To appear in the Proceedings of the World Multiconference on Systemics, Cybernetics, and Informatics (CSI'97), Caracas, Venezuela, July 1997.

# Learning of Parameter-Adaptive Reactive Controllers for Robotic Navigation

**Juan C. Santamaría** and **Ashwin Ram**
Georgia Institute of Technology
Atlanta, GA 30332-0280
{carlos — ashwin}@cc.gatech.edu

**Abstract**

Reactive controllers are widely used in mobile robots because 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 *parameter-adaptive reactive controllers*. Parameter-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. Additionally, the paper describes a multistrategy learning algorithm that combines ideas from case-based reasoning and reinforcement learning to construct a mapping between the operating conditions of the mobile robot and the appropriate controller configuration; this mapping is in turn used to adapt the controller configuration dynamically. The algorithm is implemented and evaluated in a robotic navigation system that controls a Denning MRV-III mobile robot.

**Keywords**: robotic navigation, reactive control, reinforcement learning, adaptive control.

## 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., [1, 2, 8, 9, 12, 16]).

Reactive controllers typically rely on a combination of several *task-achieving* modules, behaviors, or schemas to perform a mobile robotic task (e.g., [8, 10, 13]). 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., [8]), weighted summation (e.g., [2]), or voting (e.g., [10]). 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 that a particular object is 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 [2]). 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., [15]), 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 robot (or plant). 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., [17]). 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., [14]). 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 [6].

During the design of adaptive controllers, the available knowledge about the dynamics of the system is used to design an *adaptive policy* that the controller can use at run-time to configure itself and improve task performance. However, such an 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 policy in advance; instead, the system would have to learn an appropriate adaptive policy through its own 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 adjust itself dynamically 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 efficient adaptive policy through its own experiences, and use it to adjust the controller dynamically. Such a controller is an improvement over an adaptive reactive controller, which can only use a predefined adaptive policy 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 in four sections. The first section formally defines the concept of adaptive reactive controllers and formulates the learning task. The second section presents an algorithm that can be used to implement a learning adaptive reactive controller for autonomous robotic navigation. The third section describes the application of the proposed algorithm in a schema-based reactive controller for a Denning MRV-III robot and presents the results of a navigation task in an indoors environment. The fourth section concludes the contents of the paper.

## PROBLEM FORMULATION

The objective of this section is to formulate a theory of 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.

### Overview

The learning task of an adaptive reactive controller consists of building a mapping function between the current operating conditions of the mobile robot and an efficient configuration of the reactive controller. We call this mapping the *adaptive policy*, which a mobile robot can use to continuously adapt the reactive controller to the particular situation at hand in order to improve performance. Our fundamental assumption is that learning the adaptive policy is simpler than learning the entire control policy (see [13]). This is usually the case because the learning task occurs at a higher level of abstraction than the operating level of the reactive controller. For example, consider the following two different configurations for a reactive controller: "hurried" and "patient". Both configurations are able to take the robot to the destination; however, the former gives less importance to avoiding obstacles and more importance to approaching the goal. The later does just the opposite. An efficient adaptive policy would map the "patient" configuration to highly cluttered environments and the "hurried" one to loosely cluttered environments. Learning this kind of mapping is easier and more efficient than learning the actual movements the robot should take at every step considering the current sensory information.

The learning capability of an adaptive reactive controller provides a significant advantage; designers do not need to specify an adaptive policy since the system can synthesize one using its own experience.

### Parameter-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., [2, 4, 8, 10, 13]).

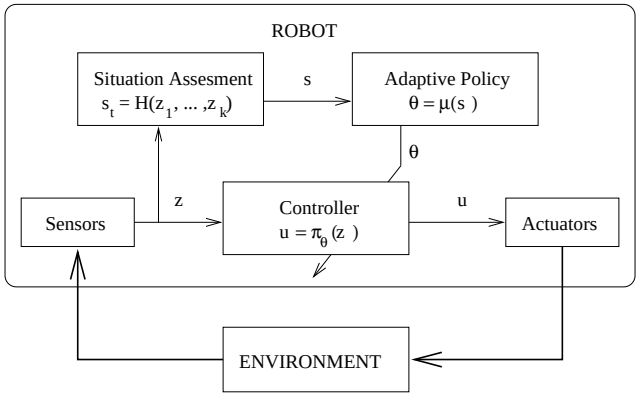


Figure 1: Adaptive Reactive Control.

Mathematically, we can represent the control 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 a parameter-adaptive reactive controller as a reactive controller that can modify its control policy through a control parameter vector $\theta = (\theta^1, \cdots, \theta^p)$. That is $u = \pi_\theta(z)$. The adaptive controller uses a *situation assessment* function $s = H(z_1, \cdots, z_k)$ to extract the current situation $s$ from the last $k$ sensations $z_1, \cdots, z_k$, and an *adaptive policy* function $\theta = \mu(s)$ to obtain the set of control parameters the reactive controller should use at any given situation. Thus, at every stage $t$, the adaptive controller uses the adaptive policy to select a new parameterization $\theta_t = \mu(s_t)$ for the control policy. Then, the controller executes the control policy for $k$ consecutive steps and uses the situation assessment function to determine the new environment situation for the next stage $s_{t+1} = H(z_1, \cdots, z_k)$. At the next stage, the entire cycle repeats when the controller uses the adaptive policy to select a new parameterization based on the new environment situation. Figure 1 shows this formulation of the adaptive reactive control problem.

The adaptive policy can be hand-designed when enough knowledge about the robot, the environment, and their interaction is known in advance [17]. However, an alternative approach is to incorporate learning algorithms into the controller and let the robot learn an appropriate adaptive policy with its own experience. In this way, the system will be capable of dealing with situations not foreseen by its designers. This is discussed next.

### Learning Task

The learning task of the autonomous navigation control problem can be stated as follows: Given a parameter-adaptive controller with a control policy $u = \pi_\theta(z)$, a situation assessment function $s = H(z_1, \cdots, z_k)$, and a desired performance reference or scalar *reward* $r = R(z_1, \cdots, z_k, u_1, \cdots, u_k)$ associated to the last $k$ sensation-action pairs of the controller, determine the control parameter vectors $\theta_t$ for all $t > 0$ such that the expected (possibly discounted) sum of rewards is maximized (Equation 2).

$$V = E \sum_{t=0}^{\infty} \gamma^t r_t \tag{2}$$

Environment situations $s$'s and control parameters $\theta$'s are usually represented using real-valued variables. Thus, the problem of learning an adaptive policy can be thought of as function synthesis. Let $\theta = \mu(s)$ denote the adaptive policy function and let $S$ denote the domain under which this function is applicable. If $s \in S$, then the expression $\theta = \mu(s)$ represents the parameter $\theta$ "recalled" in "situation" $s$ according to $\mu(\cdot)$. Thus, $\mu(\cdot)$ can be thought of as representing a mapping from $s \in S$, where $s$ represents the input description of the current situation as described by $H(\cdot)$, to appropriate controller parameters $\theta$ for this situation. Note that $s$ may include the input information $z$ from the sensors of the robot as well as internal "observations" about the state of the robot (say, the current energy level), since the appropriate parameterization $\theta$ may in general depend on both kinds of information.

The ultimate goal of the controller is to learn an adaptive policy for selecting control parameters given the environment situations that maximizes the performance measure (i.e., discounted sum of rewards) subject to the constraints of the reactive controller. Thus, the reward function is a declarative description of the desired behavior of the mobile robot. The rewards defines "what" the controller should do but it is the controller's responsibility to find out "how". This formulation is similar to the one used in reinforcement learning problems [21, 23]; however, a fundamental difference is that in reinforcement learning the controller learns the actions it should execute given any sensory information (i.e., the control policy), whereas in here the controller is given a parameterized control policy and it must learn the control parameters it should use given any environment situation as described by the situation assessment function (i.e., the adaptive policy).

The ideal mapping $\mu^*(\cdot)$ is such that when the robot is in situation $s$, then $\theta^* = \mu^*(s)$ is the optimal parameterization of the reactive controller for that situation. The exact solution of the optimal adaptive policy can be found using dynamic programming [5]; however, its computation is very expensive and requires detailed knowledge of the environment and/or the dynamics of the robot. We proposed a heuristic method inspired by the derivation of the optimal solution that is computationally efficient and does not require detailed knowledge.

## HEURISTIC APPROACH

This section presents a machine learning algorithm called LARC[1] that learns the adaptive policy for a parameter-adaptive reactive controller. The algorithm is fully implemented in SINS[2], a learning system for an adaptive schema-based reactive controller implemented in a Denning MRV-III mobile robot, and evaluated through empirical studies.

[1] Learning Adaptive Reactive Control.
[2] Self-Improving Navigational System.

1. **Initially**:
   Set the initial estimate of the value function: $w_0$
   Set the initial control parameter vector: $\theta_0$

2. **Perform**:
   At each stage $t = 1, 2, \cdots$
   (a) **Control**:
   Select best parameterization at stage $t$:
   $\hat{\theta}_t^*$, (Equation 3).
   Use the control policy: $u = \pi_{\hat{\theta}_t^*}(z)$ for $k$ steps.
   (b) **Perception**:
   Observe the resulting reward and sensation for $k$ steps:
   $r_{t+1} = R(z_1, \cdots, z_k, \pi_{\hat{\theta}_t^*}(z_1), \cdots, \pi_{\hat{\theta}_t^*}(z_k))$
   $s_{t+1} = H(z_1, \cdots, z_k)$
   (c) **Update**:
   Update the parameter-value function using TD($\lambda$):
   $w_{t+1}$, (Equation 5).

Figure 2: LARC Algorithm.

The main concept in the the derivation of the LARC algorithm is the *parameter-value function approximator*. This function maps every environment situation and control parameter combination to an estimate of the best performance measure or *long-term benefit* the controller can receive from such situation when it applies the given parameterization. Thus, the idea consists of letting the controller perform a computationally inexpensive search to select the parameterization leading to the best long-term benefit estimate according to the function approximator, and then use the outcome of each action to asymptotically improve the estimates by adjusting the function approximator. Figure 2 shows the LARC algorithm. The technical details of the LARC algorithm are discussed next.

## Parameter-Value Function Approximators

The objective of the optimal parameter-value function is to map any environment situation and control parameter combination to the best expected performance measure the controller can receive from such situation using the given parameterization. An explicit representation of the parameter-value function is important because it enables the controller to search and select the best control parameter at any given stage. However, a closed-form solution of the optimal parameter-value function is rarely available in most problems and its computation is very expensive (see [11]). A reasonable approach for explicitly representing the parameter-value function is to use a function approximator. Function approximators have been used successfully in the past in other reinforcement learning tasks (e.g.,[19, 22, 3, 18]). They use finite resources to represent the value of continuous real-valued variables. Additionally, they have weights that the controller can use to adjust the value estimates. Another advantage of function approximators is that they can generalize the performance measure associated with situations the controller actually experiences to other regions of the input space. In this way, the controller can estimate the performance measure of situations that it has never experienced before. Also, function approximators are able to represent the parameter-value function even when the variables in the environment situation and control parameters are continuous (i.e., real-valued variables).

The parameter-value function approximator, $\hat{Q}_w(s, \theta)$, is defined as

$$\begin{aligned} \hat{Q}_w(s_t, \theta_t) &\equiv \max_{\theta_{t+1}, \theta_{t+2}, \cdots} E\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \right\} \\ &= E\left\{ r_{t+1} + \gamma \hat{Q}_w\left(s_{t+1}, \theta_{t+1}^*\right) \right\} \end{aligned}$$

where $s_t$ is the environment situation, $\theta_t$ is a control parameter vector, $\theta_{t+1}^*$ is the best control parameter vector at stage $t+1$, and $w$ is a set of *weights*. This function maps every environment situation and control parameter combination to an estimate of the best performance measure the controller can receive from such situation when it applies the given parameterization. The parameter-value function approximator is important because it is used to represent and implement the adaptive policy. Specifically, the controller can find the best control parameter $\hat{\theta}_t^*$ for any given situation $s_t$ by performing the following one-stage lookahead search

$$\hat{\theta}_t^* = \arg\max_{\theta_t} E\left\{ \hat{Q}_w(s_t, \theta_t) \right\} \quad (3)$$

Equation 3 is simple: the parameter-value function measures an estimate of the best long-term benefit the controller can get for every situation-parameter pair. Thus, the best parameter vector $\hat{\theta}_t^*$ the controller can use at the current situation $s_t$ is the one that maximizes the parameter-value function approximator at the given situation. The control parameter $\hat{\theta}^*$ represent the best parameterization for situation $s_t$ given the current estimates of the parameter-value function approximator. Such estimates may be initially wrong and $\hat{\theta}^*$ may not produce good performance; however, as it will be described later, the controller will asymptotically improve the estimates of the parameter-value function approximator with the outcome of every action. In this way, the controller uses its own experience to improve the long-term benefit estimates, which in turn lead to better parameterizations of the reactive controller.

In summary, function approximators can be used to explicitly represent the optimal parameter-value function. They use finite resources to represent an estimate of the performance measure the controller can achieve from any given environment situation. The initial estimates of the function approximator may not be the optimal ones, but the controller can use the outcome of each action it executes to asymptotically improve the estimates toward their optimal values. The controller is able to accomplish this by adjusting a set of weights associated with the function approximator. Additionally, the controller can use the one-stage lookahead search to select the best control parameter it can use next. The next subsection discusses the topic of improving the estimates of the parameter-value function approximator.

### Improving Estimates

Finding the optimal parameter-value function is a computationally expensive process. A reasonable approach is use some stochastic approximation procedure (e.g., temporal difference methods [21]) to asymptotically improve the estimates towards their optimal values. The main idea consists of using the controller's experience to progressively learn the optimal parameter-value function. More specifically, the controller can incrementally learn the optimal long-term benefit values by continually exercising the current, non-optimal estimates of the parameter-value function and improving such estimates after every experience. This approach has been extensively and successfully used in reinforcement learning [22, 19, 23, 18].

At any given stage, the parameter-value function follows a recursive relation (so called the Bellman's Equation, e.g. [7])

$$\hat{Q}_w(s_t, \theta_t) = E\left\{ r_{t+1} + \gamma \hat{Q}_w\left(s_{t+1}, \theta^*_{t+1}\right)\right\} \qquad (4)$$

where $s_t$ and $\theta_t$ are the environment situation and the control parameter vectors at stage $t$. Stochastic approximation procedures exploit this recursive relation to create an update formula the controller can use to adapt the weights of the function approximator after every experience. More specifically, every time the controller executes $k$ actions and observes the next environment situation $s_{t+1}$ and reward $r_{t+1}$, it can verify whether Equation 4 holds or not by computing the error between the two consecutive predicted performance values; before executing the parameterization, $\hat{Q}_w(s_t, \hat{\theta}^*_t)$, and after executing the parameterization, $r_{t+1} + \gamma \hat{Q}_w(s_{t+1}, \hat{\theta}^*_{t+1})$. When the error is different from zero, the controller uses the update formula to adapt the weights $w$ and improve the estimate of the parameter-value function at $s_t$.

The procedure above described can be efficiently implemented using temporal difference methods [21]. Sutton defines a whole family of update formulas for temporal difference methods called TD($\lambda$), where $0 \leq \lambda \leq 1$ is a parameter used to measure the relevance of previous predictions in the current error.

A controller improves the current estimate of $\hat{Q}_{w_t}(s_t, \hat{\theta}^*_t)$ by adjusting the weights after executing every parameterization using TD($\lambda$) updates. For this purpose, every time the controller selects and executes parameterization $\hat{\theta}^*_t$ at situation $s_t$, observes the sensations $z_1, \cdots, z_k$ that result from executing that parameterization for $k$ steps, and constructs the situation $s_{t+1} = H(z_1, \cdots, z_k)$, it uses the following TD($\lambda$) formula to update the weights of the function approximator,

$$\Delta w_t = \alpha \left( r_{t+1} + \gamma \hat{Q}_{t+1} - \hat{Q}_t \right) \sum_{k=0}^{t} (\lambda\gamma)^{t-k} \nabla_{w_k} \hat{Q}_k \qquad (5)$$

where $r_{t+1} = R(z_1, \cdots, z_k, \pi_{\theta_t}(z_1), \cdots, \pi_{\theta_t}(z_k))$, $\hat{Q}_{t+1} = \hat{Q}_{w_t}(s_{t+1}, \hat{\theta}^*_{t+1})$, $\hat{Q}_t = \hat{Q}_{w_t}(s_t, \hat{\theta}^*_t)$, $\nabla_{w_k} \hat{Q}_k$ is the gradient of $\hat{Q}(\cdot)$ with respect $w$ and evaluated at situation-parameter pair $(s_k, \hat{\theta}^*_k)$, and $\alpha$ is a learning rate. The interpretation of Equation 5 is as follows: $\hat{Q}_t$ and $\hat{Q}_{t+1}$ represent the current estimates of the long-term benefit (i.e., performance measure) at states $s_t$ and $s_{t+1}$ respectively. The term $\left( r_{t+1} + \gamma \hat{Q}_{t+1} - \hat{Q}_t \right)$ represents the error incurred by $\hat{Q}_t$ in predicting the future according to $\hat{Q}_{t+1}$. In other words, the value of situation $s_t$ should be equal to the immediate reward $r_{t+1}$ plus the value of the next situation $s_{t+1}$ properly discounted. In case of error, the weights are proportionally modified in the direction of the gradient $\nabla_{w_t}$ in order to maximally reduce error. The discounted sum of the previous gradients $\nabla_{w_k}$ are also credited for the current error although their influence decays exponentially with $\lambda$.

An efficient on-line implementation of the update rule is possible using "eligibility traces" (for details, see, [20]). The idea is to maintain the value of the rightmost sum in Equation 5 in a variable or *eligibility trace*, which can be easily updated at every stage using a recursive relation. The controller performs the update every time it executes an action. Thus, the method is computationally efficient since each update is computationally inexpensive. On the other hand, the data efficiency of the method can be very low since the controller requires large amounts of data (i.e., the controller must execute many actions) to make the value function estimates converge to their optimal values.

## RESULTS

This section demonstrates the application of the algorithm proposed in the previous section for adapting the parameters of a schema-based reactive controller ([2], 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 robot. 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 [2].

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.

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 environment situation consists of the following four variables: **Obstacle-Density** provides a measure of the occupied areas around the robot that impede navigation;[3] **Absolute-Motion** measures the activity of the system; **Relative-Motion** represents the change in motion activity over an appropriate interval; and **Motion-Towards-Goal** specifies how much progress the system has actually made towards the goal. The situation assessment function computes the values of these four variables using the sonar ring and shaft encoders sensors collected over an interval of $k = 4$ steps.

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 function:

$$r = -C\,\text{virtual\_collisions} + R\left(\frac{\text{velocity}}{\text{max\_velocity}}\right)$$

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 reward every time a virtual collision[4] is detected and a positive reward every time the robot is moving; the faster it moves the stronger the reward. The constants $C$ and $R$ determine the relative importance of avoiding collisions versus maximizing velocity for a particular application. This reward function 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.

The function approximator used in the experiments is implemented using case-based methods. A description of the implementation of this type of function approximator can be found in [19]. The case-based function approximator uses cases to represent the parameter-value function. Each case represents a point in the situation-parameter space and holds its associated Q-value. In the experiment, the density threshold and the smoothing parameters were set to $\tau_d = 0.1$ and $\tau_k = 0.1$ respectively. The similarity metric was the Euclidean distance and the kernel function was the Gaussian. This produces cases with spherical receptive fields and blending of the value function using a small number of cases. The values of each case were updated using temporal difference (Equation 5) and implemented using eligibility traces following [19]. The values for the free constants were $\gamma = 0.99$, $\lambda = 0.7$, $\alpha = 0.4$, and $\epsilon = 0$. The one-stage lookahead search was performed using 6 equally spaced values for each control parameter between its minimum and maximum values.

We performed an experiment using the actual Denning MRV-III robot to verify the validity of our approach. We ran twenty learning trials and measured the accumulated time and the number of virtual collisions. A trial consisted of placing the robot at the starting point and let it run until either it reached the goal or 300 seconds had elapsed, whichever came first. The experiment was performed in an indoor room with an rectangular free space of approximately $25 \times 14$ feet and with three circular static obstacles besides the walls. Two of these obstacles were 55-gallon drums with a diameter of 2 feet and the third obstacle was another Denning robot with a diameter of $2\frac{1}{2}$ feet. Figure 3 shows an schematic layout of the robot and the laboratory room used in the experiment. Note that the boundaries of the navigable area are not flat; for example, there are desks and chairs against the walls so that the shape is irregular.

Figure 4 shows the performance of the robot plotted against the learning trials. The graphs show the time taken (top) and the number of virtual collisions (bottom) in each trial. During the first two trials the robot was not able to reach the destination point and both trials were manually terminated after five minutes. During the following trials, the robot was able to complete the task successfully and improve its performance. The robot reduced the total time taken from over 5 minutes to a final value of about 2 minutes after the first 10 trials, after which it did not improve further on this metric. Also, it reduced the number of virtual collisions from about 60 to a steady state of about 10 after the first 15 trials, which produced jerky movements along the path. The SINS system, therefore, showed a significant improvement along both metrics. Figure 5 shows the path taken by the robot in the first (1) and last (20) trials.

[3] Note that this sensory input does not provide any information about the distances or direction of the obstacles; it simply measures the density of occupied area around the robot.

[4] 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, SINS automatically diverts all its resources to focus on eliminating this imminent danger.

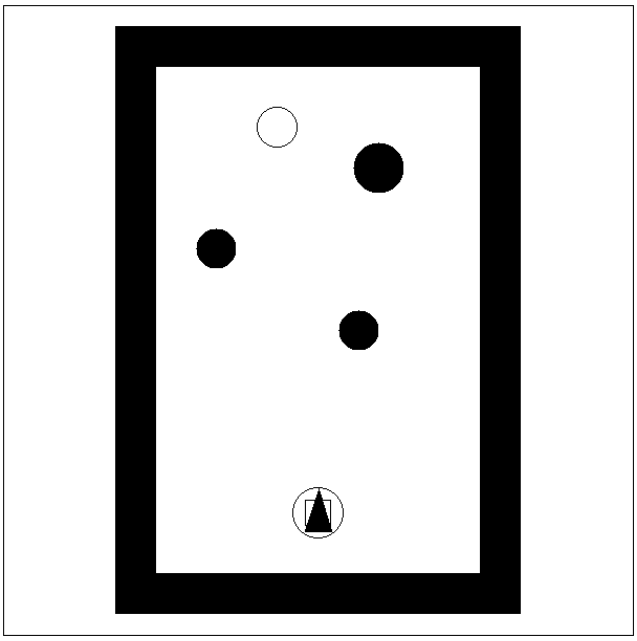

Figure 3: A schematic layout of the robot and its environment. The robot is shown at the bottom of the figure; the hollow circle near the top represents the goal; the black circles represent the static obstacles; and the black bars represent occupied areas that the robot cannot navigate through.

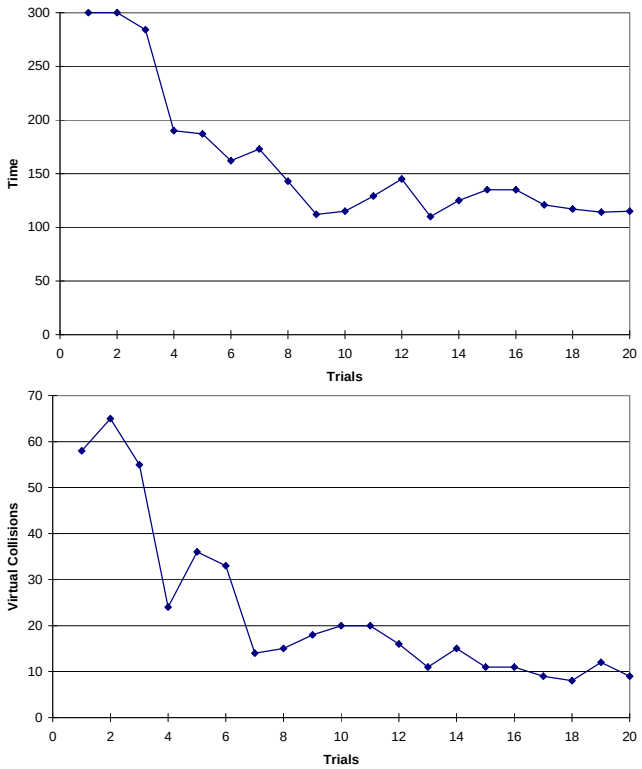


Figure 4: Performance of the real robot. Top: Total time per trial. Bottom: Number of virtual collisions per trial.

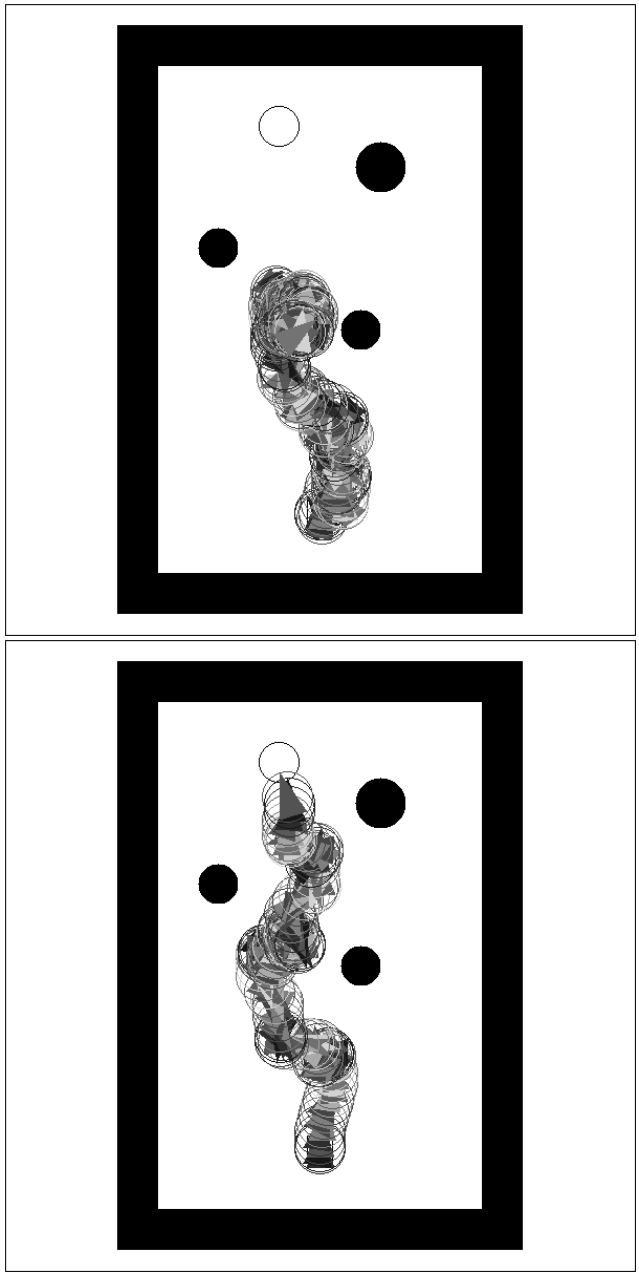

Figure 5: Actual path followed by the robot in the first and last trials: Top: trial 1. Bottom: trial 20.

## CONCLUSIONS

This paper presented a formulation of parameter-adaptive reactive controllers in terms of basic concepts from adaptive control theory, autonomous robotics, and machine learning. Additionally, it described an algorithm that is able to learn and use a mapping to adapt a reactive controller in order to improve the navigation performance of an autonomous mobile robot. The algorithm uses a multi-strategy case-based and reinforcement learning approach to progressively learn an efficient adaptive policy. The algorithm learns and uses the adaptive policy simultaneously, using its own experience to incrementally refine the policy and improve system performance. Finally, the paper presented results from a fully implemented system in a real robot.

Learning adaptive reactive controllers inherit all the advantages of traditional reactive controllers, but also are able to dynamically adjust themselves to the current operating conditions of the robot and the environment. Moreover, the are able to use its own experience to synthesize an efficient adaptive policy. This results in a significant advantage because designers do not need to specify an adaptive policy since the system can learn its own and adapt itself to the particular characteristics of the environment.

## REFERENCES


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

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

[3] C. G. Atkeson. Local trajectory optimizers. *Advances in Neural Information Processing Systems*, 6, 1993.

[4] T. Balch, G. Boone, T. Collins, H. Forbes, D. MacKenzie, and J. C. Santamaría. Io, ganymede, and callisto - a multiagent robot janitorial team. *AI Magazine*, 16(2):39–51, 1995.

[5] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[6] R. Bellman and R. Kabala. On adaptive control process. *IRE Transactions on Automatic Control*, 4:1–9, 1959.

[7] D. P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific, Belmont, MA, 1995.

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

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

[10] D. Langer, J. K. Rosenblatt, and M. Hebert. A behavior-based system for off-road navigation. *IEEE Journal on Robotics and Automation*, 10(6):776–783, 1994.

[11] R. E. Larson. A survey of dynamic programming computational procedures. *IEEE Transactions on Automatic Control*, AC-12(6):767–774, 1967.

[12] P. Maes. Situated agents can have goals. *Robotics and Autonomous Systems*, 6:49–70, 1990.

[13] S. Mahadevan and J. Connell. Scaling reinforcement learning to robotics by exploiting the subsumption architecture. In *Proceedings of the Eight International Workshop on Machine Learning*, volume 1, pages 328–332. Morgan Kaufmann, 1991.

[14] K. S. Narendra and A. M. Annaswamy. *Stable Adaptive Systems*. Prentice Hall, Englewood Cliffs, NJ, 1989.

[15] K. Ogata. *Modern Control Engineering*. Pretince Hall, Englewood Cliffs, NJ, 1990.

[16] D. W. Payton. An architecture for reflexive autonomous vehicle control. In *Proceedings of the IEEE Robotics and Automation Conference*, pages 1838–1845, 1986.

[17] A. Ram, R. C. Arkin, K. Moorman, and R. J. Clark. 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, GA, 1992.

[18] G. A. Rummery and M. Niranjan. On-line q-learning using connectionist systems. Technical Report CUED/F-INFEG/TR66, Cambridge University Department, 1994.

[19] J. C. Santamaría, R. S. Sutton, and A. Ram. Experiments with reinforcement learning in problems with continuous state and actions spaces. Technical Report UM-CS-1996-088, Department of Computer Science, University of Massachusetts, Amherst, MA, 1996.

[20] S. P. Singh and R. S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158, 1996.

[21] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[22] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems*, 8, 1996.

[23] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Univeristy of Cambridge, England, 1989.