Learning Behavioral Parameterization Using Spatio-Temporal Case-Based Reasoning*

Maxim Likhachev, **Michael Kaess**, **Ronald C. Arkin**
Mobile Robot Laboratory
College of Computing, Georgia Institute of Technology
maxim+@cs.cmu.edu, kaess@cc.gatech.edu, arkin@cc.gatech.edu

Abstract
This paper presents an approach to learning an optimal behavioral parameterization in the framework of a Case-Based Reasoning methodology for autonomous navigation tasks. It is based on our previous work on a behavior-based robotic system that also employed spatio-temporal case-based reasoning [3] in the selection of behavioral parameters but was not capable of learning new parameterizations. The present method extends the case-based reasoning module by making it capable of learning new and optimizing the existing cases where each case is a set of behavioral parameters. The learning process can either be a separate training process or be part of the mission execution. In either case, the robot learns an optimal parameterization of its behavior for different environments it encounters. The goal of this research is not only to automatically optimize the performance of the robot but also to avoid the manual configuration of behavioral parameters and the initial configuration of a case library, both of which require the user to possess good knowledge of robot behavior and the performance of numerous experiments. The presented method was integrated within a hybrid robot architecture and evaluated in extensive computer simulations, showing a significant increase in the performance over a non-adaptive system and a performance comparable to a non-learning CBR system that uses a hand-coded case library.

Index terms: *Case-Based Reasoning, Behavior-Based Robotics, Reactive Robotics.*

I. INTRODUCTION

Behavior-based control for robotics is known to provide good performance in unknown or dynamic environments. Such robotic systems require little a priori knowledge and are very fast in response to changes in the environment, as they advocate a tight coupling of perceptual data to an action. At any point of time, based on incoming sensory data, a robot selects a subset of behaviors (called a behavioral assemblage) from the set of predefined behaviors and then executes them. One of the problems, however, of this approach is that as the surrounding environment gradually changes, the parameterization of the selected behaviors should also be adjusted correspondingly. Using a constant, non-adaptive, parameterization for most of the non-trivial cases results in robot performance far from being optimal. Also, choosing the "right" set of parameters even in the case of constant parameterization is a difficult task requiring both knowledge of robot behaviors and a number of preliminary experiments. It is desirable to avoid this manual configuration of behavioral parameters in order to make mission specification as user-friendly and rapid as possible.

As one of the possible solutions to these problems, we previously proposed the use of Case-based Reasoning methodology for the automatic selection of behavioral parameters [3] for navigational tasks. This approach resulted in significant robot performance improvement and also made the manual configuration of parameters unnecessary. This method, however, required manual creation of a case library, where each case is indexed by environmental features and defines a set of behavioral parameters. Even though the creation of the library of cases needed to be performed only once for a given robot architecture, it is still very tedious work.

The idea behind this new research is to fully automate the creation of a library of cases, so that the case-based reasoning (CBR) unit automatically creates and optimizes cases in the library as the result of an automatic experimental procedure. At the training stage the system starts off with a completely empty library. As training proceeds new cases are created whenever there are no close enough matches already present in the library. Whenever a case is selected for the application, it goes through an adaptation step that, based on the previous performance of the case, adapts the case in the direction that resulted in the performance improvement. The case is then applied and its performance is re-evaluated and stored for the use by the adaptation routine the next time the case is selected. Thus, in effect, the CBR unit performs a gradient descent search in the space of behavioral parameters for each of the cases in the library. Once the training is over and a robot exhibits a good performance in the training session, the library is "frozen" and can be used in real missions. It is important to note, however, that the training does not need to be separated from actual mission execution. That is, an alternative to the above procedure is to have the robot learn the cases as it executes missions. In this case, even though

* This research is supported by DARPA/U.S. Army SMDC contract #DASG60-99-C-0081. Approved for Public Release; distribution unlimited.

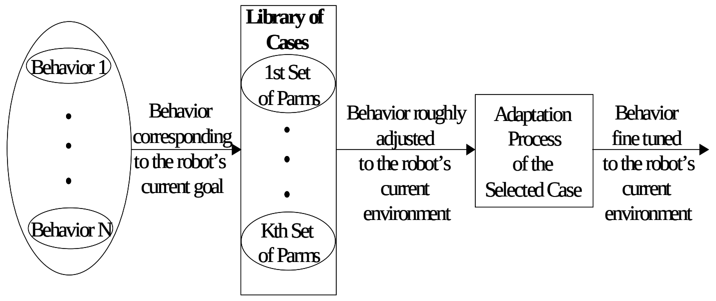
Behavior 1
Behavior N
Behavior corresponding to the robot's current goal
Library of Cases
1st Set of Parms
Kth Set of Parms
Behavior roughly adjusted to the robot's current environment
Adaptation Process of the Selected Case
Behavior fine tuned to the robot's current environment

Figure 1. Behavioral selection process with case-based reasoning unit incorporated.

the performance in the first robot missions would be far from optimal, as more and more missions are executed the robot's performance would consistently improve.

In section III, we demonstrate how a simulated robot learns new cases during training and compare its performance with both non-adaptive and non-learning CBR systems. The results show that the learning system significantly outperforms a non-adaptive system while it is comparable to a non-learning CBR system for which the library of cases was manually created as a result of extensive experimentation. Moreover, if the library of cases for the non-learning CBR system was not well optimized for actual environments that are similar to the test environments then the learning CBR system outperforms even the non-learning CBR as well.

A case-based reasoning methodology is not new to the field of robotics. It was successfully used to help in solving such problems as path planning based on past routes, high-level action-selection based on environment similarities, place learning, and acceleration of complex problem solving based on past problem solutions [4, 5, 6, 7, 8, 14]. Previous work has also been performed on the incorporation of case-based reasoning in the selection of behavior parameters by our group [1, 2, 3] (upon which this present research is partially based) as well as a few other groups [e.g., 13]. The approach described in this paper extends our previous work by learning new cases from scratch either during a training step or while executing a mission. The method has also been incorporated within a complete hybrid robot architecture and extensively evaluated in computer simulations. Extensive research has also been conducted on learning robot behaviors using other methods such as neural networks, genetic algorithms, Q-learning, and others [15, 16, 17]. In contrast to some of these methods, this research concentrates on the automatic learning of an optimal parameterization of the behaviors rather than the behaviors themselves. It also incorporates some prior knowledge using the case-based reasoning methodology during learning and thus decreases the number of experiments required for obtaining a good behavioral parameterization function as defined by a library of cases.

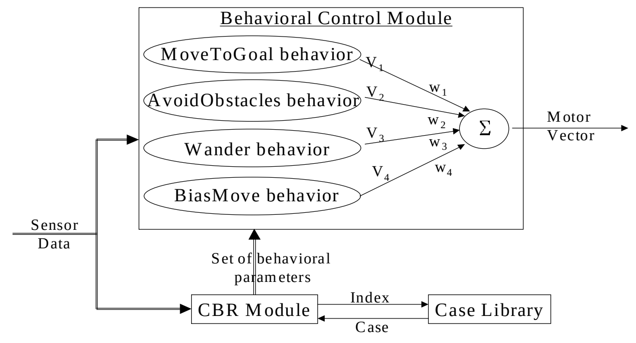

Figure 2. Interaction between behavioral control module running a GOTO behavioral assemblage and CBR unit.

II. Methodology

A. Framework

The CBR unit operates within the *MissionLab* system [9], which is a version of AuRA (Autonomous Robot Architecture) [10]. It is a hybrid architecture that consists of a schema-based reactive system coupled with a high-level deliberative planning system. The reactive component consists of primitive behaviors called motor schemas [11] grouped into sets called behavioral assemblages. Each individual primitive behavior is driven by its perceptual input (perceptual schema) producing its own motor response. The vectorial responses from each of the active schemas are added together resulting in an overall behavior output. The weighted sum of the vectors, after normalization, defines the final vector that is sent to the motor actuators. Hence, each motor schema affects the overall behavior of the robot.

It is the reactive level that the CBR unit operates at. In particular, it selects the set of parameters for a currently chosen behavioral assemblage that is best suited to the current environment. As the robot executes its mission, the CBR unit controls the switching between different sets of behavioral parameters in response to changes in the environmental type. Each such set of parameters constitutes a case in the CBR system and is indexed by environmental features. The adaptation step in the case-based reasoning subsequently fine-tunes the parameters to a specific type of environment, allowing the library of cases to be small. The overall control flow is shown in Figure 1.

The CBR unit was evaluated on the behavioral assemblage of type **GOTO** that is used for goal-directed navigation (Figure 2). The assemblage contains the following four primitive motor schemas: **MoveToGoal**, **Wander**, **AvoidObstacles** and **BiasMove**. The **MoveToGoal** schema produces a vector directed towards a specified goal location from the robot's current position. The **Wander** schema generates a random direction vector, adding an exploration component to the robot's behavior. The **AvoidObstacles** schema produces a vector repelling the robot from all of the obstacles that lie within some given distance from the robot. The **BiasMove** schema produces a vector in a certain direction in order to bias the motion behavior of the robot. The CBR module controls the following parameters:

<Noise_Gain, Noise_Persistence,
Obstacle_Sphere Obstacle_Gain,
MoveToGoal_Gain, Bias_Vector_Gain,
Bias_Vector_X, Bias_Vector_Y >

The gain parameters are the multiplicative weights of the corresponding schemas. The *Noise_Persistence* parameter controls the frequency with which the random noise vector changes its direction. *Obstacle_Sphere* controls the distance within which the robot reacts to obstacles with the **AvoidObstacles** schema. *Bias_Vector_X* and *Bias_Vector_Y* specify the direction of the vector produced by **BiasMove** schema. Thus, a case in a library is a set of values for the above parameters.

B. Overview of non-learning CBR module

This section gives a high-level overview of the non-learning CBR module. The technical details can be found in [3]. The following section then provides the details of the extensions that were made to the CBR module that make it capable of both learning new and optimizing old cases (the learning CBR module).

The overall structure of the non-learning CBR unit is similar to a traditional non-learning case-based reasoning system [5] (Figure 3). First, the sensor data and goal information are supplied to the Feature Identification sub-module of the CBR unit. This sub-module computes a spatial features vector representing the relevant spatial characteristics of the environment and a temporal features vector representing relevant temporal characteristics. The spatial feature vector is used to compute a *traversability* vector *F* of size *k*. A space around the robot is broken into k equal angular regions and each element of the vector *F* represents the degree to which the corresponding region can be traversed. The number of angular regions is configurable and depends on the type of sensors used (i.e., it is resolution-limited) and the allowed computational overhead. For all the experiments reported here, there were four angular regions used. The details of how the *traversability* vector is computed appear in [3]. In short, the traversability of a particular region decreases as the size of the largest cluster of sensor readings (readings appearing sufficiently close to each other) in this region increases, and increases as the distance between the robot and this cluster increases. The temporal features vector consists of just two elements: short-term and long-term robot velocities normalized by the robot maximum velocity. Both spatial

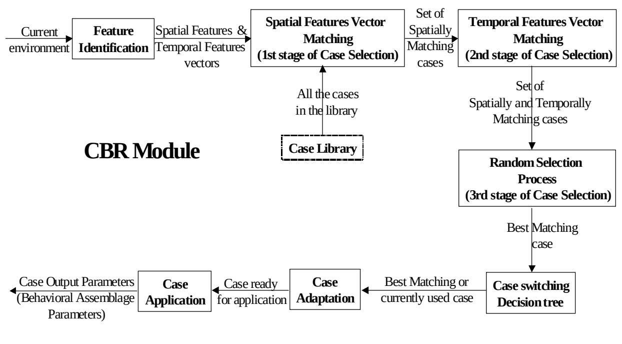
Current environment
Feature Identification
Spatial Features & Temporal Features vectors
Spatial Features Vector Matching (1st stage of Case Selection)
Set of Spatially Matching cases
Temporal Features Vector Matching (2nd stage of Case Selection)
All the cases in the library
CBR Module
Case Library
Set of Spatially and Temporally Matching cases
Random Selection Process (3rd stage of Case Selection)
Best Matching case
Case Output Parameters (Behavioral Assemblage Parameters)
Case Application
Case ready for application
Case Adaptation
Best Matching or currently used case
Case switching Decision tree

Figure 3. High-level structure of the CBR Module

and temporal vectors are passed forward for a best matching case selection.

Case selection is done in three steps. During the first stage of case selection, all the cases from the library are searched, and the weighted Euclidean distances between their *traversability* vectors (derived from their spatial feature vectors) and the environmental *traversability* vector are computed. These distances define spatial similarities of cases with the environment. The case with the highest spatial similarity is *the best spatially matching* case. However, all the cases with a spatial similarity within some delta from the similarity of the best spatially matching case are also selected for the next stage selection process. The resulting set consists of the *spatially matching* cases. At the second stage of selection, all of the spatially matching cases are searched, and the weighted Euclidean distances between their *temporal* feature vectors and the environmental temporal feature vector are computed. These distances define temporal similarities of cases with the environment. The case with the highest temporal similarity is deemed *the best temporally matching* case. Again, all the cases with a temporal similarity within some delta from the similarity value of the best temporally matching case are selected for the next stage in the selection process. These cases are *spatially and temporally matching* cases, and they are all the cases with close spatial and temporal similarity to the current environment. This set usually consists of only a few cases and is often just one case, but it is never empty. At the last selection stage, a case from the set of spatially and temporally matching cases is selected at random. Randomness in case selection is introduced in order to exercise the exploration of cases with similar features but different output parameters.

The case switching decision tree is then used to decide whether the currently applied case should still be applied or the case selected as the best matching one shoould be used instead. This protects against thrashing and overuse of cases. If the selected case is to be applied, then it goes through the adaptation and application steps. At the adaptation step, a case is fine-tuned by slightly readjusting the behavioral assemblage parameters contained in the case to better fit the current environment. At the application step these parameters are passed to the behavioral control module, which uses these parameters in the evaluation of the current behavioral assemblage.

C. Learning CBR module

This section provides an overview of the learning CBR module (Figure 4), emphasizing the extensions that were made to the non-learning CBR algorithm described previously. First, as before the sensor data and goal information are provided to the Feature Identification sub-module that operates identically to the Feature Identification sub-module in the non-learning CBR module. The resulting spatial and temporal feature vectors are then passed to the best matching case selection process.

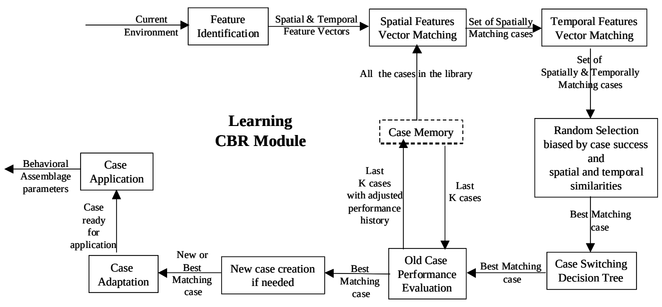
Current Environment
Feature Identification
Spatial & Temporal Feature Vectors
Spatial Features Vector Matching
Set of Spatially Matching cases
Temporal Features Vector Matching
Set of Spatially & Temporally Matching cases
All the cases in the library
Learning CBR Module
Case Memory
Random Selection biased by case success and spatial and temporal similarities
Behavioral Assemblage parameters
Case Application
Last K cases with adjusted performance history
Last K cases
Best Matching case
Case ready for application
New or Best Matching case
Best Matching case
Best Matching case
Case Adaptation
New case creation if needed
Old Case Performance Evaluation
Case Switching Decision Tree

Figure 4. High-level structure of the Learning CBR Module.

1) *Case Selection*

As before, at the spatial case selection step the spatial similarity between each case in the library and the current environment is computed as the weighted Euclidean distance between the case and environmental *traversability* vectors. Now, however, instead of selecting all the cases that have a spatial similarity within some delta from the similarity of the best spatially matching case, the cases are selected at random, with their probability of being selected proportional (according to a Gaussian function) to the difference between their spatial similarity and the spatial similarity of the best spatially matching case. Figure 5 illustrates this case selection process. Case C_1 is the best spatially matching case and has a 100 percent probability of being selected to the set of spatially matching cases. Cases C_2 and C_4 are also selected as a result of random case selection biased by their spatial similarities. The idea behind adding such randomness to the case selection process is to bias the exploration of cases by their similarities with the environment. Similarly, at the temporal case selection stage, the cases that were selected as *spatially matching cases* go through the random selection with the probability of being selected biased by the differences between their temporal similarity and the temporal similarity of the best temporally matching case. Thus, in the example in Figure 5 case C_4 is the best temporally matching case and therefore is selected for the next selection step. Case C_1 is also selected at random for the next selection step whereas C_2 is not. The cases that pass these two selection stages are also called *spatially and temporally matching* cases and are forwarded to the last case selection stage.

At the last selection step just one case is selected at random with a probability of being selected proportional to the weighted sum of case spatial similarity, temporal similarity and case success. The case success is a scalar value that reflects the performance of the case, and is described below. Thus, for the example shown in Figure 5, C_1 has a higher weighted sum of spatial and temporal similarities and success, and therefore has a higher chance of being selected than C_4. In this particular example, C_1 is indeed selected as the *best matching case*.

Once the case is selected, as before, the case switching decision tree decides whether to continue to use the currently applied case or switch onto the selected best matching case. If the switching decision tree says that the currently applied case should remain active, then nothing else needs to be done in this cycle of the CBR unit. Otherwise, the CBR unit continues its execution with the evaluation of the currently applied case performance.

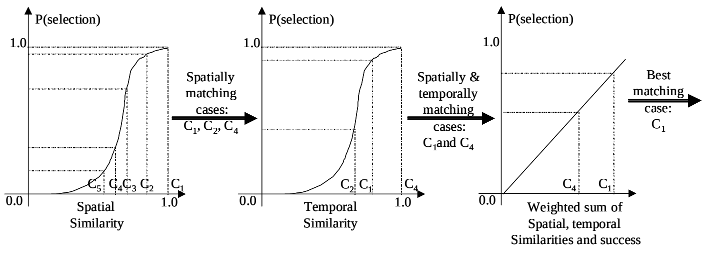

Figure 5. Case selection process.

2) *Old Case Performance Evaluation*

The velocity of the robot relative to the goal (the speed with which the robot is approaching its goal) is used as the main criteria for the evaluation of case performance. The pseudocode for the case (C) performance evaluation is given below:

```
Compute velocity $V(C)$ according to Equation (1)
If ($V(C) \le 0$ and C was applied last)
   //delayed reinforcement
   Postpone the evaluation of C until another K-1 cases are applied or C is selected for application (whichever comes first)
else
   if ($V(C) > \mu \cdot V_{max}(C)$ and $V(C)>0$)  //$\mu=0.9$
      $I(C) = \max(1, I(C) + 1)$;
   else
      $I(C) = I(C) - 1$;
   end
   $I(C)=\min(I_{max}, I(C))$;  //limit $I(C)$;$I_{max}=100$
   Update $V_{max}(C)$ according to Equation (2)
   if(C was applied last)
      if($V(C) > \mu \cdot V_{max}(C)$ and $V(C)>0$)
         Increase $S(C)$ by $\Delta$ proportional to $I(C)$;
      else
         Decrease $S(C)$ by $\Delta$;
      end
   else
      if (Robot advanced towards its goal)
         Increase $S(C)$ by $\Delta$ proportional to $I(C)$;
      else
         Decrease $S(C)$ by $\Delta$;
      end
   end
end
```

Since for some cases the task is to get the robot closer to the goal, while for other cases the task is to get the robot out of local minima such as “box canyons” created by obstacles, the robot's velocity relative to the goal may not always be the best evaluation function for case performance. Instead, a delayed evaluation of the case performance may be necessary. For this reason the information on the last K applied cases is kept. K defines a learning horizon and in this work is chosen to be 2. Thus, when a new case is about to be applied, the performance evaluation function is called on each of the following cases:

the case that was applied last; the case that was applied K cases ago and was not yet evaluated (the evaluation was postponed); and the case that was applied some time previously, that was not yet evaluated and is the very case selected for a new application. At the very beginning of the performance evaluation a check is done: if a case C was just applied and the robot did not advance towards its goal as a result of the case application ($V(C) \leq 0$, where *V(C)* denotes the average velocity of the robot relative to its goal from the time just before case C was applied up until the current time), then the case performance evaluation is postponed. Otherwise, the performance evaluation proceeds further.

Each case has a number of variables that represent the recent performance of the case and need to be updated in the performance evaluation routine. The average velocity of the robot relative to the goal for case C is computed as given:

$$V(C) = \frac{g_{t_b(C)} - g_{t_{curr}}}{t_{curr} - t_b(C)} \qquad (1)$$

where $t_b(C)$ is the time before the application of case C, t_{curr} is the current time and g_t is the distance to the goal at time t. One of the variables maintained by each case describing case performance is $V_{max}(C)$: the maximum average velocity of the robot relative to the goal as a result of the application of case C. This velocity is updated after every performance evaluation of case C. Equation 2 is a form of "maximum tracker" in which $V_{max}(C)$ very slowly decreases whenever it is larger than V(C) and instantaneously jumps to V(C) whenever $V_{max}(C)$ is smaller than V(C):

$$V_{\max}(C) = \max(V(C), \eta \cdot V_{\max}(C) + (1-\eta) \cdot V(C)) \qquad (2)$$

where η is a large time constant (0.99 for this work).

However, before $V_{max}(C)$ is updated a decision is made on whether the case resulted in performance improvement or not. The performance is considered to improve if $V(C) > \mu \cdot V_{max}(C)$ and $V(C) > 0$, where μ is close to 1. Thus, the case performance is considered to be an improvement not only when the velocity is higher than it has ever been before but also when the high velocity is reasonably sustained as a result of the case's application. The variable *I(C)* maintains the number of the last case performance improvements and is used in the adaptation step to search for the adaptation vector direction.

Finally, the case success, *S(C)*, is also updated. If the performance evaluation is not postponed, then the case success is increased if the case performance improved (the performance improvement is defined by the same formula as before) and is decreased otherwise. If, however, the case evaluation was postponed, then the case success is increased if the robot advanced sufficiently towards its goal after the case was applied and is decreased if the robot has not advanced at all. In either case the increase in the case success is proportional to the number of times the application of the case resulted in its performance improvement, *I(C)*. This adds momentum to the convergence of case success. The more there are recent case improvements, the faster the case success approaches its maximum value (1.0) indicating full convergence of the case. The case success is used in case selection to bias the selection process, and in case adaptation to control the magnitude of the adaptation vector. It will be discussed further below.

3) *Case Creation Decision*

At this step, a decision is made whether to create a new case or keep and adapt the case that was selected for the application. This decision is made based on the weighted sum of the temporal and spatial similarities of the selected case with the environment and on the success of the selected case. If the success of the selected case is high then it must be very similar to the environment, mainly spatially, in order for this case to be adapted and applied. This prevents making the case success diverge based on environments that do not correspond to the case. If the case success is low, then the case similarity may not be very close to the environment and still the case is adapted and applied. In any event, the size of the library is limited (for this work a limit of 10 cases was used) and therefore if the library is already full then the selected case is adapted and applied.

If it is decided that a new case should be created, then the new case is initialized with the same output parameters (behavioral parameters) as the selected case but input parameters (spatial and temporal feature vectors) are initialized to the spatial and temporal feature vectors of the current environment. The new case is saved to the library and then passed to the adaptation step. If no new case is created then the selected case is passed directly to the adaptation step.

4) *Case Adaptation*

Independent of whether the case to be applied is an old case or was just created, the case still goes through the adaptation process. Every case C in the library also maintains an adaptation vector, ***A****(C)* that was last used to adapt the case output parameters. If the case was just created then the adaptation vector is set to a randomly generated vector. The adaptation of a case happens in two steps. First, based on the case's recent performance, the adaptation vector is used to adapt the case C output parameter vector, ***O****(C)*:

```
if ($I(C) \leq 0$)
    //change the adaptation direction
    $\mathbf{A}(C) = -\lambda \cdot \mathbf{A}(C) + \nu \cdot \mathbf{R}$;
end
//adapt
$\mathbf{O}(C) = \mathbf{O}(C) + \mathbf{A}(C)$;
```

If the case improvement *I(C)* does not show evidence that the case was improved by the last series of adaptations, then the adaptation vector direction is reversed, decreased by a constant λ and a randomly generated vector

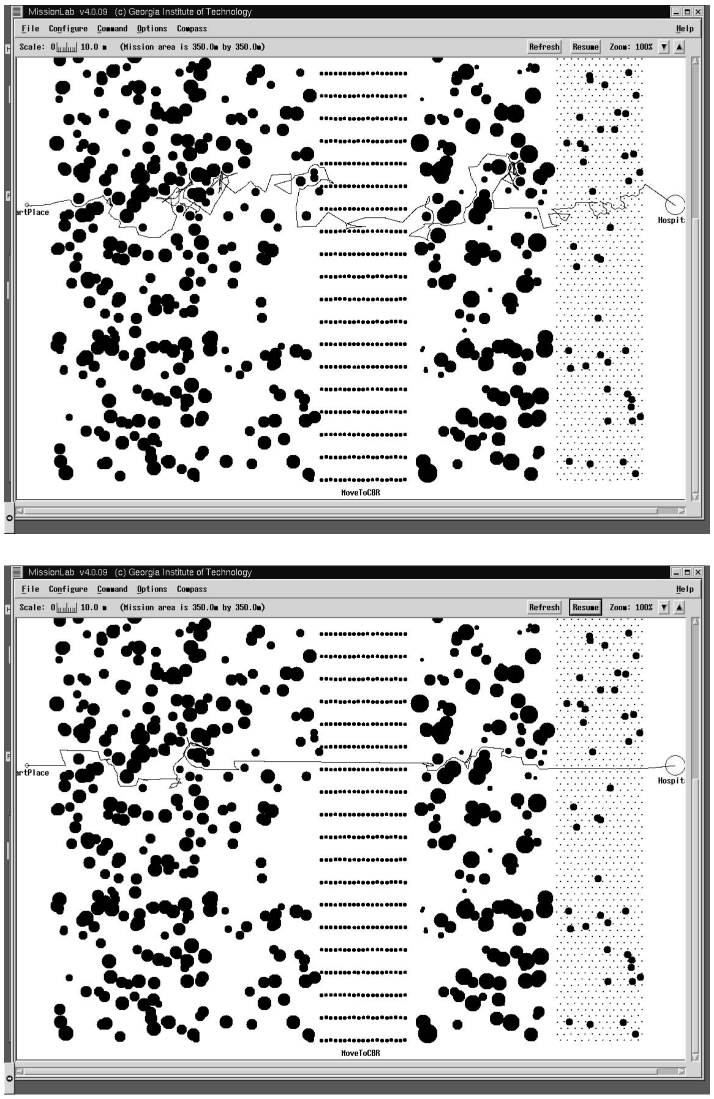
MissionLab v4.0.09 (c) Georgia Institute of Technology
File Configure Command Options Compass Help
Scale: 0 10.0 m (Mission area is 350.0m by 350.0m)
Refresh Resume Zoom: 100%
rtPlace
Hospit
MoveToCBR

Figure 6. Screenshots of training runs in a heterogeneous environment. Top: initial run that starts off with an empty library; Bottom: a run after fifty-four training runs.

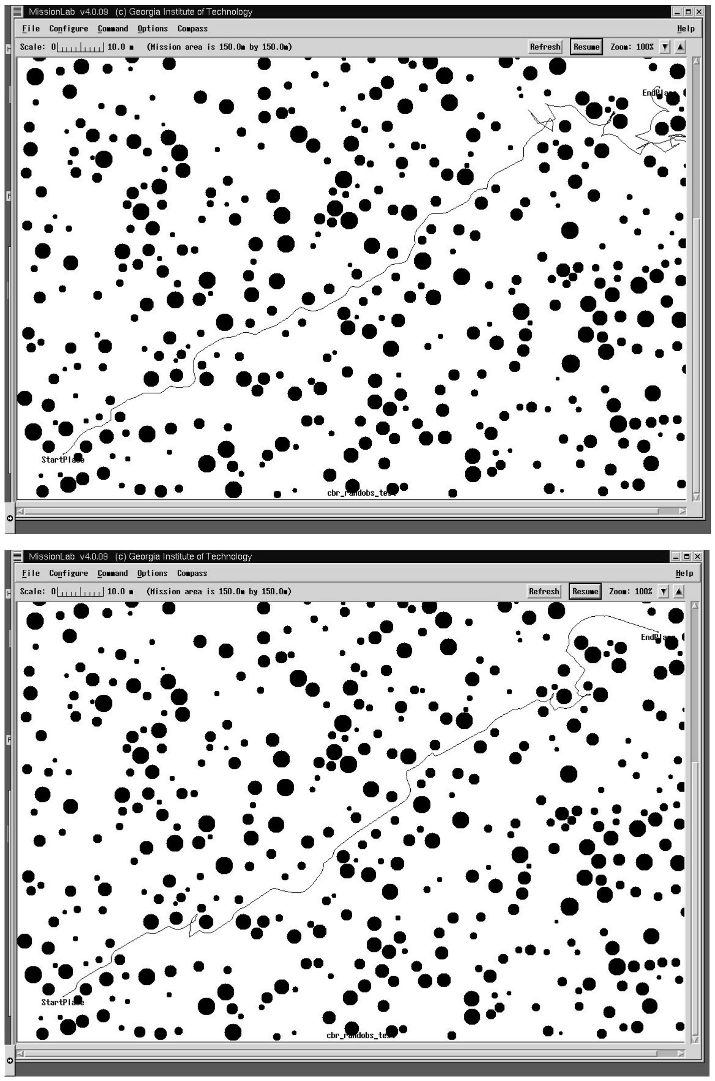
MissionLab v4.0.09 (c) Georgia Institute of Technology
File Configure Command Options Compass Help
Scale: 0 10.0 m (Mission area is 150.0m by 150.0m)
Refresh Resume Zoom: 100%
StartPlace
EndPlace
cbr_randobs_test

Figure 7. Screenshots of training runs in a homogeneous environment. Top: initial run that starts off with an empty library; Bottom: a run after fifty training runs.

R scaled by a constant v is added to assure exploration in the search for optimal parameters.

At the second adaptation step, the output parameters are altered based on the short-term and long-term relative velocities of the robot (elements of the temporal features vector). This adaptation step is similar to the adaptation step performed in the non-learning CBR module [3] and, in short, increases the *Noise_Gain* and *Noise_Persistence* behavioral parameters inverse proportionally to the short-term and long-term relative velocities of the robot. The idea is that these two parameters are increased more and more if the robot is stuck longer and longer at one place (such can be the case with difficult "box canyons").

Finally, the behavioral parameters of the case are limited by their corresponding bounds. Also, *Obstacle_Gain* is limited from below by the sum of *Noise_Gain*, *MoveToGoal_Gain* and *Bias_Vector_Gain*. This ensures that in any event the robot does not collide with obstacles.

After the case is adapted it is applied. The application is simply extracting the behavioral assemblage parameters from the adapted case and passing them to the behavioral control unit within the *MissionLab* system.

III. Experiments

A. Experimental Analysis

The performance of the system was evaluated in a simulated environment. *MissionLab* provides a simulator as well as logging capabilities, allowing the collection of the required statistical data easily.

Figures 6 and 7 demonstrate the training process of a robot. In Figure 6 the training is done on heterogeneous environments (the obstacle density and pattern change within one robot mission), whereas in Figure 7 the training is done on homogeneous environments (the obstacle density

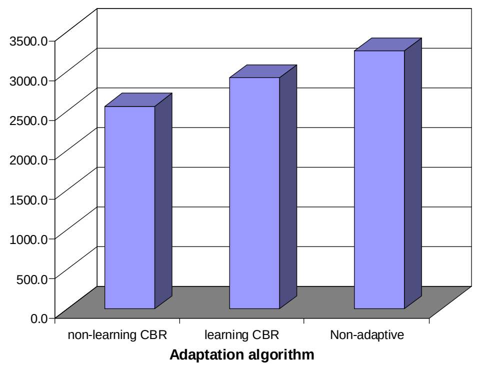

Figure 8. Average number of steps of a simulated robot in heterogeneous environments

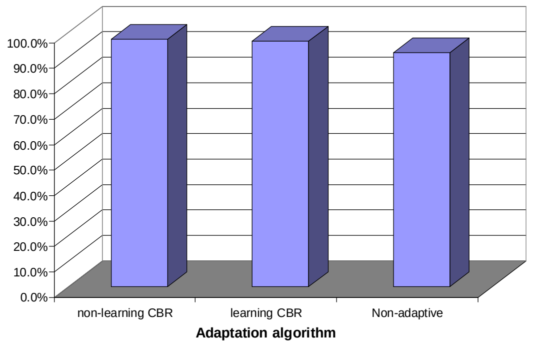

Figure 9. Mission completion rate of a simulated robot in heterogeneous environments

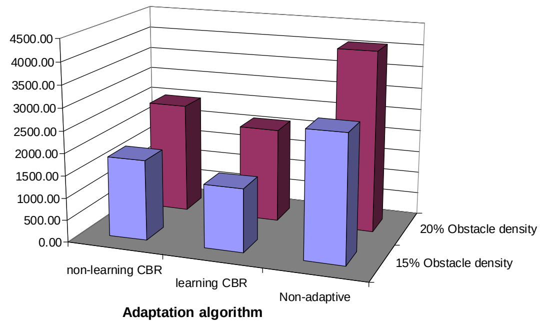

Figure 10. Average number of steps of a simulated robot in homogeneous environments

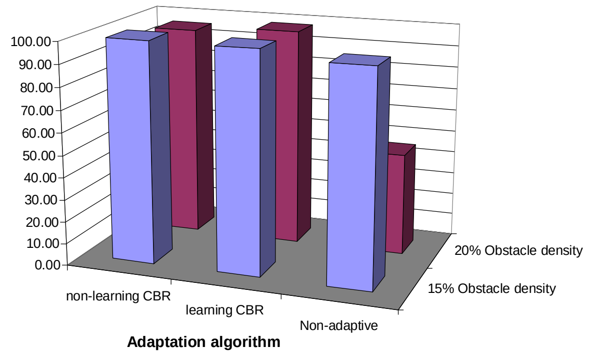

Figure 8. Mission completion rate of a simulated robot in homogeneous environments

and pattern remain constant throughout a robot mission). These were two separate training processes resulting in two different learned libraries. Figures 6(top) and 7(top) show screenshots of *MissionLab* during the first robot runs. At the beginning of both runs the libraries do not contain any cases and are created as the robot proceeds with its mission. Black dots of various sizes represent obstacles and the curved line across the picture depicts the trajectory of the robot after it completed its mission. In Figure 6 the mission area is 350 by 350 meters, whereas in Figure 7 it is 150 by 150 meters. Since the library is being created from scratch, the performance of the robot in these first runs is very poor. The search for optimal parameterization has just started in these runs and thus the robot behavior is very noisy. In contrast, after about fifty training runs for both heterogeneous and homogeneous environments, the robot successfully learned more optimal parameterizations and therefore the robot trajectory in final runs (Figures 6 and 7 bottom) is far better. A good example of learning an optimal parameterization is in the last (rightmost) grid of small obstacles in the heterogeneous environment. In order for a robot to traverse such a dense but highly ordered obstacle environment the robot has to apply what is called a "squeezing" strategy. In this strategy *Obstacle_Sphere* is decreased to its minimum while *MoveToGoal_Gain* has to prevail over *Noise_Gain*. This makes the robot squeeze between obstacles towards its goal. In the first run, this strategy is not known to the robot and it takes a long time for the robot to go through this area. In contrast, in Figure 6 (bottom) the robot successfully "squeezes" through this area along a straight line. The log files show that the robot trajectory in the final run in the heterogeneous environment is 36 percent shorter than in the initial run while in the homogenous environment the final run is 23 percent shorter than the initial run.

B. Experimental Results

Figures 8 through 11 show the statistical data gathered in the simulations. Three systems were evaluated: non-adaptive which did not use any adaptation of behavioral parameters; a system that employed the non-learning CBR module for the adaptation of behavioral parameters; and a system with the learning CBR module. Cases for the non-learning CBR module were created manually by running preliminary experiments to configure them optimally. Three libraries for the learning CBR system were created automatically by running about 250 training runs. All three libraries were evaluated and the data in the graphs contains average values over the three libraries. For the runs with the non-adaptive system, the optimal set of parameters was chosen for a given average obstacle density. This was equivalent to a user specifying the optimal parameters for a given mission.

Figures 8 and 9 show the performance of a simulated robot on a navigational task in heterogeneous environments (such as the one shown in Figure 6). Overall, the results for 37 missions in different heterogeneous environments were gathered. The performance of a robot is represented by the time steps that it takes a robot to complete its mission, as well as the percentage of completed missions. Thus, the amount of time, on average, it takes the learning CBR system to complete a mission is better than for a non-adaptive system while worse than for a non-learning one. This result is expected as the library for the non-learning CBR system was manually well optimized on the set of heterogeneous environments. The mission completion rate (Figure 9) is about equal for both non-learning and learning CBR systems. The non-adaptive system has the lowest mission success rate.

Figures 10 and 11 report the results of tests in homogeneous environments such as the one shown in Figure 7. In each of the figures, the front row is for an environment with a 15% obstacle density and the back row is for an environment with 20% obstacle density. For each environment, fifty runs were conducted for each algorithm to establish statistical significance of the results. In these tests, a system that employs learning CBR outperforms even the non-learning CBR system not to mention the non-adaptive one. This is true in terms of both criteria: the average mission execution time and mission success rate. The reason for this is that even though the non-learning CBR system performs very well in homogeneous environments it was manually optimized using a sub-set of environments used for heterogeneous environment tests. As a result, the learning CBR had an opportunity to learn cases that were better suited for the homogeneous environments than the ones that were in the library of the non-learning CBR module. Non-adaptive, on the other hand, performs far from optimally on these environments and even more importantly exhibits only 46 percent mission completion rate for denser environments (Figure 11, 20% density).

IV. Conclusion

This paper presents a robotic system that incorporated learning into a previously developed case-based reasoning module used for the selection of behavioral parameters. Not only does it significantly improve the performance of the robot in comparison to a non-adaptive system but it also potentially improves the performance over a non-learning CBR module if its library was not well optimized for test environments, as was shown in the experiments. Automatic learning of cases is also favored as the process of manually creating a CBR case library is tedious and requires knowledge of both robot behavior and the operation of the CBR module as well as numerous experiments. The case library also had to be manually re-configured every time a new robot architecture is targeted. In contrast, with the learning CBR module the process of library configuration is fully automatic, namely through training. This now makes unnecessary any configuration of behavioral parameters even to create an initial case-based reasoning library. Moreover, the robot can learn cases while executing its missions, even avoiding the automatic training process if the accompanying performance deterioration in the initial first missions is acceptable. As more and more missions are executed the better and better the parameterization becomes, resulting in enhanced robot performance.

Future work includes the evaluation of the system on real robots (Nomad 150s and ATRV-JRs). This presented research is part of a larger project involving the incorporation of a range of different learning techniques into *MissionLab*. It is planned to investigate how such multiple techniques can help in learning cases more efficiently.

V. Acknowledgments

The authors of the paper would like to thank the following people who were invaluable in the work with *MissionLab*: Dr. Douglas MacKenzie, Yoichiro Endo, William Conrad Halliburton, Mike Cramer, Alexander Stoytchev, and Dr. Tom Collins.

REFERENCES

[1] A. Ram, R. C. Arkin, K. Moorman, and R. J. Clark, "Case-based Reactive Navigation: a Method for On-line Selection and Adaptation of Reactive Robotic Control Parameters," *IEEE Transactions on Systems, Man and Cybernetics - B*, 27(30), pp. 376-394, 1997.

[2] A. Ram, J. C. Santamaria, R. S. Michalski and G. Tecuci, "A Multistrategy Case-based and Reinforcement Learning Approach to Self-improving Reactive Control Systems for Autonomous Robotic Navigation," *Proceedings of the Second International Workshop on Multistrategy Learning*, pp. 259-275, 1993.

[3] M. Likhachev and R. C. Arkin, "Spatio-Temporal Case-Based Reasoning for Behavioral Selection," *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, pp. 1627-1634, 2001.

[4] C. Vasudevan and K. Ganesan, "Case-based Path Planning for Autonomous Underwater Vehicles," *Autonomous Robots*, 3(2-3), pp. 79-89, 1996.

[5] M. Kruusmaa and B. Svensson, "A Low-risk Approach to Mobile Robot Path Planning," *Proceedings of the 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, 2, pp. 132-141, 1998.

[6] P. Gugenberger, J. Wendler, K. Schroter, H. D. Burkhard, M. Asada, and H. Kitano, "AT Humboldt in RoboCup-98 (team description)," *Proceedings of the RoboCup-98*, pp. 358-363, 1999.

[7] M. M. Veloso and J. G. Carbonell, "Derivational Analogy in PRODIGY: Automating Case Acquisition, Storage, and Utilization," *Machine Learning*, 10(3), pp. 249-278, 1993.

[8] S. Pandya and S. Hutchinson, "A Case-based Approach to Robot Motion Planning," *1992 IEEE International Conference on Systems, Man and Cybernetics*, 1, pp. 492-497, 1992.

[9] D. Mackenzie, R. Arkin, and J. Cameron, "Multiagent Mission Specification and Execution," *Autonomous Robots*, 4(1), pp. 29-57, 1997.

[10] R. Arkin and T. Balch, "AuRA: Principles and Practice in Review," *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2), pp. 175-189, 1997.

[11] R. Arkin, "Motor-Schema based Mobile Robot Navigation," *International Journal of Robotics Research*, 8(4), pp. 92-112, 1989.

[12] J. Kolodner, *Case-Based Reasoning*, Morgan Kaufmann Publishers, San Mateo, 1993.

[13] N. Chalmique Chagas and J. Hallam, "A Learning Mobile Robot: Theory, Simulation and Practice," *Proceedings of the Sixth Learning European Workshop*, pp.142-154, 1998.

[14] P. Langley, K. Pfleger, A. Prieditis, and S. Russel, "Case-based Acquisition of Place Knowledge," *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 344-352, 1995.

[15] R.P.N. Rao and O. Fuentes, "Hierarchical Learning of Navigational Behaviors in an Autonomous Robot using a Predictive Sparse Distributed Memory," *Autonomous Robots*, 5, pp. 297-316, 1998.

[16] A. Ram, R. Arkin, G. Boone, and M. Pearce, "Using Genetic Algorithms to Learn Reactive Control Parameters for Autonomous Robotic Navigation," *Journal of Adaptive Behavior*, 2(3), pp. 277-305, 1994.

[17] S. Mahadevan and J. Connell, "Automatic Programming of Behavior-Based Robots Using Reinforcement Learning," Proceedings of the Ninth National Conference of Artificial Intelligence, pp. 768-773, 1991.