MULTISTRATEGY LEARNING IN REACTIVE CONTROL SYSTEMS FOR AUTONOMOUS ROBOTIC NAVIGATION

Ashwin Ram and Juan Carlos Santamaría
College of Computing, Georgia Institute of Technology,
Atlanta, Georgia 30332-0280, U.S.A.

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

Edited by: Gheorghe Tecuci
Received: October 15, 1993 **Revised:** November 6, 1993 **Accepted:**

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 case 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.

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; Georgeff, 1987; Maes, 1990). 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., Mostow & Bhatnagar, 1987; Minton, 1988; Segre, 1988; Hammond, 1989). 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 (Brooks, 1986; Kaelbling, 1986; Payton, 1986; Arkin, 1989). 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 being used in the navigation module to get the best performance. 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 used for reactive control. 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 uses a reward that depends on the similarity of the observed mapping in the current environment to the mapping represented in the model combined with a more traditional punishment signal. This causes the system to converge towards those mappings that are consistent and beneficial over a set of experiences.

The representations used by SINS to model its interaction with the environment are initially underconstrained 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 Navigation Module

The navigation module, which uses schema-based reactive control methods, is based on the AuRA architecture (Arkin, 1989). The module 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

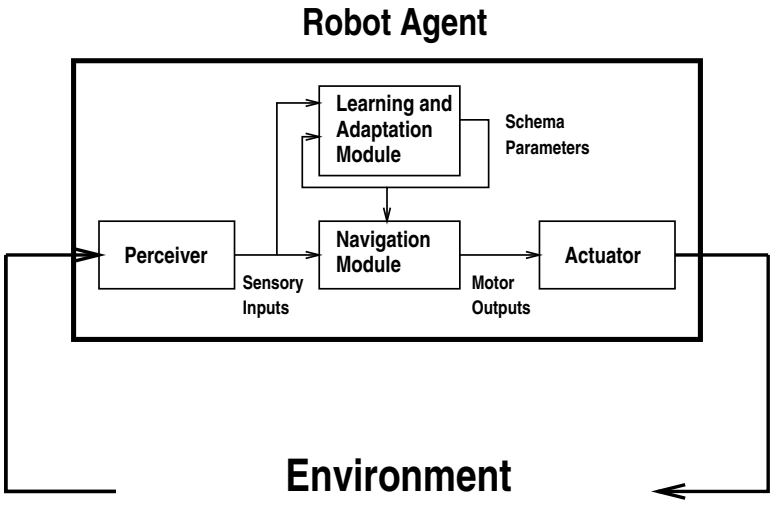

Figure 1: System architecture

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" (Ram, Arkin, Moorman, & Clark, 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 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.

Different combinations of schema parameters cause 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 enhance navigational performance, as in the ACBARR system (Ram, Arkin, Moorman & Clark, 1992). SINS adopts this approach by allowing schema parameters to be modified dynamically. However, in ACBARR, schema modification information is supplied by the designer using hand-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.

SINS uses a model consisting of associations between sensory inputs and schema parameters values. Each set of associations is represented as a case. Sensory inputs provide 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 an estimate of a quantitative variable (a sensory input or

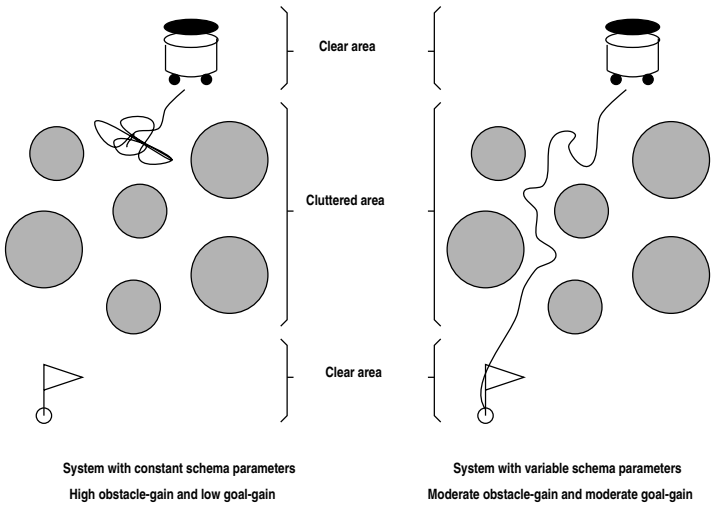

Figure 2: Typical navigational behaviors of different tunings of the reactive control module. The figure on the left shows the non-learning system with high obstacle avoidance and low goal attraction. On the right, the learning system has lowered obstacle avoidance and increased goal attraction, allowing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

a schema parameter) over a window of time. A vector represents the trend or recent history of such estimates of a variable. A case models an association between sensory inputs and schema parameters by grouping their respective vectors together. Figure 3 shows 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 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 environmental 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 for a "control interval" 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 Hammond, 1989; Kolodner, in press), and from reinforcement learn-

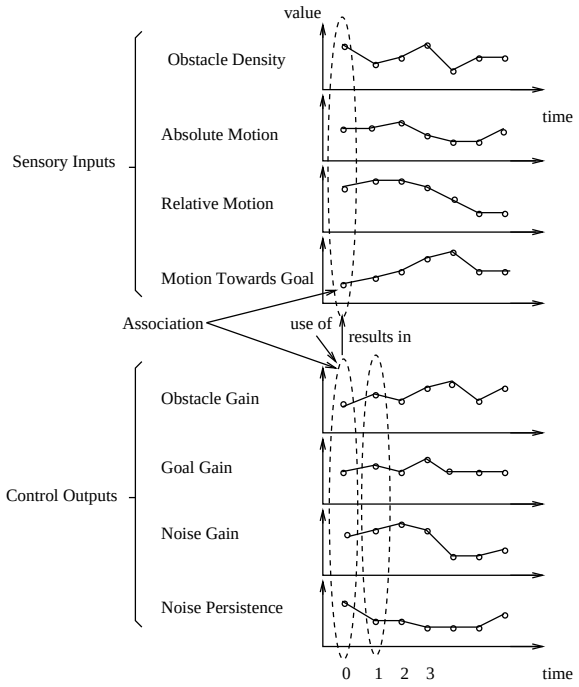

Figure 3: Sample representations showing the time history of analog values representing sensory inputs and control outputs. Associations between sensory inputs and control outputs are arranged vertically, and the sequence of associations over time is arranged horizontally. Each case in the system is represented in this manner, as is the current on-going navigational experience of the system.

ing, 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, 1989) 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. Additionally, the reinforcement learning method uses a punishment signal to reject mappings that are not beneficial to the system.

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 appropriate (as far as the system knows based on its previous experience) for that environment. The learning and adaptation 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 its predictions.

The overall algorithm is shown in table 1.

```
do
{
  /* PERCEIVE: Update input vectors */
  current_environment = perceive();

  if ( end of control interval )
    then {

      /* LEARN: Decide whether to reinforce or explore */
      if ( outcome was good )
        then {
          reinforce_schemas (previous_case, current_environment);

          /* LEARN: Decide if case should be extended */
          if ( prediction is good and at end of sequence )
            then
              extend_case (previous_case);
        }
        else
          explore_schemas (previous_case);

      /* RETRIEVE: Retrieve best case */
      best_case = retrieve_best_case(current_environment);

      /* LEARN: Decide if a new case should be created */
      if ( best_case is not a good match )
        then
          best_case = create_case (current_environment);

      /* ADAPT: Modify current set of schema parameters */
      adapt_schemas (best_case);

      /* Wait until next cycle */
      previous_case = best_case;
    }

  /* Move robot using the current set of schema parameters */
  execute();
}
while (not (goal reached or maximum number of steps exceeded ))
```

Table 1: SINS algorithm

The **perceive** function constructs and maintains a representation of the current environmental situation by reading the robot's sensors and updating the input and output vectors accordingly. This results in a set of $J = 4$ input vectors $\mathbf{E}_{\mathrm{input}_j}$, one for each sensory input j, and $K = 4$ output vectors $\mathbf{E}_{\mathrm{output}_k}$, one for each output vector k as described earlier. Then, every control interval T, the learning and adaptation module performs two main functions: It adapts the schema parameters currently in use by the reactive control module so that it performs better in the new environment, and it learns useful sequences of associations between environment situations and schema parameters.

Schema parameters are adapted using the **retrieve_best_case** and **adapt_schemas** functions. In the **retrieve_best_case** function, the case most similar to the current environment situation is selected by matching the environment's input and output vectors $\mathbf{E}_{\mathrm{input}_j}$, $\mathbf{E}_{\mathrm{output}_k}$ from the **perceive** step against the corresponding input and output vectors $\mathbf{C}^n_{\mathrm{input}_j}$, $\mathbf{C}^n_{\mathrm{output}_k}$ of the cases $\mathbf{C}^n$ in the system's memory (see figure 4). The best matching case $\mathbf{C}^{n_{\mathrm{best}}}$ and the position of the best match p_{best} are handed to the **adapt_schemas** function, which modifies the schema parameter values currently in use based on the recommendations $\mathbf{C}^{n_{\mathrm{best}}}_{\mathrm{output}_j}(p_{\mathrm{best}}+1)$ from the output vectors of the case.

Finally, the learning and adaptation module decides how to utilize information from the current experience with the best case in order to improve its case library. The system learns in three different ways: by improving the content of the case that was just used in order to make it more reliable, by creating a new case whenever the best case retrieved is not good enough, or by extending the length of the case in order to build up longer sequences of associations. The contents of a case are improved by the **reinforce_schemas** function, which reinforces the suggestions of the case if these suggestions led to a favorable outcome over the last control interval, and by the **explore_schemas** function, which uses random exploration to try out other schema parameter values if the suggested set of values did not prove useful. The outcome is evaluated by monitoring the behavior of the robot over the last control interval; collisions are undesirable, as is lack of movement.

One difference between our methods and traditional reinforcement learning is that SINS is trying to maximize consistency in "useful" behaviors as determined by a reward signal, whereas traditional reinforcement learning tries to maximize the expected utility the system is going to receive in the future as determined by the reward signal (cf. Watkins, 1989; Whitehead & Ballard, 1990). In schema-based reactive control navigation, it is inherently a good idea to modify schema parameters in an on-line fashion; however, not all modifications are equally good since some may cause the robot to collide with obstacles or not to move at all. SINS uses the reward signal to decide whether to reinforce a behavior or to explore alternative behaviors; reinforcement, when chosen, is used to reinforce behaviors that are consistent across experiences. Thus, in addition to external outcome, consistency is used as an "internal" reward signal for the reinforcement learning method.

Furthermore, traditional reinforcement learning assumes that the outcomes of the system's actions are known; it learns which actions to execute to maximize a reward. In SINS, the outcomes of what corresponds to "actions" (the adaptations to be performed on the navigation module) are not known; part of the learning task is to discover the sequences of environmental situations are likely to result from a given sequence of adaptations and, in turn, which adaptations are appropriate in different situations. Thus, SINS is learning a model of its sensorimotor interaction with the environment (represented as a set of cases) at the same time as it is learning to improve its navigational performance through on-line adaptation of its reactive control schemas.

In addition to modifying its cases, SINS can also extend its cases and learn new cases. In order to decide

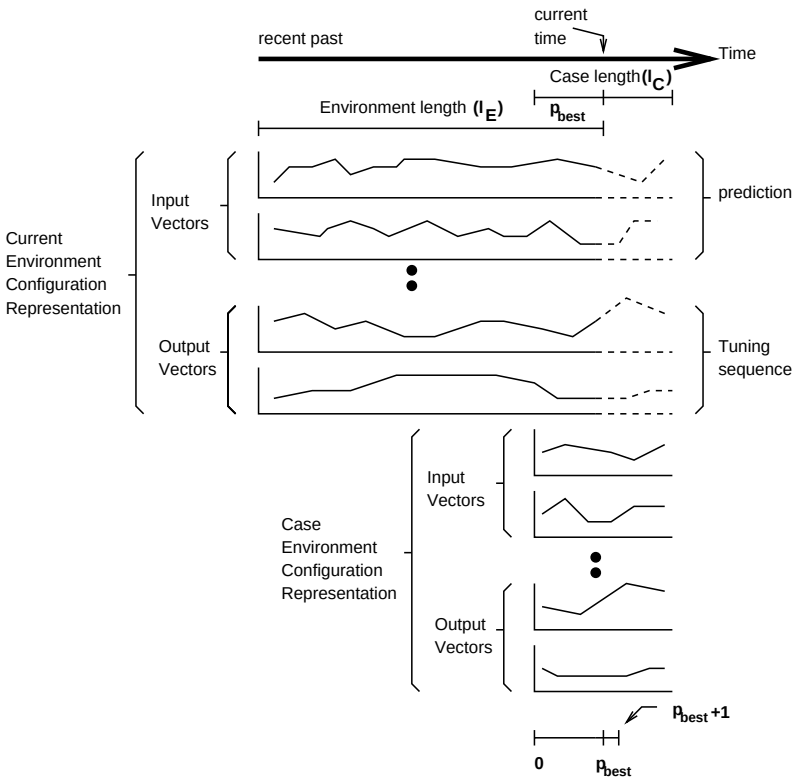

Figure 4: Schematic representation of the match process. Each graph in the case (below) is matched against the corresponding graph in the current environment (above) to determine the best match, after which the remaining part of the case is used to guide navigation (shown as dashed lines).

which kind of learning to perform in a given situation, SINS uses a relative similarity criterion to judge the appropriateness of the best matching case in the current situation. This determination is based on statistical information about the quality of match in prior applications of the case as compared to the quality of match in the current situation. If the best matching case is not as similar to the current environment situation as it has been in previous situations, the case is probably inappropriate for this situation; thus, it is better to learn a new case to represent what is probably a new class of situations. If this occurs, SINS uses the **create_new_case** function to create a new case based on the current experience and add it to the case library. To determine whether to create a new case, SINS compares the current match with the mean match plus the standard deviation of the matches over the past utilizations of the case. This ensures that new sequences of associations are created only when the available sequences of associations already captured in the case library do not fit the current environment.

The third kind of learning is carried out by the **extend_case_size** function, which extends the length of a case whenever the best case makes an accurate prediction of the next environment situation and there are no more associations in the sequence. This allows the system to increase the length of the sequence of associations only when it is confident that the sequence of the case accurately predicts how the environment changes if the suggested schema parameters are used. To estimate this confidence, the predicted values are matched with the actual environmental parameters that result; if this match is better than the mean match, the case is extended. Intuitively (as before), if the case predicts the current situation better than it predicted the previous situations that it was used in, it is likely that the current situation involves the very regularities that the case is beginning to capture; thus, it is worthwhile extending the case so as to incorporate the current situation. Alternatively, if the match is not quite as good, the case should not be modified because doing so would take it away from the regularities it has been converging towards.

Since the reinforcement formulae are based on a relative similarity criterion, 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 only the causal patterns that the reward utility identifies as useful and to use them to modify its actions by updating its schema parameters as appropriate. Useful causal patterns are those that do not cause robot collisions or do not cause the robot to stop.

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, and 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 stimuli) into an already working navigation system, while SINS improves its navigation performance by learning how to adapt the system (i.e., the navigation module) itself. 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.

We now present a detailed description of and the mathematical formulas used in the perception, matching, adaptation, and learning tasks.

2.4.1 Perception

The objective of the **perceive** function is to generate an accurate description of the current environment situation. It performs this task by shifting the previous values in each input and output vector one position back in time[1] and then calculating the current values for each input vector $\mathbf{E}_{\text{input}_j}(0), j = 1, \ldots, J$ and output vector $\mathbf{E}_{\text{output}_k}(0), k = 1, \ldots, K$, where 0 is the current position in time. The current values for the input vectors are based on the robot's sensors, and the current values for the output vectors are just the respective values of the schema parameters suggested in the previous control interval. The vectors are updated at the end of each control interval of time T.

To update the input vectors, the system monitors the values of the robot's sensors $\mathbf{Sensor}_j$ corresponding to each input vector $\mathbf{E}_{\text{input}_j}$. The sensors are monitored at each time step over the past control interval; these sensor readings are then averaged to yield the new value for the corresponding input vectors. Thus, the input vectors in the environment representation are updated using the following formula:[2]

$$\mathbf{E}_{\text{input}_j}(i) = \begin{cases} \mathbf{E}_{\text{input}_j}(i-1) & \text{if } i > 0 \\ \frac{\sum_{t=-T}^{t=0} \mathbf{Sensor}_j(t)}{T} & \text{if } i = 0 \end{cases}$$

where $\mathbf{Sensor}_j(n)$ is the sensory input that corresponds to the input vector $\mathbf{E}_{\text{input}_j}$ (sensed obstacles for **Obtacle-Density**, distance traveled for **Absolute-Motion**, relative position for **Relative-Motion**, and normal relative position for **Motion-Towards-Goal**), and t ranges over each robot step since the last control interval.

2.4.2 Retrieval and Matching

The function **retrieve_best_case** is responsible for selecting a case from the case library that best matches the current environment situation. The case similarity metric is based on the mean squared difference between each of the vector values of the case over a trending window, and the vector values of the environment. The best match window is calculated using a reverse sweep over the time axis p similar to a convolution process to find the relative position that matches best. Each case C^n in the case library is matched against the current environment using exhaustive search, which returns the best matching case $\mathbf{C}^{n_{\text{best}}}$ along with the relative position p_{best} of the match (see figure 4). After retrieving the best case, the mean and variance of the case's statistical match history are updated; these will be used later to calculate the relative similarity criterion during learning.

The case similarity metric SM of a case $\mathbf{C}$ at position p relative to the environment $\mathbf{E}$ is a value that indicates the similarity between the sequence of associations encoded in the case to the sequence of associations in the current environment situation starting at position p. The lower the value of the case similarity metric, the more similar the sequences of associations. The case similarity metric formula calculates a weighted sum of the squared difference between the corresponding vectors of the case and the environment. For the SM to be valid, p must lie between 0 and l_C.[3]

$SM(\mathbf{E}, \mathbf{C}, p) =$

[1] This is implemented using a circular buffer which does not requires copying each of the values from one cell to the next.

[2] Note that i counts back in time (i.e., $i = 0$ is the current time and $i > 0$ is the recent past.)

[3] We used $w_j = w_k = 1.0$ (i.e., input and output vectors all contribute equally in the similarity metric.)

$$\sum_{j=1}^{J} w_j \sum_{i=0}^{\min(p,l_E)} \frac{(\mathbf{E}_{\text{input}_j}(i) - \mathbf{C}_{\text{input}_j}(p-i))^2}{p+1}$$
$$+\sum_{k=1}^{K} w_k \sum_{i=0}^{\min(p,l_E)} \frac{(\mathbf{E}_{\text{output}_k}(i) - \mathbf{C}_{\text{output}_k}(p-i))^2}{p+1}$$

The best case is obtained by matching each case $\mathbf{C}^n$ in the case library at all the positions p and selecting the pair $(n_{\text{best}}, p_{\text{best}})$ that yields the lowest SM. Formally, this can be expressed as:

$$\{n_{\text{best}}, p_{\text{best}} | \min(SM(\mathbf{E}, \mathbf{C}^n, p)),\ \forall n,\ 0 \le p \le l_{C^n}\}$$

Each case $\mathbf{C}$ maintains a statistical record of the similarity metrics it has produced in the past, which is updated every time the case is retrieved as the best case. The mean ($\mathbf{C}_{SM_{\text{mean}}}$) and variance ($\mathbf{C}_{SM_{\text{var}}}$) of the case similarity metric as well as the number of times the case has been used ($\mathbf{C}_{\text{used}}$) are updated using standard formulae in descriptive statistics:

$$\begin{aligned}
\text{new } \mathbf{C}_{SM_{\text{mean}}} &= \frac{\mathbf{C}_{\text{used}}\mathbf{C}_{SM_{\text{mean}}} + SM}{\mathbf{C}_{\text{used}} + 1} \\
\text{new } \mathbf{C}_{SM_{\text{var}}} &= \frac{\mathbf{C}_{\text{used}} - 1}{\mathbf{C}_{\text{used}}}\mathbf{C}_{SM_{\text{var}}} \\
&\quad +(\text{new } \mathbf{C}_{SM_{\text{mean}}} - \mathbf{C}_{SM_{\text{mean}}})^2 \\
&\quad +\frac{(SM - \text{new } \mathbf{C}_{SM_{\text{mean}}})^2}{\mathbf{C}_{\text{used}}} \\
\text{new } \mathbf{C}_{\text{used}} &= \mathbf{C}_{\text{used}} + 1
\end{aligned}$$

2.4.3 Adaptation

The best matching case $\mathbf{C}^{n_{\text{best}}}$ is used to adapt the schema parameter values currently in use by the reactive control module. The values of output vectors for the next association $\mathbf{C}^{n_{\text{best}}}_{\text{output}_k}$ after position p_{best} are used to determine the new set of schema parameters values $\mathbf{Parameter}_k$ until the next control interval. Since learning tends to reinforce those associations that are consistently observed over several experiences, the new set of schema parameters can be expected to cause the robot to move safely and the next environment configuration that results from the movement can be expected to be the one predicted by the association. Since output vectors directly represent schema parameters, adaptation is a straightforward operation:

$$\mathbf{Parameter}_k = \mathbf{C}^{n_{\text{best}}}_{\text{output}_k}(p_{\text{best}} + 1), \quad \forall k = 1, \ldots, K$$

2.4.4 Learning

In addition to perceiving the environment, retrieving the best matching case, and adapting the schema parameters being used by the reactive control module, SINS must also learn by updating its case library based on its current experience. Three types of learning are possible: modification of the associations contained in a case, creation of a new case based on the current experience, and extension of the size of a case to yield associations over larger time windows. Modification of case contents, in turn, can be of two types: reinforcement of the associations contained in the case based on a successful experience, and exploration of alternative associations based on an unsuccessful experience.

SINS decides which kind of learning to perform using a relative similarity criterion which determines the quality of the best match. The match value of the best case, based on the case similarity metric, is compared with the match values of the case in previous situations in which it was used. If the current match is worse than the mean match value by more than a standard deviation, the case (although still the best match) is considered to be too different from the current situation, since it has been a better match to other situations in the past. In this case, the **create_case** function is invoked to create a new case containing a sequence of associations formed by copying the values of the sequence of associations in the current environmental representation:

$$\begin{aligned}
\mathbf{C}^{n_{\text{best}}}_{\text{input}_j}(0) &= \mathbf{E}^{n_{\text{best}}}_{\text{input}_j}(0), \quad \forall j = 1, \ldots, J \\
\mathbf{C}^{n_{\text{best}}}_{\text{output}_k}(0) &= \mathbf{E}^{n_{\text{best}}}_{\text{output}_k}(0), \quad \forall k = 1, \ldots, K
\end{aligned}$$

If, on the other hand, the best case matches the current situation well, it is likely that the current situation is representative of the class of situations that that case is beginning to converge towards. If the case provides good recommendations for action, its recommendations should be reinforced; if not, its recommendations should be modified. In SINS, collisions with obstacles and lack of movement are undesirable by definition of the navigation task. A set of schema parameters is considered beneficial if using it does not lead to an undesirable outcome. The objective of the learning functions is to improve the accuracy of prediction of the system's cases and, in turn, to discover those schema parameter values that result in environmental situations that are beneficial for the robot.

If the best case recommends a set of schema parameters that are not beneficial to the robot, the **explore_schemas** function is used to modify the case such that it suggests a different set of schema parameters in similar circumstances in the future. Specifically, the output vectors $\mathbf{C}^{n_{\text{best}}}_{\text{output}_k}(p_{\text{best}} + 1)$ associated with the environment situation following the best match position p_{best} are modified in a random manner since the current values are not useful to the system. The small random changes allow the system to explore the space of possible schema parameters in a controlled manner. These changes are defined by the following formula:

$$\begin{aligned}
\rho &= \min(1, \alpha \text{ collisions} + \beta\frac{\text{velocity}}{\text{max_velocity}}) \\
&\mathbf{C}^{n_{\text{best}}}_{\text{output}_k}(p_{\text{best}} + 1) = \\
&\quad (1-\rho)\ \mathbf{C}^{n_{\text{best}}}_{\text{output}_k}(p_{\text{best}} + 1) \\
&\quad + \rho\ \text{random}(\min \mathbf{C}^{n_{\text{best}}}_{\text{output}_k}, \max \mathbf{C}^{n_{\text{best}}}_{\text{output}_k}), \\
&\quad \forall k = 1, \ldots, K
\end{aligned}$$

where ρ is a "reject" value that determines the extent to which the current recommendations should be taken into account when determining the modified values. A value of $\rho = 0$ specifies that the value of the output vector should be left unchanged, and a value of $\rho = 1$ specifies that the value of output vector should be replaced completely by a new random value in the allowable range. In any given learning cycle, the value of ρ depends on α and β, which represent the importance of avoiding collisions and moving, respectively. In this paper, we used $\alpha = 0.5$ and $\beta = 1.0$.

If, on the other hand, the schema parameters suggested by the best matching case produce desirable results, the **reinforce_schemas** function is invoked. This function updates the case by making it more like the current environmental situation, so as to produce the same recommendations in similar situations in the future. This reinforcement is done using the following formulae:

$$\begin{aligned}
&\forall i = 0, \ldots, p_{\text{best}} \\
&\mathbf{C}^{n_{\text{best}}}_{\text{input}_j}(i) \quad = \quad \tfrac{\lambda}{i+1}(\mathbf{E}_{\text{input}_j}(p_{\text{best}} - i) - \mathbf{C}^{n_{\text{best}}}_{\text{input}_j}(i)), \\
&\qquad\qquad\qquad \forall j = 1, ..., J \\
&\forall i = 0, \ldots, p_{\text{best}} \\
&\mathbf{C}^{n_{\text{best}}}_{\text{output}_k}(i) \quad = \quad \tfrac{\lambda}{i+1}(\mathbf{E}_{\text{output}_k}(p_{\text{best}} - i) - \mathbf{C}^{n_{\text{best}}}_{\text{output}_k}(i)), \\
&\qquad\qquad\qquad \forall k = 1, ..., K
\end{aligned}$$

where λ determines the learning rate (0.9 in the cuurent version of SINS).

Finally, the **extend_case** function extends the sequence of associations contained in a case. The decision to extend a case is also based on a statistical relative similarity criterion. If the case's predictions $\mathbf{C}^{n_{\text{best}}}_{\text{input}_j}(p_{\text{best}} + 1)$ are similar to the resulting environment situation within a standard deviation from the mean predictive similarity, and the case does not have more associations in the sequence (that is, it cannot provide a next set of schema parameters), then the case is extended by duplicating the last association of the case:

$$\begin{aligned}
\mathbf{C}^{n_{\text{best}}}_{\text{input}_j}(p_{\text{best}} + 2) \quad &= \quad \mathbf{C}^{n_{\text{best}}}_{\text{input}_j}(p_{\text{best}} + 1), \\
&\qquad \forall j = 1, \ldots, J \\
\mathbf{C}^{n_{\text{best}}}_{\text{output}_k}(p_{\text{best}} + 2) \quad &= \quad \mathbf{C}^{n_{\text{best}}}_{\text{output}_k}(p_{\text{best}} + 1), \\
&\qquad \forall k = 1, \ldots, K
\end{aligned}$$

The net result of these learning procedures is to cause the cases in the system's case library to converge towards regularities in the system's interactions with its environment. The system learns useful sequences of schema parameters for different environment situations; these are used to guide navigation and, in turn, are updated based on navigational outcomes so as to improve the reliability of their predictions in similar situations in the future.

3 Evaluation

We evaluated the methods presented above using extensive simulations across a variety of different types of environment, performance criteria, and system configurations. The objective of these experiments was to measure the qualitative and quantitative improvement in the navigation performance of SINS (denoted "sins" in the figures), and to compare this performance against several non-learning schema-based reactive systems (the "static" systems) that do not change schema parameters and a system that changes its schema parameter values randomly after every control interval (the "random" system). Furthermore, rather than simply measure 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 using several different metrics. 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 5 shows an example runs of two of the static systems, the random system, and the SINS system on a typical randomly generated world. 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. 15% clutter corresponded to relatively easy worlds and 50% clutter to very difficult worlds. We tested the effect of three different design parameters in the SINS system: **max-cases**, the maximum number of cases that SINS is allowed to create; **max-size**, l_C, representing the maximum number of associations in a case; and **control-interval**, T, which determines how often the schema parameters in the reactive control module are adapted.

We used three non-adaptive systems for comparison. These systems were identical to the underlying navigation module in SINS, but used different fixed sets of schema parameter values. The "staticH" system used a hand-coded set that we designed manually by watching the behavior of the system on both 15% and 50% cluttered worlds. The "static15" and "static50" systems used the schema parameter values reported in Ram, Arkin, Moorman, and Clark (1992), which were hand-optimized through extensive trial and error to 15% and 50% cluttered worlds, respectively. As discussed below, SINS performed as well as the static15 system on 15% cluttered worlds and the static50 system on the 50% cluttered worlds; furthermore, SINS

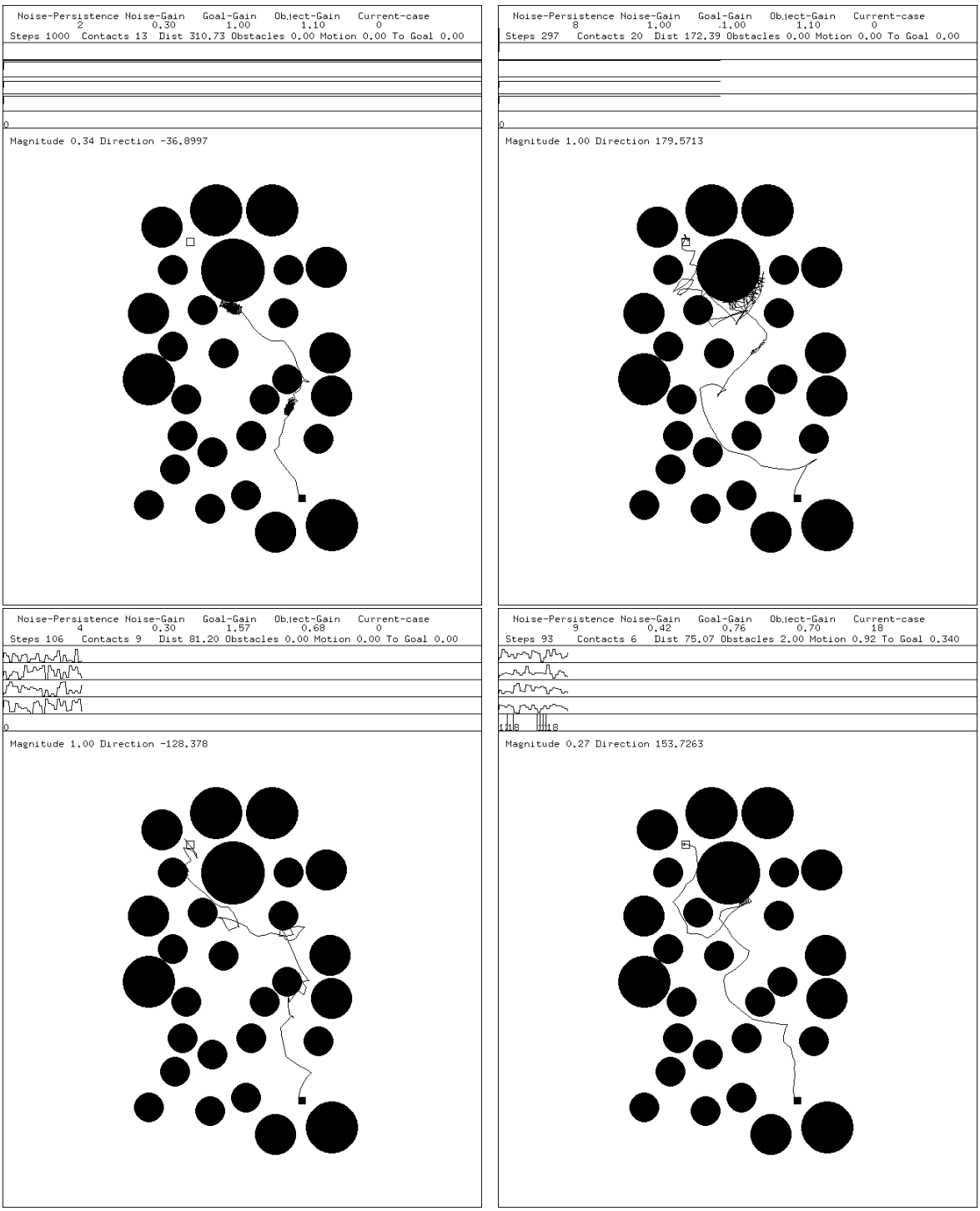

Figure 5: Sample runs of the static15 (top left), static50 (top right), random (bottom left), and SINS (bottom right) systems on a randomly generated world. The robot starts at the black box (towards the lower right side of the world) and tries to navigate to the white box. The graphs near the top of each figure show the values of each schema parameter over time.

reached this level of performance autonomously and, unlike the static systems which were optimized only for a specific clutter, was robust across sudden changes in clutter.

We used six estimators to evaluate the navigation performance of the systems. These metrics were computed using a cumulative average over several hundred test worlds to factor out the intrinsic differences in difficulty of different worlds. *Percentage 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 per world* indicates the average of number of steps that the robot takes to terminate each world; smaller values indicate better performance. *Average distance per world* indicates the total distance traveled per world on average; again, smaller values indicate better performance. *Average* $\frac{actual}{optimal}$ *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 (1000 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: Evaluation of 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.
- Experiment set 2: Evaluation of the effect of the parameters of the case-based reasoning component of the multistrategy learning system.
- Experiment set 3: Evaluation 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.
- Experiment set 4: Evaluation of the effect of changing environmental characteristics, and evaluation of the ability of the systems to adapt to new environments and learn new regularities.

3.2 Discussion of Experimental Results

The results in figures 6 through 10 show that SINS does indeed perform significantly better than its non-learning counterparts. To obtain a more detailed insight into the nature of the improvement, let us discuss the experimental results in more detail.

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 **max-size** = 10. Adaptation occurred every **control-interval** = 4 steps.

Figure 6 shows the results obtained for each estimator over 200 randomly generated worlds. Each graph compares the performance on one estimator of each of the five systems, SINS, staticH, static15, static50, and random, discussed above. Figure 6 shows the results obtained for each estimator over the 200 worlds with 50% clutter. The best configuration of SINS, which could learn up to 10 cases of maximum size 10 and with a control interval of 4, was selected to do the comparison. As shown in the graphs, SINS performed as well as or better than the other systems with respect to five out of the six estimators. Table 2 and 3 show the final improvement in the system after all the worlds with 15% and 50% clutter, respectively. For example, table 3 shows that SINS successfully navigates 100% of the worlds, the same as the static50 system optimized for 50% cluttered worlds, with 27% fewer virtual collisions. Although the non-learning system was 85% faster in time, the paths it found required the same number of steps. On average, SINS' solution paths were about the same length as those of the static50 system; however, it should be noted that the static50 system was customized for 50% cluttered worlds, whereas SINS improved its performance regardless what type of environment was dealing with. The static50 system did not perform as well in 15% cluttered worlds; although the static15 system did better, SINS compared favorably with that system in those worlds. Another important result is that SINS improved the performance independently of the initial values for the schema parameters; for example, when initialized with the same schema parameters as the staticH system, it was able to achieve performance far superior to the staticH system, and comparable with or better than the static15 system in 15% cluttered worlds and the

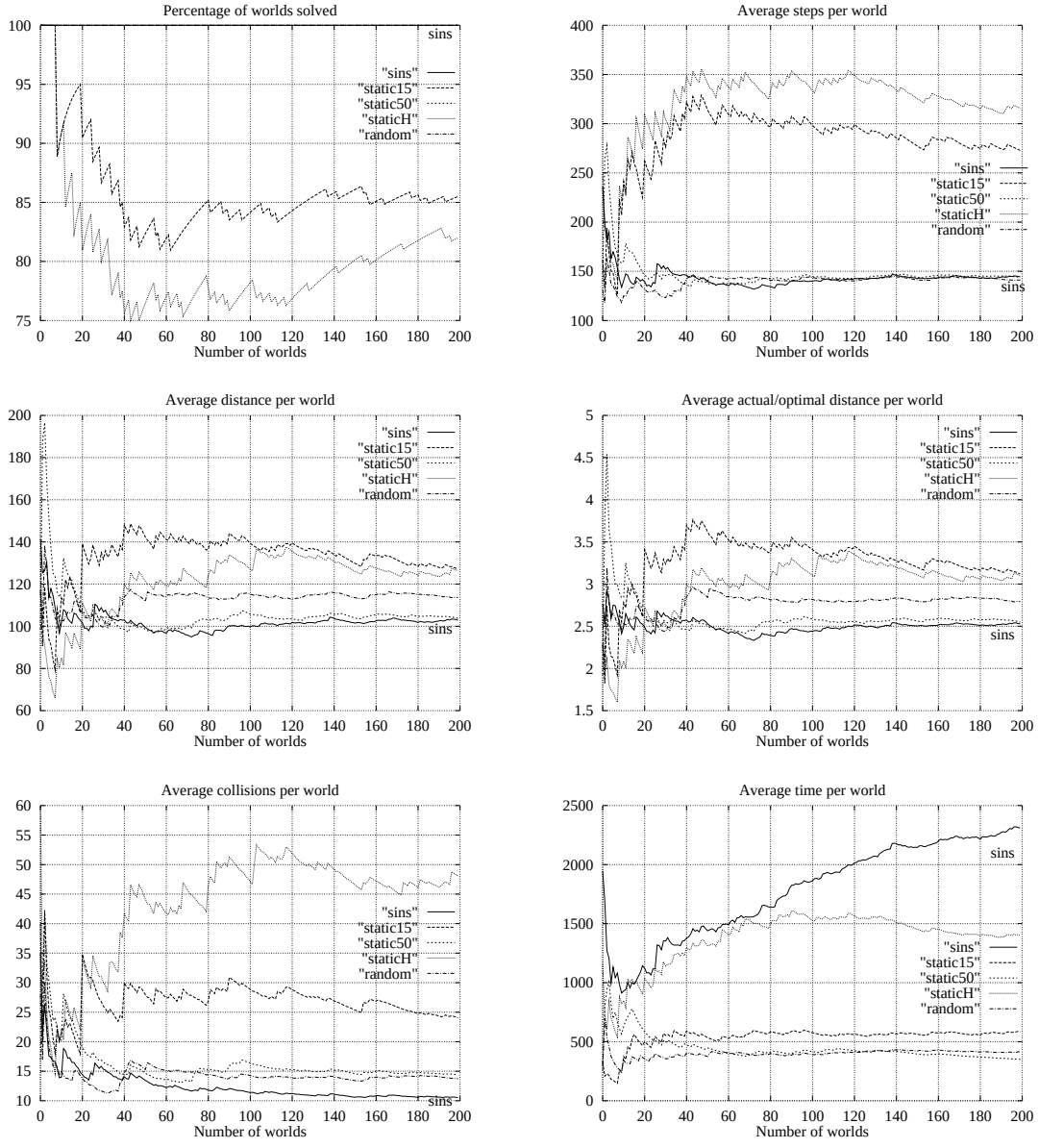

Figure 6: Cumulative performance results on 50% cluttered worlds.

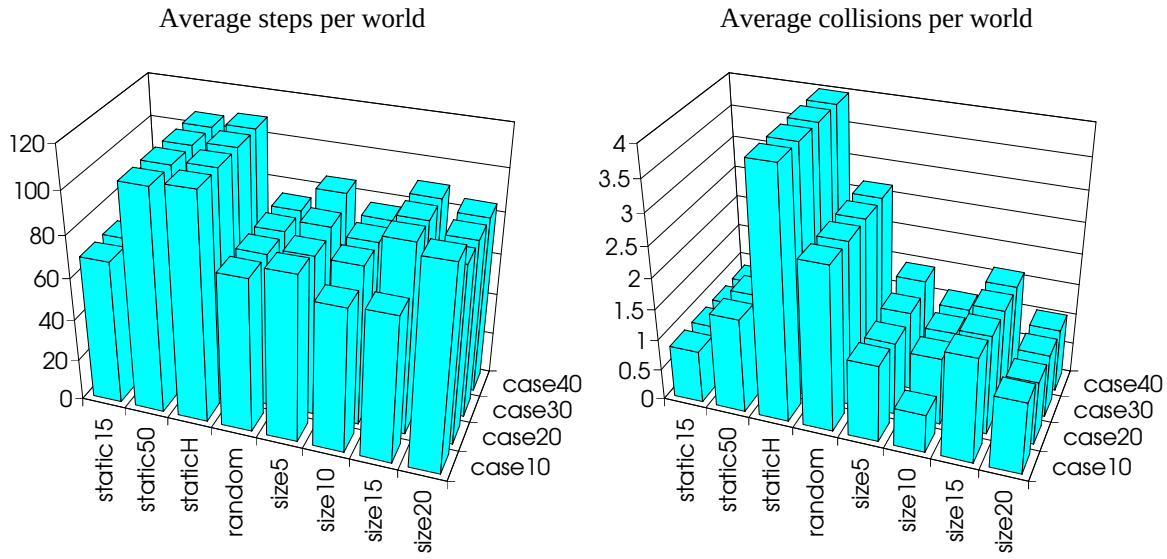

Figure 7: Effect of **max-cases** and **max-size** on 15% cluttered worlds.

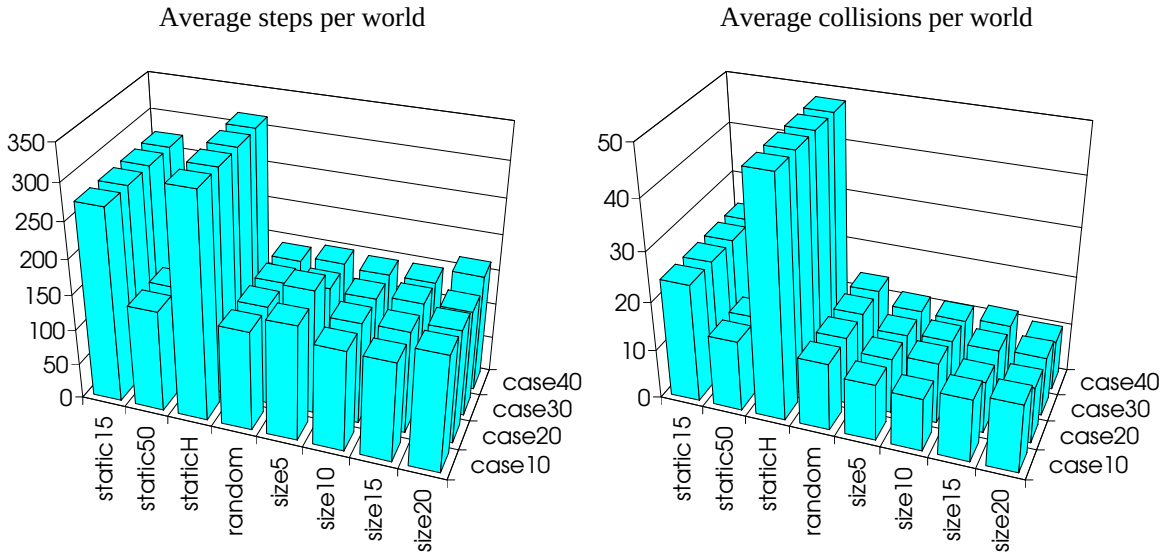

Figure 8: Effect of **max-cases** and **max-size** on 50% cluttered worlds.

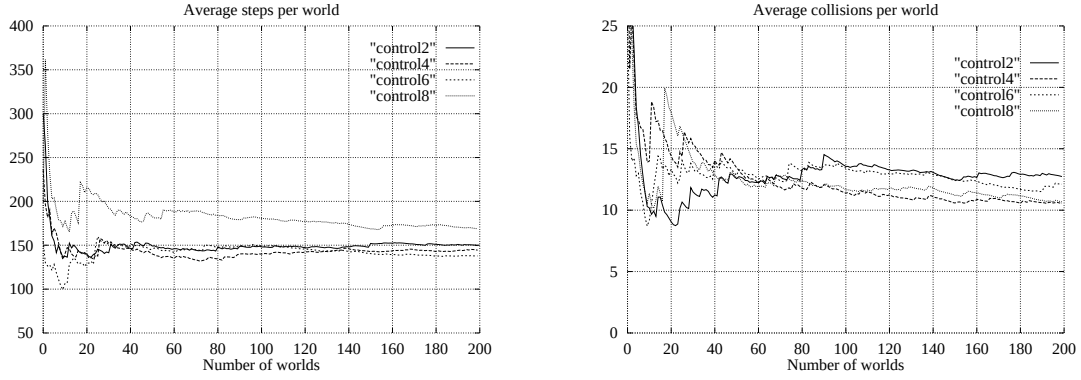

Figure 9: Effect of **control-interval** on 50% cluttered world with **max-cases**=20 and **max-size**=15.

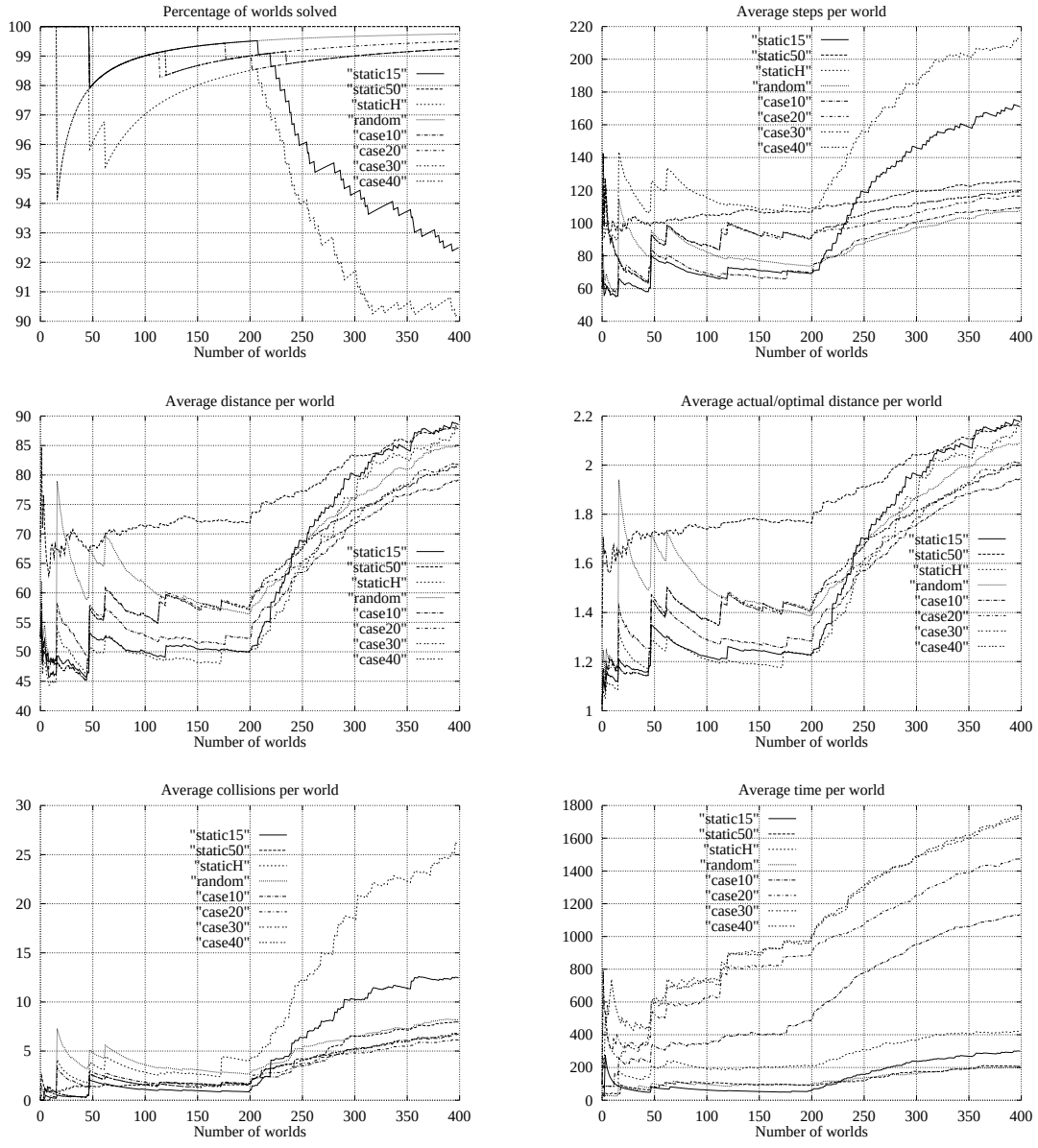

Figure 10: Effect of a sudden change in environment (after the 200th world).

	static15	*static50*	*staticH*	*random*	*SINS*
Percentage of worlds solved	99.5%	100%	98.5%	99%	99.5%
Average steps per world	69.1	106.4	108.5	73.8	68.9
Average distance per world	49.9	71.8	49.8	56.5	51.5
Average $\frac{\text{actual}}{\text{optimal}}$ distance	1.23	1.76	1.22	1.39	1.27
Average virtual collisions	0.85	1.53	3.99	2.66	0.62
Average time per world, ms	57	90	210	94	513

Table 2: Final performance results for 15% cluttered worlds.

	static15	*static50*	*staticH*	*random*	*SINS*
Percentage of worlds solved	85.5%	100%	82%	100%	100%
Average steps per world	272.7	143.6	315.5	141	141.9
Average distance per world	127.1	103.8	126.2	113.7	107.2
Average $\frac{\text{actual}}{\text{optimal}}$ distance	3.12	2.55	3.10	2.79	2.63
Average virtual collisions	24.1	14.4	48.1	13.7	10.7
Average time per world, ms	584	350	1400	413	1375

Table 3: Final performance results for 50% cluttered worlds.

static50 system in 50% cluttered worlds.

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 in SINS 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 also our assumption about the utility of adapting schema parameter values in reactive systems: 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 traveled. 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. 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 **max-size**. The parameter **control-interval** was held constant at 4, while **max-cases** was set to 10, 20, 30 and 40, and **max-size** was set to 5, 10, 15 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 15% and 50% clutter. Figures 7 and 8 show the results for two of the estimators, *average steps* and *average virtual collisions*.

All configurations of the SINS system navigated successfully in a comparable or larger percentage of the test worlds than the static systems. Regardless of the **max-cases** and **max-size** parameters, SINS could solve most of the 15% and 50% cluttered worlds. As before, the graphs show that SINS' performance was comparable to that of the static15 system in 15% cluttered worlds, and to the static50 system in 50% cluttered worlds. Thus, even if SINS is initialized with a poor set of schema parameter values, it can discover a good set of values and improve upon its initial performance.

Our experiments revealed that, in both 15% 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 (30 and 40), 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 **max-size** parameter has an appreciable effect on the performance for different environments. While in the 15% cluttered worlds, the best performance is obtained with cases of size 5, in 50% cluttered worlds, the best performance occurs with cases of size 15. The reason for this is that

complex worlds (i.e., 50% cluttered) require longer sequences of associations to ensure real progress. With shorter sequences of associations, there is a high probability that the system will get into a cycle that consists of going in and out of a local minimum point.

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 for case retrieval increases linearly with **max-cases** and with **max-size**.

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 **max-size** were held constant at 20 and 15, respectively, while **control-interval** was set to 2, 4, 6 and 8. All systems were evaluated using all six estimators on 200 randomly generated worlds of 50% clutter. The results are shown in figure 9.

Although all settings of control interval resulted in improved performance through experience, the best and worst performance in terms of *percentage of worlds solved* was obtained with **control-interval** set to 4 and 8, 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** = 4 is a good setting. Larger **control-intervals** require less case retrievals and thus improve *average time per world*; however, this gets compensated by poorer performance on other estimators.

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, 30 and 40, **max-size** set to 15, and **control-interval** set to 4, we presented the systems with 200 randomly generated worlds of 15% clutter followed by 200 randomly generated worlds of 50% clutter. The results for *average steps* and *average virtual collisions* are shown in figure 10.

The results from these experiments demonstrate the flexibility and adaptiveness of the learning methods used in SINS. Regardless of parameter settings, SINS was very robust and continued to be able to navigate successfully despite a sudden change in environmental clutter. After the 200th world, it continued to solve 100% of the worlds presented to it, with only modest deterioration in steps, distance, virtual collisions, and time in the more cluttered environments. The performance of the static systems, in contrast, deteriorated in the more cluttered environments, with the exception of static50 which was designed for such environments. The static50 started to improve in 50% worlds as compared to the 15% worlds where it was not performing as well as static15. As can be seen from the graphs, SINS performed as well as static15 in the first 200 worlds, and then as well as static50 in the next 200 worlds; furthermore, it achieved this level of performance even when it was initialized with non-optimized schema parameters. This result also suggests that the regularities that SINS captures in its cases encode strategic knowledge for navigation that is generally applicable across different types of environments. The performance of the random system does not deteriorate too badly but it was not the best in either of the two types of worlds.

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** = 20, **max-size** = 15, and **control-interval** = 4, 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."

Although SINS integrates reinforcement learning and case-based reasoning, its algorithms are somewhat different from the standard algorithms used in these areas. One of the main differences between traditional reinforcement learning and SINS is with respect to the action model used by the system. While in reinforcement learning the action model is given along with the definition of the system, in SINS it is learned through experience. One open issue in this respect is how to use this action model once it has been learned or while it is in the process of being learned. In SINS, the action model is represented as sequences of associations. In our current implementation, the system always uses the sequence of associations most similar to the current environment, but other implementations may select different sequences according to other criteria. One such criterion we are currently exploring is selecting the sequence of associations that is likely to result in a desired environmental configuration. This would enable SINS to behave in a goal-oriented fashion while still staying within a reactive framework (see also Maes, 1990).

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, in press) 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. The system benefits from the tight integration of multiple learning strategies that support and complement each other, and the on-line use of the learned knowledge in guiding the performance task.

Acknowledgements

We would like to thank Ron Arkin for his contributions to this research and for providing the AuRA robot simulator, Paul Rowland for his help with replicating the results presented in this paper, and George Tecuci for his comments on an early draft of this paper.

References

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

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

[3] S.A. Chien, M.T. Gervasio, and G.F. DeJong, *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).

[4] R.E. Fikes, P.E. Hart, and N.J. Nilsson, *Learning and Executing Generalized Robot Plans*. Artificial Intelligence, 3, 251–288, (1972).

[5] M. Georgeff, *Planning*. Annual Review of Computer Science, 2, 359–400, (1987).

[6] J.J. Grefenstette and C.L. Ramsey, *An Approach to Anytime Learning*. In Sleeman, D. and Edwards, P. (editors), Machine Learning: Proceed-

ings of the Ninth International Conference, 189–195, Aberdeen, Scotland, (1992).

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

[8] L. Kaelbling, *An Architecture for Intelligent Reactive Systems*, Technical Note 400, SRI International, (1986).

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

[10] P. Maes, *Situated Agents can have Goals*. Robotics and Autonomous Systems, 6, 49–70, (1990).

[11] S. Minton, *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).

[12] T.M. Mitchell, *Becoming Increasingly Reactive*. In Proceedings of the Eighth National Conference on Artificial Intelligence, 1051–1058, Boston, MA, (1990).

[13] J. Mostow and N. Bhatnagar, *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).

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

[15] M. Pearce, R. Arkin, and A. Ram, *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).

[16] A. Ram, *Creative Conceptual Change*. In Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society, 17–26, Boulder, CO, (1993).

[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, Georgia, (1992).

[18] A. Ram and J.C. Santamaría, *Continuous Case-Based Reasoning*. In Leake, D.B. (editor), Proceedings of the AAAI Workshop on Case-Based Reasoning, AAAI Press Technical Report WS-93-01, 86–93, Washington, DC, (1993).

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

[20] A.M. Segre, *Machine Learning of Robot Assembly Plans*, Kluwer Academic Publishers, Norwell, MA, (1988).

[21] R.S. Sutton, *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).

[22] R.S. Sutton (editor), *Special Issue on Reinforcement Learning*. Machine Learning, 8(3/4), (1992).

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

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

[25] S.D. Whitehead and D.H. Ballard, *Active Perception and Reinforcement Learning*. In Proceedings of the Seventh International Conference on Machine Learning, 179–188, Austin, TX, (1990).