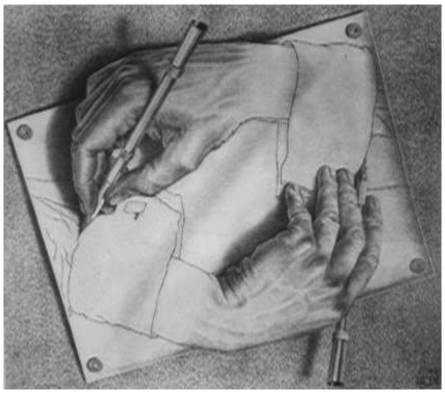
Chapter 2

Background and Related Work



This research concerns *reinforcement-learning* in *behavior-based multiagent* robot teams. Separately, each of these areas is backed by a significant body of research. Reinforcement-learning and behavior-based systems are especially well established, but the intersection of these two with multiagent systems is an emerging field. This chapter reviews the core research in robotic reinforcement learning, behavior-based robotics and learning multiagent robotic systems. The goal is to establish a base from which the research proceeds and to differentiate it from similar work.

2.1 Behavior-based robotics

In general, a robot is to satisfy a goal by selecting and executing a sequence of actions to achieve it. The goal may be a location to reach or a state of the world to be obtained (e.g. "put the soda cans in the wastebasket"). Sensors guide the selection of actions along the way. The "classic AI" approach to this problem is for the robot to process and store sensor readings with the aim of developing and maintaining an internal model of the external world. Deliberative strategies reason over the model and develop a plan (sequence of actions) for achieving the goal. Usually, intermediate failures trigger a complete re-planning. A famous example of the classic approach is Shakey the Robot [Nil84].

In contrast, over the last decade a "new wave" of robotics researchers have advanced a behavior-based view. Their central theme is to directly couple a robot's sensors and actuators so as to avoid the trouble of maintaining a model or deliberating over it. Rodney Brooks' subsumption architecture and Arkin's motor schemas are early examples [Bro86, Ark89]. Brooks asserts that world models are useless; the best model is the world itself. Space precludes a detailed review of the many behavior-based systems implemented since 1985, but some important qualities they all share include

Tight sensor to motor coupling: Sensor input is minimally processed before motor actions are selected.

Minimal representation: Many behavior-based systems do not maintain any internal state, or representation of the world at all. Some use just a few bits of memory.

Speed: Since the computational costs of reasoning over and maintaining a world model are avoided, behavior-based systems usually interact with their environment more quickly than "classic AI" systems.

Composition: The behavioral system is composed of several separate primitive behaviors. They may be arranged in layers [Bro86], or run in parallel [Ark89].

Emergent properties: The overall behavior of the system emerges through the interaction of of the several primitive behaviors.

Robust performance in dynamic environments: Since behavior-based systems are primarily sensor-oriented, they quickly respond to changes in the environment.

Two frequently raised concerns regarding behavior-based approaches are

Behaviors are hand-coded: Many existing behavior-based robotic systems *are* comprised of hand-coded behaviors. To address this, researchers are investigating automatic approaches including genetic algorithms [PAR92] and reinforcement-learning [MC92].

"The fly at the window": A fly buzzing at a window is a classic example of a behavior-based system caught in a local minimum. Behavior-based systems are often unable to recognize failure and select alternative strategies. This is one reason some argue for the integration of deliberative systems (able to detect a failure) and reactive systems (able to act quickly in the dynamic environment). Some approaches avoid the local minimum problem altogether [CG93], or use short term memory-based mechanisms to move out of them [BA93].

2.2 Motor schema-based control

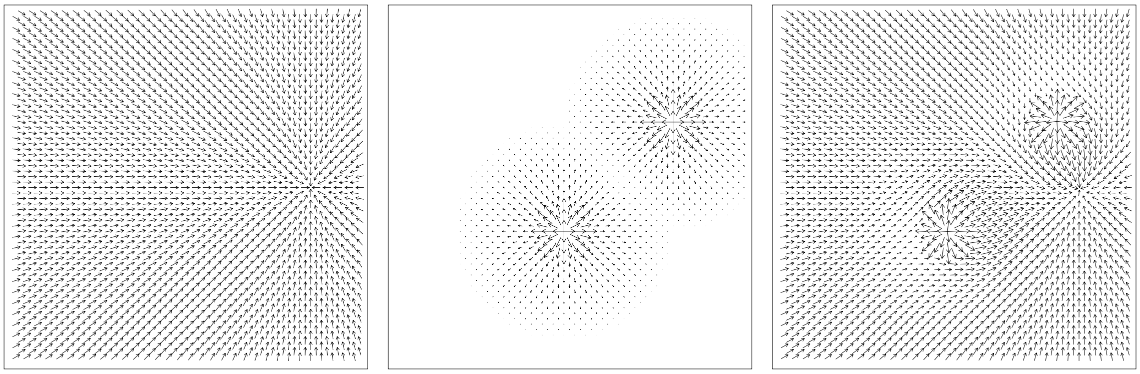


Figure 2.1: Motor schema example. The diagram on the left shows a vector field corresponding to a **move-to-goal** schema, pulling the robot to a location on the right. The center diagram shows an **avoid-obstacles** field, repelling the robot from two sensed obstacles. On the right, the two schemas are summed, resulting in a complete behavior for reaching the goal. It is important to note that the entire field is never computed, only the vectors for the robot's current location.

Motor schemas are an important example of behavior-based robot control. The motor schema paradigm is the central method in use at the Georgia Tech Mobile Robot Laboratory and is platform for this research.

Motor schemas are the reactive component of Arkin's Autonomous Robot Architecture (AuRA) [AB97]. AuRA's design integrates deliberative planning at a top level with behavior-based motor control at the bottom. The lower levels, concerned with executing the reactive behaviors are incorporated in this research.

Individual motor schemas, or primitive behaviors, express separate goals or constraints for a task. As an example, important schemas for a navigational task would include **avoid_obstacles** and **move_to_goal**. Since schemas are independent, they can run concurrently, providing parallelism and speed. Sensor input is processed by perceptual schemas embedded in the motor behaviors. Perceptual processing is minimal and provides just the information pertinent to the motor schema. For instance, a **find_obstacles** perceptual schema which provides a list of sensed obstacles is embedded in the **avoid_obstacles** motor schema.

The concurrently running motor schemas are integrated as follows. First, each produces a vector indicating the direction the robot should move to satisfy that schema's goal or constraint. The magnitude of the vector indicates the importance of achieving it. It is not so critical, for instance, to avoid an obstacle if it is distant, but crucial if close by. The magnitude of the **avoid_obstacle** vector is correspondingly small for distant obstacles and large for close ones. The importance of motor schemas relative to each other is indicated by a gain value for each one. The gain is usually set by a human designer, but may also be determined through automatic means, including on-line learning [CAR92], case-based reasoning [RS93] or genetic algorithms [PAR92]. Each motor vector is multiplied by the associated gain value and the results are summed and normalized. The resultant vector is sent to the robot hardware for execution. An example of this process is illustrated in Figure 2.1.

The approach bears a strong resemblance to potential field methods [CG93], but with an important difference: the entire field is never computed. In the example figure an entire field is shown, but this is only for visualization purposes. The robot only computes the vectors that apply to its present location and perceptual state.

2.2.1 Temporal sequencing of behavioral assemblages

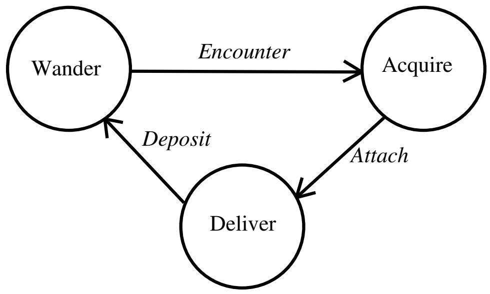


Figure 2.2: The forage FSA.

As illustrated above for navigation, motor schemas may be grouped to form more complex, emergent behaviors. Groups of behaviors are referred to as *behavioral assemblages* . One way behavioral assemblages may be used in solving complex tasks is to develop an assemblage for each sub-task and to execute the assemblages in an appropriate sequence. The steps in the sequence are separate *behavioral*

states. Perceptual events that cause transitions from one behavioral state to another are called *perceptual triggers*. A resulting task solving strategy can be represented as a Finite State Automaton (FSA). This technique is referred to as *temporal sequencing* [AM94].

As an example task where temporal sequencing is useful, consider foraging. Robot foraging behaviors have been examined in detail at Georgia Tech (see [BA95a] and [ABN93]). The forage task for a robot is to wander about the environment looking for items of interest (attractors). Upon encountering one of these attractors, the robot moves towards it, finally attaching itself. After attachment the robot navigates to the homebase where it deposits the attractor.

In this approach to solving the forage task, a robot can be in one of three behavioral states: *wander*, *acquire* and *deliver*. The robot begins in the *wander* state. If there are no attractors within the robot's field of view, the robot remains in *wander* until one is encountered. When an attractor is encountered, a transition to the *acquire* state is triggered. While in the *acquire* state, the robot moves towards the attractor and when it is sufficiently close, attaches to it. The last state, *deliver*, is triggered when the robot attaches to the attractor. While in the *deliver* state the robot carries the attractor back to home base. Upon reaching home base, the robot deposits the attractor there and reverts back to the *wander* state. An FSA for the forage task is illustrated in Figure 2.2.

2.3 Learning in behavior-based systems

Most of the behavior-based approaches presented so far in this chapter focus on static behavioral configurations. Even though behavior-based approaches are robust for many tasks and environments, they are not necessarily adaptive. We now consider some of the ways learning can be integrated into a behavior-based system.

2.3.1 Reinforcement-learning

Reinforcement learning offers a powerful set of techniques that allow a robot to learn a task without requiring its designer to fully specify how it should be carried out. If the task is feasible and feedback regarding how well the agent is doing is provided, several reinforcement learning techniques are *guaranteed* to converge (within an arbitrary ϵ) to the *optimal* solution [WD92, TVR96]. The guarantees are tempered by rather strong conditions for convergence; Q-learning for example, requires all actions to be repeatedly sampled in all states.

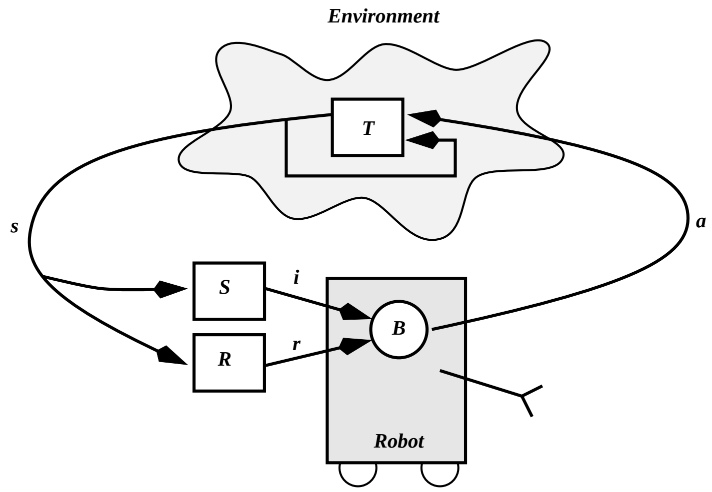


Figure 2.3: Typical model of robotic reinforcement-learning (adapted from Kaelbling).

A standard model of robotic reinforcement-learning is illustrated in Figure 2.3 [KLM96]. At each step of interaction, the robot receives input from the environment, i, which may be filtered by the robot's sensors S. It also receives a reinforcement signal r, which depends on conditions in the environment and R, the reinforcement function. r is an indication to the robot of how well it is performing. The agent's behavior B should choose actions that maximize the sum of reinforcement signals: The robot's action, a, effects a change in the environment modeled by T. Depending on the type of learning, the agent may or may not be provided T and R.

If T and R are known, an optimal B may be found using dynamic programming [Bel57]. Once the optimal behavior has been determined, the reinforcement signal r, is unnecessary and the behavior depends only on the current perceived state of the environment i. A function that selects an action depending on i is referred to as a *policy* and is denoted by $\pi(i)$. Optimal policies are labeled $\pi^*(i)$. Even though these approaches may generate optimal policies it often takes too long for practical applications [LDK95].

For many robot tasks it is not reasonable to expect that perfect models of interaction with the environment can be known a priori. When T and R are not provided, the robot must learn the best action for each situation through trial and error. Research in reinforcement learning centers on developing algorithms to compute a policy that converges to optimal as quickly as possible. Example algorithms include Dyna [Sut91], $TD(\lambda)$ [Tes92], Adaptive Heuristic Critic (AHC) [BSW90], Model-Based [AAMR88] and Q-learning [WD92].

Although reinforcement-learning bears a resemblance to supervised learning, the two are distinct. In supervised learning, the agent is presented with input/output pairs, where the given output is presumably the best choice of action. The input/output pairs correspond to s and a of the reinforcement-learning model. In the case of reinforcement-learning, the agent is provided with the present state, a reward and the next state (s_t, r_t, s_{t+1}), but the robot is not told which action would be in its best interest. It can only discover this by testing actions and evaluating rewards.

Learning agents strive for optimal performance, but what is "optimal?" It was mentioned earlier that the robot should attempt to maximize r over time. The length of time however, is important. If we know the length of an agent's life for instance, the best course is to maximize the sum of r over that entire period (this corresponds to a finite-horizon). Reinforcement-learning methods differ in their definitions of optimality; three views predominate:

- **Finite-horizon** where the the agent takes actions that maximize the sum of rewards over a finite number of steps.
- **Average-reward** where the the agent takes actions that maximize the average reward over all steps in the future.
- **Infinite-horizon discounted** where the the agent takes actions that maximize the reward over all future steps, but the future rewards are discounted geometrically.

Another important distinction between the various reinforcement-learning methods concerns their use of models. In *model-based* algorithms, the agent learns approximations to T and R, then utilizes the models to develop a policy. In *model-free* methods the agent learns the policy directly without a model. In this research we will focus on Watkins' model-free Q-learning.

2.3.2 Q-learning

Q-learning is a type of reinforcement-learning in which the value of taking each possible action in each situation is represented as a utility function, $Q(s, a)$. Where s the state or situation and a is a possible action. For the purposes of discussing Q-learning, assume the robot's sensors pass the world state on to the agent unmodified (thus s is used instead of i to signify the state). If the function is properly computed, an agent can act optimally simply by looking up the best action for any situation. The problem is to compute the $Q(s,a)$ that provides an optimal policy. Watkins [WD92] has developed an algorithm for determining $Q(s,a)$ that converges to optimal. Watkins' prefers to represent $Q(s,a)$ as a table, $Q[s,a]$ and asserts in [WD92] that the algorithm is not guaranteed to converge otherwise.

Q-learning agents seek to maximize the infinite-horizon discounted sum of r. $Q[s,a]$ is the value of choosing action a in situation s. The policy for a robot using Q-values is to chose the action that maximizes $Q[s,a]$. So

$$\pi(s) = \mathrm{argmax}_a Q[s,a]; \tag{2.1}$$

The problem then is to compute and update Q-values based on interaction with the environment. In discussing Q-values, it is useful to consider the value of reaching a particular state. For a state s_t the value of s_t is defined as:

$$V(s_t) \quad = \quad R(s_t,a_t) + \gamma R(s_{t+1},a_{t+1}) + \gamma^2 R(s_{t+2},a_{t+2}) + \gamma^3 R(s_{t+3},a_{t+3}) \ldots$$

$R(s_t,a_t)$ is the reward for being in state s_t and executing action a_t; s_{t+1} is the succeeding state. γ is the discount factor with $\gamma < 1$. $V(s_t)$ therefore, is the sum of all future rewards discounted at the rate γ. $V = V^*$ if the states are reached following an optimal policy. $V(s_t)$ may also be written recursively as

$$V(s_t) \quad = \quad R(s_t,a_t) + \gamma V(s_{t+1})$$

If each $Q[s,a]$ were set as follows, the policy of selecting actions based on them would be optimal:

$$Q[s_t,a_t] \quad = \quad R(s_t,a_t) + \gamma V^*(s_{t+1})$$

where s_{t+1} is the state reached by applying action a_t in state s_t and subsequent actions are selected optimally. Unfortunately $Q[s,a]$ cannot be computed directly because $R(s)$ and $V(s)$ aren't known initially. Watkins [WD92] introduced the following scheme for updating Q-values as an agent interacts with the environment and receives rewards:

$$Q[s_t, a_t] \quad = \quad (1-\alpha) \underbrace{Q[s_t, a_t]}_{\text{old value}} + \alpha \underbrace{(R(s_{t+1}, a_{t+1}) + \gamma \max_u Q[s_{t+1}, u])}_{\text{improved estimate}} \tag{2.2}$$

This update is applied each time action a_t "fires" in state s_t. The first term is the old Q-value, while the second is an improved estimate based on an actual reward and the estimated value of the subsequent state. α is a learning rate that indicates how much "trust" should be given the improved estimate. In the second term, $\max_u Q(s_{t+1}, u)$ is an approximation of $V(s_{t+1})$ (if $Q = Q^*$ it is easy to show that $V^*(s) = \max_u Q[s, u]$). Watkins proved this iteration converges to Q^* under the condition that the learning set includes an infinite number of episodes for each state and action. This is a strong condition, but under the stochastic conditions of his theorem, no method could be guaranteed to find the optimal policy under weaker conditions.

2.3.3 Dyna

Model-free systems like Q-learning are computationally simple, but require many experience steps to converge. Model-based systems seek to reduce the cost of experience in the real-world (as in risk of damage to the robot) by using experience to model interaction with the world, then developing a policy based on the model.

Dyna [Sut91], like Q-learning, represents the utility of executing a particular action a in a particular state s as $Q(s, a)$, but it uses models of T and R to compute Q. The models of T and R are referred to as $\hat{T}$ and $\hat{R}$ respectively. Carrying forward the notation introduced earlier for Q-learning, we will consider $Q, \hat{T}$, and $\hat{R}$ as tables.

At each step of interaction with the environment, Dyna records an experience-tuple, (s_t, a_t, s_{t+1}, r_t). Where action a_t applied in s_t results in a new state s_{t+1} and reward r_t. Next $\hat{T}$ and $\hat{R}$ are updated based on the observation using a simple statistical model. $\hat{T}[s_t, a_t, s_{t+1}]$ is the probability that state s_{t+1} results from applying action a_t in state s_t. Similarly $\hat{R}[s_t, a_t]$ is the estimated reward for executing a_t in state s_t. Also, at each step, $Q[s_t, a_t]$ is updated as follows:

$$Q[s_t, a_t] \quad = \quad \hat{R}[s_t, a_t] + \gamma \sum_{s_{t+1}} \hat{T}[s_t, a_t, s_{t+1}] \max u Q[s_{t+1}, u] \tag{2.3}$$

Dyna also performs a fixed number of additional updates to random state-action pairs, as follows:

$$Q[s_k, a_k] \quad = \quad \hat{R}[s_k, a_k] + \gamma \sum_{s_{t+1}} \hat{T}[s_k, a_k, s_{t+1}] \max uQ[s_{t+1}, u] \tag{2.4}$$

Finally, as in Q-learning, Dyna uses the Q-values to select actions.

Kaelbling evaluated Dyna and Q-learning in a simulated navigational task [KLM96]. She found Dyna to require an order of magnitude fewer steps of experience than Q-learning to converge to an optimal policy, but Dyna uses about six times more compute cycles.

2.3.4 Learning component behaviors

Reinforcement-learning is one way for a robot to learn appropriate sequences of action to attain a goal. Mahadevan and Connell [MC92] have applied Q-learning in a slightly different manner: to learn the component behaviors within a pre-defined sequence. The particular task they investigate is for a robot to find, then push a box across a room. They pre-define three behavioral states **F, P** and **U** for find-box, push-box and unwedge-box respectively; they also define conditions under which the robot transitions from one state to another. Separate reinforcement functions and tables of Q-values apply for each state.

The state vector s is composed of local sonar occupancy information, infra-red bump sensors and a "stuck" sensor. The possible actions are: go forward, turn left, turn hard left, turn right and turn hard right. Since the state space is rather large, Mahadevan sought ways to reduce it, including weighted Hamming distance and statistical clustering to group similar states. Using this approach, their robot, OBELIX was able to learn to perform better than hand-coded behaviors for box-pushing.

Mahdevan's sequence of behaviors is similar to the temporal-sequencing approach outlined earlier. An important difference is that learning takes place in the behavioral states. The significance of Mahadevan's result is that Q-learning is useful in learning sequences within sequences of behaviors; it may be applied at several levels.

2.3.5 Learning a hierarchy of behaviors

In research at Carnegie Mellon University [Lin93], Lin developed a method for Q-learning to be applied hierarchically, so that complex tasks are learned at several levels. He argues that by by decomposing the task into sub-tasks and learning at the sub-task and task level, the overall rate of learning is increased compared to monolithic learners. The approach follows these steps (from [Lin93]):

1. **Task decomposition.** A complex task is decomposed into multiple elementary tasks. The original complex task is thus reduced to the task of integrating the solutions to the elementary tasks to form the solution to the original task. Task decomposition involves designing a reward function for each elementary task.

2. **Learning elementary skills.** An elementary skill needs to be learned to solve each elementary task. Here Q-learning can be used and each elementary skill corresponds to a Q-function: $Q(s, action)$.

3. **Learning a high-level skill.** A high-level skill for coordinating the elementary skills needs to be learned in order to solve the original task. Learning a high-level skill is conceptually similar to learning and elementary skill. Again, Q-learning can be used and the high-level skill corresponds to a Q-function: $Q(s, skill)$.

In Lin's work, the job of task decomposition and assigning reward functions to sub-tasks is carried out by humans, the rest is learned by the robot. Lin's results show significantly faster convergence and better performance for agents that use this technique, compared to those learning an entire task at once.

Similarities between Lin's decomposition and temporal-sequencing for assemblages of motor schemas (Section 2.2.1) are readily apparent. Lin's sub-tasks or elementary skills correspond to behavioral assemblages, while a high-level skill is a sequence of assemblages. Learning at the high-level is equivalent to learning the state-transitions of an FSA (as in Figure 2.2) and learning the elementary skills corresponds to tuning individual states or behavioral assemblages.

A difficulty with reinforcement learning in complex tasks is that performance may converge slowly, or not at all. The problem is aggravated when only occasional (delayed) reinforcement is provided; after the task is completed for instance. Some other learning speedups Lin examined to address this include experience replay and teaching [Lin91]. Experience replay involves presenting sequences of previous experiences to the Q-learning algorithm. Presumably this serves to reduce the problems of having to gather costly or rare experiences more than once. For teaching, a human leads the robot through a series of actions to achieve the goal and the sequence of experiences thus gathered are used to train the system.

2.3.6 Distributed RL

The reinforcement learning approaches outlined so far use a centralized scheme for learning when particular sub-behaviors should be activated. Maes and Brooks [MB90] propose an alternative, distributed mechanism. In their scheme, each behavior learns for itself when it ought to be applied. They pre-define a set of behaviors and a set of binary perceptual conditions. Each behavior learns when it should be "on" or "off" based on the perceptual conditions. Positive and negative feedback are provided to guide the learning.

The behaviors learn, for each perceptual condition, *relevance* and *reliability* of the behavior to the condition. A behavior is relevant in the presence of a particular condition if it is positively correlated to positive feedback, i.e. positive feedback is likely to be received if the behavior is activated in that condition. A behavior is reliable if the probability of receiving the feedback is close to 1. The behaviors learn both negative relevance (when they should be turned off) and positive relevance for each condition. Conditions that are neither positively or negatively relevant are eventually dropped from consideration.

Maes and Brooks tested their approach on a robotic hexapod. Negative feedback is provided if either the front or rear of the robot touches the ground. Positive feedback is based on the rotation of a trailing wheel that measures forward motion. The robot was able to learn to walk, using a tripod gait in two to ten minutes. This is a significant success, but mathematical properties (rate of convergence for instance) of the technique have not yet been established rigorously.

2.4 Multi-robot systems

2.4.1 Dudek's taxonomy

The taxonomy of multiagent systems introduced by Dudek [DJW93] is becoming an important reference in multiagent literature. It provides a useful set of axes for discriminating between the many types of multiagent robot systems. The following is a synopsis of his taxonomy by dimension:

SIZE The number of robots in the environment. Types include ALONE, PAIR, LIM (a limited number of robots) and INF (unlimited).

COM Communication range; NONE, NEAR, INF.

TOP Communication topology; BROAD (broadcast), ADD (address), TREE and GRAPH.

BAND Bandwidth of the communication; ZERO, LOW, HIGH and MOTION. BAND-MOTION is a special case where the communication cost is equal to the cost of moving the robot between two locations.

ARR The rate at which the collective can spatially re-organize itself; STATIC, COMM (the members coordinate rearrangement using communication) and DYN (dynamic).

PROC The processing ability of individual units in the collective; SUM (non-linear summation), FSA (finite state automaton), PDA (push-down-automaton), TME (Turing machine equivalent).

CMP Composition; HOM (homogeneous), HET (heterogeneous).

The taxonomy provides an important context for this research. In particular, the CMP (composition) axis will be explored in terms of agent behavior.

2.4.2 Learning in behavior-based multi-robot systems

To date, only a few researchers have investigated learning in multi-robot systems, most notably Matarić [Mat92, Mat94] and Parker [Par94]. Parker developed the ALLIANCE architecture [Par94] for controlling teams of physically heterogeneous robots. The system is built on the behavior-based subsumption architecture [Bro86]. In a manner similar to temporal sequencing (Section 2.2.1), tasks are broken into sub-tasks, with groups of behaviors addressing each sub-task. At the highest level, mutually inhibitory *motivational behaviors* direct the overall behavior of the robot, activating in turn lower-level behaviors that combine to solve the sub-task.

Along with the typical sensor-based conditions that might trigger motivational behaviors Parker adds *impatience* and *acquiescence*. Impatience increases if no other robot is attempting to solve the sub-task associated with a motivational behavior, while acquiescence inhibits a behavior if the robot is not meeting with success. The combined result of the ordinary conditions, impatience and acquiescence in a group is that the group cooperates in striving to solve the overall task.

ALLIANCE was extended to L-ALLIANCE which provides for learning. Agents in L-ALLIANCE are able to learn the abilities of other robots to complete sub-task

This information, coupled with a strategy whereby the robot most suited for each task executes it, enables robot teams to significantly improve performance over other techniques.

Matarić's work is more closely related to this research because it involves the use of reinforcement learning. Her work in multi-robot learning systems is examined in Section 2.5.2.

2.5 Tasks for multi-robot systems

This research investigates the relationships between reinforcement function, performance and diversity in three multi-robot tasks: robotic foraging, soccer and formation maintenance. This section introduces each task and provides some background on the related research in each domain.

2.5.1 Robotic foraging

The forage task involves the collection of objects of interest (attractors) scattered about the environment. In a typical strategy, an agent begins by wandering about the environment looking for attractors. Upon encountering an attractor, the robot moves towards it and grasps it. After attachment, the robot returns the object to a home base. In some foraging strategies attractors may be handed off to another agent for final delivery.

Foraging has a strong biological basis. Many ant species, for instance, perform the forage task as they gather food. Foraging is also an important subject of research in the mobile robotics community; it relates to many real-world problems [Ark92, ABN93, BA95a, GM97, FM97]. Among other things, foraging robots may find potential use in mining operations, explosive ordnance disposal and waste or specimen collection in hazardous environments (e.g. the Mars Pathfinder rover).

At Georgia Tech, Arkin and Balch have investigated several homogeneous strategies for robot foraging. [Ark92, ABN93, BBC+95]. Their work specifically investigates the impact of of communication on performance in foraging teams. A motor schema approach with temporal sequencing is utilized (Figure 2.2 illustrates an example sequence from this work). The results show that foraging agents can cooperate without communicating. The investigation also found that simple communication provides an important performance advantage over no communication at all, but complex communication does not provide an additional improvement. The research is extended in this dissertation to include a more complex foraging task, several new strategies (including heterogeneous approaches) and learning.

In related research, Goldberg and Matarić have proposed a framework for investigating the relative merits of heterogeneous and homogeneous behavior in foraging tasks [GM97]. Like the research reported in this paper, their work focuses on mechanically identical, but behaviorally different agents. They propose *interference* as a metric for evaluating a foraging robot team. Interference refers to the situation where two robots attempt to occupy the same place at the same time; it is measured as the amount of time agents spend avoiding one another. Since interference may reduce the efficiency of a robot team, Goldberg suggests pack and caste arbitration as mechanisms for generating efficient behavior and reducing interference. In the pack scheme, each agent is arbitrarily assigned a place in the "pack hierarchy." Agents higher in the hierarchy are permitted to deliver attractors before the others. In the caste approach, only one agent completes the final delivery; the other robots leave their attractors on the boundary of a designated "home zone." The researcher's results indicate that interference per unit time is maximized in homogeneous foraging and minimized in pack foraging. In spite of the fact that interference is minimized in the heterogeneous pack systems, homogeneous systems perform best in terms of the number of pucks collected.

In separate research, Fontan and Matarić have investigated a territorial heterogeneous foraging strategy where the search area is equally divided between agents [FM97]. Robots hand off collected attractors from area to area, with the last agent completing delivery to the homebase. Their work indicates that performance degrades if the number of robots is increased beyond a certain maximum.

Drogoul investigates several homogeneous foraging strategies in simulation [DF94]. His research investigates the utility of laying "crumbs" as path markers for other agents. The idea was inspired by the technique of laying chemical trails to food sources utilized by many ant species [HW90]. Interestingly, the issue of agent-agent interference arises in Drogoul's work as well. In the most efficient "crumb-laying" foraging strategy, performance is reduced when the number of agents exceeds a particular mark. To address this, a "docker" behavioral strategy is explored. The docker robots are able to pass attractors from one to another while remaining in a fixed position. In robot simulations using this behavior, spontaneous chains of agents arise.

Instead of carrying attractors back to the base individually, they hand them from one to another in the chain. When resource-rich areas are discovered, performance is maximized in the docker strategy. The key drawback to this approach is the mechanical challenge of building agents able to accomplish such hand offs.

2.5.2 Learning robotic foraging

Matarić has investigated learning for multi-robot behavior-based teams in foraging tasks. Her work has focused on developing heuristic reinforcement functions for social learning [Mat94]. In one approach, the overall reinforcement, $R(t)$, for each robot is composed of separate components, D, O and V. D indicates progress towards the agent's present goal. O provides a reinforcement if the present action is a repetition of another agent's behavior. V is a measure of vicarious reinforcement; it follows the reinforcement provided to other agents. She tested this approach in a foraging task with a group of three robots. Results indicate that performance is best when the reinforcement function includes all three components. In fact the robots' behavior did not converge otherwise.

In another multi-agent learning investigation Matarić compares Q-learning with a heuristic learning strategy for foraging. The new strategy utilizes a "shaped" reinforcement function where agents are rewarded as they accomplish parts of the task. The heuristic approach is shown to perform significantly better than Q-learning and Matarić concludes that Q-learning is not appropriate for multiagent learning tasks.

The results reported in this dissertation contradict Matarić's conclusion regarding the suitability of Q-learning for multi-robot learning. In this research, multiagent teams using Q-learning converge to behaviors that perform as well as or better than human-coded approaches. This result holds in foraging, as well as soccer and cooperative movement tasks. Also, shaped reinforcement is shown to provide little or no advantage over the standard performance-based rewards used in most other reinforcement learning studies.

2.5.3 Robotic soccer

Robotic soccer is one of several task domains this research investigates. Soccer is a particularly good task for multiagent research because it includes cooperation between teammates, competition versus an opponent and unpredictable dynamic play.

In early robot soccer research, Sahota developed a system called Dynamite [Sah94]. The Dynamite test-bed utilized remotely driven cars controlled by an off-board computer. The computer was able to monitor the game through an overhead camera. He proposed *reactive deliberation* as a control scheme. In this architecture, a high level module (the Deliberator) selectively activates "action schemas" to be run at the lower level. The system did not include learning, but it was demonstrated to play soccer well. Reactive deliberation bears some resemblance to AuRA in that a higher level deliberation module selects schemas for execution by the lower level, but AuRA offers the possibility of activating and integrating multiple schemas simultaneously.

Recent interest has sparked more research in robot soccer. Kitano and Asada promote the Robot World Cup as a vehicle for multiagent research [KAK+97]. They have developed an internationally agreed upon set of rules for a game involving mobile robots and a separate simulation system using the same rules. Asada has additionally investigated learning individual skills (e.g. shooting) for robot soccer players.

2.5.4 Learning robotic soccer

Stone and Veloso have developed a multi-layered learning system for soccer [SV98]. In their approach, individual agents are taught lower-level skills first, using a neural-net technique. Higher-level behaviors are developed using decision trees. Although the mechanism is different (decision trees) the approach to training is similar to Lin's in that the lower level skills are developed first with higher-levels trained afterwards [Lin92].

Salustowicz, *et al* have investigated reinforcement learning in a simulated soccer task [SWS98]. Their research is focused on a comparison of PIPE and TD-Q learning. PIPE is genetic programming variant [Koz92], while TD-Q is based on the neural network approach introduced by Lin [Lin93] (note: TD-Q is distinct from Q-learning). The results indicate PIPE generates teams with better performance than those trained using TD-Q. The work is similar to the approach used in this research for training soccer agents, but with several important distinctions. In Salustowicz's approach the agents are implicitly homogeneous. All agents share the same policy, so it is impossible for heterogeneity to emerge. In contrast, in this work, each agent develops an individual policy that may or may not correspond to that of the other agents. Also, this research evaluates the impact of several competing reinforcement strategies (local and global) while Salustowicz's work utilizes global performance-based rewards for all training.

2.5.5 Robot formation

Formation behaviors in nature, like flocking and schooling, benefit the animals that use them in various ways. Each animal in a herd, for instance, benefits by minimizing its encounters with predators [Veh87]. By grouping, animals also combine their sensors to maximize the chance of detecting predators or to more efficiently forage for food. Studies of flocking and schooling show that these behaviors emerge as a combination of a desire to stay in the group and yet simultaneously keep a separation distance from other members of the group [CSB65]. Since groups of artificial agents could similarly benefit from formation tactics, robotics researchers and those in the artificial life community have drawn from these biological studies to develop formation behaviors for both simulated agents and robots.

Formation is important in mobile multiagent applications where sensor assets are limited. Formations allow individual team members to concentrate their sensors across a portion of the environment, while their partners cover the rest. Air Force fighter pilots for instance, direct their visual and radar search responsibilities depending on their position in a formation [For92]. Robotic scouts also benefit by directing their sensors in different areas to ensure full coverage [CGH96]. Formation is potentially applicable in many other domains such as search and rescue, agricultural coverage tasks, security patrols and so on.

In the behavior-based approach utilized in this research, formation maintenance is accomplished in two steps: first, a perceptual process, **detect-formation-position**, determines the robot's proper position in formation based on current environmental data; second, the motor process **maintain-formation**, generates motor commands to direct the robot toward the correct location. Each robot computes its proper position in the formation based on the locations of the other robots. Several motor schemas, **move-to-goal**, **avoid-static-obstacle**, **avoid-robot** and **maintain-formation** implement the overall behavior for a robot to move to a goal location while avoiding obstacles, collisions with other robots and remaining in formation. An additional background schema, **noise**, serves as a form of reactive "grease", dealing with some of the problems endemic to purely reactive navigational methods [Ark89].

In the most closely related approach, Parker simulates robots in a line-abreast formation navigating past waypoints to a final destination [Par93]. The agents are programmed using the layered subsumption architecture [Bro86]. Parker evaluates the benefits of varying degrees of global knowledge in terms of cumulative position error and time to complete the task. The approach includes a provision for obstacle avoidance, but performance in the presence of obstacles is not reported. Parker's results suggest that performance is improved when agents combine local control with information about the leader's path and the team's goal.

This research extends this earlier work by providing agents with the ability to learn formation behaviors. At this writing, the author knows of no other multi-robot formation research involving learning agents.

2.6 Social entropy theory

A precise definition of *diversity* in robot societies is important for this research. *social entropy* is proposed as an appropriate metric of diversity in robot systems. Details of social entropy in robot groups are provided in Chapter 5 and in [Bal97c, Bal97b]. Interestingly, sociologists have developed a similar (and eponymous) *social entropy theory* as a means of explaining and evaluating social structure in human groups [Bai90].[1]

Briefly, both human and robotic social entropy are based on information entropy, a measure of randomness in communication systems. Greater entropy indicates more randomness and disorder. Entropy in communication depends on the number of distinct symbols to be transmitted and the frequency of each symbol in a typical message. Similarly, social entropy depends on the number of distinct types of individuals in a society and their frequency of occurrence in the society. Both robotic and human measures of social entropy depend on a categorization of agents into groups based on differences between them.

The selection of an appropriate set of features or attributes on which to com-

[1] Although entropy was introduced as a tool in sociology as early as 1958, the application of entropy for the evaluation of robot systems is new and was developed independently.

pare individuals is a heatedly debated topic in the sociological entropy literature. The most frequently cited framework is [Bai90] by Bailey. Bailey employs a five-dimensional system of mutable variables that describe each person in a society. For each person, each variable has a particular value. People with similar attributes may be grouped together. Bailey's mutable variables are

- **I**, information: education, religious beliefs, political ideology.
- **L**, level of living: quality of life, income.
- **S**, space: location of residence.
- **T**, technology: level of technological skill.
- **O**, organization: position in organizational hierarchy.

The variables are referred to as "mutable" because an individual is able, and even likely, to change them through their life. For instance **S** is changed when someone relocates, **L** changes when a person get a raise and so on. People are also ascribed immutable characteristics like gender, time of birth (age), skin color, etc. Since in this research, the focus is on *behavioral* diversity in robots, the features for categorization are somewhat different. Agents are categorized on the basis of differences in behavior; the idea is compare their learned policies and group them according to similarities in their strategies.

As an example of how entropy might be employed for social analysis, consider how the spatial (**S**) distribution of Americans has shifted from rural areas to the cities over the last 50-100 years. When we were primarily a rural society, the value of **S** was likely to be different for nearly all citizens. As people moved to the cities, however, there was a greater and greater likelihood for many people to share the same or similar **S**. This shift has served to *decrease* the entropy of our country's spatial distribution, indicating that we have become more ordered, at least with respect to geography.

The use of entropy for similar purposes in sociology supports its use in robotics. It is important to note that sociologists hardly ever calculate a numerical value for the entropy of, say, the United States. Rather the idea is a framework for analysis. It provides a way for researchers to analyze social change and structure.

2.7 Discussion and summary

This chapter reviews the important existing work related to the dissertation. It also provides the reader with a background on the algorithms and techniques drawn from others and employed in this work. Key points:

- **Motor schema-based control** is employed as the robot behavioral programming platform [Ark89]. Motor schemas are grouped together to form behavioral assemblages. Assemblages are activated in an appropriate sequence to accomplish a task.
- **Q-learning**, a reinforcement learning technique, is used to train robots when to activate particular behaviors to accomplish a task [WD92].
- **Social entropy** is utilized as a quantitative measure of diversity in robot teams. The technique is also used in sociology for evaluating the structure of human society [Bai90].
- **Robotic tasks** including foraging, soccer and formation maintenance are explored in this research. The significant work of other researchers in these tasks is cited and reviewed.

The research in this dissertation differs from other work in several important respects. First, while other researchers are investigating performance in homogeneous and heterogeneous robot systems, here we are primarily concerned with the *origins* of heterogeneous and homogeneous behavior. The work is further distinguished by the fact that *learning* agents are the central investigative tool. No commitment is made in advance to any particular societal structure or arbitration mechanism. Instead, the robots develop their own societal solutions. This opens up the possibility that new forms of arbitration and cooperation may be discovered by the robots themselves. Finally, we are interested in measuring the diversity of the resulting society and utilize the metric of social entropy for that purpose [Bal97b].