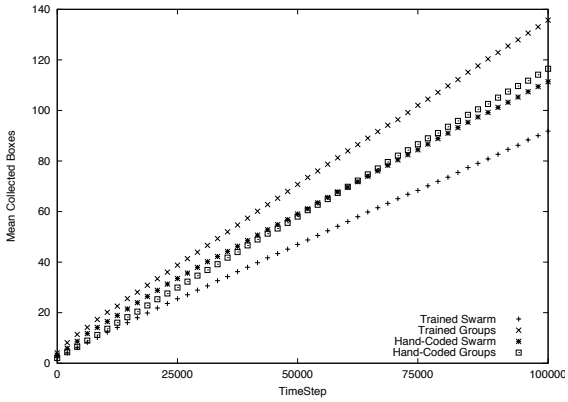


Figure 7.7: Mean number of boxes collected over time for the first experiment.

agents moved 0.1 units per timestep. Agents started at uniformly randomly distributed locations. Finally, a third experiment compared a different, even larger hierarchy against 625 independent agents.

Each experimental run lasted 100,000 timesteps, and each treatment had 100 independent runs. Treatments were gauged based on the mean number of boxes returned. Differences in results were measured at the final timestep using Bonferroni-corrected two-tailed t-tests at the 95% confidence level.

First Experiment: One Level Hierarchies The first experiment compared an entirely distributed *swarm* of 50 agents against a *group* of ten controller agents, each in charge of five basic agents. For each configuration, I performed runs with a set of trained behaviors and a set of hand-coded behaviors. Figure 7.7 shows the results of all four sets of runs.

Independent Agents Versus Hierarchies: The semi-centralized coordination available via controller agents allows specific groups of agents work on a single box, suggesting controller agents should outperform a distributed swarm. Without controller agents, agents could become stranded at boxes waiting for other agents to help pull. These waiting agents simply relied on random discovery of the box by other agents to gather enough helpers. Figure 7.7 verifies the expected improvement due to the controllers. The improvement was statistically significant in both cases.

Hand-Coded Versus Trained: Figure 7.7 shows the performance of hand-coded behaviors versus

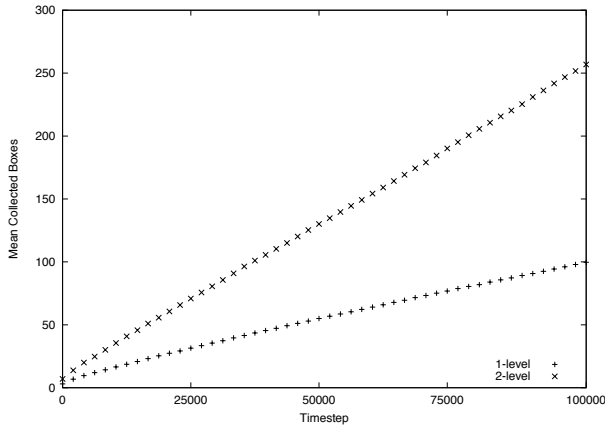


Figure 7.8: Mean number of boxes collected over time for the second experiment.

trained behaviors for a distributed swarm and group of controller agents. Hand-coded behaviors were expected to perform better due to significant training error. However, the trained solution actually performed statistically significantly better than the hand-coded solution! This was due to a more random exploration strategy which allowed agents to disperse throughout the environment better. This same exploration strategy didn't fare as well in the swarm case, however, because this strategy resulted in too many agents distributed across multiple boxes rather than pulling on the same box. While the results do not present a clear advantage to either training or programming, they do suggest that training the agents will crucially not *significantly* impair performance.

Second Experiment: 2-Level Hierarchies The second experiment compared a 1-level hierarchy with a 2-level hierarchy: two Level-2 controllers, each in charge of five Level-1 controllers, each in charge of five agents. The scenario changed to favor two levels of coordination: the environment now had eight boxes which each required five agents to pull, and two boxes which each required twenty-five agents to pull. Just as the first experiment was constructed so as to demonstrate the value of some degree of homogeneous coordination, the second experiment is meant to show the value of homogeneous coordination at two levels.

As shown in Figure 7.8, two levels significantly outperformed a single level, and for similar reasons as the first experiment. If in the one-level case a group discovered a 25-agent box, 4 other

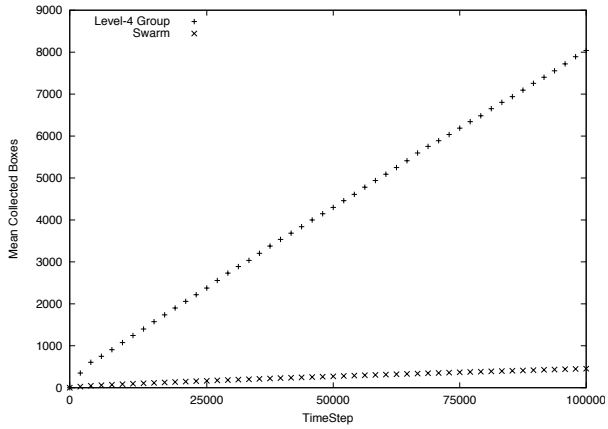


Figure 7.9: Mean number of boxes collected over time for the third experiment.

groups had to randomly discover the box before it could be moved and all the groups freed. However, two layers of coordination allowed training of 25-agent groups.

Third Experiment: Large Numbers of Agents The third experiment looked at scalability by rerunning the first experiment with a *four*-level hierarchy: a single Level-4 controller in charge of five Level-3 controllers, each in charge of five Level-2 controllers, each in charge of five Level-1 controllers, each in charge of five agents. This arrangement results in 625 agents and 156 controller agents. The larger number of agents necessitated a larger environment with different boxes: the environment expanded to 255×225 with 25 size-5 boxes, five size-25 boxes, and one size-125 box.

The four-level hierarchy was compared with 625 independent agents, and, as would be expected, this was no contest: the swarm of agents were simply outclassed, as shown in Figure 7.9. This somewhat unfair contest was not intended to show the *efficacy* of the hierarchy, but simply that it this approach is capable of scaling to large numbers of agents and more complex environments.

Chapter 8: Heterogeneous HiTAB

In many problems, a homogeneous team is incapable of solving all the challenges of the environment. For example, some areas are only accessible by climbing robots, while other areas are only visible by aerial robots. Heterogeneous teams offer a possible solution by fielding diverse capabilities.[1] Heterogeneity can be either in terms of *capability* or *behavior*. Heterogeneous capability generally refers to different physical abilities such as sensor setup, size and payload, maximum speed, degrees of freedom, and the type of terrain they can traverse. Heterogeneous behavior refers to agents which are identical in capability, but executing different behavior sets. Even if the agents contain the same behavior sets, they might be executing different behaviors to achieve coordination. For example, a group of aerial robots and another group of ground robots might work together in an area coverage problem. Within each group, the robots are considered heterogeneous since they are executing different behaviors to accomplish their particular task.

This chapter applies HiTAB to heterogeneous multiagent systems. Like the previous chapter, I arrange the agents into an agent hierarchy with agents as the leaf nodes and (possibly virtual) controller agents as non-leaf nodes. Heterogeneity can occur at any level of the agent hierarchy, from the root down to individual agents, and may be achieved in multiple ways: agents organized into squads, united by common functionality, results in a *functional organization*. Conversely, heterogeneous agents organized in squads capable of performing specific subtasks results in a *project organization*.

Controller agents in a heterogenous hierarchy have multiple agent classes as subsidiaries. Each subsidiary agent class contains at least one agent and all agents within a subsidiary class are the same. The homogenous hierarchy in the previous chapter is a special case of heterogeneous hierarchy with a single subsidiary agent class. Multiple subsidiary agent classes present the primary challenge

[1] Assembling a heterogeneous team depends on assembling the right mixture of agent capabilities. In this thesis, I assume each team is appropriate for the task, and do not worry about assembling the optimal team for the current problem. For work in dynamic team assembly, see Jones ea al [342] and the references within.

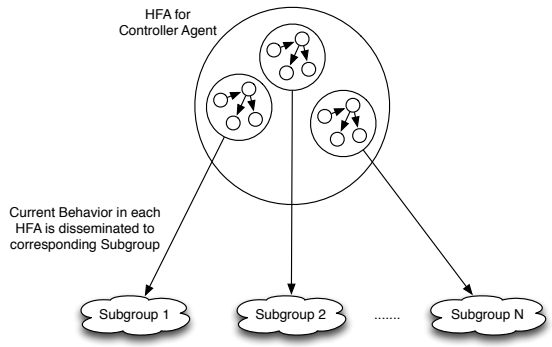


Figure 8.1: Heterogeneous HiTAB with multiple HFAs.

with heterogeneous hierarchies: determining the controller agent's behaviors. How do the controller agent's behaviors map to the groups of subsidiary agents? One solution is to represent the controller agent's behaviors as a combination of multiple HFAs: one HFA per subsidiary agent class. Another possible solution is to maintain a single HFA whose states map to all possible joint behaviors.

8.1 Multiple HFAs

A heterogeneous controller agent could maintain one HFA per subsidiary agent class (see Figure 8.1). Thus, a controller agent's basic behaviors are a collection of HFAs, each which needs to be pulsed every timestep. More formally, let $A_i = \{a_1, a_2 \ldots a_m\}$ be the set of the controller agent's subsidiary agents of class i. Then, each A_i has its own HFA $\langle S_i, B_i, F_i, T_i \rangle \in \mathscr{H}$ where S_i, B_i, F_i, and T_i are agent class-specific states, behaviors, features, and transition function.

While the individual HFAs are constructed as in the single agent case, training these HFAs is challenging due to the interactions between multiple HFAs. The coordination between agents results from collecting appropriate examples for each class: as each HFA is trained, all the other agents must be in the appropriate state for coordination. Consider training a HFA H to control agent class j. H requires its own examples of the form $\langle S_i, f_t, S_j \rangle$ where the feature vector $\vec{f}$ depends on the state of *all the other agents* in the system. Coercing the other agents into the appropriate state makes the training process highly complex, especially as the number of agent classes increases.

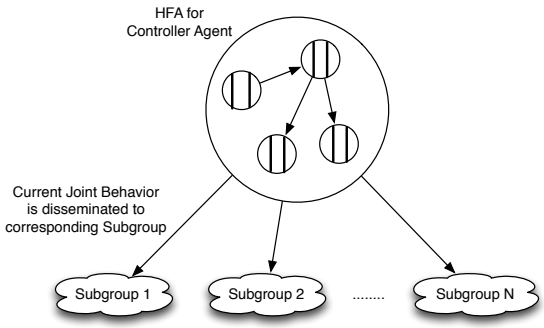


Figure 8.2: Heterogeneous HiTAB with joint behaviors.

8.2 Joint Behaviors

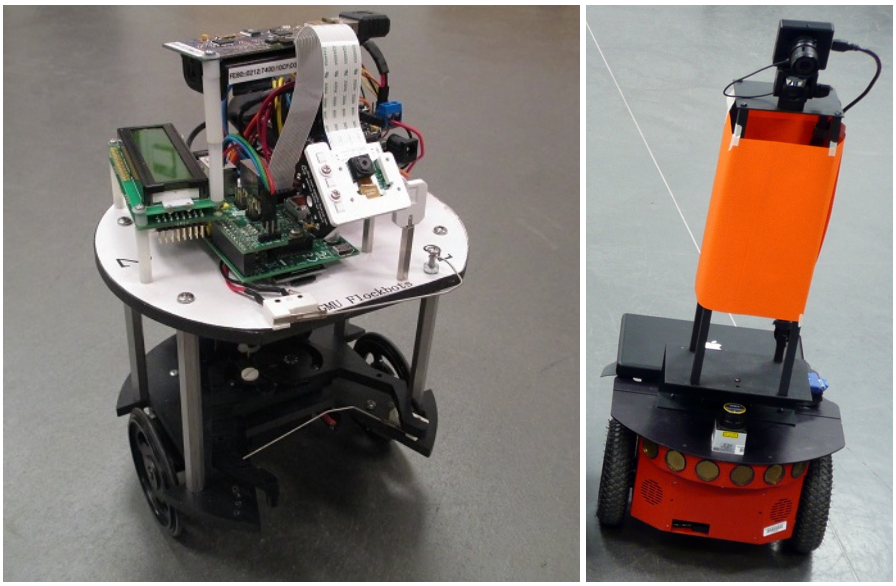
If a heterogeneous controller agent maintains a single HFA to control *all* its subsidiary agents, then the individual behaviors must control multiple groups of agents (see Figure 8.2). Called *joint behaviors* these behaviors simultaneously direct multiple groups. At each time step, every group is directed to perform the piece of the joint behavior responsible for the group. For example, the joint behavior *Forward:Left* would cause the first subsidiary group to move forward while the second group would turn left. Using joint behaviors allows the same training methodology used thus far, however, creating all possible joint behaviors results in an exponential number of behaviors, which is untenable for large numbers of subsidiary agent classes.

For joint behaviors the formal model is similar to those already presented with the only change occurring to the behaviors. The features are the same as in Chapter 7, and the states and transition functions are the same as in Chapter 4.

- $A_i = \{a_1, a_2 \ldots a_m\}$ is the set of the controller agent's subsidiary agents of class i. Each A_i is non-empty.

- $\Gamma = \{A_1, A_2, \ldots A_n\}$ is the set of all the controller agent's subsidiaries. All the the subsidiaries must be assigned a set A_i, and each A_i is non-empty.

- $B = \{JB(\Gamma)_1, JB(\Gamma)_2 \ldots JB(\Gamma)_k, B_{k+1}, B_{k+2}, \ldots B_l\}$ is the set of basic behaviors for the controller agent. Behaviors $JB(\Gamma)_i$ are an n-tuples of behaviors $\langle b_1, b_2, \ldots b_n \rangle$ where b_i is the



(a) Flockbot (b) Pioneer

Figure 8.3: Robots used in the heterogeneous robot demonstration.

behavior for all agents in A_i. Behaviors B_j are the controller agents basic behaviors, which operate the same as those of a basic agent. A controller agent is not required to have its own basic behaviors.

In this thesis, I choose to use joint behaviors rather than multiple HFAs since using joint behaviors does not alter the training process for controller agents. This simplifies the complexity of training heterogeneous teams and permits the same training procedure for both simple agents and controller agents. However, joint behaviors do introduce an extra step to the demonstrator: all joint behaviors must be manually created and added to the behavior library prior to training. While this does inject significant expert knowledge into the problem formulation, it allows significant progress in supervised multiagent training.

8.3 Robot Demonstration

To demonstrate heterogeneous agent hierarchies, I used joint behaviors to reproduce the three-robot box-pushing experiment described in [343]: the goal was to push a long box across the room using

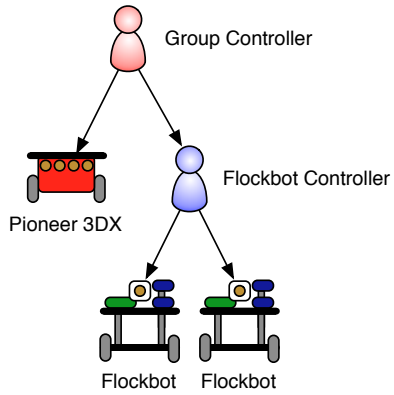


Figure 8.4: Agent Hierarchy for the Heterogeneous Demonstration.

two robots, with a third robot in the back monitoring the box's angle. The difference being, of course, that in [343] the robots were hard-coded, while I trained the robots.

The two robots pushing the box were Flockbots (Figure 8.3(a)), a small roughly 7-inch-diameter differential robot [202]. These robots are sensor poor and could not monitor the box's angle. The third robot was a Pioneer 3DX (Figure 8.3(b)), a much larger differential-drive robot equipped with a variety of sensors, including a scanning laser range finder. The Pioneer used the laser to monitor the angle of the box.

Figure 8.4 shows how the robots were arranged into a two-level hierarchy: the three robots were the leaves, and a virtual Flockbot Controller agent was responsible for coordinating the two Flockbots. The top-level virtual controller agent, Group Controller, coordinated between the Flockbot team and the Pioneer.

Each level of the agent hierarchy required different features:

- Each Flockbot used a single sensor feature, *BumpPressed*, which returned true or false if the Flockbot's bump sensor was depressed.
- 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 had triggered their **Done** states).

- The Group Controller used two features: *AllDone* and *BoxAngle*. *AllDone* informed it that both the Pioneer and Flockbot Controller had triggered their **Done** states. *BoxAngle*, based on 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 Flockbots had three basic behaviors: **Stop**, forward at full speed **Forward100** (0.4 meters per second), and forward at quarter-speed **Forward25**. The Pioneer used two basic behaviors: **Stop**, and **Forward** which moved the robot forward at a half-meter per second.

Using these basic behaviors and features, I trained additional HFAs for each robot:

- FLOCKBOTINIT: Moved the Flockbot forward until it contacted the box as indicated by *BumpPressed*.
- GOTOBOX: Kept the Pioneer within a pre-specified distance of the box using *DistanceToBox* .
- 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.

I then defined the following joint heterogeneous behaviors which mapped to basic behaviors in the Flockbot Controller:

- FLOCKBOTGROUPINIT: each Flockbot executed FLOCKBOTINIT.
- FLOCKBOTFORWARD: both Flockbots executed **Forward100**.
- CORRECTLEFT: the left Flockbot ran **Forward25** and the right Flockbot ran **Foward100**.
- CORRECTRIGHT: the left Flockbot ran **Forward100** and the right Flockbot ran **Foward25**.

Next, I trained the a single HFA for the Flockbot Controller:

- FLOCKBOTCONTROLLERINIT: executed *FlockbotGroupInit* until *AllDone* was signaled.

Before training the top-level automaton, I defined the following joint heterogeneous behaviors for the Group Controller:

- GROUPCORRECTLEFT: the Pioneer executed GOTOTOBOX and the FlockbotController executed CORRECTLEFT.
- GROUPCORRECTRIGHT: the Pioneer executed GOTOTOBOX and the FlockbotController executed CORRECTRIGHT.
- GROUPFORWARD: the Pioneer executed GOTOTOBOX while the FlockbotController ran FLOCKBOTFOWARD.

Finally, I trained the following HFAs in the Group Controller:

- 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 more naturally reside in the Flockbot Controller, because it relied on the box angle, it needed to be one level higher.
- GROUPMAIN: The top-level automaton, which first moved the robots into their initial positions, then began calling PUSH.

Figure 8.5 shows the setup of the robots in performing the box-pushing experiment. All trained HFAs are shown in Figure 8.6.

The experiment was successful at producing the desired box-pushing behavior. As mentioned in Figure 8.6, some of the transition edges mention a learned distance β and learned angle α. As trained, these values came to approximately $\beta = 960$ mm and $\alpha \approx 11.5°$(exact values differed slightly from edge to edge, not varying more than $1.7°$).

Table 8.1 shows the number of user-provided demonstration samples required to train each of the above HFA. We note that PUSH and FLOCKBOTINIT required more samples due to the loops present in those two HFAs which are not present in the other, simpler HFAs. As a result, PUSH and FLOCKBOTINIT took approximately 10 minutes to train while the rest of the HFAs took approximately 2 minutes.

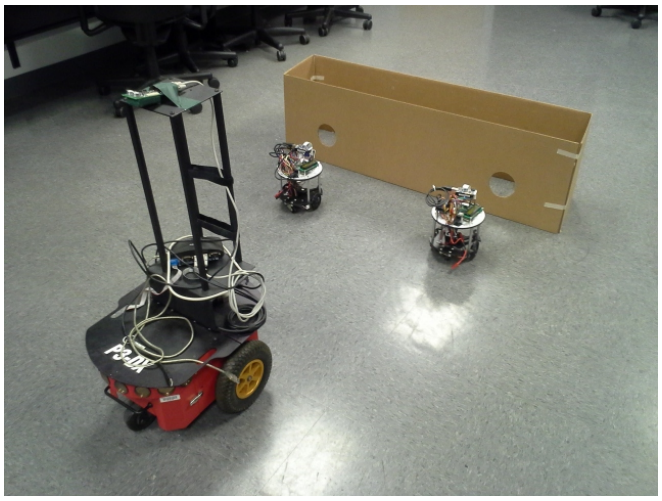


Figure 8.5: Experimental Setup

Table 8.1: Number of samples to train each HFA

Behavior	Number of Examples
Group Main	4
Push	10
Flockbot Controller Init	3
Pioneer Init	3
Goto Box	5
Flockbot Init	10
Total Samples	**35**

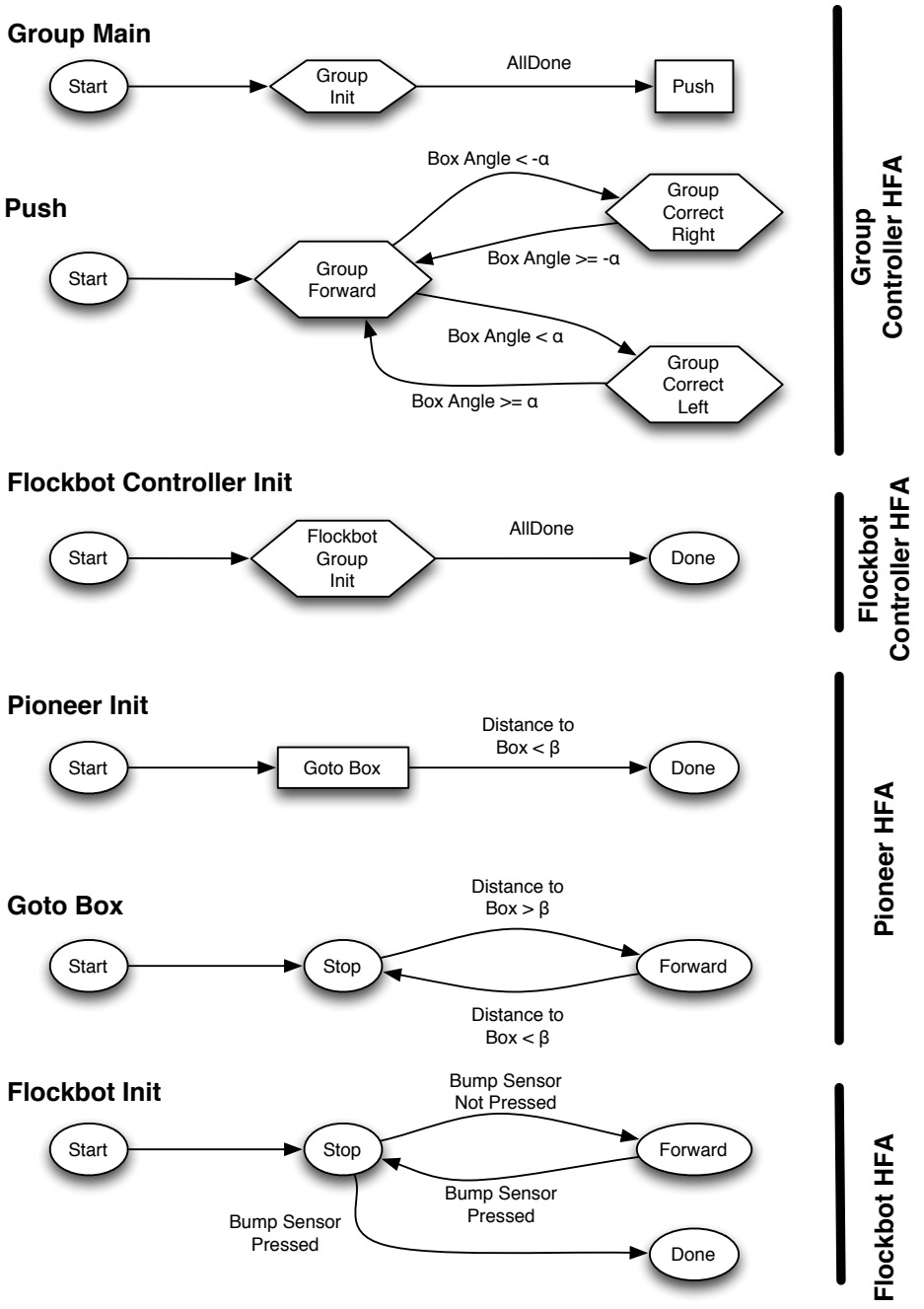


Figure 8.6: Trained HFAs for box pushing

This demonstration succeeded in rapidly reproducing a complex, heterogeneous multi-robot behavior from the literature. The rapid training, as evidenced by Table 8.1, came about due to simplifying the problem through behavior decomposition and the agent hierarchy. However, the simplification resulted from a large injection of user knowledge (e.g., knowing that the Flockbots needed to rotate the box, I created a joint behavior to do just that). Such user knowledge does reduce the problem to programming aided by machine learning, but does allow progress versus highly expensive, knowledge-poor approaches used elsewhere.

Chapter 9: Conclusions

This thesis developed a learning from demonstration system, called HiTAB, to train multiple homogeneous and heterogenous agents in complex, dynamic environments. HiTAB reduces the dependence of human experts to program new agent behaviors by applying machine learning to a database of state/action pairs provided by a human demonstrator. Notably, HiTAB employes a behavior hierarchy to reduce the number of examples to train complex behaviors. By manually decomposing behaviors into a hierarchy of smaller behaviors, the size of the learning space is reduced, thus, permitting training of complex behaviors with minimal examples.

Chapter 4 shows how HiTAB is applied to a single agent. Experiments showed that HiTAB can train a single agent in a variety of tasks which involve a high degree of dynamism. The success of HiTAB at RoboCup validated a claim of LfD proponents that trained behaviors can perform as well as hard-coded behaviors. Additionally, experiments demonstrated how novices can quickly train complex behaviors using HiTAB in minimal time (due to HiTAB's behavior decomposition).

When using HiTAB, the demonstrator sometimes provides incorrect examples during training, which result in errant behaviors. Since HiTAB uses minimal examples, it becomes important to identify and correct these errors. Chapter 5 presented two algorithms for identifying errant examples, and (potentially) removing them. The goal was to modify the example database rather than rely on the noise-insensitivity of the classification algorithm. Both unlearning algorithms aimed to distinguish between noise and undersampling of the learning space. If examples are simply noise, they are potentially removed. Examples that represent undersampled regions of the learning space are left in the database.

The thrust of this thesis was applying HiTAB to the multiagent case, which presents a fundamental problem: closing the gap between macro-level behaviors and micro-level behaviors for each agent. Chapter 6 introduced behavior bootstrapping as a method to reduce the gap by training a single agent in the context of other agents. Chapter 7 presented agent hierarchies as another technique to reduce

the gulf between micro- and macro-level behaviors. Agents were arranged into a tree with leaf nodes as actual agents, and internal, non-leaf nodes as controller agents. By arranging the agents into a tree, agents at each level are only responsible for a small number of subordinate agents, which allows rapid training due to the minimal number of required examples. While agent hierarchies inherit the disadvantages of tree structures (e.g., not robust to agent failure), they have an overriding scaling advantage: at any level of the tree, an agent must deal with only a fixed number of agents. HiTAB is a rare application of supervised machine learning to the multiagent problem domain.

Finally, Chapter 8 extended agent hierarchies to the heterogenous case. A major challenge in training heterogenous agents is their coordination architecture: how does a controller agent's HFA modify the behavior of multiple subsidiary agent classes? I take the approach of training a single HFA for each controller, where the controller agent's basic behaviors are joint behaviors among the subsidiary agent classes. As a result, the controller agent's HFA has the same structure as subsidiary agents which greatly reduces the complexity of training, and also permits the use of the same training methods for all (controller and non-controller) agents in the system.

9.1 Future Work

Several lines of future work suggested themselves while completing this thesis.

- This thesis demonstrated two primary multiagent training techniques: behavior bootstrapping and agent hierarchies. One demonstration missing is training large number of agents to perform stageful behavior (i.e., combine heterogeneous agent hierarchies with behavior bootstrapping).

- While HFAs are a good representation of agent behaviors, perhaps other representation would be more apropos. HiTAB's HFAs are not as expressive as other representations. For example, Petri Nets are inherently parallel, while stack automata and arbitrary functions are more computationally expressive. Another option is hierarchical reinforcement learning which uses an on-line algorithm to learn behavior hierarchies for exploration [344].

- Decision trees were chosen for HiTAB due to their ease of handling different types of data and that they do not require significant amounts of training data. However, they are limited to

axis-parallel splits of the learning space. Other classifiers, while requiring more data, could partition the space in a non-linear fashion.

- Currently, the agent hierarchies are fixed a priori, and, thus, are brittle to agent failure or changing tasks. Two lines of future work include automatically building the agent hierarchy, and reforming the hierarchy according to an external condition such as agent failure or requiring the system to (temporarily) solve a new task.

- Another approach for training coordination (which shows promise) is to modify the feature vector to include the state of all other agents [337].

- The HFAs within HiTAB use a one-to-one mapping between states and behaviors. HiTAB uses "wrapper" behaviors to achieve a many-to-one mapping between behaviors and states. Currently, these wrappers are manually constructed due to a lack of data. Developing a learning algorithm to automatically discover this many-to-one mapping is a future work opportunity.

- The current HiTAB GUI suffered from several problems which could be addressed by HRI-based techniques. In particular, managing training of multiple heterogeneous agents become cumbersome due to the large number of features and behaviors (each agent could have its own set of behaviors and features). A more intuitive method to train the agents is required, especially for large systems. One possible addition is to store common controller agent behaviors and features so the user does not need to program them.

- This thesis applied HiTAB to relatively small models (e.g., soccer keepaway, box foraging). In general, there is nothing to prevent HITAB's application to large scale multiagent models involving many (thousands, hundred to thousands) of complex agents with unpredictable, non-linear interactions. For example, modeling the rise of the Mongol empire [345], modeling conflict in Eastern Africa [346], or modeling the rise and fall of the Pueblo Culture in Arizona [347].

Appendix A: Software and Hardware Development

While performing the work in this thesis, I constructed, and contributed to, several software and hardware projects. This appendix presents information about these projects (and some research performed not related to the thesis). MASON, in particular, featured heavily during the thesis since the HiTAB system is constructed on top of MASON. Additionally, the RoboPatriots were a major component of the thesis, particularly, the research presented in Chapter 4.

A.1 Software

I am co-developer of ECJ[1], a popular open-source evolutionary computation toolkit written in Java. ECJ is designed to be highly flexible with nearly all the classes dynamically determined at runtime by user-provided parameter files. The library is highly efficient, and supports many evolutionary computation techniques including genetic algorithms, genetic programming, evolutionary strategies, coevolution, island models, particle swarm optimization, differential evolution, master/slave evaluation, multi-objective optimization, and meta-evolution. Distributed under a BSD-style license, ECJ targets large-scale projects with heavy experimental needs, and as such, is highly "hackable".

Initially, I used ECJ to evolve kernels for support vector machines in a classification task [348]. That work illustrated the need for very large-scale evolutionary computation methods, thus, I extended ECJ to operate in a cloud environment for Parabon, Inc. [349]. Using the massively distributed version of ECJ, I examined multiagent aspects of evolutionary computation, specifically cooperative coevolution [350, 351].

While I was co-developing ECJ, I was also co-developing ECJ's sister project, MASON[2], an open-source Java-based multiagent simulation toolkit [297]. MASON was designed to dovetail with ECJ to permit automatic discovery of agent behaviors. MASON was intended to support simulations of large numbers of agents on a single machine without any domain-specific features. To this end, MASON separates the model from the visualization, and MASON models are completely serializable.

[1] `http://cs.gmu.edu/~eclab/projects/ecj`
[2] `http://cs.gmu.edu/~eclab/projects/mason`

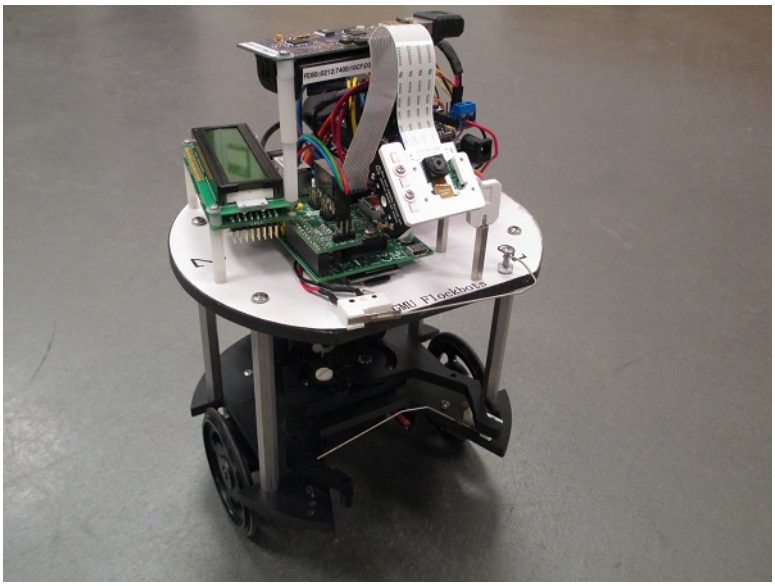


Figure A.1: A completed FlockBot: a roughly circular two-platform differential-drive mobile robot 7” in diameter. The robot is constructed nearly entirely of off-the-shelf material, and is intended to be much more powerful computationally and functionally than most inexpensive swarmbots.

Both these features allow running a model on a high-performance back-end machine, pause the model and transfer to another machine for visualization. Additionally, MASON models are entirely encapsulated, permiting multiple models to run on a single machine.

MASON models are typically simulated worlds either in 2D or 3D grid or continuous worlds which are (normally) weak models of a real-world environment. To extend the applicability of MASON, I added GIS capability [352]. Now models can use any information contained within a GIS system to guide agent behavior.

A.2 Hardware

A.2.1 FlockBots

I helped design and construct the FlockBots[3], an open-source robot specification for an inexpensive (less than $500) differential-drive robot [202] (See Figure A.1). The goal was to develop a robot

[3] `http://cs.gmu.edu/~eclab/projects/robots/flockbots/pmwiki.php`

suitable for "swarm" style multi-robot research with as much functionality as possible as cheaply as possible. The low price allows us to create a collaborative swarm of eight FlockBots using primarily off-the-shelf parts. Each FlockBot has:

- A servo controlled gripper capable of grabbing cans. The gripper has a digital bump sensor for determining when an object is within the gripper.
- A flat push region for pushing tall boxes, with a digital bump sensor to know when the robot is pushing against an object.
- Wheel encoders.
- Five Sharp IR ranging sensors, placed at -90, -45, 0 (straight ahead), 45, and 90 degrees.
- A six-cell NiMH battery.

The original hardware design of the FlockBots was based around a Gumstix embedded computer, connected to a Waysmall daughter board via I^2C. The Waysmall provided a serial port for communication with a CMUCam2 vision system, and an I2C connection to a Brainstem microcontroller. The Waysmall had an additional serial port for console login to the Gumstix. The CMUCam2 performed various computer vision tasks (notably, color blob tracking), while the Brainstem controlled three servos and the analog and digital sensors. The Gumstix ran Linux while the Brainstem was programmed in a proprietary language called TEA, a C-derivative. The Gumstix provides Bluetooth communication, but not wireless ethernet.

Shortly after we completed the first version of the FlockBots, Gumstix released a robotics-oriented micro controller called the RoboStix, which replaced the Waysmall daughter board and the Brainstem. The RoboStix was programmed in C, and handled all the functionality of the Waysmall and Brainstem in a smaller, more user-friendly package. The knowledge gained while developing the first two versions of the FlockBots guided the development of the RoboPatriots (discussed below).

After several years, the FlockBots were showing their age, so we redesigned them. After raising the high platform so the robots could grab regular sized soda cans, we replaced the Gumstix and Brainstem with inexpensive hobbyist electronics. A Raspberry Pi embedded computer replaced the

Gumstix and CMUCam2, and an Arduino Uno replaced the Brainstem. A dedicated camera plugs directly into the Raspberry Pi. The Raspberry Pi and Arduino Uno communicate via serial which allows reprogramming of the Uno via the Pi. The Raspberry Pi has 802.11 wireless via a USB dongle. The Arduino Uno runs embedded C++ rather than Processing due to the ease of reprogramming from the Raspberry Pi. The increased processing power of the Raspberry Pi allows us to run OpenCV for all vision-related tasks.

Motivated by the FlockBots, several co-authors and I studied ways to conduct ant foraging using robots and movable sensor motes (called beacons) to simulate ant pheromones [2]. Simulation experiments demonstrated the efficacy of the method, despite the addition or removal of obstacles or environmental features.

A.2.2 RoboPatriots

The RoboPatriots are a team of three humanoid robots constructed by the GMU Computer Science Department, and I am the primary developer [311–315]. Meant for dynamic multirobot tasks such as the robot soccer competition RoboCup [79], behaviors for the RoboPatriots are typically constructed by hand. The RoboPatriot 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.

Humanoid Hardware

Like the FlockBots, the RoboPatriot hardware has changed over time. Throughout the history of the RoboPatriots, we have based the hardware of commercially available hardware and electronics, which allowed us to focus on embodied AI rather than hardware development.

2009 RoboPatriots We choose the Kondo KHR-1HV as our robot base. Each robot has 20 degrees of freedom (DOF) controlled via servo motors. There are 6 DOF per leg, 3 DOF per arm, and 2 DOF in the neck. The sixteen KRS-788HV digital servos used in the arms and legs produce 10 kg-cm of torque at a speed of 0.14 sec. / 60 degrees. The two KRS-4024 servos in the shoulders produce

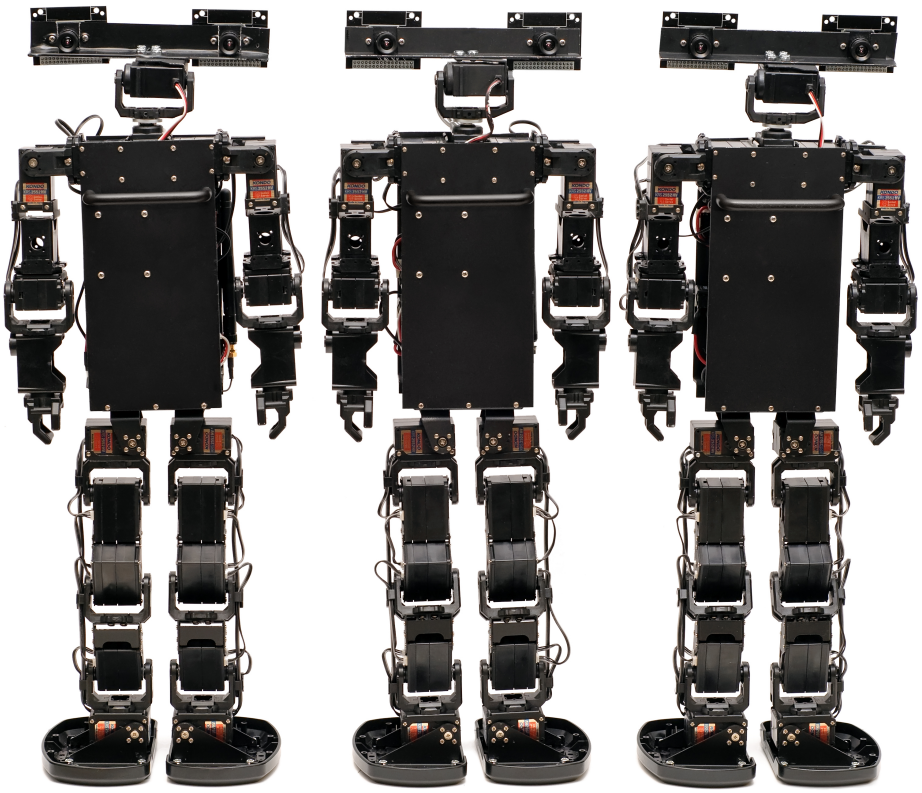


Figure A.2: Three complete RoboPatriots: constructed from off the shelf-parts, the aim was to develop a relatively inexpensive humanoid robot capable of competing in the Humanoid league of RoboCup.

10.5 kg-cm of torque at a speed of 0.17 sec. / 60 degrees. These servos are controlled via the RCB-3 controller board. In addition, two KRG-3 single axis gyros connect to the RCB-3.

The main processing was handled by a 600 MHz Verdex embedded computer attached to a Robostix microcontroller [353]. The Verdex communicates with the Robostix via I^2C and with the RCB-3 via serial at 115200 bps. A custom inverter board facilitates communication.

We replaced the stock head with a pan-tilt mount constructed from two standard micro-servos and a CMUCam3 [354]. The pan-tilt servos are controlled from the CMUCam3. The CMUCam3 communicates with the Verdex through a serial bridge on the Robostix: data travels via I^2C from the Verdex to the Robostix, and from the Robostix to the CMUCam3 via serial at 115200 bps.

The CMUCam3 is an embedded vision system based on the NXP LPC2106 60 MHz processor and an Omnivision CMOS sensor. The RGB CMOS sensor has CIF resolution (352x288), and operates at 26 frames per second. The CMUCam3 has an open-source API allowing customized software development.

The robots each have a 12 V, 2100 mAh battery. A power regulator board ensures proper voltages to each component. Each robot is equipped with 802.11b wireless.

2010 RoboPatriots After the 2009 competition, we realized our robots contained too much plastic and that the hardware architecture was too complicated. So, we completely redesigned the robots with easily obtainable equipment. The robot base was the Kondo KHR-3HV. Each robot has 3 DOF per arm, 5 DOF per leg, and 2 DOF in the neck. The sixteen Kondo KRS-2555HV digital servos used in the arms and legs produce 14 kg-cm of torque at a speed of 0.14 sec / 60 degrees. The 2555HV servos communicate via a serial protocol over RS485 and are controlled by the RCB-4 servo controller board. In addition, two KRG-3 single axis gyros connect to the RCB-4. The two Kondo KRS-788HV digital servos used our pan/tilt mount produce 10 kg-cm of torque at a speed of 0.14 sec. / 60 degrees. These servos are controlled by the Surveyor SVS vision system via PWM.

Our main sensor is the Surveyor SVS (Stereo Vision System) [355]. The SVS consists of two OmniVision OV 7725 camera modules connected to two independent 600 MHz Blackfin BF537 processors. The two OmniVision camera modules are mounted on a pan/tilt mount with 10.5 cm

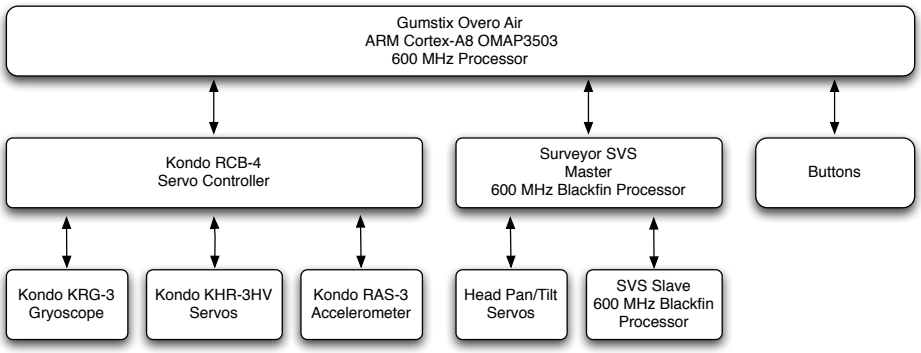


Figure A.3: The hardware architecture of the 2012 RoboPatriots, and the information flow between components.

separation. Each camera module operates at 640x480 resolution with a 90-degree field of view. The two processors are connected by a dedicated SPI bus, and the camera unit provides a Lantronix Matchport 802.11g wireless unit. Two single axis RAS-2 dual-axis accelerometers connect to the SVS. A custom inverter board allows serial communication at 115200 bps between the RCB-4 and the SVS.

Each robot has a 11.1V 2200 mAh battery.

2011 RoboPatriots For 2011, we kept the same servos and servo controller, but added an additional computer: a Gumstix Overo Air [353]. The Air is a 600 MHz OMAP 3503 processor with 256 MB of flash and 256 MB RAM. The Air runs embedded Linux, and provides 802.11 b/g. The Air communicates with the RCB-4 over a dedicated serial bus with a custom inverter circuit for logic level shifting and signal inversion. The SVS and Air are connected via an SPI bus. The Air and SVS are mounted on a custom motherboard which also provides power distribution, USB connections, and sensor connections. See Figure A.4 for a prototype.

For 2012, we kept this hardware configuration. Figure A.3 shows the hardware architecture and information flow between components.

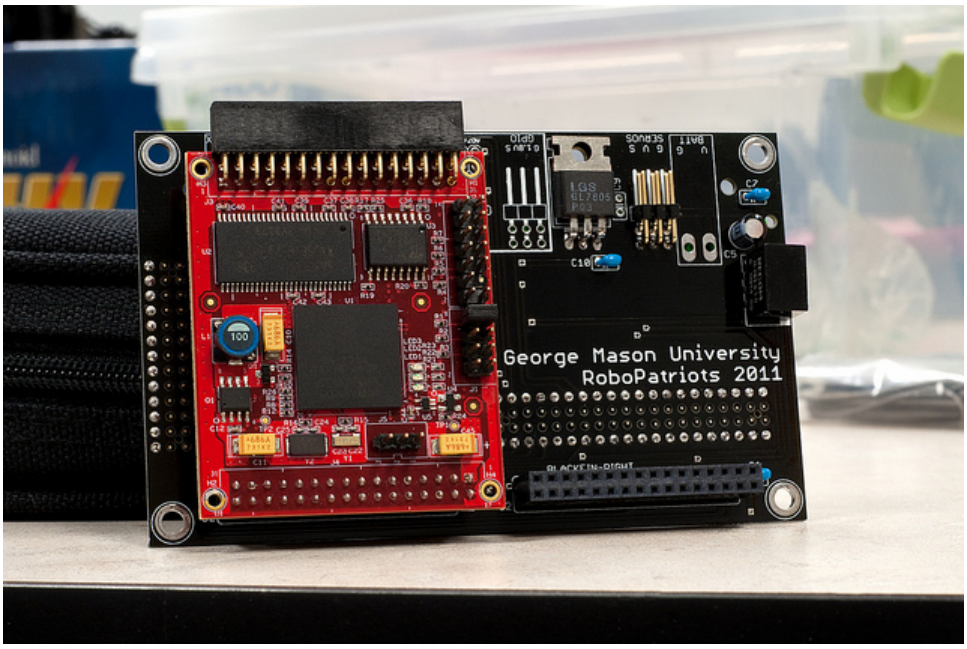


Figure A.4: A prototype of the integrated motherboard connecting the Gumstix computer and SVS modules.

Humanoid Software

Based on the hardware, all the RoboPatriots' software is built from scratch, with minimal use of libraries. The goalie and attackers each run a state machine for high-level decision making. Predefined motions are stored on the RCB4 and are not dynamically modifiable (i.e., all motions are open loop). However, we can interrupt motions during execution and can run arbitrary length cyclic motions (e.g., we can execute N walking steps based on dynamic sensor information). The software architecture is split across the RCB-4, SVS and Gumstix as follows:

- The RCB-4 Servo controller handles gyro stabilization and execution of predefined motions. Motions are a series of stop-frames executed in an open-loop fashion.

- The SVS performs vision related tasks.

- The Gumstix detects falls, handles communication, and runs the state machine.

The Brain The Gumstix runs the state-machine used for high-level decision making in a single loop. Every pass through the loop, the code first checks if the robot fell over and, if necessary, stands up and resets internal state. Next, any messages received from the referee box are processed, and internal state is updated accordingly. Finally, the code requests updated information from the cameras, and uses this information when stepping the main behavior.

The RoboPatriots execute a straight-forward version of soccer: search for the ball, approach the ball, align towards the goal, align for kicking, kick the ball, and repeat. While admittedly simple, this hard-coded version of soccer served the RoboPatriots well. In four years, we twice advanced to the second round and have scored several goals.

Vision Initially, we relied on the black-box vision algorithms of the CMUCam2 for detecting the ball and goals. Upgrading to the CMUCam3 provided a way to directly program computer vision algorithms on a dedicated image sensor; however the CMUCam3s proved to be highly susceptible to static shock, so we upgraded again to the SVS, which allowed us to perform stereo-vision with dedicated hardware.

Both modules of the SVS execute the same code to handle color tracking with the master requesting information from the slave when the master cannot see the tennis ball. Objects of interest are detected via a simple two pass state machine. In the first pass, the image is subsampled by examining every fifth column and fifth row. Pixels in this subsampled image are marked with either a color of interest (orange, blue, yellow, green, or white) or a "not interested" value. A second pass over the subsampled image looks for runs of color and applies specific detection conditions to prevent false positives. The Blackfins run the detection loop continuously at 15 – 20 fps.

The master camera also control the pan/tilt servos in the neck: it uses a simple PD-controller to keep the tennis ball centered within the master's frame; and pans the head when looking for the goals.

Bibliography

Bibliography

[1] L. Panait and S. Luke, "Learning ant foraging behaviors," in *Proceedings of the International Conference on the Simulation and Synthesis of Living Systems*, 2004.

[2] B. Hrolenok, S. Luke, K. Sullivan, and C. Vo, "Collaborative foraging using beacons," in *Proceedings of Autonomous Agents and Multi-Agent Systems Conference (AAMAS)*, 2010.

[3] A. Hailegiorgis, W. Kennedy, M. Roleau, J. Bassett, M. Coletti, G. Balan, and T. Gulden, "An agent based model of climate change and conflict among pastoralists in east africa," in *Proceedings of International Congress on Environmental Modeling and Software Modeling for Environment's Sake*, D. A. Swayne, W. Yang, A. A. Voinov, A. Rizzoli, and T. Filatova, Eds., 2010.

[4] W. Kennedy, A. Hailegiorgis, M. Rouleau, J. Bassett, M. Colletti, G. Balan, and T. Gulden, "An agent-based model of conflict in east Africa and the effect of watering holes," in *Proceedings of the Conference on Behavior Representation in Modeling and Simulation*, 2010.

[5] G. Balan and S. Luke, "History-based traffic control," in *Proceedings of Autonomous Agents and Multi-Agent Systems Conference (AAMAS)*, 2006, pp. 616 – 621.

[6] J. B. Dunham, "An agent-based spatially explicit epidemiological model in MASON," *Journal of Artificial Societies and Social Simulation*, vol. 9, no. 1, 2006.

[7] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, 2009.

[8] P. Evrard, E. Gribovskaya, S. Calinon, A. Billard, and A. Kheddar, "Teaching physical collaborative tasks: Object-lifting case study with a humanoid," in *Proceedings of the IEEE-RAS International Conference on Humanoid Robotics (Humanoids)*, 2009, pp. 399–404.

[9] L. Peternel, T. Petriç, E. Oztop, and J. Babiç, "Teaching robots to cooperate with humans in dynamic manipulation tasks based on multi-modal human-in-the-loop approach," *Autonomous Robots*, vol. 36, no. 1-2, pp. 123–136, 2014.

[10] S. Kim, C. H. Kim, B. You, and S. Oh, "Stable whole-body motion generation for humanoid robots to imitate human motions," in *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2009, pp. 2518–2524.

[11] D. Kulić, C. Ott, D. Lee, J. Ishikawa, and Y. Nakamura, "Incremental learning of full body motion primitives and their sequencing through human motion observation," *International Journal of Robotics Research*, 2011.

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

[13] P. E. Black, "Finite state machine," in *Dictionary of Algorithms and Data Structures*, V. Pieterse and P. E. Black, Eds., Aug 2013. [Online]. Available: http://www.nist.gov/dads/HTML/finiteStateMachine.html

[14] A. Girault, B. Lee, and E. A. Lee, "Hierarchical finite state machines with multiple concurrency models," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 6, pp. 742–760, 1999.

[15] J. E. Hopcroft, *Introduction to automata theory, languages, and computation*. Pearson Education India, 1979.

[16] R. M. Keller, *Computer Science: Abstraction to Implementation*. Harvey Mudd College, 2001.

[17] P. Hong, M. Turk, and T. S. Huang, "Gesture modeling and recognition using finite state machines," in *Proceeding of International Conference on Automatic Face and Gesture Recognition*. IEEE, 2000, pp. 410–415.

[18] M. Mohri, "On some applications of finite-state automata theory to natural language processing," *Natural Language Engineering*, vol. 2, no. 01, pp. 61–80, 1996.

[19] A. V. Aho, *Compilers: Principles, Techniques and Tools*, 2nd ed. Pearson Education India, 2003.

[20] P. Gladyshev and A. Patel, "Finite state machine approach to digital event reconstruction," *Digital Investigation*, vol. 1, no. 2, pp. 130–149, 2004.

[21] S. H. Collins and A. Ruina, "A bipedal walking robot with efficient and human-like gait," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2005, pp. 1983–1988.

[22] R. Le Hy, A. Arrigoni, P. Bessiere, and O. Lebeltel, "Teaching bayesian behaviors to video game characters," *Robotics and Autonomous Systems*, vol. 47, no. 2, pp. 177–185, 2004.

[23] E. F. Moore, "Gedanken-experiments on sequential machines," *Automata studies*, vol. 34, pp. 129–153, 1956.

[24] G. H. Mealy, "A method for synthesizing sequential circuits," *Bell System Technical Journal*, vol. 34, no. 5, pp. 1045–1079, 1955.

[25] D. Carmel and S. Markovitch, "Learning models of intelligent agents," in *Proceedings of the American Association of Artificial Intelligence Conference (AAAI)*, 1996, pp. 62–67.

[26] S. Bie and J. Persson, "Behavior-based control of the ERS-7 AIBO Robot," Lund University, Tech. Rep., 2004. [Online]. Available: http://ai.cs.lth.se/xj/StefanBie/report.pdf

[27] C. Armbrust, D. Schmidt, and K. Berns, "Generating behavior networks from finite-state machines," in *Proceedings of German Conference on Robotics*. VDE, 2012, pp. 1–6.

[28] D. Mescheder, K. Tuyls, and M. Kaisers, "Opponent modeling with POM-DPs," in *Proceedings of 23nd Belgium–Netherlands Conference on Artificial Intelligence (BNAIC)*, 2011, pp. 152–159.

[29] D. Harel, "Statecharts: A visual formalism for complex systems," *Science of computer programming*, vol. 8, no. 3, pp. 231–274, 1987.

[30] M. Yannakakis, "Hierarchical state machines," in *Theoretical Computer Science: Exploring New Frontiers of Theoretical Informatics*, ser. Lecture Notes in Computer Science, J. van Leeuwen, O. Watanabe, M. Hagiya, P. Mosses, and T. Ito, Eds. Springer Berlin Heidelberg, 2000, vol. 1872, pp. 315–330.

[31] V. Sklyarov, "Hierarchical finite-state machines and their use for digital control," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 2, pp. 222–228, 1999.

[32] W. Kohn and T. Skillman, "Hierarchical control systems for autonomous space robots," in *Proceedings of AIAA Conference in Guidance, Navigation and Control*, vol. 1, 1988, pp. 382–390.

[33] M. Risler, "Behavior control for single and multiple autonomous agents based on hierarchical finite state machines," Ph.D. dissertation, Technische Universitat Darmstadt, 2010.

[34] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society, Series C*, vol. 28, no. 1, pp. 100 – 108, 1979.

[35] M. A. Figueiredo and A. K. Jain, "Unsupervised learning of finite mixture models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 3, pp. 381–396, 2002.

[36] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. John Wiley and Sons, 2001.

[37] R. E. Schapire, "The boosting approach to machine learning: An overview," in *Proceedings of the MSRI Workshop on Nonlinear Estimation and Classification*, 2002.

[38] ——, "The boosting approach to machine learning: An overview," in *Nonlinear estimation and classification*. Springer, 2003, pp. 149–171.

[39] K. O. Arras, O. M. Mozos, and W. Burgard, "Using boosted features for the detection of people in 2d range data," in *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2007, pp. 3402–3407.

[40] Y. Sun, M. S. Kamel, A. K. Wong, and Y. Wang, "Cost-sensitive boosting for classification of imbalanced data," *Pattern Recognition*, vol. 40, no. 12, pp. 3358–3378, 2007.

[41] T. Kudo and Y. Matsumoto, "A boosting algorithm for classification of semi-structured text," in *Proceedings of Conference on Empirical Methods in Natural Language Processing*, vol. 4, 2004, pp. 301–308.

[42] R. Lawrence, A. Bunn, S. Powell, and M. Zambon, "Classification of remotely sensed imagery using stochastic gradient boosting as a refinement of classification tree analysis," *Remote sensing of environment*, vol. 90, no. 3, pp. 331–336, 2004.

[43] T. M. Mitchell, *Machine Learning*. The McGraw-Hill Companies, 1997.

[44] J. P. González and Ümi Özgüner, "Lane detection using histogram-based segmentation and decision trees," in *Proceedings of Intelligent Transportation Systems*, 2000.

[45] R. Kohavi and J. R. Quinlan, "Data mining tasks and methods: Classification: decision-tree discovery," in *Handbook of data mining and knowledge discovery*. Oxford University Press, Inc., 2002, pp. 267–276.

[46] H.-P. Huang and C.-C. Liang, "Strategy-based decision making of a soccer robot system using a real-time self-organizing fuzzy decision tree," *Fuzzy Sets and Systems*, vol. 127, no. 1, pp. 49–64, 2002.

[47] L. Steels, "Language games for autonomous robots," *Intelligent Systems, IEEE*, vol. 16, no. 5, pp. 16–22, 2001.

[48] J. Bruce, T. Balch, and M. Veloso, "Fast and inexpensive color image segmentation for interactive robots," in *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*, vol. 3. IEEE, 2000, pp. 2061–2066.

[49] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[50] S. K. Murthy, S. Kasif, and S. Salzberg, "A system for induction of oblique decision trees," *Journal of Artificial Intelligence Research*, vol. 2, pp. 1–32, 1994.

[51] E. Cantu-Paz and C. Kamath, "Inducing oblique decision trees with evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 1, pp. 54–68, 2003.

[52] J. Quinlan, "Simplifying decision trees," *International Journal of Man-Machine Studies*, vol. 27, no. 3, pp. 221 – 234, 1987.

[53] L. Breiman, J. H. Friedman, C. J. Stone, and R. A. Olshen, *Classification and Regression Trees*. CRC Press, 1984.

[54] J. Mingers, "Expert systems-rule induction with statistical data," *Journal of the Operational Research Society*, pp. 39–47, 1987.

[55] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81 – 106, 1986.

[56] G. V. Kass, "An exploratory technique for investigating large quantities of categorical data," *Journal of the Royal Statistical Society, Series C*, vol. 29, no. 2, pp. 119 – 127, 1980.

[57] J. R. Quinlan, *C4.5: Programs for Machine Learning*, 1st ed., ser. Morgan Kaufmann Series in Machine Learning. Morgan Kaufmann, January 1993.

[58] I. Steinwart and A. Christmann, *Support vector machines*. Springer, 2008.

[59] B. Schölkoph and A. J. Smola, *Learning with Kernels*. MIT Press, 2002.

[60] C.-C. Chang and C.-J. Lin, "Training ν-support vector classifiers: theory and algorithms," *Neural computation*, vol. 13, no. 9, pp. 2119–2147, 2001.

[61] B. Schölkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett, "New support vector algorithms," *Neural Computation*, vol. 12, pp. 1207 – 1245, 2000.

[62] E. Fix and J. L. Hodges Jr, "Discriminatory analysis-nonparametric discrimination: consistency properties," DTIC Document, Tech. Rep., 1951.

[63] A. M. Kibriya and E. Frank, "An empirical comparison of exact nearest neighbor algorithms," in *Knowledge Discovery in Databases: PKDD 2007*. Springer, 2007, pp. 140–151.

[64] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, Sept. 1975.

[65] M. Wooldridge, *An Introduction to Multiagent Systems*, 2nd ed. Wiley Publishing, 2009.

[66] G. Weiss, Ed., *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999.

[67] M. J. Matarić and F. Michaud, "Behavior based systems," in *Springer Handbook of Robotics*. Springer, 2008, ch. 38, pp. 891 – 909.

[68] J. S. Albus, "Outline for a theory of intelligence," *IEEE Transaction on Systems, Man and Cybernetics*, vol. 21, no. 3, pp. 473–509, 1991.

[69] R. Brooks, "A robust layered control system for a mobile robot," *IEEE Transactions on Robotics*, vol. RA-2, pp. 14–23, April 1986.

[70] R. J. Firby, "An investigation into reactive planning in complex domains." in *Proceedings of the American Association of Artificial Intelligence Conference (AAAI)*, vol. 87, 1987, pp. 202–206.

[71] M. Schoppers, "Universal plans for reactive robots in unpredictable environments." in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, vol. 87. Citeseer, 1987, pp. 1039–1046.

[72] R. C. Arkin, "Towards the unification of navigational planning and reactive control," in *Proceedings of AAAI Spring Symposium on Robot Navigation*, 1989, pp. 1 – 5.

[73] G. N. Saridis, "Intelligent robotic control," *IEEE Transactions on Automatic Control*, vol. 28, no. 5, pp. 547–557, 1983.

[74] M. J. Matarić, "Reinforcement learning in the multi-robot domain," in *Robot Colonies*. Springer, 1997, pp. 73–83.

[75] M. Mouad, L. Adouane, P. Schmitt, D. Khadraoui, and P. Martinet, "Control architecture for cooperative mobile robots using multi-agent based coordination approach," in *Proceedings of 6th National Conference on Control Architectures of Robots*, 2011.

[76] P. Švestka and M. H. Overmars, "Coordinated path planning for multiple robots," *Robotics and Autonomous Systems*, vol. 23, no. 3, pp. 125–152, 1998.

[77] M. B. Dias and A. Stentz, "A free market architecture for distributed control of a multirobot system," in *Proceedings of 6th International Conference on Intelligent Autonomous Systems*, 2000, pp. 115–122.

[78] H.-S. Shim, H.-S. Kim, M.-J. Jung, I.-H. Choi, J.-H. Kim, and J.-O. Kim, "Designing distributed control architecture for cooperative multi-agent system and its real-time application to soccer robot," *Robotics and Autonomous Systems*, vol. 21, no. 2, pp. 149–165, 1997.

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

[80] T. Huntsberger, P. Pirjanian, A. Trebi-Ollennu, H. Das Nayar, H. Aghazarian, A. J. Ganino, M. Garrett, S. S. Joshi, and P. S. Schenker, "CAMPOUT: a control architecture for tightly coupled coordination of multirobot systems for planetary surface exploration," *IEEE Transaction on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 33, no. 5, pp. 550–559, 2003.

[81] J. Kok, C. Warmer, and I. Kamphuis, "Powermatcher: multiagent control in the electricity infrastructure," in *Proceedings of Autonomous Agents and Multi-Agent Systems Conference (AAMAS)*. ACM, 2005, pp. 75–82.

[82] R. Negenborn, "Multi-agent model predictive control with applications to power networks multi-agent model predictive control with applications to power networks multi-agent model predictive control with applications to power networks," Ph.D. dissertation, Delft University of Technology, 2007.

[83] R. Simmons, S. Singh, D. Hershberger, J. Ramos, and T. Smith, "First results in the coordination of heterogeneous robots for large-scale assembly," in *Proceedings of the International Symposium on Experimental Robotics (ISER)*, 2000.

[84] D. Hooper and G. Peterson, "HAMR: A hybrid multi-robot control architecture," in *Proceedings of the International Florida Artificial Intelligence Research Society Conference (FLAIRS)*, 2009.

[85] P. Stone and M. Veloso, "Layered learning and flexible teamwork in robocup simulation agents," in *RoboCup-99: Robot Soccer World Cup III*, ser. Lecture Notes in Computer Science, M. Veloso, E. Pagello, and H. Kitano, Eds. Springer Berlin / Heidelberg, 2000, vol. 1856, pp. 65–72.

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

[87] J. Saunders, C. Nehaniv, and K. Dautenhahn, "Teaching robots by molding behavior and scaffolding the environment," in *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2006.

[88] M. N. Nicolescu, "A framework for learning from demonstration, generalization and practice in human-robot domains," Ph.D. dissertation, University of Southern California, 2003.

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

[90] R. W. Beard, J. Lawton, and F. Y. Hadaegh, "A coordination architecture for spacecraft formation control," *IEEE Transactions on control systems technology*, vol. 9, no. 6, pp. 777–790, 2001.

[91] J. Albus, "RCS: A cognitive architecture for intelligent multi-agent systems," in *Proceedings of Performance Metrics for Intelligent Systems Workshop (PerMIS)*, 2004.

[92] J. S. Albus, C. L. Pape, I. N. Robinson, T. Chiueh, A. D. McAulay, Y.-H. Pao, and Y. Takefuji, "RCS: A reference model architecture for intelligent control," *Computer*, vol. 25, no. 5, pp. 56–79, 1992.

[93] D. Goldberg and M. J. Mataric, "Design and evaluation of robust behavior-based controllers," in *Robot Teams: From Diversity to Polymorphism*, T. Balch and L. E. Parker, Eds. A. K. Peters, 2002, pp. 315–344.

[94] D. Goldberg and M. Matarić, "Coordinating mobile robot group behavior using a model of interaction dynamics," in *Proceedings of the Third Annual Conference on Autonomous Agents*. ACM, 1999, pp. 100–107.

[95] R. Grabowski, L. E. Navarro-Serment, C. Paredis, and P. Khosla, "Heterogeneous teams of modular robots for mapping and exploration," *Autonomous Robots*, vol. 8, no. 3, pp. 293 – 308, June 2000.

[96] J. McLurkin and D. Yamins, "Dynamic task assignment in robot swarms," in *Proceedings of Robotics: Science and Systems (RSS)*, 2005.

[97] L. Parker, "ALLIANCE: An architecture for fault tolerance multi-robot cooperation," *IEEE Transactions on Robotics*, vol. 14, no. 2, 1998.

[98] T. Vu, J. Go, G. Kaminka, M. Veloso, and B. Browning, "MONAD: a flexible architecture for multi-agent control," in *Proceedings of Autonomous Agents and Multi-Agent Systems Conference (AAMAS)*, 2003, pp. 449–456.

[99] L. E. Navarro-Serment, R. Grabowski, C. J. Paredis, and P. K. Khosla, "Millibots: The development of a framework and algorithms for a distributed heterogeneous robot team," *IEEE Robotics and Automation Magazine*, vol. 9, no. 4, pp. 31 – 40, December 2002.

[100] Y. Shoham, R. Powers, and T. Grenager, "If multi-agent learning is the answer, what is the question?" *Artificial Intelligence*, vol. 171, no. 7, pp. 365–377, 2007.

[101] K. Tuyls and G. Weiss, "Multiagent learning: Basics, challenges, and prospects," *AI Magazine*, vol. 33, no. 3, p. 41, 2012.

[102] P. Stone and M. Veloso, "Multiagent systems: A survey from a machine learning perspective," *Autonomous Robots*, vol. 8, no. 3, pp. 345–383, 2000.

[103] J.-H. Kim and P. Vadakkepat, "Multi-agent systems: a survey from the robot-soccer perspective," *Intelligent Automation & Soft Computing*, vol. 6, no. 1, pp. 3–17, 2000.

[104] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," *Autonomous Agents and Multi-Agent Systems*, vol. 11, no. 3, pp. 387–434, 2005.

[105] H. H. Bui, S. Venkatesh, and D. Kieronska, "Learning other agents' preferences in multi-agent negotiation using the Bayesian classifier," *International Journal of Cooperative Information Systems*, vol. 08, no. 04, pp. 275–293, 1999.

[106] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, "Autonomous inverted helicopter flight via reinforcement learning," in *Proceedings of International Symposium on Experimental Robotics*. Springer, 2004, pp. 363–372.

[107] P. Stone and M. Veloso, "Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork," *Artificial Intelligence*, vol. 110, no. 2, pp. 241 – 273, 1999.

[108] B. Zupan, M. Bohanec, I. Bratko, and J. Demsar, "Machine learning by function decomposition," in *Proceedings of International Conference on Machine Learning (ICML)*, 1997, pp. 421–429.

[109] W.-P. Lee, J. Hallam, and H. H. Lund, "Learning complex robot behaviors by evolutionary computing with task decomposition," in *Proceedings of European Workshop on Learning Robots*, A. Birk and J. Demiris, Eds., 1997.

[110] A. Lockerd and C. Breazeal, "Tutelage and socially guided robot learning," in *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*, vol. 4. IEEE, 2004, pp. 3475–3480.

[111] M. Nicolescu, O. Jenkins, and A. Olenderski, "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.

[112] M. Nicolescu, O. Jenkins, and A. Stanhope, "Fusing robot behaviors for human-level tasks," in *Proceedings of the International Conference on Development and Learning (ICDL)*. IEEE, 2007, pp. 76–81.

[113] S. Whiteson, N. Kohl, R. Miikkulainen, and P. Stone, "Evolving soccer keepaway players through task decomposition," *Machine Learning*, vol. 59, no. 1-2, pp. 5–30, 2005.

[114] V. Khare, X. Yao, B. Sendhoff, Y. Jin, and H. Wersing, "Co-evolutionary modular neural networks for automatic problem decomposition," in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol. 3, 2005, pp. 2691–2698 Vol. 3.

[115] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun, "Support vector machine learning for interdependent and structured output spaces," in *Proceedings of International Conference on Machine Learning (ICML)*. ACM, 2004, p. 104.

[116] V. Eyharabide and A. Amandi, "Automatic task model generation for interface agent development," *Inteligencia artificial*, vol. 9, no. 26, pp. 49–57, 2005.

[117] A. Garland, "Learning hierarchical task models by demonstration," Mitsubishi Electric Research Laboratories, Tech. Rep. TR-2001-03, 2001.

[118] L. C. Cobo, C. L. I. Jr, and A. L. Thomaz, "Automatic task decomposition and state abstraction from demonstration," in *Proceedings of Autonomous Agents and Multi-Agent Systems Conference (AAMAS)*, 2012.

[119] L. C. Cobo, P. Zeng, C. L. I. Jr, and A. L. Thomaz, "Automatic state abstraction from demonstration," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

[120] R. A. Jacobs, M. I. Jordan, and A. G. Barto, "Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks," *Cognitive Science*, vol. 15, no. 2, pp. 219 – 250, 1991.

[121] P. Stone, "Layered learning in multiagent systems," Ph.D. dissertation, Carnegie Mellon University, 1998.

[122] S. Whiteson and P. Stone, "Concurrent layered learning," in *Proceedings of Autonomous Agents and Multi-Agent Systems Conference (AAMAS)*, 2003, pp. 193–200.

[123] S. Gustafson and W. Hsu, "Layered learning in genetic programming for a cooperative robot soccer problem," in *Genetic Programming*, J. Miller, M. Tomassini, P. Lanzi, C. Ryan, A. Tettamanzi, and W. Langdon, Eds. Springer Berlin / Heidelberg, 2001, vol. 2038, pp. 291–301.

[124] P. E. Utgoff and D. J. Stracuzzi, "Many-layered learning," *Neural Computation*, vol. 14, no. 10, pp. 2497–2529, 2002.

[125] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," in *Proceedings of International Conference on Machine Learning (ICML)*, D. H. Fisher, Ed. Morgan Kaufmann, 1997, pp. 12–20.

[126] M. Yeasin and S. Chaudhuri, "Automatic robot programming by visual demonstration of task execution," in *Proceedings of International Conference on Advanced Robotics (ICAR)*, 1997.

[127] H. Veeraraghavan and M. M. Veloso, "Learning task specific plans through sound and visually interpretable demonstrations," in *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2008, pp. 2599–2604.

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

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

[130] G. Hovland, P. Sikka, and B. McCarragher, "Skill acquisition from human demonstration using a hidden markov model," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, vol. 3. IEEE, 1996, pp. 2706–2711.

[131] K. Dixon and P. K. Khosla, "Learning by observation with mobile robots: A computational approach," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[132] J. Saunders, C. L. Nehaniv, and K. Dautenhahn, "An experimental comparison of imitation paradigms used in social robotics," in *Proceedings of IEEE Robot and Human Interactive Communication (ROMAN)*. IEEE Press, 2004, pp. 691–696.

[133] C. L. Nehaniv and K. Dautenhahn, "The correspondence problem," in *Imitation in Animals and Artifacts*, K. Dautenhahn and C. L. Nehaniv, Eds. Cambridge, MA, USA: MIT Press, 2002, pp. 41–61.

[134] C. Sammut, S. Hurst, D. Kedzier, and D. Michie, "Learning to fly," in *Proceedings of International Conference on Machine Learning (ICML)*. Morgan Kaufmann, 1992, pp. 385–393.

[135] J. E. Young, K. Ishii, T. Igarashi, and E. Sharlin, "User-centered programming by demonstration: Stylistic elements of behavior," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

[136] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Movement imitation with nonlinear dynamical systems in humanoid robots," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2002, pp. 1398–1403.

[137] A. Billard and M. J. Mataric, "Learning human arm movements by imitation: Evaluation of a biologically inspired connectionist architecture," *Robotics and Autonomous Systems*, vol. 37, no. 2 – 3, pp. 145 – 160, November 2001.

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

[139] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of International Conference on Machine Learning (ICML)*, C. E. Brodley, Ed. ACM, 2004.

[140] O. Jenkins, M. Mataric, and S. Weber, "Primitive-based movement classification for humanoid imitation," in *Proceedings of the IEEE-RAS International Conference on Humanoid Robotics (Humanoids)*, 2000.

[141] O. C. Jenkins and M. J. Matarić, "Deriving action and behavior primitives from human motion data," in *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2002, pp. 2551 – 2556.

[142] N. S. Pollard and J. K. Hodgins, "Generalizing demonstrated manipulation tasks," in *Algorithmic Foundations of Robotics V*, J.-D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson, Eds. Springer, 2003, pp. 523–540.

[143] E. Drumwright, O. C. Jenkins, and M. J. Mataric, "Exemplar-based primitives for humanoid movement classification and control," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[144] A. Fod, M. J. Mataric, and O. C. Jenkins, "Automated derivation of primitives for movement classification," *Autonomous Robots*, vol. 12, no. 1, pp. 39–54, 2002.

[145] C. Breazeal and B. Scassellati, "Robots that imitate humans," *Trends in Cognitive Sciences*, vol. 6, no. 11, pp. 481–487, 2002.

[146] D. Kulic, D. Lee, C. Ott, and Y. Nakamura, "Incremental learning of full body motion primitives for humanoid robots," in *Proceedings of the IEEE-RAS International Conference on Humanoid Robotics (Humanoids)*, Dec. 2008, pp. 326–332.

[147] A. Olenderski, M. N. Nicolescu, and S. J. Louis, "Robot learning by demonstration using forward models of schema-based behaviors," in *Proceedings of the International Conference on Informatics in Control, Automation and Robotics*, J. Filipe, J. Andrade-Cetto, and J.-L. Ferrier, Eds. INSTICC Press, 2005, pp. 263–269.

[148] R. Dillmann, "Teaching and learning of robot tasks via observation of human performance," *Robotics and Autonomous Systems*, vol. 47, no. 2-3, pp. 109–116, 2004.

[149] M. N. Nicolescu and M. J. Mataric, "Task learning through imitation and human-robot interaction," in *Models and Methods of Imitation and Social Learning in Robots, Humans, and Animals*, K. Dautenhahn and C. Nehaniv, Eds., 2005.

[150] M. Matarić, "Getting humanoids to move and imitate," *IEEE Intelligent Systems*, vol. 15, no. 18–24, 2000.

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

[152] M. Mehta, S. Ontañón, T. Amundsen, and A. Ram, "Authoring behaviors for games using learning from demonstration," in *Proceedings of Workshop on Case-Based Reasoning for Computer Games*, 2009.

[153] T. Lozano-Perez, "Robot programming," in *Proceedings of IEEE*, vol. 71, 1983, pp. 821–841.

[154] A. Levas and M. Selfridge, "A user friendly high-level robot teaching system," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 1984.

[155] B. Dufay and J.-C. Latombe, "An approach to automatic robot programming based on inductive learning," in *International Journal of Robotics Research*, vol. 3, 1984, pp. 3 – 20.

[156] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, "Robot programming by demonstration," in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Springer, 2007, ch. 59.

[157] Y. Kuniyoshi, M. Inaba, and H. Inoue, "Teaching by showing: Generating robot programs by visual observation of human performance," in *Proceedings of International Symposium on Industrial Robotics*, 1989.

[158] ——, "Learning by watching: Extracting reusable task knowledge from visual observation of human performance," *IEEE Transactions on Robotics*, vol. 10, no. 6, pp. 799 – 822, 1994.

[159] S. B. Kang and K. Ikeuchi, "A robot system that observes and replicates grasping tasks," in *Proceedings of International Conference on Computer Vision*, 1995.

[160] C. P. Tung and A. C. Kak, "Automatic learning of assembly task using a DataGlove system," in *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*, 1995.

[161] K. Ikeuchi and T. Suehiro, "Toward an assembly plan from observation. Part I: Task recognition with polyhedral objects," *IEEE Transactions on Robotics*, vol. 10, no. 3, pp. 368 –385, June 1994.

[162] M. Ito, K. Noda, Y. Hoshino, and J. Tani, "Dynamic and interactive generation of object handling behaviors by a small humanoid robot using a dynamic neural network model," *Neural Networks*, vol. 19, no. 3, pp. 323 – 337, April 2006.

[163] T. Inamura, N. Kojo, and M. Inaba, "Situation recognition and behavior induction based on geometric symbol representation of multimodal sensorimotor patterns," in *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2006.

[164] S. Liu and H. Asada, "Teaching and learning of deburring robots using neural networks," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 1993, pp. 339 – 345.

[165] A. Billard and G. Hayes, "DRAMA: A connectionist architecture for control and learning in autonomous robots," *Adaptive Behavior*, vol. 7, no. 1, pp. 1 – 14, 1999.

[166] M. Kaiser and R. Dillmann, "Building elementary robot skills from human demonstration," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 1996, pp. 2700 – 2705.

[167] R. Dillmann, M. Kaiser, and A. Ude, "Acquisition of elementary robot skills from human demonstration," in *Proceedings of International Symposium on Intelligent Robotic Systems (SIRS)*, 1995, pp. 1 – 38.

[168] J. Yang, Y. Xu, and C. Chen, "Hidden markov model approach to skill learning and its application in telerobotics," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 1993, pp. 396 – 402.

[169] C. Breazeal, M. Berlin, A. Brooks, J. Gray, and A. L. Thomaz, "Using perspective taking to learn from ambiguous demonstrations," *Robotics and Autonomous Systems*, vol. 54, no. 5, pp. 385–393, 2006.

[170] B. Scassellati, "Imitation and mechanisms of joint attention: A developmental structure for building social skills on a humanoid robot," in *Computation for Metaphors, Analogy, and Agents*, C. L. Nehaniv, Ed. Springer, 1999, pp. 176 – 195.

[171] H. Kozima and H. Yano, "A robot that learns to communicate with human caregivers," in *Proceedings of International Workshop on Epigenetic Robotics*, 2001.

[172] H. Ishiguro, T. Ono, M. Imai, and T. Kanda, "Development of an interactive humanoid robot robovie - an interdisciplinary approach," in *Robotics Research*, R. Jarvis and A. Zelinsky, Eds. Springer Berlin / Heidelberg, 2003, vol. 6, pp. 179 – 191.

[173] K. Nickel and R. Stiefelhagen, "Pointing gesture recognition based on 3D-tracking of face, hands and head orientation," in *Proceedings of International Conference on Multimodel Interfaces*, 2003, pp. 140 – 146.

[174] M. Ito and J. Tani, "Joint attention between a humanoid robot and users in imitation game," in *Proceedings of the International Conference on Development and Learning (ICDL)*, 2004.

[175] V. Hafner and Freédéric, "Learning to interpret pointing gestures: experiments with four-legged autonomous robots," in *Biomimetic Neural Learning for Intelligent Robots*, S. Wermter, G. Palm, and M. Elshaw, Eds. Springer Berlin / Heidelberg, 2005.

[176] P. Dominey, M. Alvarez, B. Gao, M. Jeambrun, A. Cheylus, A. Weitzenfeld, A. Martinez, and A. Medrano, "Robot command, interrogation and teaching via social interaction," in *Proceedings of the IEEE-RAS International Conference on Humanoid Robotics (Humanoids)*, 2005.

[177] M. Pardowitz, S. Knoop, R. Dillmann, and R. D. Zollner, "Incremental learning of tasks from user demonstrations, past experiences and vocal comments," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 37, no. 2, pp. 322 – 332, 2007.

[178] A. L. Thomaz, M. Berlin, and C. Breazeal, "Robot science meets social science: An embodied computational model of social referencing," in *Proceedings of Workshop Towards Social Mechanisms of Android Science*, 2005, pp. 7 – 17.

[179] C. Breazeal and L. Aryananda, "Recognition of affective communicative intent in robot-directed speech," *Autonomous Robots*, vol. 12, no. 1, pp. 83 – 104, 2002.

[180] B. Jansen and T. Belpaeme, "A computational model of intention reading in imitation," *Robotics and Autonomous Systems*, vol. 54, no. 5, pp. 394 – 402, May 2006.

[181] A. Chella, H. Dindo, and I. Infantino, "A cognitive framework for imitation learning," *Robotics and Autonomous Systems*, vol. 54, no. 5, pp. 403 – 408, 2006.

[182] C. Nehaniv and K. Dautenhahn, "Of hummingbirds and helicopters: An algebraic framework for interdisciplinary studies of imitation and its applications," in *Interdisciplinary Approachs to Robot Learning*, J. Demiris and A. Birk, Eds. World Scientific Press, 2000, pp. 136 – 161.

[183] A. Billard, S. Calinon, and F. Guenter, "Discriminative and adaptive imitation in uni-manual and bi-manual tasks," *Robotics and Autonomous Systems*, vol. 54, no. 5, pp. 370 – 384, 2006.

[184] Y. Demiris and B. Khadhouri, "Hierarchical attentive multiple models for execution and recognition of actions," *Robotics and Autonomous Systems*, vol. 54, no. 5, pp. 361 – 369, 2006.

[185] S. Chernova and M. Veloso, "Confidence-based multi-robot learning from demonstration," *International Journal of Social Robotics*, vol. 2, no. 2, pp. 195–215, 2010.

[186] S. Chernova, "Confidence-based robot policy learning from demonstration," Ph.D. dissertation, Carnegie Mellon University, 2009.

[187] D. Grollman and O. Jenkins, "Dogged learning for robots," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2007, pp. 2483–2488.

[188] D. H. Grollmand and O. C. Jenkins, "Can we learn finite state machine robot controllers from interactive demonstration?" in *From Motor Learning to Interaction Learning in Robots*. Springer, 2010, pp. 407–430.

[189] B. Browning, L. Xu, and M. Veloso, "Skill acquisition and use for a dynamically-balancing soccer robot," in *Proceedings of the American Association of Artificial Intelligence Conference (AAAI)*, 2004, pp. 599–604.

[190] M. Blokzijl-Zanker and Y. Demiris, "Multi-robot learning by demonstration," in *Proceedings of Autonomous Agents and Multi-Agent Systems Conference (AAMAS)*, 2012.

[191] S. Raza, S. Haider, and M.-A. Williams, "Teaching coordinated strategies to soccer robots via imitation," in *Proceedings of International Conference on Robotics and Biomimetics*, 2012.

[192] S. Raza, "On teaching collaboration to a team of autonomous agents via imitation," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

[193] M. F. Martins and Y. Demiris, "Learning multirobot joint action plans from simultaneous task execution demonstrations," in *Proceedings of Autonomous Agents and Multi-Agent Systems Conference (AAMAS)*, 2010, pp. 931–938.

[194] B. Takács and Y. Demiris, "Balancing spectral clustering for segmenting spatio-temporal observations of multi-agent systems," in *Proceedings of the International Conference on Data Mining (ICDM)*, 2008, pp. 580–587.

[195] H. B. Suay, R. Toris, and S. Chernova, "A practical comparison of three robot learning from demonstration algorithms," *International Journal of Social Robotics*, vol. 4, no. 4, pp. 319 – 330, 2012.

[196] A. Alissandrakis, C. L. Nehaniv, K. Dautenhahn, and J. Saunders, "Evaluation of robot imitation attempts: Comparison of the system's and the human's perspectives," in *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2006.

[197] S. Calinon and A. G. Billard, "What is the teacher's role in robot programming by demonstration? Toward benchmarks for improved learning," *Journal of Interaction Studies*, vol. 8, no. 3, 2007.

[198] M. Pomplun and M. Matarić, "Evaluation metrics and results of human arm movement imitation," in *Proceedings of the IEEE-RAS International Conference on Humanoid Robotics (Humanoids)*, 2000.

[199] A. Steinfeld, T. Fong, D. Kaber, M. Lewis, J. Scholtz, A. Schultz, and M. Goodrich, "Common metrics for human-robot interaction," in *Proceedings of the 1st ACM SIGCHI/SIGART Conference on Human-robot Interaction*, 2006, pp. 33–40.

[200] The swarms project. [Online]. Available: http://www.swarms.org/

[201] M. J. Matarić, "Interaction and intelligent behavior," Ph.D. dissertation, Massachusetts Institute of Technology, 1994.

[202] S. Luke, K. Andrea, M. Bowen, D. Fleming, K. Sullivan, B. Hrolenok, C. Vo, A. Bovill, R. Steck, B. Davidson, J. Youngblood, S. Aziz, F. Clavijo, S. A. Hadid, P. Naudus, L. Lister, R. Green, D. Nguyen, V. Mukkavilli, C. Chmara, E. Sax, G. Barton, V. Selvaraj, A. Gupta, J. Fowler, and C. Gilbert, "The Flockbots," http://cs.gmu.edu/∼eclab/projects/robots/flockbots/, 2014.

[203] M. Dorigo, E. Tuci, V. Trianni, R. Groß, S. Nouyan, C. Ampatzis, T. H. Labella, R. O'Grady, M. Bonani, and F. Mondada, "SWARM-BOT: design and implementation of colonies of self-assembling robots," in *Computational Intelligence: Principles and Practice*, G. Y. Yen and D. B. Fogel, Eds. IEEE Computational Intelligence Society, 2006, ch. 6, pp. 103 – 135.

[204] K. Konolige, C. Ortiz, R. Vincent, B. Morisset, A. Agno, M. Eriksen, D. Fox, B. Limketkai, J. Ko, B. Stewart, and D. Schulz, "Centibots: Very large scale distributed robotic teams," in *Proceedings of IFIP Congress Topical Sessions*, 2004.

[205] S. Curtis, M. Brandt, G. Bowers, G. Brown, C. Cheung, C. Cooperider, M. Desch, N. Desch, J. Dorband, K. Gregory, K. Lee, A. Lunsford, F. Minetto, W. Truszkowski, R. Wesenberg, J. Vranish, M. Abrahantes, P. Clark, T. Capon, M. Weaker, R. Watson, P. Olivier, and M. L. Rilee, "Tetrahedral robotics for space exploration," *Aerospace and Electronic Systems Magazine, IEEE*, vol. 22, no. 6, pp. 22 –30, June 2007.

[206] MAGIC competition. [Online]. Available: http://www.dsto.defence.gov.au/MAGIC2010/

[207] G. Calderón-Meza and L. Sherry, "Analysis of stakeholder benefits of NextGen trajectory-based operations," in *Proceedings of Integrated Communications Navigation and Surveillance Conference (ICNS)*, 2010.

[208] UAS challenge. [Online]. Available: http://www.nasa.gov/directorates/armd/uaschallenge/index.html

[209] BORDERS competition. [Online]. Available: http://www.borders.arizona.edu/cms/announcements/borders-small-unmanned-aircraft-system-competition

[210] UAVForge. [Online]. Available: http://www.uavforge.net/uavhtml/

[211] M. Dastani, J. Dix, and P. Novak, "The first contest on multi-agent systems based on computational logic," in *Computational Logic in Multi-Agent Systems*, ser. Lecture Notes in Computer Science, F. Toni and P. Torroni, Eds. Springer Berlin / Heidelberg, 2006, vol. 3900, pp. 373–384.

[212] Multiagnet programming contest. [Online]. Available: http://multiagentcontest.org/

[213] L. Parker, "Cooperative robotics for multi-target observation," *Intelligent Automation and Soft Computing*, vol. 5, no. 1, pp. 5–19, 1999.

[214] ——, "Distributed algorithms for multi-robot observation of multiple moving targets," *Autonomous Robots*, vol. 12, no. 3, pp. 231–255, 2002.

[215] L. Parker and C. Touzet, "Multi-robot learning in a cooperative observation task," in *Robotic Systems 4*, L. Parker, G. Bekey, and J. Barhem, Eds. Springer, 2000, pp. 391–401.

[216] S. Luke, K. Sullivan, G. Balan, and L. Panait, "Tunably decentralized algorithms for cooperative target observation," in *Proceedings of Autonomous Agents and Multi-Agent Systems Conference (AAMAS)*, Utrecht Netherlands, July 2005.

[217] K. Sullivan and S. Luke, "Autonomous UUV control via decentralized algorithms," in *Proceedings of IEEE Autonomous Underwater Vehicles Workshop*, Sebasco Estates, ME, June 2004.

[218] I. Harmati and K. Skrzypczyk, "Robot team coordination for target tracking using fuzzy logic controller in game theoretic framework," *Robotics and Autonomous Systems*, vol. 57, no. 1, pp. 75 – 86, January 2009.

[219] C. J. Tomlin, G. J. Pappas, and S. S. Sastry, "Conflict resolution for air traffic management: A study in multiagent hybrid systems," *IEEE Transactions on Automatic Control*, vol. 43, no. 4, pp. 509 – 521, 1998.

[220] J. C. Hill, J. K. Archibald, W. C. Stirling, and R. L. Frost, "A multi-agent system architecture for distributed air traffic control," in *Proceedings of AIAA Guidance, Navigation and Control Conference*, 2005.

[221] K. Tumer and A. Agogino, "Improving air traffic management with a learning multiagent system," *IEEE Intelligent Systems*, vol. 24, no. 1, pp. 18 –21, Jan – Feb 2009.

[222] H. Santana, G. Ramalho, V. Corruble, and B. Ratitch, "Multi-agent patrolling with reinforcement learning," in *Proceedings of Autonomous Agents and Multi-Agent Systems Conference (AAMAS)*, 2004.

[223] M. P. Johnson, F. Fang, , and M. Tambe, "Patrol strategies to maximize pristine forest area," in *Proceedings of the American Association of Artificial Intelligence Conference (AAAI)*, 2012.

[224] M. P. Johnson, F. Fang, M. Tambe, and H. Albers, "Designing patrol strategies to maximize pristine forest area," in *Proceedings of Autonomous Agents and Multi-Agent Systems Conference (AAMAS)*, 2012.

[225] Z. Wang and V. Kumar, "Object closure and manipulation by multiple cooperating mobile robots," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2002.

[226] J. Spletzer, A. K. Das, R. Fierro, C. J. Taylor, V. Kumar, and J. P. Ostrowski, "Cooperative localization and control for multi-robot manipulation," in *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2001.

[227] M. J. Mataric, M. Nilsson, and K. T. Simsarian, "Cooperative multi-robot box pushing," in *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*, 1995.

[228] A. Zaerpooraa, M. N. Ahmadabadiaa, M. R. Baruniaa, and Z. Wang, "Distributed object transportation on a desired path based on constrain and move strategy," *Robotics and Autonomous Systems*, vol. 50, no. 2 – 3, pp. 115 – 128, 2005.

[229] A. Drogoul, J. Ferber, and A. D. J. Ferber, "From tom thumb to the dockers: Some experiments with foraging robots," in *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, 1992.

[230] D. Stillwell, "Toward the development of a material transport system using swarms of ant-like robots," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 1993.

[231] A. Martinoli and F. Mondada, "Collective and cooperative group behaviors: Biologically inspired experiments in robotics," in *Proceedings of the Fourth International Symposium on Experimental Robotics*, 1995.

[232] T. Wang and H. Zhang, "Collective sorting with multiple robots," in *Proceedings of International Conference on Robotics and Biomimetics*, 2004.

[233] O. Ali, B. S. Germain, J. V. Belle, P. Valckenaers, H. V. Brussel, and J. V. Noten, "Multi-agent coordination and control system for multi-vehicle agricultural operations," in *Proceedings of Autonomous Agents and Multi-Agent Systems Conference (AAMAS)*, Richland, SC, 2010, pp. 1621–1622.

[234] A. Gadre, "Learning strategies in multi-agent systems – applications to the herding problem," Master's thesis, Virginia Tech, 2001.

[235] S. E. Russell, D. Carr, M. Dragone, G. M. O'Hare, and R. W. Collier, "From bogtrotting to herding: a UCD perspective," *Annals of Mathematics and Artificial Intelligence*, vol. 61, no. 4, pp. 349–368, Apr. 2011.

[236] A. Schultz, J. J. Grefenstette, and W. Adams, "Robo-shepherd: Learning complex robotic behaviors," in *Robotics and Manufacturing: Recent Trends in Research and Applications*, vol. 6. ASME Press, 1996, pp. 763–768.

[237] C. H. Yong and R. Miikkulainen, "Cooperative coevolution of multi-agent systems," Department of Computer Sciences, The University of Texas at Austin, Tech. Rep. AI07-338, 2001.

[238] T. Balch and R. C. Arkin, "Behavior-based formation control for multirobot teams," *IEEE Transactions on Robotics*, vol. 14, no. 6, pp. 926 –939, December 1998.

[239] M. Egerstedt and X. Hu, "Formation constrained multi-agent control," *IEEE Transactions on Robotics*, vol. 17, no. 6, pp. 947 –951, December 2001.

[240] R. W. Beard, J. Lawton, and F. Y. Hadaegh, "A coordination architecture for spacecraft formation control," *IEEE Transactions on Control Systems Technology*, vol. 9, no. 6, pp. 777 –790, November 2001.

[241] J. Lawton, B. J. Young, and R. W. Beard, "A decentralized approach to elementary formation maneuvers," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, vol. 3, 2000, pp. 2728 –2733.

[242] R. Fierro, A. Das, V. Kumar, and J. Ostrowski, "Hybrid control of formations of robots," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1, 2001, pp. 157 – 162.

[243] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," in *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*, Switzerland, October 2002.

[244] M. Benewitz, W. Burgard, and S. Thrun, "Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots," *Robotics and Autonomous Systems*, vol. 41, no. 2 – 3, pp. 89 – 99, 2002.

[245] C. R. Kube and H. Zhang, "Collective robotic intelligence," in *Proceedings of Second International Conference on Simulation of Adaptive Behavior*, 1992.

[246] C. W. Reynolds, "Flocks, herds, and schools: A distributed behavioral model," *Computer Graphics*, vol. 21, no. 4, pp. 25 – 34, July 1987.

[247] R. Olfati-Saber, "Flocking for multi-agent dynamic systems: algorithms and theory," *Automatic Control, IEEE Transactions on*, vol. 51, no. 3, pp. 401 – 420, March 2006.

[248] Computer animation. [Online]. Available: http://www.h2g2.com/approved_entry/A3421045

[249] M. Turpin, N. Michael, and V. Kumar, "Trajectory design and control for aggressive formation flight with quadrotors," *Autonomous Robotics*, vol. 33, no. 1 – 2, pp. 143 – 156, 2012.

[250] Q. Lindsey, D. Mellinger, and V. Kumar, "Construction of cubic structures with quadrotor teams," in *Proceedings of Robotics: Science and Systems (RSS)*, 2011.

[251] D. Mellinger, M. Shomin, N. Michael, and V. Kumar, "Cooperative grasping and transport using multiple quadrotors," in *Distributed Autonomous Robotic Systems*, ser. Springer Tracts in Advanced Robotics, A. Martinoli, F. Mondada, N. Correll, G. Mermoud, M. Egerstedt, M. A. Hsieh, L. E. Parker, and K. Støy, Eds. Springer Berlin / Heidelberg, 2013, vol. 83, pp. 545–558.

[252] J. Wawerla, G. S. Sukhatme, and M. J. Matarić, "Collective construction with multiple robots," in *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[253] A. Stroupe, T. Huntsberger, A. Okon, H. Aghazarian, and M. Robinson, "Behavior-based multi-robot collaboration for autonomous construction tasks," in *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2005, pp. 1495 – 1500.

[254] Y. Meng and J. Gan, "A distributed swarm intelligence based algorithm for a cooperative multi-robot construction task," in *Proceedings of Symposium on Swarm Intelligence*, 2008, pp. 1 – 6.

[255] J. France and A. A. Ghorbani, "A multiagent system for optimizing urban traffic," in *Proceedings of IEEE International Conference on Intelligent Agent Technology*, October 2003, pp. 411 – 414.

[256] K. Dresner and P. Stone, "Multiagent traffic management: A reservation-based intersection control mechanism," in *Proceedings of Autonomous Agents and Multi-Agent Systems Conference (AAMAS)*, July 2004, pp. 530 – 537.

[257] G. Balan, "Computational issues in long-term fairness among groups of agents," Ph.D. dissertation, George Mason University, 2009.

[258] B. Abreu, L. Botelho, A. Cavallaro, D. Douxchamps, T. Ebrahimi, P. Figueiredo, B. Macq, B. Mory, L. Nunes, J. Orri, M. J. Trigueiros, and A. Violante, "Video-based multi-agent traffic surveillance system," in *Proceedings of the Intelligent Vehicles Conference*, 2000.

[259] D. Fox, W. Burgard, H. Kruppa, and S. Thrun, "A probabilistic approach to collaborative multi-robot localization," *Autonomous Robotics*, vol. 8, pp. 325 – 344, 2000.

[260] L. Paz, P. Jensfelt, J. Tardos, and J. Neira, "EKF SLAM updates in O(n) with divide and conquer SLAM," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, April 2007, pp. 1657–1663.

[261] L. M. Paz, J. D. Tardos, and J. Neira, "Divide and conquer EKF SLAM in O(n)," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1107 – 1120, 2008.

[262] H. Temeltas and D. Kayak, "SLAM for robot navigation," *IEEE Aerospace and Electronic Systems Magazine*, vol. 23, no. 12, pp. 16–19, Dec. 2008.

[263] L. Zhang, X. Meng, and Y. Chen, "Unscented transform for SLAM using Gaussian mixture model with particle filter," in *Proceedings of International Conference on Electronic Computer Technology*, Feb. 2009, pp. 12–17.

[264] S. Thrun, Y. Liu, D. Koller, A. Ng, Z. Ghahramani, and H. Durrant-Whyte, "Simultaneous localization and mapping with sparse extended information filters," *The International Journal of Robotics Research*, vol. 23, no. 7–8, pp. 693 – 716, 2004.

[265] R. Eustice, M. Walter, and J. Leonard, "Sparse extended information filters: insights into sparsification," in *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*, Aug. 2005, pp. 3281–3288.

[266] Y.-J. Lee and J.-B. Song, "Visual SLAM in indoor environments using autonomous detection and registration of objects," in *Proceedings of IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, Aug. 2008, pp. 671–676.

[267] D. Herath, S. Kodagoda, and G. Dissanayake, "New framework for simultaneous localization and mapping: Multi map SLAM," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, May 2008, pp. 1892–1897.

[268] Y.-J. Lee and J.-B. Song, "Autonomous selection, registration, and recognition of objects for visual SLAM in indoor environments," in *Proceedings of International Conference on Control, Automation, and Systems*, Oct. 2007, pp. 668–673.

[269] L. Paz, P. Pinies, J. Tardos, and J. Neira, "Large-scale 6-DOF SLAM with stereo-in-hand," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 946–957, Oct. 2008.

[270] D. Cole and P. Newman, "Using laser range data for 3D SLAM in outdoor environments," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, May 2006, pp. 1556–1563.

[271] D. Fox, W. Burgard, H. Kruppa, and S. Thrun, "Collaborative multi-robot localization," in *Proceedings of German Conference on Artificial Intelligence*, 1999.

[272] ——, "Efficient multi-robot localization based on Monte Carlo approximation," in *Proceedings of International Symposium of Robotics Research*, 1999.

[273] R. Madhavan, K. Fregene, and L. E. Parker, "Distributed heterogeneous outdoor multi-robot localization," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, May 2002.

[274] S. I. Roumeliotis and G. A. Bekey, "Distributed multi-robot localization," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, vol. 18, 2002, pp. 781 – 795.

[275] T. C. H. Sit, Z. Liu, M. H. A. Jr., and W. K. G. Seah, "Multi-robot mobility enhanced hop-count based localization in ad hoc networks," *Robotics and Autonomous Systems*, vol. 55, pp. 244 – 252, 2007.

[276] I. M. Rekleitis, G. Dudek, and E. E. Milios, "Multi-robot cooperative localization: A study of trade-offs between efficiency and accuracy," in *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[277] A. I. Mourikis and S. I. Roumeliotis, "Performance analysis of multirobot cooperative localization," *IEEE Transactions on Robotics*, 2006.

[278] M. Moors, F. Schneider, and D. Wildermuth, "Relative position estimation in a group of robots," in *Proceedings of the International Conference on Climbing and Walking Robots*, 2003.

[279] A. Gasparri, S. Panzieri, and F. Pascucci, "A fast conjunctive resampling particle filter for collaborative multi-robot localization," in *Proceedings of Autonomous Agents and Multi-Agent Systems Conference (AAMAS)*, 2008.

[280] R. Grabowski, L. E. Navarro-Serment, C. J. Paredis, and P. K. Khosla, "Heterogenous teams of modular robots," in *Robot Teams*, T. Balch and L. E. Parker, Eds. A. K. Peters, 2002, pp. 293 – 314.

[281] W. Burgard, M. Moors, D. Fox, and R. S. S. Thrun, "Collaborative multi-robot exploration," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2000.

[282] W. Burgard, M. Moors, C. Stachniss, and F. Schneider, "Coordinated multi-robot exploration," *IEEE Transactions on Robotics*, vol. 21, no. 5, pp. 376 – 386, June 2005.

[283] J. Ko, B. Stewart, D. Fox, K. Konolige, and B. Limketkai, "A practical, decision-theoretic approach to multi-robot mapping and exploration," in *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2003.

[284] D. Fox, J. Ko, K. Konolige, B. Limketkai, D. Schultz, and B. Stewart, "Distributed multi-robot exploration and mapping," in *Proceedings of the IEEE*, vol. 94, no. 7, July 2006, pp. 1325 – 1339.

[285] A. Howard, "Multi-robot simultaneous localization and mapping using particle filters," in *Proceedings of Robotics: Science and Systems (RSS)*, Cambridge, USA, June 2005.

[286] ——, "Multi-robot simultaneous localization and mapping using particle filters," *The International Journal of Robotics Research*, vol. 25, no. 12, pp. 1243 – 1256, December 2006.

[287] X. S. Zhou and S. I. Roumeliotis, "Multi-robot SLAM with unknown initial correspondence: The robot rendezvous case," in *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2006.

[288] M. Di Marco, A. Garulli, A. Giannitrapani, and A. Vicino, "Simultaneous localization and map building for a team of cooperating robots: a set membership approach," *IEEE Transactions on Robotics*, vol. 19, no. 2, pp. 238–249, Apr 2003.

[289] J. Nieto, J. Guivant, E. Nebot, and S. Thrun, "Real time data association for FastSLAM," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[290] S.-Y. Chien, H. Wang, and M. Lewis, "Human vs. algorithmic path planning for search and rescue by robot teams," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 2010.

[291] R. Hodoshima, M. Guarnieri, R. Kurazume, H. Masuda, T. Inoh, P. Debenest, E. F. Fukushima, and S. Hirose, "HELIOS tracked robot team: Mobile RT system for special urban search and rescue operations," *Journal of Robotics and Mechatronics*, vol. 23, no. 6, pp. 1041 – 1054, 2011.

[292] C. Brom, J. Vyhnánek, J. Lukavská, D. Waller, and R. Kadlec, "A computational model of the allocentric and egocentric spatial memory by means of virtual agents, or how simple virtual agents can help to build complex computational models," *Cognitive Systems Research*, vol. 17 – 18, no. 0, pp. 1 – 24, 2012.

[293] B. Moulin, W. Chaker, J. Perron, P. Pelletier, J. Hogan, and E. Gbei, "MAGS project: Multi-agent geosimulation and crowd simulation," in *Spatial Information Theory. Foundations of Geographic Information Science*, W. Kuhn, M. Worboys, and S. Timpf, Eds. Springer Berlin / Heidelberg, 2003, vol. 2825, pp. 151–168.

[294] B. Ulicny and D. Thalmann, "Crowd simulation for interactive virtual environments and vr training systems," in *Computer Animation and Simulation*, ser. Eurographics, N. Magnenat-Thalmann, D. Thalmann, W. Hansmann, W. Purgathofer, and F. Sillion, Eds. Springer Vienna, 2001, pp. 163–170.

[295] W. Li and J. M. Allbeck, "Populations with purpose," in *Proceedings of the 4th international conference on Motion in Games*, 2011, pp. 132–143.

[296] K. Sullivan and S. Luke, "Analyzing relocatable targets using multiagent simulation and evolutionary computation," in *Proceedings of 72nd Military Operations Research Society Annual Symposium*, June 2004.

[297] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan, "MASON: A multiagent simulation environment," *Simulation*, vol. 81, no. 7, pp. 517 – 527, July 2005.

[298] K. Sullivan, S. Luke, and B. Hrolenok, "Can you do me a favor?" in *Proceedings of Trust in Agent Societies Workshop*, 2010.

[299] M. I. Lichbach, *The cooperator's dilemma*. University of Michigan Press, 1996.

[300] Y. Shoham, R. Powers, and T. Grenager, "On the agenda(s) of research on multi-agent learning," in *Proceedings of Artificial Multiagent Learning. Papers from the 2004 AAAI Fall Symposium. Technical Report FS-04-02*, 2004.

[301] B. Banerjee, R. Mukherjee, and S. Sen, "Learning mutual trust," in *Working Notes of AGENTS-00 Workshop on Deception, Fraud and Trust in Agent Societies*, 2000, pp. 9–14.

[302] R. Mukherjee and S. Sen, "Towards a pareto-optimal solution in general-sum games," in *Agents-2001 Workshop on Learning Agents*, 2001.

[303] L. Panait, K. Sullivan, and S. Luke, "Lenience towards teammates helps in cooperative multiagent learning," in *Proceedings of Autonomous Agents and Multi-Agent Systems Conference (AAMAS)*, Hakodate, Japan, May 2006.

[304] ——, "Lenient learners in cooperative multiagent systems," in *Proceedings of Autonomous Agents and Multi-Agent Systems Conference (AAMAS)*. ACM, 2006, pp. 801–803.

[305] K. Sullivan, L. Panait, G. Balan, and S. Luke, "Can good learners always compensate for poor learners?" in *Proceedings of Autonomous Agents and Multi-Agent Systems Conference (AAMAS)*, Hakodate, Japan, May 2006.

[306] M. Potter and K. De Jong, "A cooperative coevolutionary approach to function optimization," in *Proceedings of the Parallel Problem Solving from Nature Conference*, 1994, pp. 249–257.

[307] L. Bull, "Evolutionary computing in multi-agent environments: Partners," in *Proceedings of the Seventh International Conference on Genetic Algorithms*, T. Back, Ed. Morgan Kaufmann, 1997, pp. 370–377.

[308] R. P. Wiegand, W. Liles, and K. De Jong, "An empirical analysis of collaboration methods in cooperative coevolutionary algorithms," in *Proceedings of the 2001 Genetic and Evolutionary Computation Conference (GECCO-2001)*, E. Cantu-Paz *et al*, Ed., 2001, pp. 1235–1242.

[309] S. Kapetanakis and D. Kudenko, "Reinforcement learning of coordination in cooperative multi-agent systems," in *Proceedings of the American Association of Artificial Intelligence Conference (AAAI)*, 2002.

[310] K. Sullivan, S. Luke, and V. A. Ziparo, "Hierarchical learning from demonstration on humanoid robots," in *Proceedings of Humanoid Robots Learning from Human Interaction Workshop*, Nashville, TN, 2010.

[311] K. Sullivan, B. Davidson, C. Vo, B. Hrolenok, and S. Luke, "RoboPatriots: George Mason University 2008 RoboCup team," in *Proceedings of the 2008 RoboCup Workshop*, 2008.

[312] K. Sullivan, C. Vo, B. Hrolenok, and S. Luke, "RoboPatriots: George Mason University 2009 RoboCup team," in *Proceedings of the 2009 RoboCup Workshop*, 2009.

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

[314] K. Sullivan, C. Vo, and S. Luke, "RoboPatriots: George Mason University 2011 RoboCup team," in *Proceedings of the 2011 RoboCup Workshop*, 2011.

[315] K. Sullivan, K. Russell, K. Andrea, B. Stout, and S. Luke, "RoboPatriots: George Mason University 2012 RoboCup team," in *Proceedings of the 2012 RoboCup Workshop*, 2012.

[316] M. Lagarde, P. Andry, C. Giovannangeli, and P. Gaussier, "Learning new behaviors: Toward a control architecture merging spatial and temporal modalities." in *Proceedings of Robotics: Science and Systems (RSS)Workshop on Interactive Robot Learning*, 2008.

[317] M. T. Gervasio and J. L. Murdock, "What were you thinking? Filling in missing dataflow through inference in learning from demonstration," in *Proceedings of Intelligent User Interfaces (IUI)*, 2009.

[318] M. Gamon, "Sentiment classification on customer feedback data: noisy data, large feature vectors, and the role of linguistic analysis," in *Proceedings of the 20th International Conference on Computational Linguistics*, 2004.

[319] E. Kalapanidas, N. Avouris, M. Craciun, and D. Neagu, "Machine learning algorithms: A study on noise sensitivity," in *Proceedings of First Balkan Conference in Informatics*, 2003.

[320] D. F. Nettleton, A. Orriols-Puig, and A. Fornells, "A study of the effect of different types of noise on the precision of supervised learning techniques," *Artificial Intelligence Review*, vol. 33, pp. 275–306, 2010.

[321] J. Huang, A. J. Smola, A. Gretton, K. M. Borgwardt, and B. Schölkopf, "Correcting sample bias by unlabeled data," in *Proceedings of Neural Information Processing Conference (NIPS)*, 2006.

[322] T. S. Furey, N. Cristianini, N. Duffy, D. W. Bednarski, M. Schummer, and D. Haussler, "Support vector machine classification and validation of cancer tissue samples using microarray expression data," *Bioinformatics*, vol. 16, no. 10, pp. 906–914, 2000.

[323] K. Lakshminarayan, S. A. Harp, R. Goldman, and T. Samad, "Imputation of missing data using machine learning techniques," in *Proceedings of Knowledge Discovery and Data Mining (KDD)*, 1996.

[324] J. M. Jereza, I. Molinab, P. J. García-Laencinac, E. Albad, N. Ribellesd, M. Martíne, and L. Francoa, "Missing data imputation using statistical and machine learning methods in a real breast cancer problem," *Artificial Intelligence in Medicine*, vol. 50, no. 2, pp. 105–115, 2010.

[325] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," in *Second European Conference on Computational Learning Theory*, 1995, pp. 23–37.

[326] N. A. Syed, H. Liu, and K. K. Sung, "Handling concept drifts in incremental learning with support vector machines," in *Proceedings of Knowledge Discovery and Data Mining (KDD)*, 1999.

[327] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental Bayesian approach testing on 101 object categories," *Computer Vision and Image Understanding*, vol. 106, no. 1, pp. 59–70, 2007.

[328] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang, "Incremental learning for robust visual tracking," *International Journal of Computer Vision*, vol. 77, no. 1–3, pp. 125–141, 2008.

[329] J. G. V. Ganti, R. Ramakrishnan, and W.-Y. Loh, "BOAT — optimistic decision tree construction," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, vol. 28, 1999, pp. 169–180.

[330] R. S. Michalski, I. Mozetic, J. Hong, and N. Lavrac, "The multi-purpose incremental learning system AQ15 and its testing application to three medical domains," in *Proceedings of the American Association of Artificial Intelligence Conference (AAAI)*, 1986.

[331] R. Polikar, L. Upda, S. S. Upda, and V. Honavar, "Learn++: An incremental learning algorithm for supervised neural networks," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 31, no. 4, pp. 497–508, Nov 2001.

[332] G. Cauwenberghs and T. Poggio, "Incremental and decremental support vector machine learning," in *Proceedings of Neural Information Processing Conference (NIPS)*, 2001.

[333] C. P. Diehl and G. Cauwenberghs, "SVM incremental learning, adaptation and optimization," in *Proceedings of Neural Information Processing Conference (NIPS)*, vol. 4, 2003, pp. 2685–2690.

[334] J. Ma, J. Theiler, and S. Perkins, "Accurate on-line support vector regression," *Neural Computation*, vol. 15, pp. 2683–2703, 2003.

[335] K. Sullivan, A. Molla, B. Squires, and S. Luke, "Unlearning from demonstration," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

[336] K. Bache and M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[337] K. Sullivan, E. Wei, D. Wicke, and S. Luke, "Training heterogeneous teams of robots," in *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*, submitted.

[338] K. Sullivan and S. Luke, "Real-time training of team soccer behaviors," in *Proceedings of the RoboCup Symposium*, 2012.

[339] P. Stone, R. S. Sutton, and G. Kuhlmann, "Reinforcement learning for RoboCup-soccer keepaway," *Adaptive Behavior*, vol. 13, no. 3, pp. 165–188, 2005.

[340] M. M. Veloso, P. Stone, and M. Bowling, "Anticipation as a key for collaboration in a team of agents: A case study in robotic soccer," in *Proceedings of SPIE Sensor Fusion and Decentralized Control in Robotic Systems*, 1999, pp. 134 – 143.

[341] K. Sullivan and S. Luke, "Learning from demonstration with swarm hierarchies," in *Proceedings of Autonomous Agents and Multi-Agent Systems Conference (AAMAS)*, 2012.

[342] E. Jones, B. Browning, M. B. Dias, B. Argall, M. Veloso, and A. T. Stentz, "Dynamically formed heterogeneous robot teams performing tightly-coordinated tasks," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2006, pp. 570 – 575.

[343] M. Mataric, M. Nilsson, and K. Simsarin, "Cooperative multi-robot box-pushing," in *Intelligent Robots and Systems 95: Human Robot Interaction and Cooperative Robots.*, vol. 3, Aug 1995, pp. 556–561 vol.3.

[344] A. G. Barto, G. Konidaris, and C. Vigorito, "Behavioral hierarchy: Exploration and representation," in *Computational and Robotic Models of the Hierarchical Organization of Behavior*. Springer, 2013, pp. 13–46.

[345] C. Cioffi-Revilla, S. Luke, D. C. Parker, J. D. Rogers, W. W. Fitzhugh, W. Honeychurch, B. Fr ohlich, P. D. Priest, and C. Amartuvshin, "Agent-based modeling simulation of social adaptation and long-term change in inner asia," in *Proceedings of the First World Congress on Advancing Social Simulation*, 2007.

[346] I. Skoggard and W. Kennedy, "An interdisciplinary approach to agent-based modeling of conflict in eastern africa," *Practicing Anthropology*, vol. 35, no. 1, pp. 29 – 33, 2013.

[347] R. L. Axtell, J. M. Epstein, J. S. Dean, G. J. Gumerman, A. C. Swedlund, J. Harburger, S. Chakravarty, R. Hammond, J. Parker, and M. Parker, "Population growth and collapse in a multiagent model of the kayenta anasazi in long house valley," in *Proceedings of the National Academy of Science*, vol. 99, 2002.

[348] K. Sullivan and S. Luke, "Evolving kernel for support vector machine classification," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, D. Thierens, Ed., vol. II, London, England, July 2007, pp. 1702 – 1707.

[349] K. Sullivan, S. Luke, C. Larock, S. Cier, and S. Armentrout, "Opportunistic evolution: Efficient evolutionary computation on large-scale computational grids," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) Late Breaking Papers*, 2008.

[350] D. Rothman, S. Luke, and K. Sullivan, "Do multiple trials help univariate methods?" in *Proceedings of the Congress on Evolutionary Computation Conference (CEC)*, 2011.

[351] S. Luke, K. Sullivan, and F. Abidi, "Large scale empirical analysis of cooperative coevolution," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2011.

[352] K. Sullivan, M. Coletti, and S. Luke, "GeoMason: Geospatial support for MASON," in *Proceedings of NSF Franco-American Workshop on Spatial Models of Social Complexity*, 2011.

[353] Gumstix inc. [Online]. Available: http://www.gumstix.com

[354] A. Rowe, A. G. Goode, D. Goel, and I. Nourbakhsh, "CMUcam3: an open programmable embedded vision sensor," Robotics Institute, Carnegie Mellon University, See also www. cmucam.org. CMU-RI-TR-07-13, 2007.

[355] Surveyor stereo vision system (SVS). [Online]. Available: http://www.surveyor.com/stereo/stereo_info.html

BIOGRAPHY

Keith M Sullivan graduated from West Lafayette Jr./Sr. High School in 1993. He received a Bachelor of Science in Mathematics and Bachelor of Arts in Journalism from Indiana University in 1998. After working for the Navy in Rhode Island, he returned to graduate school, receiving his Masters of Science in Computer Science in 2005 and PhD in Computer Science in 2015, both from George Mason University. His research interests include robotics, multiagent learning, and stochastic optimization. He helped develop MASON, an open-source Java multiagent simulation toolkit, and ECJ, a Java-based, open-source evolutionary computation toolkit. He created the RoboPatriots, GMU's humanoid robot soccer team, and co-developed the FlockBots, an open-source differential drive robot meant for embodied multiagent systems research.