Multistrategy Learning of Adaptive Reactive Controllers

Juan Carlos Santamaría and **Ashwin Ram**
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280
E-mail: {`carlos,ashwin`}`@cc.gatech.edu`
Phone: (404) 894-4995
Fax: (404) 894-9846

Technical Report GIT-CC-97-05

January 1997

Abstract

Reactive controllers has been widely used in mobile robots since they are able to achieve successful performance in real-time. However, the configuration of a reactive controller depends highly on the operating conditions of the robot and the environment; thus, a reactive controller configured for one class of environments may not perform adequately in another. This paper presents a formulation of *learning adaptive reactive controllers*. *Adaptive* reactive controllers inherit all the advantages of traditional reactive controllers, but in addition they are able to adjust themselves to the current operating conditions of the robot and the environment in order to improve task performance. Furthermore, *learning* adaptive reactive controllers can learn when and how to adapt the reactive controller so as to achieve effective performance under different conditions. The paper presents an algorithm for a learning adaptive reactive controller that combines ideas from case-based reasoning and reinforcement learning to construct a mapping between the operating conditions of a controller and the appropriate controller configuration; this mapping is in turn used to adapt the controller configuration dynamically. As a case study, the algorithm is implemented in a robotic navigation system that controls a Denning MRV-III mobile robot. The system is extensively evaluated using statistical methods to verify its learning performance and to understand the relevance of different design parameters on the performance of the system.

Keywords: Reactive control, multistrategy learning, case-based reasoning, reinforcement learning, robotic navigation.

Contents

1 Introduction

Autonomous mobile robots must perform many complex information processing tasks in real-time. Furthermore, they must operate successfully under changing environments. These requirements impose several challenges on their control systems. To be successful, an autonomous robotic control system must be able to process incoming sensory information, decide what action to execute next, and carry out that action without missing any time deadlines. Reactive controllers has been widely used in mobile robots since they are able to achieve successful performance in real-time (e.g., Agre, 1987; Arkin, 1989; Brooks, 1986; Kaelbling, 1990; Maes, 1990; Payton, 1986).

Reactive controllers typically rely on a combination of several *task-achieving* modules, behaviors, or schemas to perform a mobile robotic task (e.g., Brooks, 1986; Langer, Rosenblatt, & Hebert, 1994; Mahadevan & Connell, 1991). That is, a robotic task is decomposed into several subtasks that the robot must accomplish and execute concurrently. Typically, the system designer programs specific modules that accomplish each subtask by considering relevant information from the robot's sensors to control the robot's actuators. Each module has a stimulus-response type of relationship with the world. The response of the robot is the result of the interaction of all the responses in the system and can be computed according to different schemes, such as subsumption (e.g., Brooks, 1986), weighted summation (e.g., Arkin, 1989), or voting (e.g., Langer, Rosenblatt, & Hebert, 1994). There are many advantages of such controllers. Reactive controllers are able to execute actions in real-time since the modules act like quick "reflexes" to environmental inputs. This allows mobile robots to react to sudden changes in the environment. Reactive controllers do not use complex internal representations to keep an accurate model of the world, nor do they rely on executing expensive planning processes operating on that model, which may consume important resources and deteriorate the response time of the robot. Instead, each module in a reactive controller extracts only the relevant information required to execute its particular task. For example, a robot does not need to recognize a chair in order to avoid it. For this task, an avoid-obstacle module only needs to know at what distance an obstacle is located from the robot to suggest an appropriate response (e.g., the "avoid-static-obstacle" motor schema of Arkin, 1989). Thus, reactive controllers are characterized by having robust navigational capabilities and rapid, real-time response to the environment.

Nevertheless, there is much room for improvement in reactive controllers. Like classical controllers (see, e.g., Ogata, 1990), reactive controllers have several parameters that affect the performance of the controlled process. Thus, the performance of any given task executed by a controller will depend highly on the parameters of the controller and on the operating conditions of the plant (or robot). For example, a reactive controller may guide a mobile robot successfully through areas with different number of obstacles. However, if the robot is to accomplish the task at high performance levels, then different controller parameters will be required for operating on areas with different numbers or configurations of obstacles (e.g., Ram, Arkin, Moorman, & Clark, 1992). The problem of designing classical controllers that adapt themselves dynamically has been addressed by researchers in the subarea of control theory known as *adaptive control* (see, e.g., Narendra & Annaswamy, 1989). Adaptive control refers to the control of partially known systems in which designers know enough about a system to select a particular class of controllers, but an efficient configuration of the controller is impossible to determine since there is not enough knowledge about the dynamics of the system to be controlled. Thus, the controller is designed in such a way that it improves its performance by observing the outputs of the process and choosing the appropriate configuration accordingly. As the process unfolds, additional information becomes available and improved configurations become possible (Bellman, 1959).

During the design of adaptive controllers, the available knowledge about the dynamics of the system is used to design an *adaptive law* that the controller can use at run-time to configure itself and improve task performance. However, such approach is not always directly applicable in the design of reactive controllers. In order to be flexible, mobile robots must be designed to interact with unknown environments. Since the dynamics of the process to be controlled is composed from the dynamics of the robots and the environment, the amount of information known in advance is usually not enough to use the traditional design tools from adaptive control theory. In particular, it may not be possible to determine an adaptive law in advance; instead, the system would have to learn an appropriate adaptive law through experience.

In this paper we present a formulation of *learning adaptive reactive controllers* for mobile robots. An *adaptive* reactive controller is a controller that inherits all the advantages of traditional reactive controllers, but also it is able to dynamically adjust itself to the current operating conditions of the robot and the environment to improve task performance. Furthermore, a *learning* adaptive reactive controller is one which is able to learn an adaptive law through its own experiences that can be used to adjust the controller dynamically. Such a controller is an improvement over an adaptive reactive controller, which can only use a predefined adaptive law to configure the controller at run time. Combining ideas from adaptive control theory and machine learning, we propose an algorithm for a learning adaptive reactive controller. The algorithm is implemented in an autonomous navigational system for a Denning MRV-III mobile robot.

This paper is organized as follows. Section 2 reviews necessary background from adaptive control theory and formulates the problem in terms of mobile robot control. Section 3 presents an algorithm that can be used to implement a learning adaptive reactive controller for autonomous robotic navigation. Section 4 describes the application of the proposed algorithm to a schema-based reactive controller for a Denning MRV-III robot. Section 5 describes the experiments, results, and analysis of results used to evaluate the proposed algorithm. Section 6 discusses relevant points of the proposed algorithm from the point of view of the theory behind its design. Finally, Section 7 concludes the paper.

2 Problem Formulation

2.1 Overview

The objective of this section is to formulate a theory of learning adaptive reactive control and to express the autonomous robotic navigation task in terms of that theory. Autonomous robotic navigation is defined as the task of moving a robot safely from an initial location to a destination location in an obstacle-ridden terrain. In most real-world applications, designers do not have complete knowledge about the environment in which the robot is to navigate; in addition, the robot must often operate in many different environments. Both situations require that a reactive controller be able to adapt itself to the particular operating conditions in order to achieve effective performance. Thus, being able to design and implement adaptive reactive controllers is essential for building mobile robots that can perform effectively and flexibly in real-world environments. An adaptive reactive controller would have a significant advantage over a traditional one since it would not only be able to respond in a rapid and robust manner but would also be able to regulate itself to different operating conditions, thus improving the performance of the navigation task. In addition, a learning adaptive reactive controller would not only be able to adapt itself to the operating conditions but would also learn the adaptive law required to perform such adaptations. The learning capability provides a significant advantage; designers do not need to specify an adaptive law since the system can synthesize one using its own experience.

2.2 Adaptive Reactive Controllers

Reactive controllers must perform the same function of standard controllers, namely to implement a *control policy* which is a mapping between sensory information and output commands designed to accomplish the robot's task. However, reactive controllers have two major differences from standard controllers. First, a purely reactive controller does not use sensory information to update internal models of the world that then guide action; instead, it directly uses current sensory information to select the control commands to execute next. Second, reactive controllers are usually designed as a combination of modules that interact, each module being responsible for implementing a specific subtask such as avoiding collisions or moving towards a goal. These two characteristics allow reactive controllers to perform robustly in real-time since the mapping between sensors and actuators is implemented as quick reflexes without the reasoning necessary to update detailed world models (e.g., Arkin, 1989; Balch, Boone, Collins, Forbes, MacKenzie, & Santamaría, 1995; Brooks, 1986; Langer, Rosenblatt, & Hebert, 1994; Mahadevan & Connell, 1991).

Mathematically, we can represent the policy of a reactive controller as a function mapping the set of inputs z delivered by the robot's sensors to the set of outputs u sent to the robot's actuators:

$$u = \pi(z) \tag{1}$$

where $u = (u_1, \cdots, u_m)$ and $z = (z_1, \cdots, z_n)$.

We define an adaptive reactive controller as a reactive controller that can modify its control policy $\pi(\cdot)$ on the fly.[1] For example, the controller might use a pre-programmed model to select an appropriate control policy based on its observation of the current situation. An adaptive reactive controller may be classified as *parameter adaptive* or *structurally adaptive*. In the former case, the controller is able to modify a predefined set of components (specifically, modules) through the use of parameters that can be adjusted dynamically. In the latter case, the controller is able to modify the structure of the controller by introducing or eliminating components altogether and by modifying the interconnections between them. Thus, an adaptive controller is parameter adaptive when it can, for example, modify the intensity of the response of a module by varying a scalar or *gain*, such as the minimum allowable distance of approach to an obstacle which influences the

[1]Note that the two tasks, one of using the inputs z to modify a control policy and the other of using the control policy to determine actuator outputs, may be implemented using a single algorithm which accomplishes both tasks in an integrated manner. However, since these two subtasks are *functionally* distinct parts of the overall task of adaptive reactive control, it is useful to distinguish them in a teleological analysis that takes a design stance (Dennett, 1987) towards the problem of adaptive reactive control.

"avoid-obstacle" behavior. In contrast, a controller is structurally adaptive if it can interchange a module, say "move-to-dark-area", with another, such as "move-to-bright-area", or when it can change the combination mechanism of module responses from, say, subsumption to weighted summation.

Structurally adaptive controllers, if designed successfully, might be able to cope with more drastic environmental changes and achieve better levels of performance than parameter adaptive controllers. The intuitive reason for this is that the former contains a much larger set of possible solutions than the latter. Thus, there may exist solutions obtainable through modification of the structure of the controller that are not obtainable merely through parameter adjustments. However, structurally adaptive controllers are very difficult to design because stability properties cannot be guaranteed due to their mathematical intractability (Narendra & Annaswamy, 1989). Extensive knowledge about the particular process to be controlled must be known before additional assumptions can be made and stability properties can be guaranteed. Parameter adaptive controllers, in contrast, tend to be more tractable than structurally adaptive controllers because, in the former, there exists a continuous relationship between the parameters and the behavior of the system, whereas changes in the structure of the controller may produce radically different system behaviors. In general, there is no way to guarantee the performance characteristics of these behaviors. In order that the design of our adaptive reactive controller and its tractability and stability properties not be dependent on specific knowledge about the dynamics of the environment and the mobile robot, we have chosen to concentrate on parameter-adaptive reactive controllers in this article. Additionally, due to the continuous relationship between the parameters and the behavior of the system, parameter-adaptive reactive controllers can rely on a sufficient, although not necessary, condition to guarantee stability: in an adaptive system, as long as the adaptations are performed at a much slower rate than the dynamics of the system, the system can remain stable (Albus, 1991).

The operation of a controller is successful if it can maintain specific outputs of the system within prescribed limits. In the case of autonomous robotic navigation, the controller's task is to guide the robot to the destination point while avoiding obstacles. Quantitatively, this may be stated as the determination of the actuator commands $u(\cdot)$ to keep the error $e = y_p - y_m$ between some measure of the robot performance y_p and a desired measure y_m within prescribed values.[2] The performance measure of the robot may consist, among other things, of a subset of preprocessed sensory inputs (z) such as the current position and the number of collisions at a given instant. The desired input consists of the desired position (the destination) and number of collisions (zero in most cases). Note that all perceived inputs may or may not have a desired value specified. For example, the robot may perceive the distance to the nearest obstacle in addition to its location, but the objective might specify only a destination location without specifying a desired value for the robot's distance to the nearest obstacle.

If the dynamics of the robot and the structure of the environment are completely known, the structure of the controller can be determined and its parameters chosen as to minimize a desired performance index, such as distance traveled and total number of collisions. When the structure of the environment is unknown, however, the control problem can be viewed as an adaptive control problem since the dynamics of the system (in this case, the robot and its environment) are not known completely in advance and different controller parameters might be appropriate at different

[2]The variables e, y_p, and y_m may be multivariables. In other words, they can be vectorial variables, for example $\vec{e} = (e_1, \ldots, e_n)$. We have avoided vectorial notation to simplify the presentation in this section.

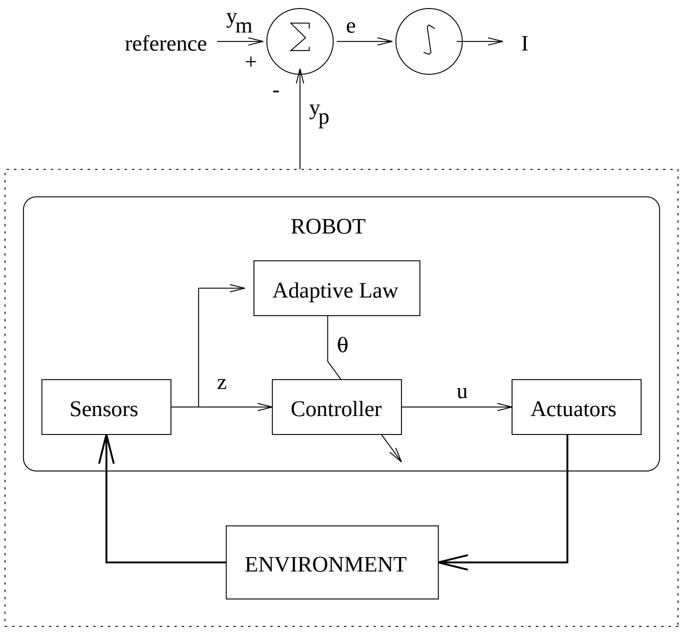

Figure 1: Adaptive Reactive Control.

times. Figure 1 shows this formulation of the adaptive reactive control problem.

The autonomous navigation control problem can be stated as follows: Given a robot R with an sensor-actuator relationship $\{z(\cdot), u(\cdot)\}$ and a desired performance reference y_m, determine $u(t)$ for all $t > 0$ such that the performance index given in Equation 2 is minimized:

$$I = \int_0^\infty (y_p - y_m)^T Q(y_p - y_m)dt \tag{2}$$

In this formulation it is assumed that the output set U of the robot is generated by its controller C. A practical requirement is that $u(\cdot)$ is bounded such that it can be physically realizable. The reference input y_m is assumed to be bounded and physically realizable. Knowledge about the robot and the structure of the environment helps in defining the reference input. The structure of the controller C is determined using knowledge about the robot and the environment. The controller is responsible for generating the output commands $u(\cdot)$ using all the sensory information $z(\cdot)$ available to it.

Assuming that the controller is parameterized by a vector θ, for any given robot and environment, there exists a parameter vector θ^* such that Equation 2 is minimized. In most cases, there would exist more than one θ^*, each corresponding to a different set of controller parameters that can be used to guide the robot. Q in Equation 2 is a positive-definite matrix to guarantee that I is a convex function. The term $(y_p - y_m)$ represents the difference vector between the current performance measure and the reference at a given instance in time. The difference is squared and weighted by the matrix Q and then integrated over the entire working period ($t \in [0, \infty)$). Thus, I is a measure of the deviation of the performance of the system from the reference over the entire working period. For example, I might track the total number of collisions or the length of the path followed to the destination location.

Ideally, the vector θ should vary over time so that I is minimized. The rule by which θ is adjusted over time is called the *adaptive law*. Summarizing, the problem of designing an adaptive reactive controller can be stated as follows:

1. Determine the bounds of the robot's outputs u and the set of realizable y_m.
2. Determine the structure of a reactive controller $C(\theta)$, parameterized by the vector θ.
3. Determine the adaptive law for adjusting θ such that a performance index I given by Equation 2 is minimized.

Adaptive reactive controllers can be hand-designed. That is, with enough knowledge about the robot, the environment, and their interaction, the system designer can synthesize an appropriate adaptive law such that the performance index improves as compared to a non-adaptive reactive controller (e.g., Ram, Arkin, Moorman, & Clark, 1992). However, an alternative approach is to incorporate learning algorithms into the controller and let the robot learn an appropriate adaptive law with its own experience. This is discussed next.

2.3 Learning Adaptive Reactive Controllers

The main requirement of a parameter-adaptive reactive controller is to be able to apply an adaptive law to adjust the controller parameters at run-time in order to optimize performance. An adequate law can be synthesized at design time only when there exists enough information about the system and its dynamics. However, to design a system capable of dealing with situations not foreseen by its designer, it is necessary that the system be able not only to apply but also to learn and refine its adaptive law using its own experience.

In robotic navigation, sensory information and control parameters are usually represented using analog values. Thus, the problem of learning an adaptive law can be thought of as function synthesis. Let M denote a general memory device and let D denote the domain under which this memory is applicable. If $x \in D$, then the expression $\theta = M(x)$ represents the parameter θ "recalled" in "situation" x from memory M. Thus M can be thought of as representing a mapping from $x \in D$, where x represents the input description of the current situation, to appropriate controller parameters θ for this situation. Note that x may include input information z from the sensors of the robot as well as internal "observations" about the state of the robot (say, the current value of θ), since the appropriate parameterization θ may in general depend on both kinds of information.

The ideal mapping M^* is such that when the robot is in situation x, then $\theta^* = M^*(x)$ is the optimal parameterization for the reactive controller for that situation. Since M^* is not known in advance due to the lack of detailed knowledge of the environment and/or the dynamics of the robot, the objective of the learning algorithm is to incrementally refine the mapping M with every experience such that $M \rightarrow M^*$ as $t \rightarrow \infty$, assuming continuous operation. At any given time, the current mapping M can be used as an adaptive law; in turn, the outcome of the suggested controller parameterization in a given situation can be observed and used to upgrade the mapping in a continuous, on-line manner.

In conclusion, in our formulation, reactive controllers can be classified into three categories:

Category 2.1 (Reactive Controller) *A controller composed of task-achieving modules. Each module specializes in accomplishing a specific subtask. The response of the robot is the result of the interaction of all the modules. A general algorithm for a reactive controller is outlined in Figure 2.*

Category 2.2 (Adaptive Reactive Controller) *A reactive controller that is able to adjust itself to the current operating conditions of the robot and the environment to improve task performance. Adaptive reactive controllers can be parameter adaptive or structurally adaptive. A parameter-adaptive reactive controller has a predefined adaptive law that it uses to parameterize the reactive controller at run-time and improve performance. A general algorithm for a parameter-adaptive reactive controller is outlined in Figure 3.*

Category 2.3 (Learning Adaptive Reactive Controller) *An adaptive reactive controller that is able to learn an appropriate adaptive law with its own experience. A general algorithm for a learning parameter-adaptive reactive controller is outlined in Figure 4.*

In this article, we present a machine learning algorithm called LARC[3] that learns the adaptive law for a parameter-adaptive reactive controller. The algorithm is fully implemented in SINS[4], a learning system for an adaptive schema-based reactive controller for a Denning MRV-III mobile robot, and evaluated through extensive empirical studies. The significance of the results is determined through statistical analysis of the empirical results and is discussed in the context of the theory underlying the implementation. The technical details of the proposed method are discussed next.

3 Approach

We propose a multistrategy case-based and reinforcement learning algorithm, called LARC, as a mechanism to learn when and how to adapt the parameters of a reactive controller in a mobile robot. With experience, the algorithm is able to learn and apply an adequate adaptive law to improve task performance autonomously. The algorithm is based on a combination of ideas from case-based reasoning and learning, which deals with the issue of using past experiences to deal with and learn from novel situations (see Hammond, 1989; Kolodner, 1993), and from reinforcement learning, which deals with the issue of strengthening the tendency to produce actions leading to satisfactory states of affairs (see Sutton, Barto, & Williams, 1991; Thorndike, 1911).

In the LARC algorithm, the adaptive law is represented by a mapping M consisting of a set of cases containing specific associations between situations and control parameters. Each case is applicable in a specific region of the input space D and contains the best guess so far of which control parameters θ the controller is to use in that region of the input space. Cases are formed and refined by merging past experiences in common regions of the input space and by remembering which control parameters are useful as measured by a reward signal. System performance is improved as a consequence of both synthesizing an adaptive law in regions of the input space not

[3]Learning Adaptive Reactive Control.
[4]Self-Improving Navigational System.

```
For every perception-action cycle of the robot:
1.  Read inputs $z$ from robot's sensors.
2.  Compute next output commands $u = \pi(z)$.
3.  Send output commands $u$ to robot's actuators.
```

Figure 2: Outline of a general algorithm for a reactive controller.

```
For every perception-action cycle of the robot:
1.  Read inputs $z$ from robot's sensors.
2.  Use adaptive law to decide new controller parameters $\theta$.
3.  Compute next output commands $u = \pi(z, \theta)$.
4.  Send output commands $u$ to robot's actuators.
```

Figure 3: Outline of a general algorithm for a parameter-adaptive reactive controller.

```
For every perception-action cycle of the robot:
1.  Read inputs $z$ from robot's sensors.
2.  Learn and update the adaptive law.
3.  Use adaptive law to decide new controller parameters $\theta$.
4.  Compute next output commands $u = \pi(z, \theta)$.
5.  Send output commands $u$ to robot's actuators.
```

Figure 4: Outline of a general algorithm for a learning parameter-adaptive reactive controller.

considered previously and optimizing the adaptive law in those regions of the input space used more often.

The main role of the case-based learning strategy is to remember previous successful parameterizations of the controller and use (or modify) these parameterizations when the robot faces similar circunstances. The main role of the reinforcement learning strategy is to strengthen the content of the cases that contain parameterizations that tend to produce useful results. Thus, the basic learning scheme involves the following three steps: (1) Given a situation x, use the current mapping M to generate a controller parameterization $\theta = M(x)$. (2) Observe the behavior of the robot R as governed by the controller $C(\theta)$ under the suggested parameterization. (3) Update M based on the outcome according to some optimization criteria. The following subsections describes the details of the representational structures and algorithms involved.

3.1 Case Representation

The mapping between situations and parameterizations is represented using a set of cases that are learned and modified through experience. Each case consists of a time sequence of associations between situations and control parameters. Situations and control parameters are represented by points in the input and output spaces respectively. The input space consists of a n-tuple of n continuous sensory variables sampled at a given instant in time; each variable corresponds to an observable or computable input z. Similarly, the output space consists of a p-tuple of p continuous control parameters sampled at the same instant in time; the p-tuple represents the θ at that instant. Thus, an association represent a situation in the space covered by the robot's sensory information and the control parameters that were active while the robot traversed this situation. A sequence of associations captures a history of situations and their corresponding control parameters over a short window in time. Using this formulation, a case represents a portion of an adaptive law which is applicable only at a specific region of the input space. More precisely, since cases are retrieved by matching the recent history of input variables z and output variables θ against the corresponding graphs represented in the cases, the adaptive law is represented by a mapping $\theta(t_{n+1}) = M((z(t_n), \ldots, z(t_{n-k+1}), \theta(t_n), \ldots, \theta(t_{n-k+1})))$. In other words, as mentioned earlier, the situation description x used to retrieve cases consists of external sensory observations z as well as "observations" of internal state (specifically, θ) over a suitable window of time represented by k samples taken at $t = t_{n-k+1}, \ldots, t_n$. This is discussed in more detail below. Figure 5 shows an example of a case. The complete adaptive law is represented by a set of such cases.

The purpose of the learning algorithm is to use previous navigational experiences to create and refine cases that capture *consistent* sequences of *useful* associations. A sequence of associations is consistent over time when a controller that uses similar control parameters under similar situations produces similar results. Such sequences are advantageous because they are a reliable source of information when the robot faces new situations similar to situations experienced in the past. For example, when a robot is traversing a new situation it could use the same or similar control parameters as those used in previous similar situations and hope to get similar results. However, consistent associations may not always be useful. For example, if a robot detects it is near an obstacle and lowers the importance of avoiding such obstacle, it is very likely that a collision will occur. Such a sequence of associations is consistent but not useful since it deteriorates the navigation performance. The learning algorithm uses an external reward signal to remember only

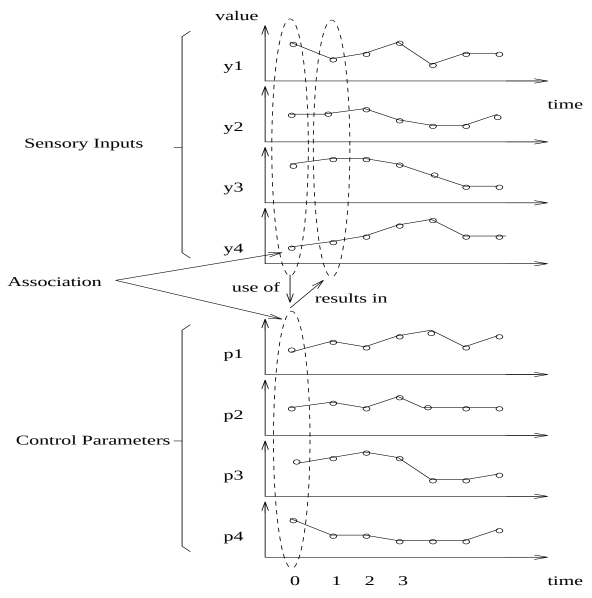

Figure 5: Sample representations showing the time history of values representing perceived inputs and control parameters. Associations between sensory inputs and control parameters are arranged vertically and the sequence of associations are arranged horizontally.

those associations that are consistently useful so that, after several experiences, cases will contain consistent sequences of useful associations.

Situations and control parameters are represented using tuples of real-valued variables. Analogical representations such as real-valued variables are useful for several reasons. First, sensory information can be quickly processed using real-valued functions to produce a tuple that can be used to designate the current environmental situation. Fast information processing is a requirement because reactive controllers must act in real-time. Second, analogical representations, due to their fine grain-size, can be modified gradually with new incoming information. Gradual modification of cases ensures that the mapping from situations to control parameters is updated continuously and incrementally with every single experience. Such gradual modification of the mapping function helps to ensure stability in the system since the total behavior of the system is being modified in a continuous fashion. Third, analogical representation allow extrapolation of values. This is useful when a new parameterization is required in a region of the input space that has not been explored before. The parameters of adjacent regions can be used to extrapolate the values in the required region. Finally, analogical representations are able to represent a wide range of situations and control parameters. This allow cases to capture any sequence of associations as long as they are proven to be consistent and useful.

An additional innovation introduced in our work is the time-history representation of situations and control parameters. Keeping time-history information allows explicit representation of causal patterns between situations and control parameters. Representation of causal patterns is a requisite for the detection and capture of consistent sequence of associations. Such representations also allow the system to learn a broader class of navigational strategies. For example, the system might learn what to do when it is *approaching* an obstacle (a temporal concept) rather than merely what to do when it is *near* an obstacle (a static concept). Note that time-history representations are different from representations of temporal concepts (such as velocity); a time-history representation captures the successive values of the represented variable (which may in fact be a temporal variable such as velocity) over time.

Finally, note that the case representation allows the system to learn navigational strategies in general rather than learning about a particular environment. For example, the system might learn a general strategy such as "when approaching a dense cluster of obstacles, increase the obstacle-avoidance gain and go around the obstacles" rather than a specific recommendation for action for a given environment (such as "this particular world has an obstacle at such-and-such location, so reduce velocity by 75% when the robot arrives at such-and-such position").

3.2 The LARC Algorithm

The adaptation and learning algorithm must perform two tasks simultaneously. It must use past experiences to construct a mapping between situations and control parameters (i.e., learning the adaptive law) and it must use the mapping to determine the best set of control parameters to use in every given situation (i.e., using the adaptive law). The strategy of the algorithm is to come up with a partition of the input space into regions (or clusters) and to search for the best control parameters to be used in each particular region. Partitioning the input space into several regions is possible due to the extrapolation property of analogical representations, which exploits the assumption that neighboring points in the input space should have similar control parameters. Thus, partitioning the input space into regions allows the mapping to be used under situations not experienced before and avoids the impossible task of associating the appropriate control parameters to every single point in the input space. An additional requirement on the algorithm is that it must learn the mapping incrementally by continuously incorporating new information gathered from every experience. This is necessary because as the robot moves, the environmental situation changes and new control parameters may be required. Thus, the parameter adaptation algorithm must be able to suggest a best guess set of control parameters to use during a new situation and then observe the results of the suggested controller parameterization to decide how to improve task performance during future similar situations.

The requirements described above are accomplished by a multistrategy algorithm that combines ideas from case-based learning, reinforcement learning, competitive learning, and minimum distance clustering. The partition of the input space is accomplished by merging past similar experiences into cases. Every case corresponds to a region in the input space and every new experience is used to refine the boundaries and the centroid of that region. The association of useful control parameters to every case is accomplished by modifying the control parameters associated to a region whenever a reward or punishment signal is received or whenever the results of the application of these parameters is not consistent over time. In this way, only associations that are both useful and consistent are encoded into cases.

In summary, the LARC algorithm learns to partition the input space and to associate adequate control parameters simultaneously and incrementally with every experience. In terms of our formulation, the LARC algorithm learns and applies a piece-wise general memory device M in which each case represents a portion of the mapping $\theta = M(x)$ by indicating what parameterization θ should be used under a particular situation x. Thus, the LARC algorithm compares the perceived inputs z with reference inputs or cases that consists of previous similar experiences, and adapts the controller by supplying the θ that results from this comparison.

We now discuss the technical details of the LARC algorithm. Although the different phases of the algorithm are tightly integrated, in order to facilitate presentation of the algorithm, the issue of partitioning the input space will be considered first. The issue of associating useful control parameters will be considered after the process of learning cases is described.

3.2.1 Learning Cases

Case-based reasoning and learning deals with the issue of using past experiences to solve new problems. However, when the domain is represented using real-valued variables, remembering each single experience as a case is impractical due to bounds imposed by limits on the available memory. The alternative is to combine several similar experiences together to create an "average" or "virtual" experience or centroid in a region and let a case represent this centroid instead of each single experience (see Porter, Bareiss, & Holte, 1990, for another example of a case-based reasoning system, PROTOS, where cases represent "prototypical" experiences). In this way, each case represents all the points in the input space that are close enough to its centroid so that any information in the case can be generalized to the points inside the region that it represents. One problem with this alternative is that there is no a priori information about where such centroids should be located or how big the regions should be. The centroids and the region's boundaries must therefore be learned with each experience. Since experiences come one after another, cases must be learned gradually and incrementally. This means the content of a case depends not only on a past experience, but also on alterations introduced by subsequent similar experiences. This approach deviates from standard case-based reasoning where each case contains a previous experience or a generalization of a previous one and future similar experiences do not modify the content of an existing case (but see Ram, 1993).

The basic idea underlying learning of cases is that of competitive learning, which intuitively consists of letting all existing cases compete to incorporate each new experience and allowing only the "winner" to modify itself to become more similar to the new experience. More formally, competitive learning can formulated as follows: Assume a series of samples of a vectorial "sensorial" observable[5] $\mathbf{X} = \mathbf{X}(t_0) \in \Re^n$, where t_0 is a particular point in the time coordinate, and a set of reference cases $\{\mathbf{C}_i \in \Re^n, i = 1, \ldots, N\}$. Assume that the $\mathbf{C}_i$ have been initialized in some proper way; random selection will suffice. For every given $\mathbf{X}$, the best match $\mathbf{C}_{i^*}$ according to some distance metric $d(\mathbf{X}, \mathbf{C}_i)$ is updated to match even more closely to the current $\mathbf{X}$. In other words, the best matching case $\mathbf{C}_{i^*}$, where i^* is the index of the case with the minimum distance $d(\mathbf{X}, \mathbf{C}_i)$, is altered in some way to reduce the distance $d(\mathbf{X}, \mathbf{C}_{i^*})$, and all the other cases $\mathbf{C}_i$ with $i \neq i^*$ are left intact. The result of this procedure is that the different cases tend to become specifically "tuned" to different domains of the input variable $\mathbf{X}$ in such a way that the cases $\mathbf{C}_i$ tend to be located in the input space $\Re^n$ such that they approximate to the probability density function $f(\cdot)$ in the sense of some minimal residual error (see Kohonen, 1990). In other words, those areas of the input space that are used more often are covered with more cases than areas of the input space use less often.

[5]A sample of the sensorial observable represents the values of the outputs delivered by the sensors of the robot.

The quantization error incurred by partitioning the input space with cases is given by:

$$error = \int \|\mathbf{X} - \mathbf{C}_{i*}\|^r f(\mathbf{X}) d\mathbf{X} \tag{3}$$

where $d\mathbf{X}$ is the volume differential in the input space, r is a constant, and the index i^* of the best-matching case is a function of $\mathbf{X}$:

$$\|\mathbf{X} - \mathbf{C}_{i*}\| = \min_{\text{all } i} \|\mathbf{X} - \mathbf{C}_i\| \tag{4}$$

For a squared-error criterion ($r = 2$), Equation 3 is the Euclidean metric between the environment situation $\mathbf{X}$ and the best-matching case $\mathbf{C}_{i*}$. In this case, the following steepest-descent gradient-step optimization rule ("delta rule") defines the optimal values for the $\mathbf{C}_i$ asymptotically:

$$\begin{array}{ll} \mathbf{C}_{i*}^{\text{new}} = \mathbf{C}_{i*}^{\text{old}} + \alpha_{i*}(\mathbf{X} - \mathbf{C}_{i*}^{\text{old}}) & \\ \mathbf{C}_{i}^{\text{new}} = \mathbf{C}_{i}^{\text{old}} & \text{for } i \neq i^* \end{array} \tag{5}$$

where α_{i*} is a gain coefficient that decreases monotonically with every iteration and $0 < \alpha_{i*} < 1$. This is the simplest analytical description of competitive learning, which has been widely used for vector quantization in digital telecommunication engineering as a method for unsupervised clustering (see Gray, 1984; Linde, Buzo, & Gray, 1980; Shanmugam, 1979).

However, the previous formulation is only applicable for samples of the vectorial variable $\mathbf{X}$ at a specific time. As mentioned previously, one of the objectives of the LARC algorithm is to learn *sequences* of samples that are consistent in time, that is, those sequences that tend to reoccur when used under similar situations. Thus, to keep track of sequences of situations such as the ones shown in Figure 5, the above formulation must be extended to take into account multiple samples of $\mathbf{X}$ and $\mathbf{C}_i$ along the time dimension. For this purpose, the distance function $d(\cdot,\cdot)$ needs to extended to measure the match between $\mathbf{X}(t)$ and $\mathbf{C}_i$ at different relative time positions. Such a distance function can be formulated as follows: Let t denote a discrete time index ($t = -l_Z, \cdots, -2, -1, 0$) where 0 represents the current time and negative values represent the recent past. Let $\mathbf{C}_i(m)$ represent one sample in a sequence of l_{C_i} tuples in $\Re^n$, $\mathbf{C}_i(m) \in \{\mathbf{C}_i(0), \cdots, \mathbf{C}_i(l_{C_i})\}$. Then $d(\mathbf{X}, \mathbf{C_i}, t)$, the distance between $\mathbf{X}$ and $\mathbf{C}_i$ at time position t, can be defined as follows:

$$d(\mathbf{X}, \mathbf{C_i}, t) = \frac{1}{t} \sum_{m=0}^{m=t} \|\mathbf{X}(m-t) - \mathbf{C}(m)\|^2 \tag{6}$$

where $t \in [0, \cdots, l_{C_i}]$ (refer to Figure 6 for a graphical representation).

The best-matching $\mathbf{C}_{i*}$ and the position in time of the best match t^* is determined by the following equation:

$$d(\mathbf{X}, \mathbf{C}_{i*}, t^*) = \min_{\text{all } i, 0<t<l_{C_i}} \{d(\mathbf{X}, \mathbf{C}_i, t)\} \tag{7}$$

Equation 7 represents a process which matches all the cases in all relative time positions against the given sequence of situations, and selects the case having the minimum Euclidean distance per unit of time from the set of current available cases.

The update rule for the best-matching case $\mathbf{C}_{i*}$ must also be modified to take into account multiple samples along the time dimension. Equation 8 shows the new version of this rule:

$$\begin{array}{lcll} \mathbf{C}_{i*}^{\text{new}}(m) & = & \mathbf{C}_{i*}^{\text{old}}(m) + \frac{\alpha_{i*}}{t^*-m+1}(\mathbf{X}(m-t^*) - \mathbf{C}_{i*}^{\text{old}}(m)) & \text{for } m = 0, \cdots, t^* \\ \mathbf{C}_{i}^{\text{new}} & = & \mathbf{C}_{i}^{\text{old}} & \text{for } i \neq i^* \end{array} \tag{8}$$

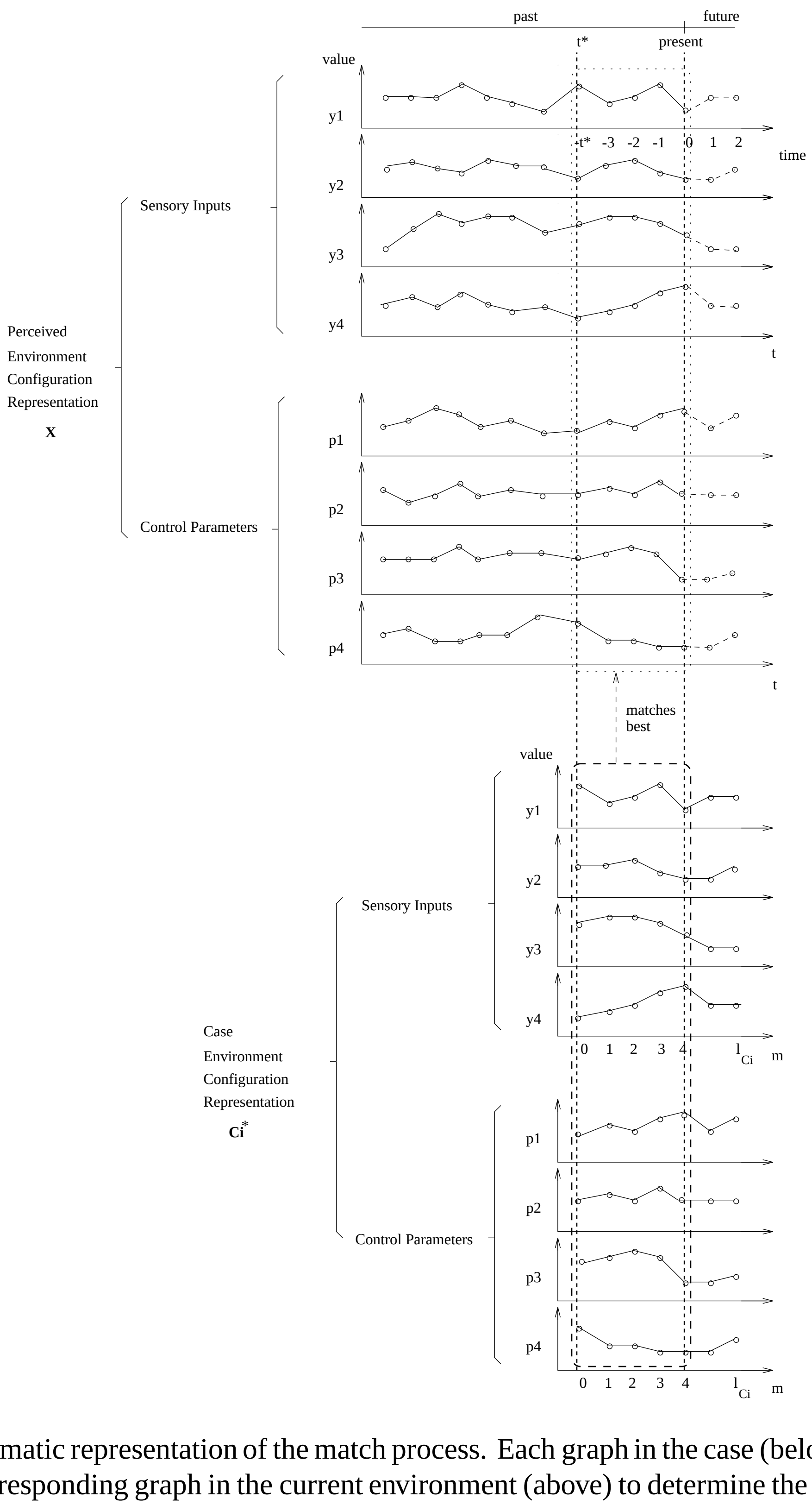

Figure 6: Schematic representation of the match process. Each graph in the case (below) is matched against the corresponding graph in the current environment (above) to determine the best-matching case $\mathbf{C}_{i^*}$ and the time position t^* of the best match.

Equation 8 represents a process which updates each case sample $\mathbf{C}_{i^*}(m)$ to match its corresponding situation sample $\mathbf{X}(m - t^*)$ with different strengths. The sample corresponding to the current time is updated more heavily than the samples taken back in time. In other words, the coefficient for updating the sample at the current time $t = 0$ is α_{i^*}, and decreases linearly down to $\frac{\alpha_{i^*}}{t^*+1}$ as the position $t = m - t^*$ of the sample in the time dimension goes from $t = 0$ down to $t = -t^*$. This update rule has the nice effect of altering the cases evenly with their use, since the first samples in a case are used more frequently than its later samples and may be exposed to the update rule more often.

Two important decisions regarding case learning are when to create a new case (as opposed to using and modifying an existing case) and when to incorporate new samples to the end of a case. The first question addresses the issue of deciding when it is time to create a new case to map a region of the input space. Creating new cases is important because this allows the system to refine the mapping in regions of the input space that have not been considered before or that have not been covered in sufficient detail. The second question addresses the issue of capturing consistent sequences of samples into a case. This allow cases to grow in length to capture longer sequences of samples.

The decision rule to create new cases is based on the idea of minimum distance clustering, which consists of using a threshold to decide whether a new experience should be incorporated into the closest case or used to create a new case. The threshold explicitly defines the border of the regions represented by each case and is updated with every experience until it converges to its best value. Given T_{i^*} as the threshold value associated with the best-matching case $\mathbf{C}_{i^*}$, the decision rule is defined as follows:

$$\text{decision rule:} \begin{cases} d(\mathbf{C}_{i^*}, \mathbf{X}, t^*) \leq T_{i^*} & \text{Adapt } \mathbf{C}_{i^*} \text{ using Equation 8.} \\ d(\mathbf{C}_{i^*}, \mathbf{X}, t^*) > T_{i^*} & \text{Create a new case using } \mathbf{X}. \end{cases} \tag{9}$$

The update rule for T_{i^*} is defined by the following equation:

$$T_{i^*}^{\text{new}} = T_{i^*}^{\text{old}} + \beta(\mu_{d_{i^*}} + \gamma\sigma_{d_{i^*}} - T_{i^*}^{\text{old}}) \tag{10}$$

where β is a learning constant, γ is an integer constant, and $\mu_{d_{i^*}}$ and $\sigma^2_{d_{i^*}}$ are the mean and variance of the distance obtained during previous retrievals of case $\mathbf{C}_{i^*}$. In other words, the threshold value T_{i^*} is always updated towards a limiting value that is equal to the mean of the distance of the experiences to the centroid of the case plus a multiple of its standard deviation. This means that a new case is created only when the distance between the current situation and the best-matching case is greater than "usual". Intuitively, if the best available case is not as good a match to the new situation as it has been to the situations in which it has been useful in the past, then it is likely that the system is encountering a situation for which the best available case isn't a very good match and a new case needs to be created.

The purpose of the decision rule for case learning (Equation 9) and the update rule (Equation 10) is to create regions in the input space with sizes that are inversely proportional to the value of the probability density function of the input space. Thus, the portions of the input space that are used more often are mapped with smaller regions than portions of the input space used less often. In this way, as depicted in Figure 7, the mapping can be dynamically refined and specialized to those portions of the input space used more often.

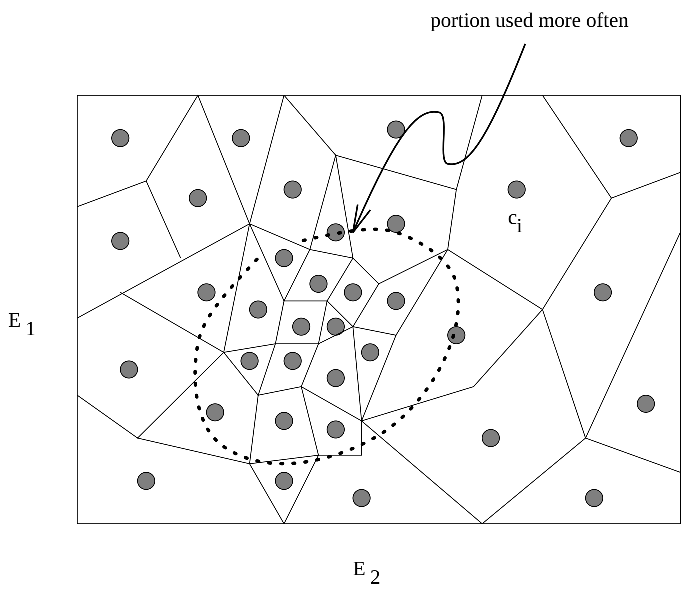

Figure 7: Quantization of the input space in region with different sizes. The size of each region depends on the frequency of use of each portion of the input space.

The decision rule used to extend the length of the best-matching case is based on how well the case is able to predict the next situation given the current sequence of situations. Intuitively, if the best-matching case runs short in matching the current sequence of situations, then the case is extended to incorporate the current sample. That is, if there is no other case that is closest to end of the sequence, then the best decision is to extend the case that best matches the beginning of the sequence. In this way, the next time the case is retrieved under similar situations it will be longer and able to predict a longer sequence of situations. Mathematically, this can be accomplished by modifying Equation 7 to match cases at all relative time positions, including those that go one unit beyond the length l_{C_i} of the matching case $\mathbf{C}_i$. If the best-matching case $\mathbf{C}_{i^*}$ occurs at relative time position $t^* = l_{C_i} + 1$, then case $\mathbf{C}_{i^*}$ is running short and must be extended. Equation 11 specifies the procedure:

$$\text{If } t^* = l_{C_{i^*}} + 1, \quad \text{then} \left\{ \begin{array}{l} l^{\text{new}}_{C_{i^*}} = l^{\text{old}}_{C_{i^*}} + 1 \\ \mathbf{C}_{i^*}(l^{\text{new}}_{C_{i^*}}) = \mathbf{X}(0) \end{array} \right. \tag{11}$$

The above specifications are achieved by an algorithm that learns to partition the input space into several cases, as described in Figure 8. The algorithm works by incorporating new experiences sequentially, creating new cases or updating the content of existing cases as required. Its asymptotic behavior is to partition the input space into regions that minimize the quantization error as expressed in Equation 3. The cases created by this algorithm may be thought of as *feature-sensitive detectors* that represent consistent sequences of associations in the input space. The sequences that are captured into cases are those that are intrinsic to the environmental situation and the dynamics of the system. The main characteristics of this algorithm are that it creates and self-organizes cases to partition the input space based on the probability density function of the input space. Also, it works in an incremental manner since it incorporates new samples step-by-step. Finally, the algorithm can construct prototypical representations of sequences of situations that have proven to be consistent over time.

```
For every perception-action cycle of the robot:
1.  Take a sample of the current sensory information:
    $[X_1(0),\cdots,X_n(0)] = [\texttt{variable}_1,\cdots,\texttt{variable}_n]$
2.  Retrieve the best-matching case $\mathbf{C}_{i^*}$ using Equation 7.
    Best match distance:  $d^* = d(\mathbf{X},\mathbf{C}_{i^*},t^*)$
3.  Update case distance statistics (mean and variance of $d^*$):  $\mu_{d^*},\sigma^2_{d^*}$
4.  Update decision threshold $T_{i^*}$ for best matching case:
    $T^{\text{new}}_{i^*} = T^{\text{old}}_{i^*} + \beta(\mu_{d_{i^*}} + \gamma\sigma_{d_{i^*}} - T^{\text{old}}_{i^*})$
5.  Apply decision rule to create a new case:
    If $d^* < T_{i^*}$
       Update $\mathbf{C}_{i^*}$ using Equation 8.
       best = $i^*$
    Else
       Create a new case:  $\mathbf{C}_{\text{new}} = \mathbf{X}(0)$
       best = new
6.  Apply decision rule to extend the best-matching case:
    If $t^* = l_{C_{i^*}} + 1$
       Incorporate a new sample to $\mathbf{C}_{\text{best}}$:
       $l_{C_{\text{best}}} = l_{C_{\text{best}}} + 1$
       $\mathbf{C}_{\text{best}}(l_{C_{\text{best}}}) \leftarrow \mathbf{X}(0)$
7.  Prepare for next sample:  $\mathbf{X}(t-1) \leftarrow \mathbf{X}(t),\ \forall t \in [-l_Z+1,\cdots,0]$
```

Figure 8: Case Learning Algorithm.

The cases created by this algorithm represent consistent sequences that have occurred in the past. Control parameters are associated with each case, representing the specific parameterization of the reactive controller that should be used under the situation encoded by the case. However, while these sequences are "correct" in the sense that they represent generalizations about the system-environment interaction, they may or may not be useful to the actual goal that the system is currently pursuing. In order to learn useful control parameters, an external reward signal is used to bias the case learning algorithm. This is discussed next.

3.2.2 Learning Useful Control Parameters

Reinforcement learning deals with the issue of increasing the tendency to execute actions that produce satisfactory states of affairs. Basically, when an agent performs an action in a given state, it receives an external reward signal. If the reward is positive, the agent strengthens the association between the action and the state so that the agent will tend to execute the same action in future similar states. Conversely, if the reward signal is negative, the agent weakens this association in order to decrease the tendency to perform that same mistake in the future. Combining the algorithm for learning cases described previously with reinforcement learning will allow the system to learn cases that are not just consistent over time but that are also useful according to the reward signal.

Given a set of cases $\mathbf{C}_i$, and assuming that the observable $\mathbf{X} \in \Re^n$ consists of a concatenation of an environmental situation vector $y_p = \mathbf{X}^{\text{E}} \in \Re^{n_E}$ and a control parameter vector $\theta = \mathbf{X}^{\text{P}} \in \Re^{n_P}$ where $n = n_E + n_P$, then the mapping $M(\cdot)$ consists of finding out the best-matching case $\mathbf{C}_{i^*}$ for a given situation x and returning the parameter vector associated with the case $\theta = \mathbf{C}^{\text{P}}_{i^*}(t^*)$, where $\mathbf{C}^{\mathrm{P}}$ is the notation for extracting the control parameter vector from the concatenation vector. To reinforce parameters that are useful, the update procedure should not blindly update the best-matching case to be more similar to the current situation; instead, it should take into account the reward feedback in such a way that control parameters that produce beneficial rewards are remembered while control parameters that produce perjudicial rewards are modified. This is achieved by comparing the reward obtained during the current situation with the expected reward obtained during past similar situations, and updating the control parameters only if the reward obtained during the current situation is better than the expected reward of the case. In other words, a case is adapted to be more similar to the current situation only if the current situation is "better" than the situation encoded in the case as evaluated by the reward signal. Thus, each case $\mathbf{C}_i$ must keep track of the expected reward or *utility* U_i received every time it is used. The update rule, originally stated in Equation 8, must be applied only when the reward received during the current situation is greater than the utility of the case.

The introduction of the reward signal acts as a filter for training data to the case-based learning algorithm, thereby providing an "input bias" (Cox & Ram, 1994) to the algorithm. Only those sequences that are considered relatively better than the sequence in the best-matching case are incorporated and learned. This rule produces the reinforcement effect of remembering those parameters that lead to good rewards and forgetting those that lead to bad rewards. To promote self-improvement in performance, one more condition is required: The best parameterization of the reactive controller in the given environmental situation must be selected. To do this, all cases that are "close enough" to the current situation are considered, and the one that promises the best utility is selected. This approach is somewhat different from standard case-based reasoning in which the best matching case is used to guide action; here, several of the best matches are considered and the most useful one is chosen. Equation 12 specifies the procedure:

$$\mathbf{C}_{\text{best}} = \max_{\text{all } i, 0<t<l_{C_i}} \{U_i \mid d(\mathbf{X}, \mathbf{C}_i, t) < T_{i^*}\} \tag{12}$$

The decision threshold T_{i^*} associated to the best-matching case is used to define what is "close enough". That is, only those cases whose distance to the current situation is less than T_{i^*} are considered. If the utility of the best case selected by Equation 12 is not "good enough" (as determined by a fixed threshold), then a random set of parameters is selected for the reactive controller. Random search is the only strategy for optimization when there is no additional knowledge about the direction in which to change the parameters in order to optimize the external reward. Heuristic functions, if available for a particular application, can be easily incorporated at this step to focus the search for useful parameters.

Figure 9 shows the complete LARC algorithm that integrates the procedure for partitioning the input space into cases with reinforcement learning of useful control parameters. The proposed algorithm shares the characteristics of the algorithm in Figure 8, namely, it learns to partition the input space into regions that minimize the quantization error by incorporating new experiences in a sequential fashion. Additionally, it uses an external reward signal to remember only those control parameters that produce useful results. Thus, the cases learned by this algorithm represent not only a set of feature-sensitive detectors, but also specialized mappings between environmental situations and control parameters that produce useful performance.

```
For every perception-action cycle of the robot:
1.  Take a sample of the current sensory information and reward signal:
    $[X^{\mathrm{E}}_1(0), \cdots, X^{\mathrm{E}}_{n_E}(0), X^{\mathrm{P}}_1(0), \cdots, X^{\mathrm{P}}_{n_P}(0)] = [y_{p_1}, \cdots, y_{p_{n_E}}, \theta_1, \cdots, \theta_{n_P}]$
    $R_Z =$ reward
2.  Update utility of previous case given $R_Z$.
3.  Retrieve the best-matching case $\mathbf{C}_{i^*}$ using Equation 7.
    Best match distance:  $d^* = d(\mathbf{X}, \mathbf{C}_{i^*}, t^*)$
4.  Update case distance statistics (mean and variance of $d^*$):  $\mu_{d^*}, \sigma^2_{d^*}$
5.  Update decision threshold $T_{i^*}$ for best matching case:
    $T^{\mathrm{new}}_{i^*} = T^{\mathrm{old}}_{i^*} + \beta(\mu_{d_{i^*}} + \gamma\sigma_{d_{i^*}} - T^{\mathrm{old}}_{i^*})$
6.  Apply decision rule to create a new case:
    If $d^* < T_{i^*}$
        If $R_Z > U_{i^*}$ Update $\mathbf{C}_{i^*}$ using Equation 8.
    Else
        Create a new case:  $\mathbf{C}_{\mathrm{new}} = \mathbf{X}(0)$
7.  Apply decision rule to extend the best-matching case:
    If $t^* = l_{C_{i^*}} + 1$
        Incorporate a new sample to $\mathbf{C}_{\mathrm{best}}$:
        $l_{C_{\mathrm{best}}} = l_{C_{\mathrm{best}}} + 1$
        $\mathbf{C}_{\mathrm{best}}(l_{C_{\mathrm{best}}}) \leftarrow \mathbf{X}(0)$
8.  Select best case for new parameterization using Equation 12:
    Use suggested parameters only if the expected reward is good enough:
    If $U_{\mathrm{best}} >$ Utility
        parameters $\leftarrow \mathbf{C}^{\mathrm{P}}_{\mathrm{best}}$
    Else
        parameters $\leftarrow$ random
9.  Prepare for next sample:  $\mathbf{X}(t-1) \leftarrow \mathbf{X}(t),\ \forall t \in [-l_Z + 1, \cdots, 0]$
```

Figure 9: The LARC Algorithm.

4 Case Study

This section demonstrates the application of the algorithm proposed in the previous section for adapting the parameters of a schema-based reactive controller (Arkin, 1989), resulting in a learning adaptive reactive controller for robotic navigation. In schema-based reactive control, basic behaviors or *motor schemas* such as obstacle avoidance and movement towards a goal individually recommend specific motor actions which are then combined to produce the final action of the agent. Potential fields or forces are used to represent the recommendations of the motor schemas; thus, a final motor action can be computed as the sum of the forces of each motor schema and delivered to the robot's effectors. Each motor schema uses current sensory information from the environment and internal control parameters to compute its potential field, which recommends the direction and speed at which the robot is to move given current environmental conditions. For example, the motor schema AVOID-STATIC-OBSTACLE directs the system to move itself away from detected obstacles, and its control parameter **Obstacle-Gain** determines the magnitude of the repulsive potential field generated by the obstacles perceived by the system. The forces produced by all the schemas are then summed to produce a resulting force that directs the actual movement of the robot. Simple behaviors, such as wandering, obstacle avoidance, and goal following, can combine to produce complex emergent behaviors in a particular environment. Different emergent behaviors can be obtained by modifying the simple behaviors. A detailed description of schema-based reactive control methods can be found in Arkin (1989).

Different combinations of schema parameters cause different emergent behaviors to be exhibited by the system (see figure 10). Traditionally, parameters are fixed and determined ahead of time by the system designer or through optimization techniques such as genetic algorithms (e.g., Ram, Arkin, Boone, & Pearce, 1994). However, on-line adaptation of control parameters can enhance navigational performance (see Ram, Arkin, Moorman, & Clark, 1992). While earlier approaches (such as Ram, Arkin, Moorman, & Clark, 1992) use a pre-defined adaptive law to improve performance, we have argued that it is beneficial for the system to learn an adaptive law through experience. The following subsections describe in detail a system called SINS (Self-Improving Navigation System) that uses the proposed LARC algorithm to learn to adapt the control parameters of a schema-based reactive controller.

4.1 System Architecture

The SINS mobile robotic system consists of a navigation module, which uses the AuRA schema-based reactive control architecture (Arkin, 1989) implemented on a Denning MRV-III robot, and an on-line adaptation and learning module, which uses the LARC learning method described earlier. The navigation module is the reactive controller, which is responsible for moving the robot through the environment from the starting location to the desired goal location while avoiding obstacles along the way. The adaptation and learning module has two responsibilities. First, it must perform on-line adaptation of the reactive control parameters being used in the navigation module to get the best performance, that is, it must apply the adaptive law. The control parameters are based on recommendations from cases that the system has learned through its experience. Second, it must monitor the progress of the system and incrementally modify the cases library to get better

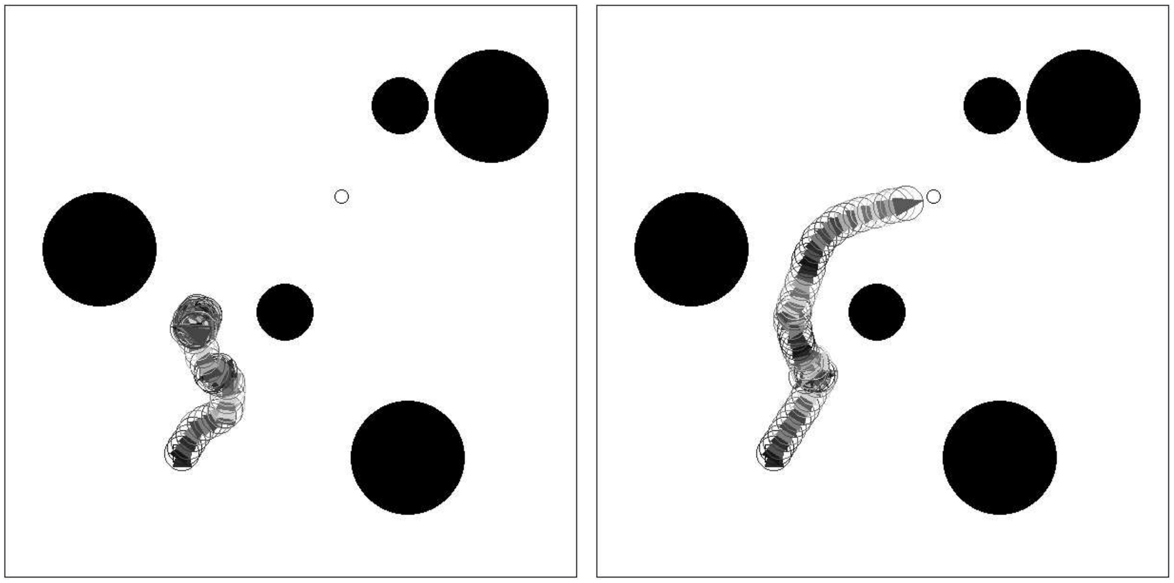

Figure 10: Typical navigational behaviors of different tunings of the reactive control module. The figure on the left shows the non-learning system with high obstacle avoidance and low goal attraction. On the right, the learning system has lowered obstacle avoidance and increased goal attraction, allowing it to "squeeze" through the obstacles and then take a relatively direct path to the goal (up-center).

performance in the future, that is, it must learn the adaptive law. Figure 11 shows the system functional architecture, which is an instantiation of Figure 1 for this particular implementation.

The reactive controller in this system uses three motor schemas: AVOID-STATIC-OBSTACLE, MOVE-TO-GOAL, and NOISE. AVOID-STATIC-OBSTACLE directs the system to move itself away from detected obstacles. MOVE-TO-GOAL schema directs the system to move towards a particular point in the terrain. The NOISE schema makes the system move in a random direction; it is used to escape from local minima and, in conjunction with other schemas, to produce wandering behaviors. Each motor schema has a set of parameters that control the potential field generated by the motor schema. In this research, we used the following parameters: **Obstacle-Gain**, associated with AVOID-STATIC-OBSTACLE, determines the magnitude of the repulsive potential field generated by the obstacles perceived by the system; **Goal-Gain**, associated with MOVE-TO-GOAL, determines the magnitude of the attractive potential field generated by the goal; **Noise-Gain**, associated with NOISE, determines the magnitude of the noise; and **Noise-Persistence**, also associated with NOISE, determines the duration for which a noise value is allowed to persist.

4.2 Input Space Representation

The observable $\mathbf{X}$ consists of the concatenation of an environmental situation vector $\mathbf{X}^{\text{E}} = y_p$ of four variables and a control parameter vector $\mathbf{X}^{\text{P}} = \theta$ of four variables. The four variables for the environmental situation vector are **Obstacle-Density-Ahead**, **Obstacle-Density-Behind**, **Obstacle-Density-Right**, and **Obstacle-Density-Left**. Each of these variables provide a measure of the occupied areas that impede navigation in the direction towards, contrary to, right of, and left

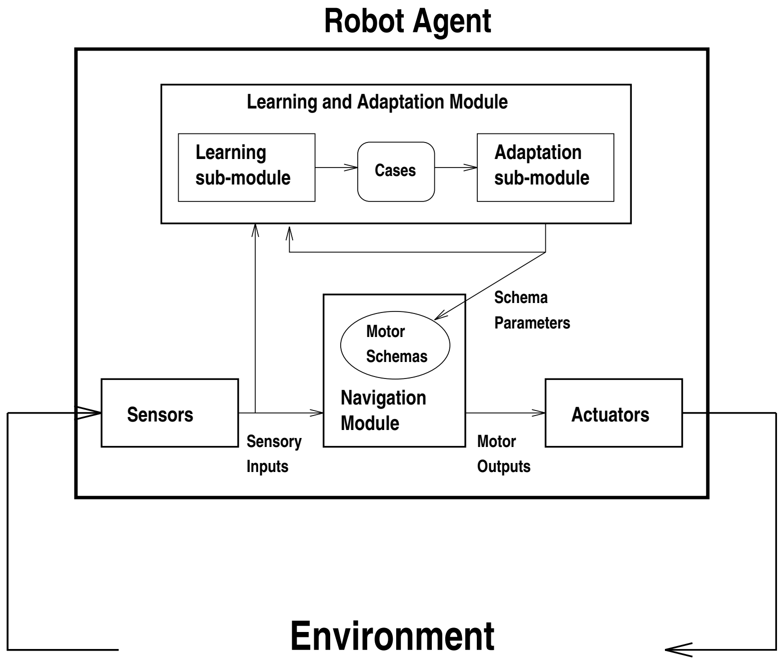

Figure 11: Functional architecture of the SINS system.

of the direction of the perceived goal respectively. These input vectors are computed and constantly updated using the information received from the robot's sensors. The four variables for the control parameter vector represent the schema parameter values used to adapt the navigation module, one for each of the schema parameters (**Obstacle-Gain**, **Goal-Gain**, **Noise-Gain**, and **Noise-Persistence**) discussed earlier. The values are set periodically according to the recommendations of the best case. The new values remain constant over a *control interval* until the next setting period.

The reactive controller operates on a Denning MRV-III robot. The robot uses a Denning sonar ring which has twenty-four laboratory grade Polaroid Ultrasonic Rangefinders equally spaced over 360 degrees in a plane parallel to the floor. Range data is obtained by emitting ultrasonic pulses from the sensors. The four input variables are calculated by averaging the range data from the six consecutive Rangefinders that lay in each of the four quadrants (e.g., ahead, left, right, behind). Figure 12 shows a diagram of the robot's ultrasonic sensors and the corresponding input variables.

4.3 Reward Signal

The selection of the reward signal depends on processing and performance constraints. Processing constraints refer to the ability of the robot to compute the reward signal taking into account the information delivered by the sensors. For example, a common problem faced by reactive controllers is that they may direct the robot inside a box canyon. Such a situation occurs when the robot, using its current sensory capabilities, determines that it can move between two obstacles beyond which the destination lies, but then later discovers that there is no (or not enough) space to go between them and thereby gets trapped in a local minimum. A reward signal that punishes an action that drives the robot into a box canyon is computationally expensive since it must perform complex

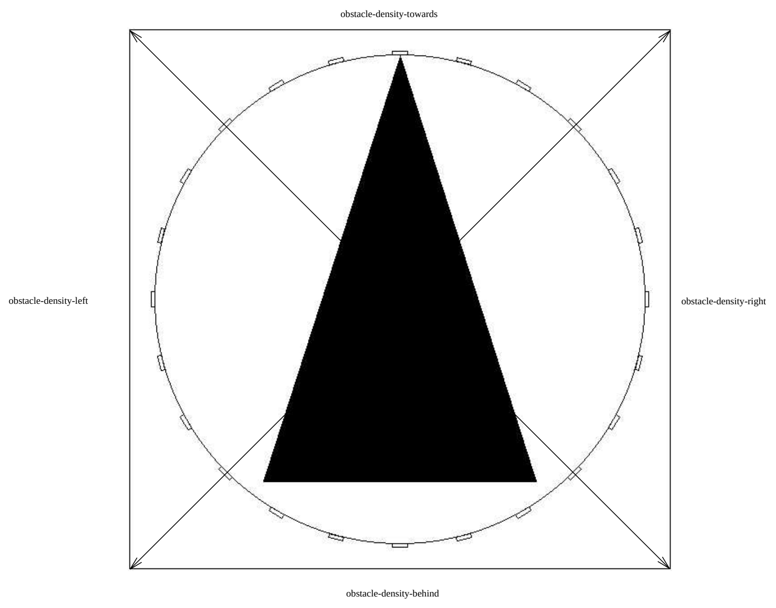

Figure 12: Ultrasonic sensors and the input variables.

information processing using the robot's sensory information to determine if the robot is inside of a box canyon. Performance constraints refer to the metrics used to measure the performance of the system. The reward signal should be selected to reward actions that help to improve the performance metric used to evaluate the robot's task. For example, in autonomous robot navigation it is often important to reach the destination point quickly while avoiding collisions. Then, the reward signal should be selected to punish detected collisions and periods of inactivity (i.e., periods in which the robot does not move or moves very slowly).

In the current system, successful navigation is defined as reaching the destination point as fast as possible and without any collisions. Thus, the robot needs to minimize the time taken to reach the goal with the fewest number of collisions. This is represented in the following reward signal:

$$r = \alpha \text{ virtual\_collisions} + \beta \left(\frac{\text{max\_velocity} - \text{velocity}}{\text{max\_velocity}}\right) \tag{13}$$

This formula is based on the assumption that the only sensory information available to the robot consists of the 24 ultrasonic sensors and the shaft encoders. This means that the robot receives a negative signal every time a virtual collision[6] is detected and a positive signal every time the robot is moving; the faster it moves the stronger the signal. The coefficients α and β determine the relative importance of avoiding collisions versus maximizing velocity for a particular application. This reward signal is not computationally expensive to evaluate, it uses information provided by the robot sensors, and it is consistent with the performance metric defined for successful autonomous robotic navigation.

[6]Since there is a real robot involved we can not afford actual physical collisions. The term virtual collision to refer to the situation when the robot come so close to an obstacle that a physical collision is imminent. When this occurs, AuRA automatically diverts all its resources to focus on eliminating this imminent danger.

5 Evaluation

This section describes the evaluation of the LARC algorithm in the context of the SINS system. The objective of the evaluation is twofold: first, to verify that the algorithm actually learns a useful adaptive law, which improves the performance of the navigation task; and second, to determine how design decisions affect the performance of the system under different environmental conditions. The first objective can be verified by constructing an empirical model that relates the performance metric of the system with the amount of experience or *experience level*. We expect the model to show a statistically significant improvement in the performance of the system with experience level. The second objective, optimization of the system configuration, is useful because it enable us to shed some light on how a range of design decisions affect the behavior of the system and how to select appropriate design parameters to improve the performance of the system. The first objective enable us to verify the theory supporting the proposed multistrategy learning algorithm, and the second objective enables us to understand the relationship between design decisions and system performance under different environmental conditions. To achieve these objectives, we evaluated the LARC algorithm presented above in the context of the SINS system using extensive simulations across a variety of different types of environments, performance criteria, and system configurations.

Two characteristics of the mobile robotic system under study are its complexity and the large amount of interaction with its environment. The complexity of the system is due to the multistrategy learning algorithm, which can adapt the control parameters after every control interval and modify the behavior of the system. The large amount of interaction is due to the intrinsic nature of reactive control and the learning algorithm. A direct consequence of these characteristics is that the behavior of the system has many sources of variability, which cause any performance metric defined to evaluate the system's behavior to vary as well. This in turn makes it difficult to assess the significance of the performance of the system in a specific situation since the system may perform differently under the same or similar circumstances in a different set of runs. Furthermore, the components of the system are tightly coupled and highly interactive. Thus, for example, straightforward learning curves or ablation studies are inadequate to evaluate this system.

To solve this problem, we used a systematic evaluation methodology (proposed by Santamaría & Ram, 1994) to evaluate the performance of the mobile robotic system. In this methodology, statistical tools are employed to analyze the change in the performance of the system in terms of changes in design parameters and environmental characteristics. In the analysis, the system is evaluated through systematic experiments defined to filter out undesirable sources of variability. The result of the analysis is an empirical model that relates the performance of the system to the design decisions and environmental characteristics. This empirical model can be used to understand the behavior of the system in terms of the theory and configuration of the system, to select the best system configuration for a given environment, and to predict how the system will behave in response to changing problem characteristics. Systematic empirical analysis based on statistical tools can also be used to verify the significance of the system's performance.

We developed a software package, `driver++`, which is an implementation of Arkin's (1989) AuRA architecture on a Denning MRV-III. `driver++` can work in two modes: real and simulated. In real mode, `driver++` receives information from sensors and sends commands to the robot through a radio link. In simulated mode, `driver++` simulates both the values of the Rangefinders as specified by a given arrangement of obstacles in a simulated environment and the dynamics of the real robot as specified by motor commands. The results presented in this section are based on experiments performed using `driver++` in simulated mode, since this allows us to run several hundred experiments with systematically different design parameters and environmental configurations.

In these experiments, the robot navigates in randomly generated environments consisting of rectangular bounded worlds. Each environment contains circular obstacles, a start location, and a destination location. The location, number, and radius of the obstacles are randomly determined to create environments of varying amounts of *clutter*, defined as the ratio of free space to occupied space. In our experiments, we used 15% cluttered worlds which correspond to "typical" laboratory and similar indoor environments in the real world.

The performance of the mobile robotic system varies across different worlds, system configurations, and amounts of experience. Moreover, due to the nature of the task and the architecture of the system, the robot can perform differently given the same world and case library. The reason for this is that the adaptation and learning module tunes the navigation module randomly when no appropriate case exists; this allows the system to explore and discover new feature-sensitive detectors or *regularities* with better expected utility. This means that any performance metric used to evaluate system needs to be treated as a random variable and statistical estimation techniques should be used to assess its mean value.

In this article, we will focus on how two factors influence the performance of the robot: maximum number of cases and utility threshold. The former is relevant to the case-based learning aspect of the algorithm and the latter to the reinforcement learning aspect. We will also consider how the experience level influences the performance of the robot and verify that the system indeed improves its performance as the experience level increases.

5.1 Experimental Design and Data Collection

To collect data for the evaluation analysis, we performed several runs on the system. A run consisted of placing the robot at the start location and letting it run until it reached the destination location. The data for the estimators was obtained after the system terminated each run. This was to ensure that we were consistently measuring the effect of learning across experiences rather than within a single experience (which is less significant on worlds of this size anyway).

We evaluated the performance of the robot using the median value among five replicates of the time it takes to solve a world. The reason for this is that the median is a robust estimator of the mean and is not too sensitive to outliers. Outliers are common in schema-based reactive control since the system can get trapped in local minima points, resulting in a significant change in the behavior of the system. An experiment consisted of measuring the time the robot takes to solve a world across five independent runs under the same conditions (i.e., same case library, number of cases, case size, and level of experience, world clutter) and reporting the median among the five runs as the response variable.

Two sets of experiments were designed to satisfy the objectives of our evaluation. In the first set (called "learning profile" experiments), we ran the system under the same 15% cluttered world and with an initial configuration based on preliminary experiments. In the second set (called "system

Table 1: Design for Experiment 1.

Maximum Cases	15
Utility Threshold	.35
Experience Levels	1-30
World Clutter	15%
Replicates	5
Response Variable	Time
Total runs	150

Table 2: Design for Experiment 2.

Maximum Cases	10, 15, 20
Utility Threshold	.25, .35, .45
Experience Level	25
World Clutter	15%
Replicates	5
Response Variable	Time
Total runs	45

profile" experiments), we ran different system configurations by systematically varying the factors around the initial configuration. In this way, we determined the influence of each design parameter on the performance of the system; this allows us to decide how to change the system's configuration for best performance. Both sets of experiments also allowed us to verify that the LARC algortihm does in fact improve the performance of the system through experience. Tables 1 and 2 show the design of each experiment.

5.2 Learning Profile

We used the data collected from the first experiment to construct an empirical model to verify improvement in the performance of the system with experience. The desired model has the median time (T) as the response variable and experience level (E) and its squared term (E^2) as the predictors or regressors. In this way, if such a model is found to be statistically significant (i.e., the model shows that the amount of experience is related to the response variable), we can conclude that the system actually learns and improves performance under the given configuration and environmental conditions. Equation 14 shows the complete hypothetical model for this experiment.

$$\mathrm{T} = \alpha_0 - \alpha_{\mathrm{E}}\mathrm{E}' + \alpha_{\mathrm{EE}}\mathrm{E}'^2 + \varepsilon \tag{14}$$

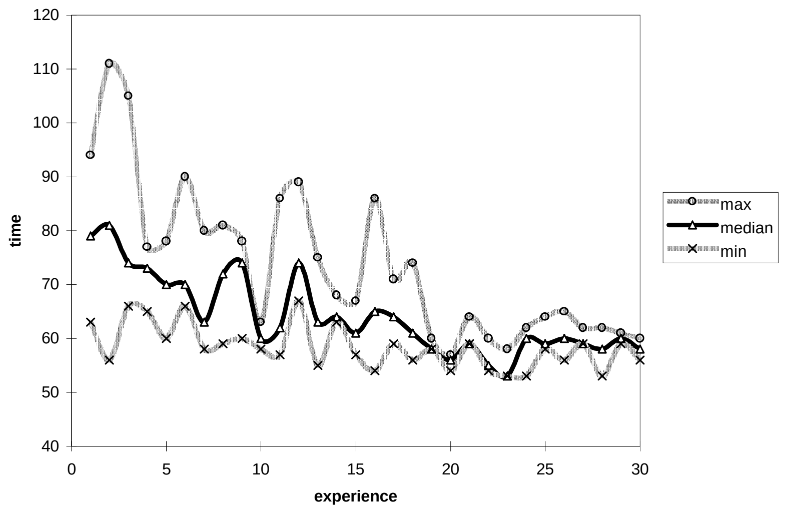

Figure 13: Data from Experiment 1. The graph shows the median curve (solid) and the maximum and minimum responses (shaded) from the five replicate runs at each experience level.

where E' is the standardized[7] value of the variable E (i.e., $E' = \frac{E-\bar{E}}{\sqrt{var(E)}}$). Assuming that the mathematical relationship between the response variable and the independent variables is "smooth", a second order polynomial expression of that relationship, such as the one proposed by the model, is a good approximation. Intuitively, this approach corresponds to fitting a statistical model to a suitable sample of traditional "learning curves" obtained from the system and analyzing the model for statistical significance.

Figure 13 shows the data gathered during experiment set 1. The graph shows the median curve (solid) and the maximum and minimum responses (shaded) from five replicate runs at each experience level. Figure 14 shows the median time and the least mean squared fitted model given by Equation 14; as can be seen from the figure, the model describes the actual performance of the system very well. Table 3 shows the analysis of variance (ANOVA) of the regression and shows that the fitted model is statistically significant (P-value $= 0.0000$). That is, given the data from this experiment, we can not accept the hypothesis that the coefficients from Equation 14 are all zero, and therefore we accept the alternative hypothesis. This means we can conclude that the system learns with experience and that the improvement in performance is significant. Table 4 shows the statistical results for each individual parameter in the model, their significance, and the 95% confidence interval estimation of its value.

5.3 System Profile

We used the data collected from the second set of experiment to analyze the impact of the design parameters on the learning performance of the system. The second experiment is a 3^2 factorial design in which the two system design factors being investigated, maximum number of cases and

[7]Use of standardized values instead of the original values helps to reduce roundoff errors and other problems with multicollinearity between independent variables (see Neter, Wasserman, & Kutner, 1989).

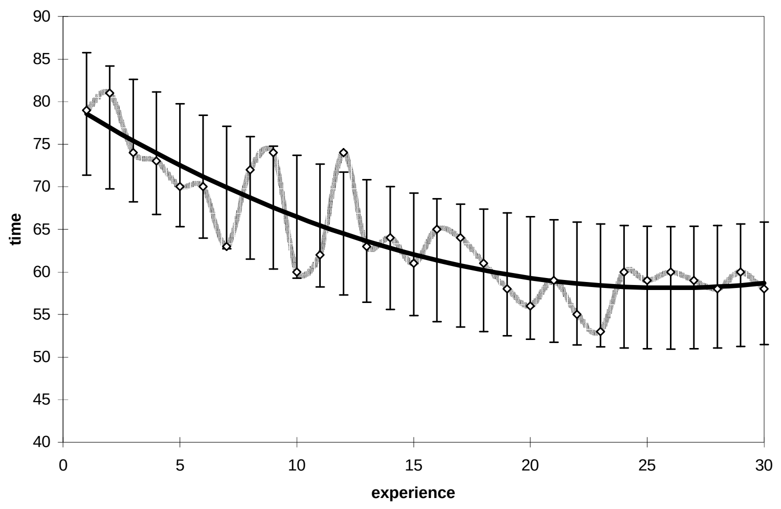

Figure 14: Fitted model from Experiment 1. The graph shows the median curve (shaded) and the fitted model (solid) with error bars at each experience level.

Table 3: ANOVA table for the regression model of Experiment 1.

Source	df	SS	MS	F	P-value
Regression	2	1202.31	601.16	44.365	2.93E-09
Residual	27	365.85	13.55		
Total	29	1568.16			

Table 4: Model coefficients for Equation 14.

Coefficients	Value	Std. Error	P-value	95% C.I.
α_0	61.70	1.01	0.000	(59.63,63.77)
α_E	-6.04	0.68	0.000	$(-7.44,-4.64)$
α_{EE}	2.55	0.78	0.003	(0.95,4.14)

Table 5: Model coefficients for Equation 15.

Coefficients	Value	Std. Error	P-value	95% C.I.
β_0	59.80	0.94	0.000	(57.89,61.71)
β_C	0.00	0.43	1.000	(−0.87,0.87)
β_U	2.89	0.43	0.000	(2.02,3.76)
β_{CC}	0.00	0.61	1.000	(−1.24,1.24)
β_{UU}	−0.20	0.61	0.740	(−1.44,1.03)
β_{CU}	0.00	0.43	1.000	(−0.88,0.88)

utility threshold, are systematically varied around the values used during the first experiment.[8] A factorial experimental design allows us to verify if the factors have any influence in the performance of the system. Thus, we can determine how to change the factors to maximize the expected outcome in the performance of the system.

This experiment provides information on how the considered design parameters affect the performance of the system at experience level 25. Table 2 shows the values of this experimental setting. The reason for evaluating the system at an experience level of 25 is because this is approximately the point in which performance is maximized according to the fitted model from Experiment 1 (see Figure 14).

Equation 15 shows the complete hypothetical model for this experiment. The model include the effects of both design parameters: maximum number of cases (C) and utility threshold (U), the effects of their quadratic terms (C^2 and U^2), as well as their second order interaction (CU):

$$\begin{aligned} T &= \beta_0 + \beta_C C' + \beta_U U' + \\ &\quad \beta_{CC} C'^2 + \beta_{UU} U'^2 + \\ &\quad \beta_{CU} C'U' + \\ &\quad \varepsilon \end{aligned} \tag{15}$$

Considering this model, the direction of steepest ascent in which to change the system configuration parameters to optimize performance can be found using standard calculus techniques, i.e., by calculating the gradient of Equation 15.

Table 5 shows the statistical results for each individual parameter in the model as well as the 95% confidence interval estimation of its value. Table 6 shows the analysis of variance (ANOVA) of the regression and shows that the fitted model is statistically significant (P-value $= 0.0000$). That is, given the data from this experiment, we can not accept the hypothesis that the coefficients from Equation 15 are all zero, and therefore we accept the alternative hypothesis. Additionally, from table 5 we can conclude that the maximum number of cases to use is irrelevant to the performance metric (i.e., both terms associated with the maximum number of cases, β_C and β_{CC}, and the quadratic term for utility threshold have high P-values).

Given the estimation of the parameters showed in Table 5, the direction of steepest ascent is along the U' axis in the positive direction due to a positive β_U. Future experimental designs can

[8] A 3^2 factorial design consists of systematically varying two factors using three values or levels for each factor. The total number of runs is $3^2 = 9 \times$ # of replicates.

Table 6: ANOVA table for the regression model of Experiment 2.

Source	df	SS	MS	F	P-value
Regression	5	368.40	73.68	9.140	8.11E-11
Residual	39	314.40	8.06		
Total	44	682.80			

explore this alternative and find a configuration that optimizes the response of the system. A similar analysis can be carried out to optimize performance for other applications in the domain of interest.

5.4 Discussion of Results

The performance of the robot is very complex and depends not only on simple terms but also on their interactions. The evaluation shows that the median time the system takes to solve a 15% cluttered world decreases with the experience level. Figure 13 also shows that the variability in the response also decreases with experience. This demonstrate another aspect of the learning behavior of the system, namely, that the performance not only improves with experience but also becomes more predictive. In other words, as the system increases its experience, it tends to behave similarly under similar situations and this reduces variability in the performance.

Experiment set 2 showed that a change in the system's configuration by increasing the utility threshold improves the performance at experience level of 25. The value of the maximum number of cases does not influence the performance at that level of experience. This was, to us, a surprising result; however, we then analyzed the memory of cases for each of the system's configurations and found that the system had created only 10 cases after 30 runs. Moreover, experiment set 2 suggests that in order to explore the possibility of further improvement in the performance of the system, we would need to run additional experiments with a higher utility threshold (the direction of steepest ascent). Intuitively, this makes sense since the system will use the control parameters from cases that have produce at least the required level of utility, or randomized control parameters otherwise (step 8 of algorithm in Figure 9). It is also possible that in a more complex application, more than 10 cases will be required to adequately map the system-environment interaction, which will increase the significance of the maximum number of cases that the system is allowed to construct. In addition, other design decisions may also play an important role in the performance of the system; this is an issue for future research but one that can be addressed using the above evaluation methodology.

In summary, the evaluation was useful to verify and understand several aspects of the proposed adaptive reactive controller. In particular:

- The evaluation showed that the robot does improve its performance with experience (Equation 14 and Figure 14) and that this improvement is statistically significant (Tables 4 and 3).
- Considering the two design parameters, maximum number of cases and utility threshold, the evaluation showed that the performance of the system in a 15% cluttered world depends on the utility threshold and not the maximum number of cases (Equation 15 and Tables 5 and 6).

- The evaluation suggests a higher utility threshold would improve the performance of the system (Equation 15); thus, these experiments have predictive value as well.
- The evaluation showed how that the system not only learns to improve performance, but also decreases the variability in its behavior as experience increases (Figure 13).

6 Discussion and Related Work

The performance of a reactive controller can be enhanced by learning when and how to adapt its control parameters according to the environmental situation. A multistrategy learning algorithm that combines ideas from case-based reasoning and reinforcement learning was proposed to learn how to perform this adaptation. The algorithm is able to learn cases that partition the the input space into regions such that the quantization error is minimized. This minimization is useful because it allows the system to use the control parameters associated with the centroid of a region for all the points that are inside that region. Furthermore, the algorithm uses a reward signal to reinforce the association of appropriate control parameters to each region. The algorithm is able to learn consistent sequences of samples, which are likely to produce similar performance results if used to face similar situations in the future. Finally, the algorithm learns incrementally with every experience. This allows the system to use the current level of knowledge to produce the best response it can, and then use the feedback obtained from its own experience to improve its knowledge and perform better in the future. The following subsections discuss each of these points in more detail along with related work in the area, and highlight the important assumptions underlying the theory.

6.1 Learning an Adaptive Law

Parameter-adaptive reactive controllers can be more robust and achieve better performance than reactive controllers because they are able to adjust their parameters to particular operating conditions faced by the agent. However, adaptive reactive controllers require the use of an adaptive law that must be determined at design time. A learning adaptive reactive controller can learn and synthesize an adaptive law using its own experience. This type of controller inherits all the advantages of the adaptive reactive controller, but does not require a predefined adaptive law.

We formulated the problem of learning adaptive reactive controllers, and proposed the LARC algorithm which can simultaneously learn and use an adaptive law for a reactive controller. The algorithm gradually and incrementally learns an adaptive law by detecting and remembering sequences of situations and control parameters that are consistent over time and are useful according to a reward signal. The algorithm uses previous similar prototypical experiences to find the best parameterization (as far as it can determine) in any given situation. The algorithm is implemented on top of a schema-based reactive controller and validated on simulated and actual Denning MRV-III platforms.

6.2 Self-Organization

Self-organization refers to the ability of an unsupervised learning system to develop specific detectors of different signal patterns. Unsupervised learning tasks differs from supervised learning tasks in the type of information supplied to the system. In supervised learning, a "teacher" must supply the class to which each training example belongs, whereas in unsupervised learning the goal is to look for regularities in the training examples. Self-organization in learning adaptive reactive controllers is useful because it is unlikely that designers would have detailed knowledge of how to characterize the environment in terms of the agent's sensors. Thus, it is not easy to select the appropriate control parameters values the reactive controller must use when facing specific environments. A system that is able to characterize environmental situations autonomously can associate useful control parameters to environmental situations and adapt the reactive controller successfully.

The LARC algorithm is based in part on the idea of competitive learning. The main consequence of this is that the input space is quantized into regions such that the quantization error is minimized. The assumption is that finer quantization of portions of the input space used more often helps to improve the performance because the system can use different control parameter values under the environmental situations used more frequently. This means that the system self-organizes to create and adapt cases in situations where it needs them the most. An additional feature of the LARC algorithm is that the input space represents time sequences of sensorimotor inputs, allowing the system to learn regularities that encode causal relationships between sequences of inputs and outputs.

6.3 Incremental Learning

The LARC algorithm incorporates new knowledge incrementally with every experience. Thus, the system learns in a continuous or "anytime" manner (Grefenstette & Ramsey, 1992). This is useful because at every control interval, the system is able to use its current level of knowledge to adapt the control parameters to the current environmental situations. Then, during the next cycle, the system can use the results of its actions to adapt and improve its current level of knowledge. The assumption behind this idea is that the relationship of control parameters and the behavior of the system is continuous. Given this assumption, the system is able to explore and find useful control parameters by slightly modifying the control parameters suggested by the best-matching case. Such random modification will produce only a small alteration in the intended behavior of the system, but enough to perform exploration to improve performance.

6.4 Feature-Sensitive Detectors

The self-organization property of the multistrategy learning algorithm results in the creation of cases. Cases contain sequences of samples that are consistent over time. This means that each case develops into a feature-sensitive detector that correspond to a regularity in the interaction of the robot and its environment. Feature-sensitive detectors are useful because they force the system to perceive its environment in terms of these regularities, which means that the robot perceives its environment in terms of “affordances” between environmental situations and control parameters (see Gibson, 1966, for a definition of affordance and Effken & Shaw, 1992, for a discussion on the application of affordances in robotics applications). For any given situation, the best-matching case corresponds to the best strategy to follow in the adaptation sequence of control parameters under the current environmental situation. The assumption behind this method is that the interaction between the system and the environment can be characterized by a finite set of causal patterns or associations between the sensory inputs and the actions performed by the system. The method allows the system to learn only the causal patterns that the reward utility identifies as useful and to use them to modify its actions by updating its control parameters as appropriate.

6.5 Case-Based Reasoning and Learning

Case-based reasoning systems have traditionally been used to perform high-level reasoning in problem domains that can be adequately described using discrete, symbolic representations (see Kolodner, 1993). The LARC algorithm may be thought of as a method for “continuous” case-based reasoning which shares many of the fundamental assumptions of what might be called “discrete” case-based reasoning in symbolic problem domains. Learning is integrated with performance. Performance is guided by previous experience. New problems are solved by retrieving cases and adapting them. New cases are learned by evaluating proposed solutions and testing them on a real or simulated world. The basic problem-solving mechanism relies on the retrieve-adapt-apply-learn cycle common to case-based reasoning systems.

However, the requirements of continuous problem domains are significantly different in ways that do not permit ready application of traditional case-based reasoning methods. For example, a robotic navigation task requires representations of perceptual and motor control information, unlike the symbolic, high-level information used in traditional case-based reasoning programs (e.g., Hammond, 1989; Ram, 1993) and other similar approaches (e.g., Porter, Bareiss, & Holte, 1990). The input is a continuous stream of perceptual data from ultrasonic and other sensors; the data itself is analog in the sense that the value of an input parameter can vary infinitesimally (within the limits of the digitization and sampling parameters). In addition, perception and action must be tightly integrated with what corresponds to planning and execution into a continuous, real-time process. This also implies that cases must not only be retrieved and applied but also retroactively modified through experience (e.g., Ram, 1993).

6.6 Machine Learning for Robotic Control

Our research is related to and builds on prior research in many areas of machine learning and artificial intelligence, which has been discussed throughout this article. In this section, we focus on previous research on the problem of learning the parameters or structure of mobile robot controllers. Most of this research emphasizes a particular learning algorithm or a specific architecture.

Genetic algorithms have been used to determine near-optimal parameters for a reactive controller (Ram, Arkin, Boone, & Pearce, 1994). This approach consists of using genes to represent the floating-point values of the parameters of the controller. Then, controllers are evolved by evaluating their fitness under different simulated environments. This unsupervised learning method reduces greatly the task of configuring reactive controllers under specific environments, but it lacks the ability to adjust itself dynamically according to the current operating conditions during the performance of the task.

Maes and Brooks (1990) propose an algorithm to learn to activate and deactivate behaviors. The algorithm is able to learn the correlation between rewards and behavior activation through experience. Given this correlation, the algorithm is capable of selecting and activating only those behaviors that are relevant and consistent. Relevant behaviors are those that are positively correlated with positive feedback or negatively correlated with negative feedback. Reliable behaviors are those that produce consistent positive feedback when they are active. Following the terminology presented in this paper, this approach can be categorized as a limited version of a structurally-adaptive reactive controller. The controller can learn when its behaviors should become active using positive and negative feedback.

Sutton, Barto, and Williams (1991) suggest the use of reinforcement learning for direct adaptive control. Reinforcement learning methods combine methods for adjusting action-selections with methods for estimating the long-term consequences of actions. The basic idea in reinforcement learning algorithms such as Q-learning (Watkins, 1989) is to estimate a real-valued function, $Q(\cdot,\cdot)$, of states and actions, where $Q(x,a)$ is the expected discounted sum of future rewards for performing action a in state x and performing optimally thereafter. However, in most implementations of Q-learning, the function $Q(\cdot,\cdot)$ is implemented as a lookup table which requires that the states and actions must be discrete sets (e.g., when states are represented using continuous variables, they must be quantized into predefined ranges). Thus, they are not applicable to systems where states and actions are represented with real values, as is necessary to continuously adapt the parameters of a reactive controller based on sensory information (but see Lin, 1991, for an example of a Q-learning system that uses continuous state representations and discrete actions).

Ram, Arkin, Moorman, & Clark (1992) proposed the ACBARR system, a case-based system that can adapt the control parameters of a reactive controller in an on-line manner. ACBARR is able to retrieve relevant cases given the current environmental situation the robot is navigating through. The retrieved case contains hand-coded rules that are used to adapt the parameters in the current situation. According to the terminology presented earlier, ACBARR can be categorized as a (non-learning) parameter-adaptive reactive controller. However, the adaptive law used to adjust the parameters is not learned by the system. The LARC algorithm proposed here is inspired by the ACBARR algorithm but it can also learn an appropriate adaptive law to be used to improve performance.

7 Conclusions

7.1 Future Work

The methods presented in this article were evaluated using one instance of a schema-based reactive controller. Although the formulation presented in section 3 is independent of the controller architecture, experiments with different controllers will be required to validate the generality of the approach. Additionally, there are other tasks besides autonomous robotic navigation that are suitable for reactive controllers. Two examples are autonomous legged locomotion and autonomous boat navigation. In autonomous legged locomotion, the control parameters of the reactive controller must be adjusted to obtain coordination of the legs so that the robot can move safely and efficiently. Autonomous boat navigation is similar to autonomous robotic navigation but with the additional challenge of dealing with drift and superficial fluid currents. These tasks are different from autonomous robotic navigation in that the interaction between the system and the environment is governed by different physical laws. Evaluating the proposed approach under other tasks will be required to validate the theory supporting the LARC algorithm, namely, to show that the learning system can capture regularities in the interaction of the robot and its environment and can use them to improve performance.

There is still much room for improvement in adaptive reactive controllers. The methods proposed here deal only with parameter-adaptive reactive controllers. More work is required to integrate ideas from adaptive control, machine learning, and artificial intelligence to deal with structurally-adaptive reactive controllers. Such controllers will be able to learn additional architectural components and component relationships to improve system performance. In the case of schema-based reactive controllers, this would correspond to learning *assemblages* of motor schemas. An assemblage consists of a set of motor schemas that are active simultaneously and cooperate with each other to obtain a common objective. Different assemblages may be required to navigate during quite different types of environment. For example, a robot may use one assemblage while navigating inside a building and then switch to a different assemblage while navigating outside the building. In the first assemblage, only the motor schemas that help the robot navigate through a structured environment with walls, doors, and furniture will be active. In the second assemblage, only the motor schemas that help to navigate in open terrains will be active. Note that on-line local adaptation during the use of each assemblage using LARC or a similar algorithm will still be required to adapt to different local environment situations.

7.2 Summary

Learning adaptive reactive controllers can improve the performance of the navigation task in autonomous robots. We proposed a formulation of learning adaptive reactive controllers in terms of basic concepts from adaptive control theory, autonomous robotics, and machine learning. Additionally, we proposed an algorithm that is able to learn and use a mapping to tune a parameter-adaptive reactive controller in order to improve the performance of an autonomous mobile robot. The algorithm uses a multistrategy case-based reasoning and reinforcement learning approach to partition the input space into regions such that it can create a mapping between environmental situations and control parameters. The mapping is represented as a set of cases and encodes the adaptive law for the controller. The algorithm can learn and use the mapping simultaneously, using its own experience to incrementally refine the mapping so that the performance of the system can be improved in future situations. The theory supporting the algorithm consists of capturing regularities in the interaction of the system and its environment and using them during future similar situations. Regularities are casual patterns that are consistent over time; they help to improve system performance because of the predictive power obtained through this consistency. Thus, an agent that is able to perceive its surrounding environment in terms of these regularities is able to detect affordances for improving its performance. Finally, we presented results from systematic empirical evaluations of an implemented system that verify that the algorithm improves the performance of a robotic navigation system through experience.

Acknowledgements

This research was supported by the Army Research Labs under contract #DAKF11-91-D-0004-0035 and by the Georgia Institute of Technology. We thank Ron Arkin for his help with the robotics aspects of this research and Chris Atkeson for his detailed comments on an earlier draft of this paper.

References

[1] Agre, P., & Chapman, D. (1987). Pengi: An implementation of a theory of activity. In *Proceedings of the American Association of Artificial Intelligence (AAAI-87)* (vol. 1), (pp. 268–272). Los Altos, CA:Morgan Kaufmann.

[2] Albus, J.S., (1991). Outline for a theory of intelligence. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(3), 473–509.

[3] Arkin, R.C. (1989). Motor schema-based mobile robot navigation. *International Journal of Robotics Research*, 8(4), 92-112.

[4] Balch, T., Boone, G., Collings, T., Forbes, H., MacKenzie, D., & Santamar'ıa, J.C. (1995). Io, Ganymede, and Callisto - a Multiagent Robot Janitorial Team. *AI Magazine*, 16(2), 39–51.

[5] Bellman, R., & Kalaba, R. (1959). On adaptive control processes. *IRE Transactions on Automatic Control*, 4, 1–9.

[6] Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation RA-2*, 1, 14–23.

[7] Cox, M.T., & Ram, A. (1994). Failure-Driven Learning as Input Bias. In *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, (pp. 231–236). Hillsdale, NJ:LEA.

[8] Effken, J.A., & Shaw, R.E. (1992). Ecological Perspectives on the New Artificial Intelligence. *Ecological Psychology*, 4(4):247–270.

[9] Gibson, J.J. (1966). *The senses considered as perceptual systems.* Boston: Houghton Mifflin. 1966.

[10] Gray, R.M. (1984). Vector Quantization, *IEEE ASSP Magazine* (vol. 1), (pp. 4–29).

[11] Grefenstette, J.J. & Ramsey, C.L. (1992). An Approach to Anytime Learning. In *Proceedings of the Ninth International Conference on Machine Learning*, (pp. 189–195), Aberdeen, Scotland.

[12] Hammond, K.J. (1989). *Case-Based Planning: Viewing Planning as a Memory Task.* Academic Press, Boston, MA.

[13] Kaelbling, L.P. (1990). *Learning in embedded Systems*. PhD thesis, Standford University, Department of Computer Science, Standford, CA. Technical Report TR-90-04.

[14] Kohonen, T. (1990). The Self-Organizing Map. In *Proceedings of the IEEE*. 78(9), 1464–1480.

[15] Kolodner, J.L. (1993). *Case-Based Reasoning*, Morgan Kaufmann, San Mateo, CA.

[16] Langer, D., Rosenblatt, J.K., & Hebert, M. (1994). A Behavior-Based System For Off-Road Navigation. *Accepted in IEEE Transactions in Robotics and Automation*.

[17] Lin, L.J., (1991). Programming Robots using Reinforcement Learning, Planning and Teaching. In *Proceedings of the Ninth National Conference on Artificial Intelligence* (vol. 2), (pp. 781–786), Menlo Park, CA:AAAI Press.

[18] Linde, Y., Buzo, A. & Gray, R.M. (1980). An algorithm for vector quantization. *IEEE Transactions on Communication* (vol. COM-28), 84–95.

[19] Maes, P. (1990). Situated Agents Can Have Goals. *Robotics and Autonomous Systems*, 6, 49–70.

[20] Maes, P., & Brooks, R.A. (1990). Learning to Coordinate Behaviors. In *Proceedings of the Eight National Conference on Artificial Intelligence* (vol. 2), (pp. 796–802), Cambridge, MA:MIT Press.

[21] Mahadevan, S., & Connell, J. (1991). Scaling Reinforcement Learning to Robotics by Exploiting the Subsumption Architecture. In *Proceedings of the Eight International Workshop on Machine Learning*, (pp. 328–332). San Mateo, CA:Morgan Kaufmann.

[22] Narendra, K.S., & Annaswamy, A. M. (1989). *Stable Adaptive Systems.* Prentice Hall, New Jersey.

[23] Neter, J., Wasserman, W., & Kutner, M.H. (1989). *Applied Regression Models.* Prentice Hall, New Jersey.

[24] Ogata, K. (1990). *Modern Control Engineering.* Prentice Hall, New Jersey.

[25] Payton, D.W. (1990). Internalized Plans: A Representation for Action Resources. *Robotics and Autonomous Systems*, 6, 89–103.

[26] Payton, D.W. (1986). An Architecture for Reflexive Autonomous Vehicle Control. In *Proceedings of the IEEE Robotics and Automation Conference*, (pp. 1838–1845).

[27] Porter, B., Bareiss, R., & Holte, R.C. (1990). Concept Learning and Heuristic Classification in Weak-Theory Domains. In *Journal of Artificial Intelligence*, 45, 229–263.

[28] Ram, A., Arkin, R.C., Moorman, K., & Clark, R.J. (1992). *Case-Based Reactive Navigation: A Case-Based Method for On-Line Selection and Adaptation or Reactive Control Parameters in Autonomous Robotic Systems.* Technical Report GIT-CC-92/57, College of Computing, Georgia Institute of Technology, Atlanta, Georgia.

[29] Ram, A. (1993). Indexing, Elaboration, and Refinement: Incremental Learning of Explanatory Cases. *Machine Learning*, 10(3), 201–248.

[30] Ram, A., Arkin, R.C., Boone, G., & Pearce, M., Using Genetic Algorithms to Learn Reactive Control Parameters for Autonomous Robotic Navigation. *Adaptive Behavior*, 2(3), 277–305.

[31] Santamaría, J.C., & Ram, A. (1994). Systematic Evaluation of Design Decisions in CBR Systems. In it Proceedings of the AAAI Case-Based Reasoning Workshop, (pp. 23–29), Seattle, Washington.

[32] Shanmugam, K.S., (1979). *Digital and Analog Communication Systems.* New York, John Wiley and Sons.

[33] Sutton, R.S., Barto, A.G., & Williams, R.J., Reinforcement Learning in direct adaptive optimal control. In *Proceedings of the American Control Conference*, (pp. 2143–2146), Boston, MA.

[34] Thorndike, E.L. (1911). *Animal Intelligence*, Hafner, Darien, CT.

[35] Watkins, C.J.C.H. (1989). *Learning from Delayed Rewards*, Ph.D. thesis, University of Cambridge, England.