Proceedings of the Second International Workshop on Multistrategy Learning, Harpers Ferry, WV, May 1993.

A Multistrategy Case-Based and Reinforcement Learning Approach to Self-Improving Reactive Control Systems for Autonomous Robotic Navigation

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

Abstract

This paper presents a self-improving reactive control system for autonomous robotic navigation. The navigation module uses a schema-based reactive control system to perform the navigation task. The learning module combines case-based reasoning and reinforcement learning to continuously tune the navigation system through experience. The case-based reasoning component perceives and characterizes the system's environment, retrieves an appropriate case, and uses the recommendations of the case to tune the parameters of the reactive control system. The reinforcement learning component refines the content of the cases based on the current experience. Together, the learning components perform on-line adaptation, resulting in improved performance as the reactive control system tunes itself to the environment, as well as on-line learning, resulting in an improved library of cases that capture environmental regularities necessary to perform on-line adaptation. The system is extensively evaluated through simulation studies using several performance metrics and system configurations.

Keywords: Robot navigation, reactive control, case-based reasoning, reinforcement learning, adaptive control.

1 Introduction

Autonomous robotic navigation is defined as the task of finding a path along which a robot can move safely from a source point to a destination point in an obstacle-ridden terrain, and executing the actions to carry out the movement in a real or simulated world. Several methods have been proposed for this task, ranging from high-level planning methods to reactive control methods.

High-level planning methods use extensive world knowledge and inferences about the environment they interact with (Fikes, Hart & Nilsson, 1972; Sacerdoti, 1975). Knowledge about available actions and their consequences is used to formulate a detailed plan before the actions are actually executed in the world. Such systems can successfully perform the path-finding required by the navigation task, but only if an accurate and complete representation of the world is available to the system. Considerable high-level knowledge is also needed to learn from planning experiences (e.g., Hammond, 1989a; Minton, 1988; Mostow & Bhatnagar, 1987; Segre, 1988). Such a representation is usually not available in real-world environments, which are complex and dynamic in nature. To build the necessary representations, a fast and accurate perception process is required to reliably map sensory inputs to high-level representations of the world. A second problem with high-level planning is the large amount of processing time required, resulting in significant slowdown and the inability to respond immediately to unexpected situations.

Situated or reactive control methods have been proposed as an alternative to high-level planning methods (Arkin, 1989; Brooks, 1986; Kaelbling, 1986; Payton, 1986). In these methods, no planning is performed; instead, a simple sensory representation of the environment is used to select the next action that should be performed. Actions are represented as simple behaviors, which can be selected and executed rapidly, often in real-time. These methods can cope with unknown and dynamic environmental configurations, but only those that lie within the scope of predetermined behaviors. Furthermore, such methods cannot modify or improve their behaviors through experience, since they do not have any predictive capability that could account for future consequences of their actions, nor a higher-level formalism in which to represent and reason about the knowledge necessary for such analysis.

We propose a self-improving navigation system that uses reactive control for fast performance, augmented with multistrategy learning methods that allow the system to adapt to novel environments and to learn from its experiences. The system autonomously and progressively constructs representational structures that aid the navigation task by supplying the predictive capability that standard reactive systems lack. The representations are constructed using a hybrid case-based and reinforcement learning method without extensive high-level reasoning. The system is very robust and can perform successfully in (and learn from) novel environments, yet it compares favorably with traditional reactive methods in terms of speed and performance. A further advantage of the method is that the system designers do not need to foresee and represent all the possibilities that might occur since the system develops its own “understanding” of the world and its actions. Through experience, the system is able to adapt to, and perform well in, a wide range of environments without any user intervention or supervisory input. This is a primary characteristic that autonomous agents must have to interact with real-world environments.

This paper is organized as follows. Section 2 presents a technical description of the system, including the schema-based reactive control component, the case-based and reinforcement learning methods, and the system-environment model representations, and places it in the context of related work in the area. Section 3 presents several experiments that evaluate the system. The results shown provide empirical validation of our approach. Section 4 concludes with a discussion of the lessons learned from this research and suggests directions for future research.

2 Technical Details

2.1 System Description

The Self-Improving Navigation System (SINS) consists of a navigation module, which uses schema-based reactive control methods, and an on-line adaptation and learning module, which uses case-based reasoning and reinforcement learning methods. The navigation module 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. The adaptation sub-module performs on-line adaptation of the reactive control parameters to get the best performance from the navigation module. The adaptation is based on recommendations from cases that capture and model the interaction of the system with its environment. With such a model, SINS is able to predict future consequences of its actions and act accordingly. The learning sub-module monitors the progress of the system and incrementally modifies the case representations through experience. Figure 1 shows the SINS functional architecture.

The main objective of the learning module is to construct a model of the continuous sensorimotor interaction of the system with its environment, that is, a mapping from sensory inputs to appropriate behavioral (schema) parameters. This model allows the adaptation module to control the behavior of the navigation module by selecting and adapting schema parameters in different environments. To learn a mapping in this context is to discover environment configurations that are relevant to the navigation task and corresponding schema parameters that improve the navigational performance of the system. The learning method is unsupervised and, unlike traditional reinforcement learning methods, does not rely on an external reward function (cf. Watkins, 1989; Whitehead & Ballard, 1990). Instead, the system’s “reward” depends on the similarity of the observed mapping in the current environment to the mapping represented in the model. This causes the system to converge towards those mappings that are consistent over

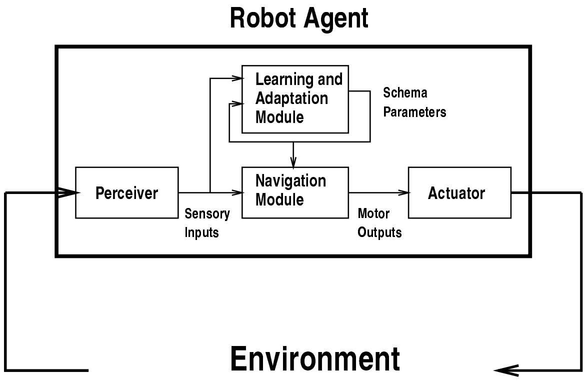


Figure 1: System architecture

a set of experiences.

The representations used by SINS to model its interaction with the environment are initially under-constrained and generic; they contain very little useful information for the navigation task. As the system interacts with the environment, the learning module gradually modifies the content of the representations until they become useful and provide reliable information for adapting the navigation system to the particular environment at hand.

The learning and navigation modules function in an integrated manner. The learning module is always trying to find a better model of the interaction of the system with its environment so that it can tune the navigation module to perform its function better. The navigation module provides feedback to the learning module so it can build a better model of this interaction. The behavior of the system is then the result of an equilibrium point established by the learning module which is trying to refine the model and the environment which is complex and dynamic in nature. This equilibrium may shift and need to be re-established if the environment changes drastically; however, the model is generic enough at any point to be able to deal with a very wide range of environments.

We now present the reactive module, the representations used by the system, and the methods used by the learning module in more detail.

2.2 The Schema-Based Reactive Control Module

The reactive control module is based on the AuRA architecture (Arkin, 1989), and consists of a set of motor schemas that represent the individual motor behaviors available to the system. Each schema reacts to sensory information from the environment, and produces a velocity vector representing the direction and speed at which the robot is to move given current environmental conditions. The velocity vectors produced by all the schemas are then combined to produce a potential field 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. This allows the system to interact successfully in different environmental configurations requiring different navigational “strategies” (Clark, Arkin, & Ram, 1992).

A detailed description of schema-based reactive control methods can be found in Arkin (1989). In this research, we used 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 to wander in a random direction. 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.

Different combinations of schema parameters produce different behaviors to be exhibited by the system (see figure 2). Traditionally, parameters are fixed and determined ahead of time by the system designer. However, on-line selection and modification of the appropriate parameters based on the current environment can en-

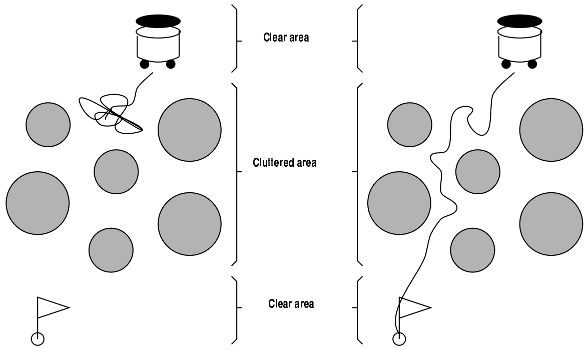


Figure 2: Typical navigational behaviors of different tunings of the reactive control module. The figure on the left shows the non-learning system with high obstacle avoidance and low goal attraction. On the right, the learning system has lowered obstacle avoidance and increased goal attraction, allowing it to "squeeze" through the obstacles and then take a relatively direct path to the goal.

hance navigational performance (Clark, Arkin, & Ram, 1992; Moorman & Ram, 1992). SINS adopts this approach by allowing schema parameters to be modified dynamically. However, in their systems, the cases are supplied by the designer using hand-coded coded cases. Our system, in contrast, can learn and modify its own cases through experience. The representation of our cases is also considerably different and is designed to support reinforcement learning.

2.3 The System-Environment Model Representation

The navigation module in SINS can be adapted to exhibit many different behaviors. SINS improves its performance by learning how and when to tune the navigation module. In this way, the system can use the appropriate behavior in each environmental configuration encountered. The learning module, therefore, must learn about and discriminate between different environments, and associate with each the appropriate adaptations to be performed on the motor schemas. This requires a representational scheme to model, not just the environment, but the interaction between the system and the environment. However, to ensure that the system does not get bogged down in extensive high-level reasoning, the knowledge represented in the model must be based on perceptual and motor information easily available at the reactive level.

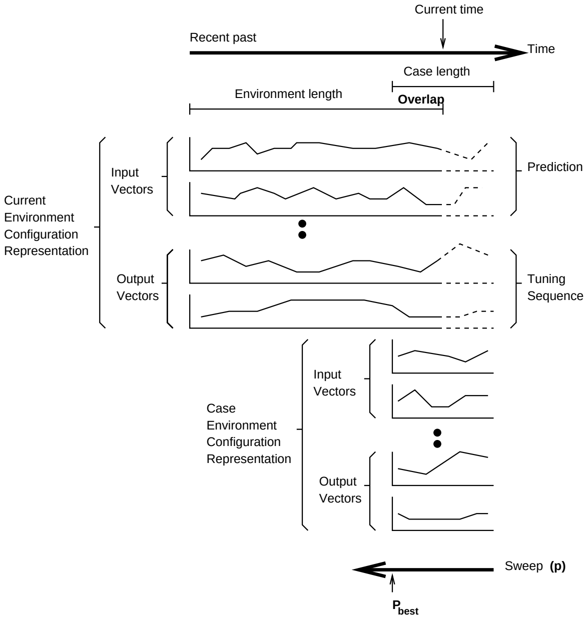


Figure 3: Sample representations showing the time history of analog values representing perceived inputs and schema parameters. Each graph in the case (below) is matched against the corresponding graph in the current environment (above) to determine the best match, after which the remaining part of the case is used to guide navigation (shown as dashed lines).

SINS uses a model consisting of associations between the sensory inputs and schema parameters values. Each set of associations is represented as a case. Sensory inputs provides information about the configuration of the environment, and schema parameter information specifies how to adapt the navigation module in the environments to which the case is applicable. Each type of information is represented as a vector of analog values. Each analog value corresponds to a quantitative variable (a sensory input or a schema parameter) at a specific time. A vector represents the trend or recent history of a variable. A case models an association between sensory inputs and schema parameters by grouping their respective vectors together. Figure 3 show an example of this representation.

This representation has three essential properties. First, the representation is capable of capturing a wide range of possible associations between of sensory inputs and schema parameters. Second, it permits continuous progressive refinement of the associations. Finally, the representation captures trends or patterns of input and output values over time. This allows the system to detect patterns over larger time windows rather than having to make a decision based only on instantaneous values of perceptual inputs.

In this research, we used four input vectors to characterize the environment and discriminate among different environment configurations: **Obstacle-Density** provides a measure of the occupied areas that impede navigation; **Absolute-Motion** measures the activity of the system; **Relative-Motion** represents the change in motion activity; and **Motion-Towards-Goal** specifies how much progress the system has actually made towards the goal. These input vectors are constantly updated with the information received from the sensors.

We also used four output vectors to 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 case that best matches the current environment. The new values remain constant until the next setting period.

The choice of input and output vectors was based on the complexity of their calculation and their relevance to the navigation task. The input vectors were chosen to represent environment configurations in a generic manner but taking into account the processing required to produce those vectors (e.g., obstacle density is more generic than obstacle position, and can be obtained easily from the robot's ultrasonic sensors). The output vectors were chosen to represent directly the actions that the learning module uses to tune the navigation module, that is, the schema parameter values themselves.

2.4 The On-Line Adaptation And Learning Module

This module creates, maintains and applies the case representations used for on-line adaptation of the reactive module. The objective of the learning method is to detect and discriminate among different environment configurations, and to identify the appropriate schema parameter values to be used by the navigation module, in a dynamic and an on-line manner. This means that, as the system is navigating, the learning module is perceiving the environment, detecting an environment configuration, and modifying the schema parameters of the navigation module accordingly, while simultaneously updating its own cases to reflect the observed results of the system's actions in various situations.

The method 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 (e.g., see Kolodner, 1988; Hammond, 1989b), and from reinforcement learning, which deals with the issue of updating the content of system's knowledge based on feedback from the environment (e.g., see Sutton, 1992). However, in traditional case-based planning systems (e.g., Hammond, 1989a) learning and adaptation requires a detailed model of the domain. This is exactly what reactive planning systems are trying to avoid. Earlier attempts to combine reactive control with classical planning systems (e.g., Chien, Gervasio, & DeJong, 1991) or explanation-based learning systems (e.g., Mitchell, 1990) also relied on deep reasoning and were typically too slow for the fast, reflexive behavior required in reactive control systems. Unlike these approaches, our method does not fall back on slow non-reactive techniques for improving reactive control.

To effectively improve the performance of the navigation task, the learning module must find a consistent mapping from environment configurations to control parameters. The learning module captures this mapping in the learned cases, each case representing a portion of the mapping localized in a specific environment configuration. The set of cases represents the system's model of its interactions with the environment, which is adapted through experience using the case-based and reinforcement learning methods. The case-based method selects the case best suited for a particular environment configuration. The reinforcement learning method updates the content of a case to reflect the current experience, such that those aspects of the mapping that are consistent over time tend to be reinforced. Since the navigation module implicitly provides the bias to move to the goal while avoiding obstacles, mappings that are consistently observed are those that tend to produce this behavior. As the system gains experience, therefore, it improves its own performance at the navigation task.

Each case represents an observed regularity between a particular environmental configuration and the effects of different actions, and prescribes the values of the schema parameters that are most appropriate (as far as the system knows based on its previous experience) for that environment. The learning module performs the following tasks in a cyclic manner: (1) **perceive** and represent the current environment; (2) **retrieve** a case whose input vector represents an environment most similar to the current environment; (3) **adapt** the schema parameter values in use by the reactive control module by installing the values recommended by the output vectors of the case; and (4) **learn** new associations and/or adapt existing associations represented in the case to reflect any new information gained through the use of the case in the new situation to enhance the reliability of their predictions.

A detailed description of each step would require more space than is available in this paper; however, a short description of the method follows. The **perceive** step builds a set of four input vectors $\mathbf{E}_{\text{input}_j}$, one for each sensory input j described earlier, which are matched against the corresponding input vectors $\mathbf{C}_{\text{input}_j}$ of the cases in the system's memory in the **retrieve** step. The case similarity metric SM is based on the mean squared difference between each of the vector values $C^k_{\text{input}_j}(i)$ of the kth case $\mathbf{C}^k$ over a trending window l_C, and the vector values $E_{\text{input}_j}(i)$ of the environment $\mathbf{E}$ over a trending window of a given length l_E:

$$SM(\mathbf{E}, \mathbf{C}^k, p) = \\ \sum_{j=1}^{4} w_j \sum_{i=0}^{\min(l_E - p, l_C)} \frac{(E_{\text{input}_j}(i+p) - C_{\text{input}_j}(i))^2}{(\min(l_E - p, l_C) - p)^2}$$

The match window p_{best} is calculated using a reverse sweep over the time axis p similar to a convolution process to find the relative position (represented by $\min(l_E - p, l_C)$) that matches best. The best matching case $\mathbf{C}^{k_{\text{best}}}$, satisfying the equation:

$$\{k_{\text{best}}, p_{\text{best}} | \min(SM(\mathbf{E}, \mathbf{C}^k, p)), \forall k, 0 \leq p \leq l_C\}$$

is handed to the **adapt** step, which selects the schema parameter values $\mathbf{C}^k_{\text{output}_j}$ from the output vectors of the case and modifies the values currently in use using a reinforcement formula which uses the case similarity metric as a scalar reward. Thus the actual adaptations performed depend on the goodness of match between the case and the environment, and are given by:

$$\mathbf{C}^{k_{\text{best}}}_{\text{output}_j} \min(l_E - p_{\text{best}}, l_C) \times \\ |1 - RSM| \text{random}(0, \max_k \mathbf{C}^{k_{\text{best}}}_{\text{output}_j})$$

where RSM is the relative similarly metric discussed below. The random factor allows the system to "explore" the search space locally in order to discover regularities, since the system does not start with prior knowledge that can be used to guide this search.

Finally, the **learn** step uses statistical information about prior applications of the case to determine whether information from the current application of the case should be used to modify this case, or whether a new case should be created. The vectors encoded in the cases are adapted using a reinforcement formula in which a *relative similarity measure* is used as a scalar reward or reinforcement signal. The relative similarity measure RSM, given by $(SM - SM_{\text{best}})/(\overline{SM} - SM_{\text{best}})$ quantifies how similar the current environment configuration is to the environment configuration encoded by the case relative to how similar the environment has been in previous utilizations of the case. Intuitively, if case matches the current situation better than previous situations it was used in, it is likely that the situation involves the very regularities that the case is beginning to capture; thus, it is worthwhile modifying the case in the direction of the current situation. Alternatively, if the match is not quite as good, the case should not be modified because that will take it away from the regularity it was converging towards. Finally, if the current situation is a very bad fit to the case, it makes more sense to create a new case to represent what is probably a new class of situations.

Thus, if the RSM is below a certain threshold (0.1 in this paper), the input and output case vectors are updated using a gradient descent formula based on the similarity measure:

$$C_j^{k_{\text{best}}}(i) =$$

$$\alpha \min(l_E - p, l_C)(E_j(i+p) - C_j^{k_{\text{best}}}(i)),$$
$$0 \le i \le l_C$$

where the constant α determines the learning rate (0.5 in this paper). In the **adapt** and **learn** steps, the overlap factor $\min(l_E - p_{\text{best},l_C})$ is used to attenuate the modification of early values within the case which contribute more to the selection of the current case.

Since the reinforcement formula is based on a relative similarity measure, the overall effect of the learning process is to cause the cases to converge on stable associations between environment configurations and schema parameters. Stable associations represent regularities in the world that have been identified by the system through its experience, and provide the predictive power necessary to navigate in future situations. 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 these causal patterns and to use them to modify its actions by updating its schema parameters as appropriate.

Genetic algorithms may also be used to modify schema parameters in a given environment (Pearce, Arkin, & Ram, 1992). However, while this approach is useful in the initial design of the navigation system, it cannot change schema parameters during navigation when the system faces environments that are significantly different from the environments used in the training phase of the genetic algorithm. Another approach to self-organizing adaptive control is that of Verschure, Kröse, & Pfeifer (1992), in which a neural network is used to learn how to associate conditional stimulus to unconditional responses. Although their system and ours are both self-improving navigation systems, there is a fundamental difference on how the performance of the navigation task is improved. Their system improves its navigation performance by learning how to incorporate new input data (i.e., conditional stimulus) into an already working navigation system, while SINS improves its navigation performance by learning how to adapt the system itself (i.e., the navigation module). Our system does not rely on new sensory input, but on patterns or regularities detected in perceived environment. Our learning methods are also similar to Sutton (1990), whose system uses a trial-and-error reinforcement learning strategy to develop a world model and to plan optimal routes using the evolving world model. Unlike this system, however, SINS does not need to be trained on the same world many times, nor are the results of its learning specific to a particular world, initial location, or destination location.

3 Evaluation

The methods presented above have been evaluated using extensive simulations across a variety of different types of environment, performance criteria, and system configurations. The objective of these experiments is to measure qualitatively and quantitatively improvement of the navigation performance of SINS (the “adaptive system”), and to compare this performance against a non-learning schema-based reactive system (the “static system”) and a system that changes the schema parameter values randomly after every control interval (the “random system”). Rather than simply measure the improvement in performance in SINS by some given metric such as “speedup”, we were interested in systematically evaluating the effects of various design decisions on the performance of the system across a variety of metrics in different types of environments. To achieve this, we designed several experiments, which can be grouped into four sets as discussed below.

3.1 Experiment Design

The systems were tested on randomly generated environments consisting of rectangular bounded worlds. Each environment contains circular obstacles, a start location, and a destination location, as shown in figure 2. Figure 4 shows an actual run of the static and adaptive systems on one of the randomly generated worlds. The location, number and radius of the obstacles were randomly determined to create environments of varying amounts of **clutter**, defined as the ratio of free space to occupied space. We tested the effect of three different parameters in the SINS system: **max-cases**, the maximum number of cases that SINS is allowed to create; **case-length**, l_C, representing the time window of a case; and **control-interval**, which determines how often the schema parameters in the reactive control module are adapted.

We used six estimators to evaluate the navigation performance of the systems. These metrics were computed using a cumulative average over the test worlds to factor out the intrinsic differences in difficulty of different worlds. *Average number of worlds solved* indicates in how many of the worlds posed the system actually found a path to the goal location. The optimum value is 100% since this would indicate that every world presented was successfully solved. *Average steps* indicates the average of number of steps that the robot takes to terminate each world; smaller values indicate better performance. *Average distance* indicates the total distance traveled per world on average; again, smaller values indicate better performance. *Average* $\frac{\textit{actual}}{\textit{optimal}}$ *distance per world* indicates the ratio of the total distance traveled and the Euclidean distance between the start and end points, averaged over the solved worlds. The optimal value is 1, but this is only possible in a world without obstacles. *Average virtual collisions per world* indicates the total number of times the robot came within a predefined distance of an obstacle. Finally, *average time per world* indicates the total time the system takes to execute a world on average.

The data for the estimators was obtained after the systems terminated each world. 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). The execution is terminated when the navigation system reaches its destination or when the number of steps reaches an upper limit (3000 in the current evaluation). The latter condition guarantees termination since some worlds are unsolvable by one or both systems.

In this paper, we discuss the results from the following sets of experiments:

Experiment set 1: Effect of the multistrategy learning method. We first evaluated the effect of our multistrategy case-based and reinforcement learning method by comparing the performance of the SINS system against the static and random systems. SINS was allowed to learn up to 10 cases (**max-cases** $=$ 10), each of **case-length** $=$ 4. Adaptation occurred every **control-interval** $=$ 4 steps. Figure 5 shows the results obtained for each estimator over the 200 worlds. Each graph compares the performance on one estimator of each of the three systems, static, random and adaptive, discussed above.

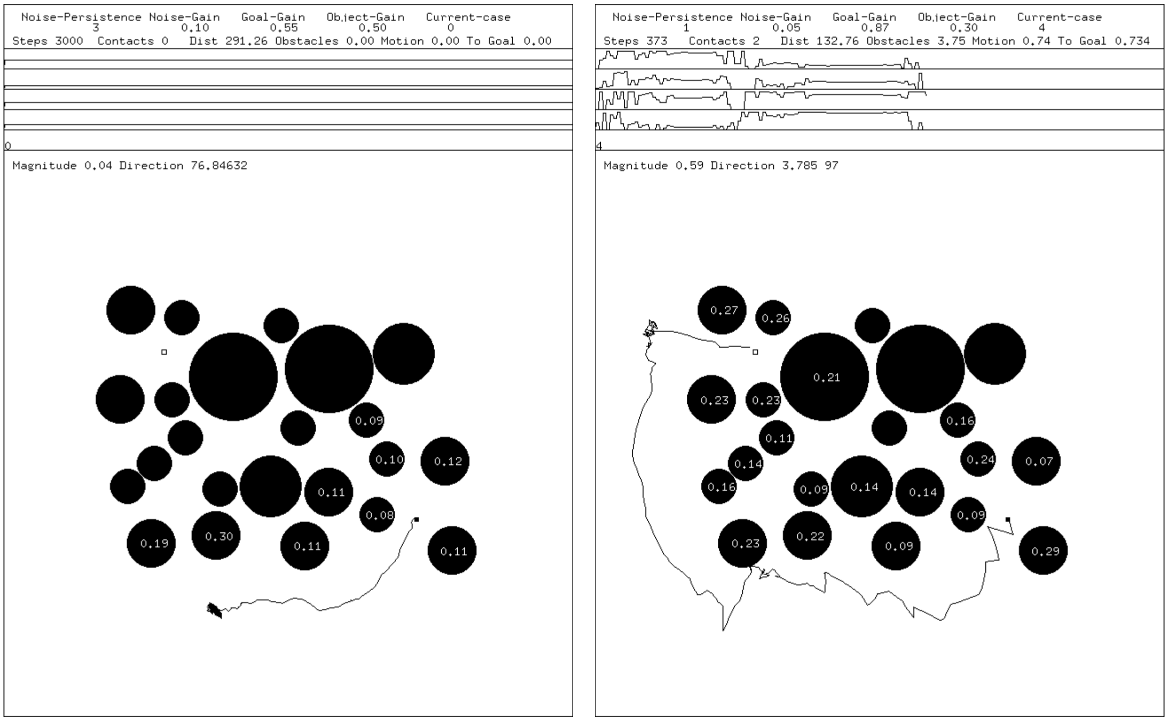
Experiment set 2: Effect of case parameters. This set of experiments evaluated the effect of two parameters of the case-based reasoning component of the multistrategy learning system, that is, **max-cases** and **case-length**. **control-interval** was held constant at 4, while **max-cases** was set to 10, 20, 40 and 80, and **case-length** was set to 4, 6, 10 and 20. All these configurations of SINS, and the static and random systems, were evaluated using all six estimators on 200 randomly generated worlds of 25% and 50% clutter. The results are shown in figures 6 and 7.

Experiment set 3: Effect of control interval. This set of experiments evaluated the effect of the **control-interval** parameter, which determines how often the adaptation and learning module modifies the schema parameters of the reactive control module. **max-cases** and **case-length** were held constant at 10 and 4, respectively, while **control-interval** was set to 4, 8, 12 and 16. All systems were evaluated using all six estimators on 200 randomly generated worlds of 50% clutter. The results are shown in figure 8.

Experiment set 4: Effect of environmental change. This set of experiments was designed to evaluate the effect of changing environmental characteristics, and to evaluate the ability of the systems to adapt to new environments and learn new regularities. With **max-cases** set to 10, 20, 40 and 80, **case-length** set to 4, 6 and 10, and **control-interval** set to 4, we presented the systems with 200 randomly generated worlds of 25% clutter followed by 200 randomly generated worlds of 50% clutter. The results are shown in figure 9.

3.2 Discussion of Experimental Results

The results in figures 5 through 9 show that SINS does indeed perform significantly better than its non-learning counterpart. To obtain a more detailed insight into the nature of the improvement, let us discuss the experimental results in more detail.



```
Noise-Persistence Noise-Gain   Goal-Gain   Object-Gain   Current-case
          3            0.10         0.55        0.50          0
Steps 3000  Contacts 0   Dist 291.26 Obstacles 0.00 Motion 0.00 To Goal 0.00
0
Magnitude 0.04 Direction 76.84632
```

```
Noise-Persistence Noise-Gain   Goal-Gain   Object-Gain   Current-case
          1            0.05         0.87        0.30          4
Steps 373   Contacts 2   Dist 132.76 Obstacles 3.75 Motion 0.74 To Goal 0.734
4
Magnitude 0.59 Direction 3.785 97
```

Figure 4: Sample runs of the static and adaptive systems on a randomly generated world. The system starts at the filled box (towards the lower right side of the world) and tries to navigate to the unfilled box. The figure on the left shows the static system. On the right, the adaptive system has learned to "balloon" around the obstacles, temporarily moving away from the goal, and then to "squeeze" through the obstacles (towards the end of the path) and shoot towards the goal. The graphs at the top of the figures plot the values of the schema parameters over the duration of the run.

Experiment set 1: Effect of the multistrategy learning method. Figure 5 shows the results obtained for each estimator over the 200 worlds. As shown in the graphs, SINS performed better than the other systems with respect to five out of the six estimators. Figure 10 shows the final improvement in the system after all the worlds. SINS successfully navigates 93% of the worlds, a 541% improvement over the non-learning system, with 22% fewer virtual collisions. Although the non-learning system was 39% faster, the paths it found required over 4 times as many steps. On average, SINS' solution paths were 25% shorter and required 76% fewer steps, an impressive improvement over a reactive control method which is already good at navigation.

The average time per world was the only estimator in which the self-improving system performed worse. The reason for this behavior is that the case retrieval process is very time consuming. However, since in the physical world the time required for physical execution of a motor action outweighs the time required to select the action, the time estimator is less critical than the distance, steps, and solved worlds estimators. Furthermore, as discussed below, better case organization methods should reduce the time overhead significantly.

The experiments also demonstrate an somewhat unexpected result: the number of worlds solved by the navigation system is increased by changing the values of the schema parameters even in a random fashion, although the random changes lead to greater distances travelled. This may be due to the fact that random changes can get the system out of "local minima" situations in which the current settings of its parameters are inadequate. However, consistent changes (i.e., those that follow the "regularities" captured by our method) lead to better performance than random changes alone.

Experiment set 2: Effect of case parameters. All configurations of the SINS system navigated successfully in a larger percentage of the test worlds than the static system. Regardless of the

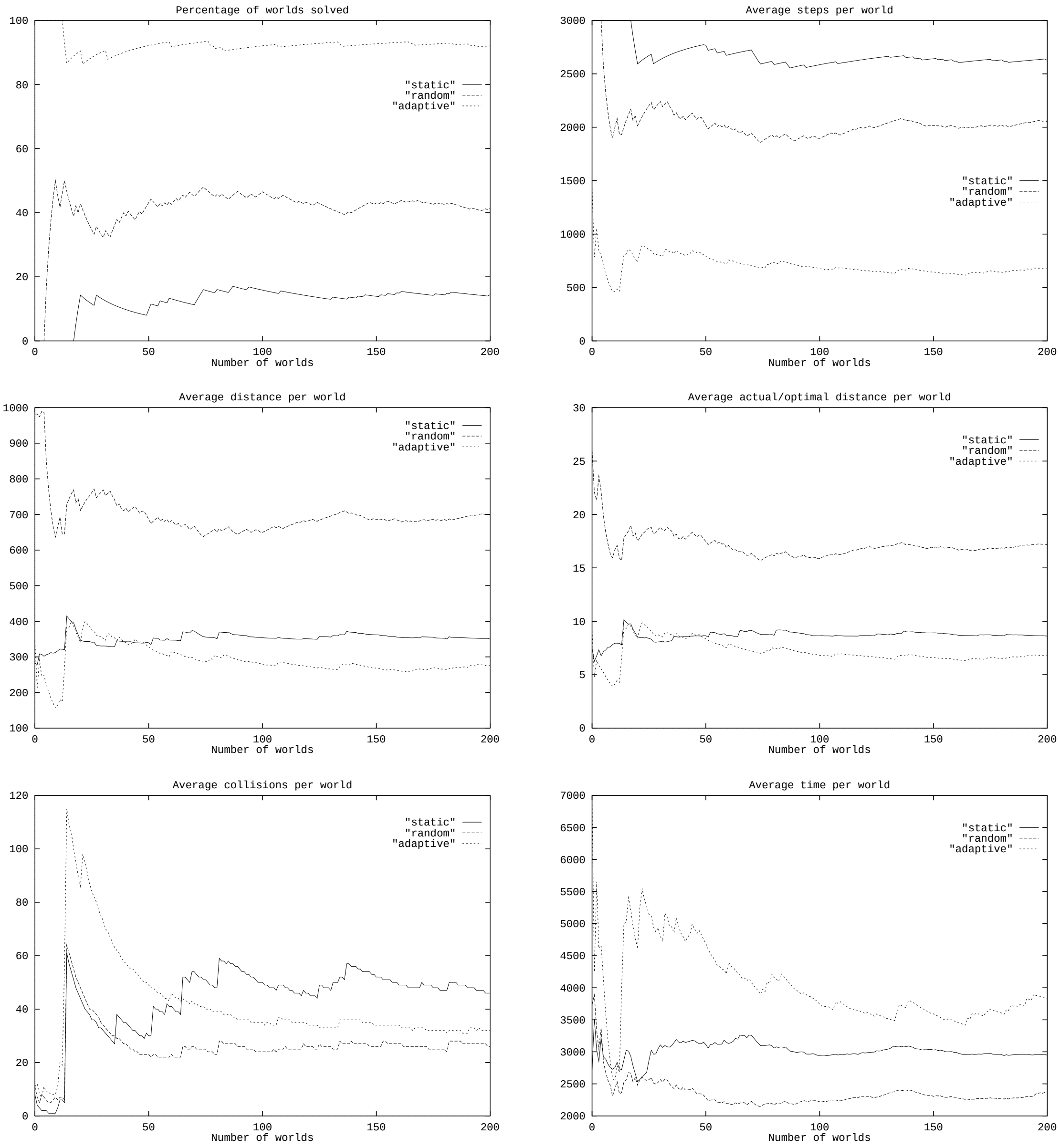
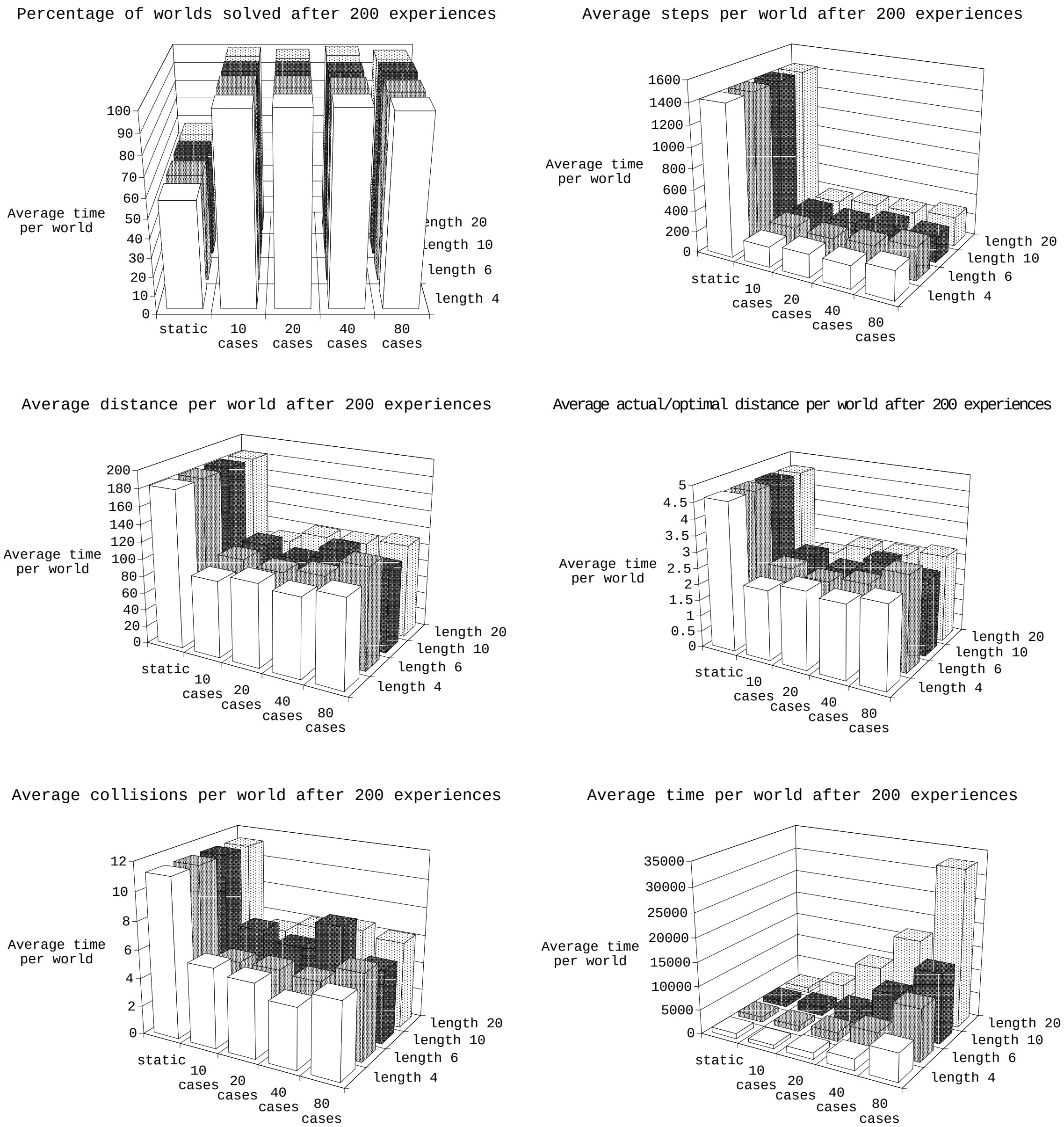


Figure 5: Cumulative performance results.



Percentage of worlds solved after 200 experiences

Average steps per world after 200 experiences

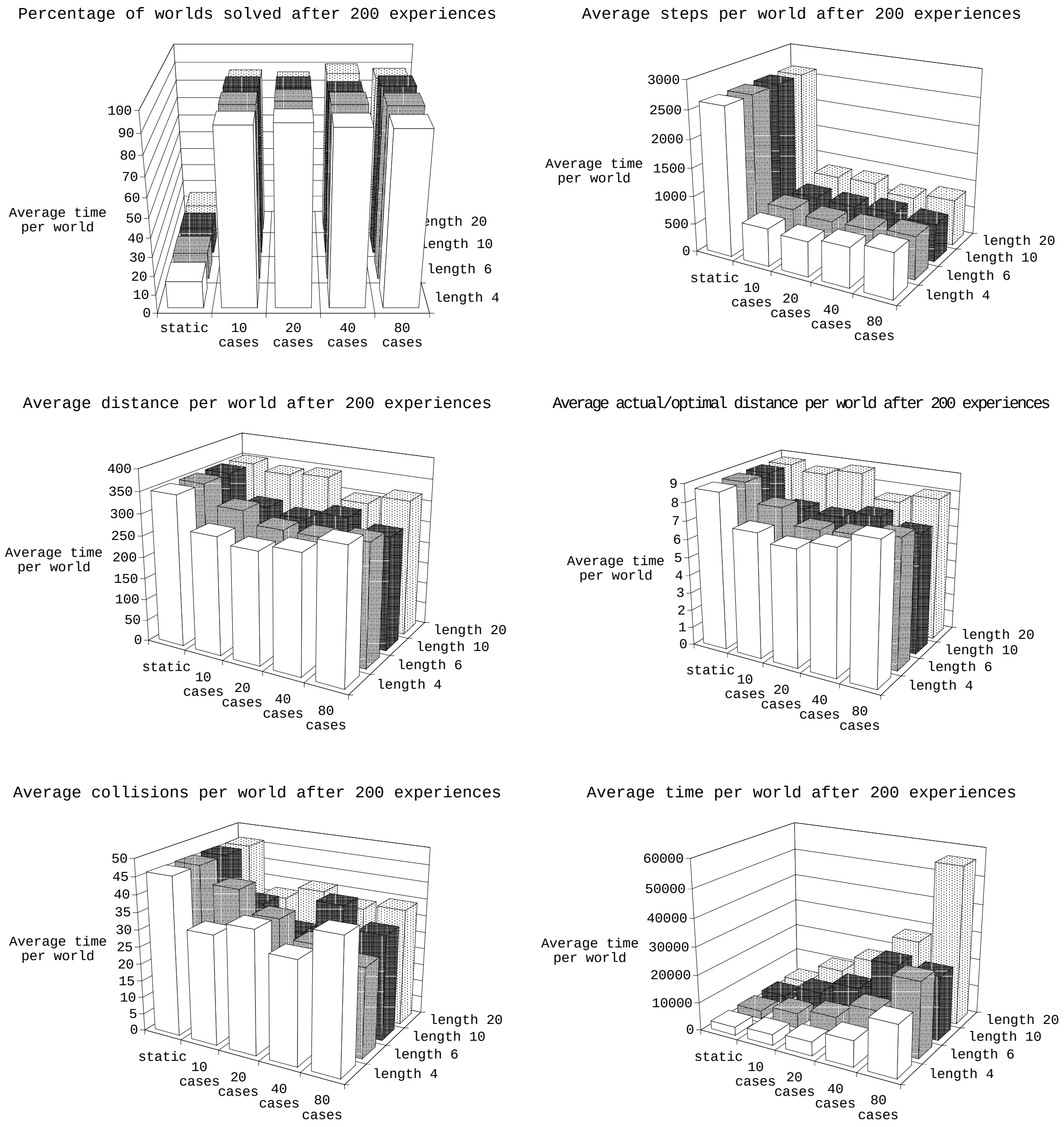
Average distance per world after 200 experiences

Average actual/optimal distance per world after 200 experiences

Average collisions per world after 200 experiences

Average time per world after 200 experiences

Figure 6: Effect of **max-cases** and **case-length** on 25% cluttered worlds.



Percentage of worlds solved after 200 experiences

Average steps per world after 200 experiences

Average distance per world after 200 experiences

Average actual/optimal distance per world after 200 experiences

Average collisions per world after 200 experiences

Average time per world after 200 experiences

Figure 7: Effect of **max-cases** and **case-length** on 50% cluttered worlds.

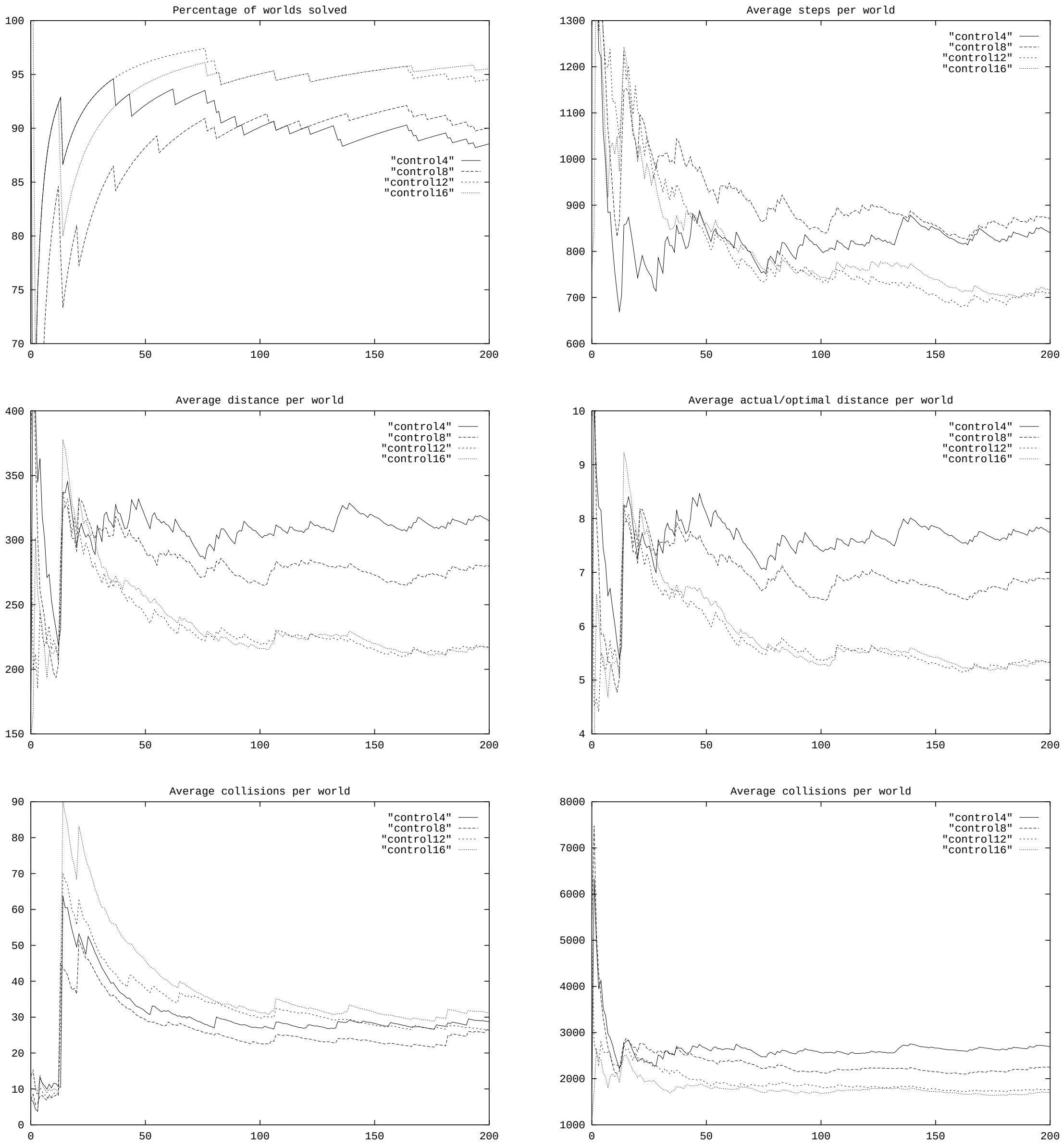


Figure 8: Effect of **control-interval**.

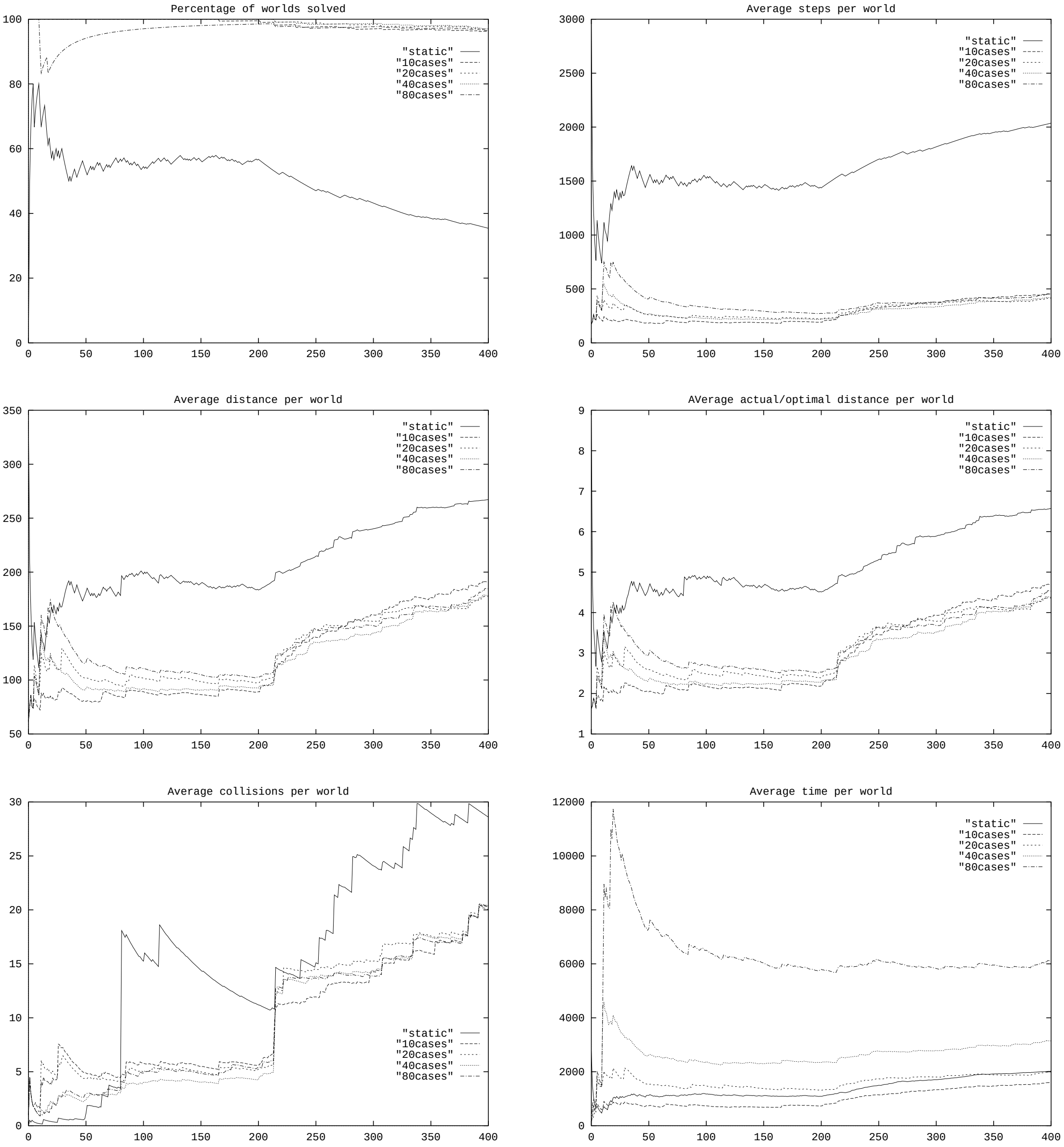


Figure 9: Effect of a sudden change in environment (after the 200th world).

	static	*random*	*adaptive*
Percentage of worlds solved	14.5%	41.5%	93%
Average steps per world	2624.6	2046.8	618.4
Average distance per world	350.5	696.5	261.2
Average $\frac{\text{actual}}{\text{optimal}}$ distance	8.6	17.1	6.4
Average virtual collisions	46.1	26.4	35.7
Average time per world, ms	2947.8	2352.5	4878.3

Figure 10: Final performance results.

max-cases and **case-length** parameters, SINS could solve most of the 25% cluttered worlds (as compared with 55% in the static system) and about 90% of the 50% cluttered worlds (as compared with 15% in the static system). Although it could be argued that an alternative set of schema parameters might lead to better performance in the static system, SINS would also start out with those same settings and improve even further upon its initial performance.

Our experiments revealed that, in both 25% and 50% cluttered worlds, SINS needed about 40 worlds to learn enough to be able to perform successfully thereafter using 10 or 20 cases. However, with higher numbers of cases (40 and 80), it took more trials to learn the regularities in the environment. It appears that larger numbers of cases require more trials to train through trial-and-error reinforcement learning methods, and furthermore there is no appreciable improvement in later performance, The **case-length** parameter did not have an appreciable effect on performance in the long run, except on the *average number of virtual collisions* estimator which showed the best results with case lengths of 4 and 10.

As observed earlier in experiment set 1, SINS requires a time overhead for case-based reasoning and thus loses out on the *average time* estimator. Due to the nature of our current case retrieval algorithm, the time required increases linearly with **max-cases** and with **case-length**. In 25% cluttered worlds, values of 10 and 4, respectively, for these parameters provide comparable performance.

Experiment set 3: Effect of control interval. Although all settings resulted in improved performance through experience, the best and worst performance in terms of *average number of worlds solved* was obtained with **control-interval** set to 12 and 4, respectively. For low **control-interval** values, we expect poorer performance because environment classification cannot occur reliably. We also expect poorer performance for very high values because the system cannot adapt its schema parameters quickly enough to respond to changes in the environment. Other performance estimators also show that **control-interval** = 12 is a good setting. Larger **control-intervals** require less case retrievals and thus improve **average time**; however, this gets compensated by poorer performance on other estimators.

Experiment set 4: Effect of environmental change. The results from these experiments demonstrate the flexibility and adaptiveness of the learning methods used in SINS. Regardless of parameter settings, SINS continued to be able to navigate successfully despite a sudden change in environmental clutter. It continued to solve about 95% of the worlds presented to it, with only modest deterioration in steps, distance, virtual collisions and time in more cluttered environments. The performance of the static system, in contrast, deteriorated in the more cluttered environment.

Summary: These and other experiments show the efficacy of the multistrategy adaptation and learning methods used in SINS across a wide range of qualitative metrics, such as flexibility of the system, and quantitative metrics that measure performance. The results also indicate that a good configuration for practical applications is **max-cases** = 10, **case-length** = 4, and **control-interval** = 12, although other settings might be chosen to optimize particular performance estimators of interest. These values have been determined empirically. Although the empirical results can be explained intuitively, more theoretical research is needed to analyze why these particular values worked best.

4 Conclusions

We have presented a novel method for augmenting the performance of a reactive control system that combines case-based reasoning for on-line parameter adaptation and reinforcement learning for on-line case learning and adaptation. The method is fully implemented in the SINS program, which has been evaluated through extensive simulations.

The power of the method derives from its ability to capture common environmental configurations, and regularities in the interaction between the environment and the system, through an on-line, adaptive process. The method adds considerably to the performance and flexibility of the underlying reactive control system because it allows the system to select and utilize different behaviors (i.e., different sets of schema parameter values) as appropriate for the particular situation at hand. SINS can be characterized as performing a kind of constructive representational change in which it constructs higher-level representations (cases) from low-level sensorimotor representations (Ram, 1993).

In SINS, the perception-action task and the adaptation-learning task are integrated in a tightly knit cycle, similar to the "anytime learning" approach of Grefenstette & Ramsey (1992). Perception and action are required so that the system can explore its environment and detect regularities; they also, of course, form the basis of the underlying performance task, that of navigation. Adaptation and learning are required to generalize these regularities and provide predictive suggestions based on prior experience. Both tasks occur simultaneously, progressively improving the performance of the system while allowing it to carry out its performance task without needing to "stop and think."

In contrast to traditional case-based reasoning methods which perform high-level reasoning in discrete, symbolic problem domains, SINS is based on a new method for "continuous case-based reasoning" in problem domains that involve continuous information, such as sensorimotor information for robot navigation (Ram & Santamaría, 1993). There are still several unresolved issues in this research. The case retrieval process is very expensive and limits the number of cases that the system can handle without deteriorating the overall navigational performance, leading to a kind of utility problem (Minton, 1988). Our current solution to this problem is to place an upper bound on the number of cases allowed in the system. A better solution would be to develop a method for organization of cases in memory; however, conventional memory organization schemes used in case-based reasoning systems (see Kolodner, 1992) assume structured, nominal information rather than continuous, time-varying, analog information of the kind used in our cases.

Another open issue is that of the nature of the regularities captured in the system's cases. While SINS' cases do enhance its performance, they are not easy to interpret. Interpretation is desirable, not only for the purpose of obtaining of a deeper understanding of the methods, but also for possible integration of higher-level reasoning and learning methods into the system.

Despite these limitations, SINS is a complete and autonomous self-improving navigation system, which can interact with its environment without user input and without any pre-programmed "domain knowledge" other than that implicit in its reactive control schemas. As it performs its task, it builds a library of experiences that help it enhance its performance. Since the system is always learning, it can cope with major environmental changes as well as fine tune its navigation module in static and specific environment situations.

References

Arkin, R.C., Motor Schema-Based Mobile Robot Navigation, *The International Journal of Robotics Research*, 8(4):92–112, 1989.

Brooks, R., A Robust Layered Control System for a Mobile Robot, *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, 1986.

Chien, S.A., Gervasio, M.T., & DeJong, G.F., On Becoming Decreasingly Reactive: Learning to Deliberate Minimally, in Birnbaum, L. & Collins, G. (editors), *Proceedings of the Eighth*

International Workshop on Machine Learning, 288–292, Chicago, IL, 1991.

Clark, R.J., Arkin, R.C., & Ram, A., Learning Momentum: On-Line Performance Enhancement for Reactive Systems, in *Proceedings of the IEEE International Conference on Robotics and Automation*, 111–116, Nice, France, 1992.

Fikes, R.E., Hart, P.E., & Nilsson, N.J., Learning and Executing Generalized Robot Plans. *Artificial Intelligence*, 3:251–288, 1972.

Grefenstette, J.J. & Ramsey, C.L. An Approach to Anytime Learning, in Sleeman, D. & Edwards, P. (editors), *Machine Learning: Proceedings of the Ninth International Conference*, 189–195, Aberdeen, Scotland, 1992.

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

Hammond, K.J. (editor), *Proceedings of the Second Case-Based Reasoning Workshop*, Pensacola Beach, FL, Morgan Kaufman, 1989b.

Kaelbling, L., An Architecture for Intelligent Reactive Systems, Technical Note 400, SRI International, 1986.

Kolodner, J.L. (editor), *Proceedings of a Workshop on Case-Based Reasoning*, Clearwater Beach, FL, Morgan Kaufman, 1988.

Kolodner, J.L., *Case-Based Reasoning*, Morgan Kaufmann, San Mateo, CA, 1992 (in press).

Minton, S., *Learning Effective Search Control Knowledge: An Explanation-Based Approach*, PhD thesis, Technical Report CMU-CS-88-133, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, 1988.

Mitchell, T.M., Becoming Increasingly Reactive, in *Proceedings of the Eighth National Conference on Artificial Intelligence*, 1051–1058, Boston, MA, 1990.

Moorman, K. & Ram, A., A Case-Based Approach to Reactive Control for Autonomous Robots, in *Proceedings of the AAAI Fall Symposium on AI for Real-World Autonomous Mobile Robots*, Cambridge, MA, 1992.

Mostow, J. & Bhatnagar, N., FAILSAFE — A Floor Planner that uses EBG to Learn from its Failures, in *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, 249–255, Milan, Italy, 1987.

Payton, D., An Architecture for Reflexive Autonomous Vehicle Control, in *Proceedings of the IEEE Conference on Robotics and Automation*, 1838–1845, 1986.

Pearce, M., Arkin, R., & Ram, A., The Learning of Reactive Control Parameters through Genetic Algorithms, In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 130–137, Raleigh, NC, 1992.

Ram, A., Creative Conceptual Change, in *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*, Boulder, CO, 1993 (to appear).

Ram, A. & Santamaría, J.C., Continuous Case-Based Reasoning, in Leake, D.B. (editor), *Proceedings of the AAAI Workshop on Case-Based Reasoning*, Washington, DC, 1993 (to appear).

Sacerdoti, E.D., A Structure for Plans and Behavior, Technical Note 109, Stanford Research Institute, Artificial Intelligence Center. Summarized in P.R. Cohen & E.A. Feigenbaum's Handbook of AI, Volume III, pages 541-550, 1975.

Segre, A.M., *Machine Learning of Robot Assembly Plans*, Kluwer Academic Publishers, Norwell, MA, 1988.

Sutton, R.S., Integrated Architectures for Learning, Planning, and Reacting based on Approximating Dynamic Programming, in *Proceedings of the Seventh International Conference on Machine Learning*, 216–224, Austin, TX, 1990.

Sutton, R.S. (editor), *Machine Learning*, 8(3/4), special issue on Reinforcement Learning, 1992.

Verschure, P.F.M.J., Kröse, B.J.A., & Pfeifer, R., Distributed Adaptive Control: The Self-Organization of Structured Behavior. *Robotics and Autonomous Systems*, 9:181–196, 1992.

Watkins, C.J.C.H., *Learning from Delayed Rewards*, PhD thesis, University of Cambridge, England, 1089.

Whitehead, S.D. & Ballard, D.H., Active Perception and Reinforcement Learning, in *Proceedings of the Seventh International Conference on Machine Learning*, 179–188, Austin, TX, 1990.