Spatio-Temporal Case-Based Reasoning for Efficient Reactive Robot Navigation[†]

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

Abstract

This paper presents an approach to automatic selection and modification of behavioral assemblage parameters for autonomous navigation tasks. The goal of this research is to make obsolete the task of manual configuration of behavioral parameters, which often requires significant knowledge of robot behavior and extensive experimentation, and to increase the efficiency of robot navigation by automatically choosing and fine-tuning the parameters that fit the robot task-environment well in real-time. The method is based on the Case-Based Reasoning paradigm. Derived from incoming sensor data, this approach computes spatial features of the environment. Based on the robot's performance, temporal features of the environment are then computed. Both sets of features are then used to select and fine-tune a set of parameters for an active behavioral assemblage. By continuously monitoring the sensor data and performance of the robot, the method reselects these parameters as necessary. While a mapping from environmental features onto behavioral parameters, i.e., the cases, can be hard-coded, a method for learning new and optimizing existing cases is also presented. This completely automates the process of behavioral parameterization. The system was integrated within a hybrid robot architecture and extensively evaluated using simulations and indoor and outdoor real world robotic experiments in multiple environments and sensor modalities, clearly demonstrating the benefits of the approach.

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 in time a robot selects a subset of behaviors, called a behavioral assemblage, based on incoming sensory data from the set of predefined behaviors and then executes them. One of the problems of this approach, however, 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 non-trivial cases, results in the robot performance being far from 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. It is also desirable to avoid the requirement of knowing when and which types of environments a robot will encounter during its mission.

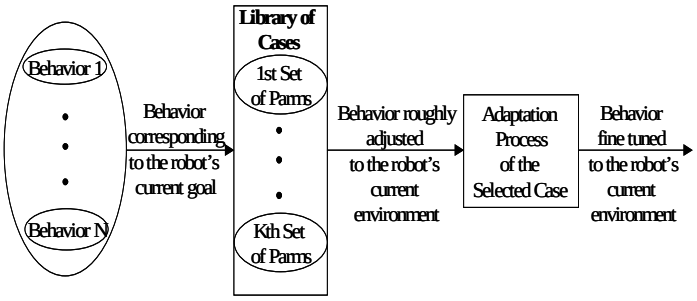


Figure 1. Behavioral selection process with Case-Based Reasoning module incorporated.

This paper presents a solution to these problems by incorporating Case-Based Reasoning into the behavior selection process. Case-Based Reasoning is a form of learning in which specific instances or experiences are stored in a library of cases, and are later retrieved and adapted to similar situations when they occur [12]. The Case-Based Reasoning (CBR) module operates at the reactive level of the robot. Given a particular behavioral assemblage that the robot executes, the CBR module selects the set of parameters for the chosen behaviors that is best suited for the current environment. As the robot executes its mission, the CBR module controls the switching between different sets of behavioral parameters in response to changes in the environment. Each such set of parameters constitutes a case in the CBR library of cases and is indexed by spatial and temporal features of the environment. Spatial features are computed from the sensory data of the robot,

[†] This research is supported by DARPA/U.S. Army SMDC contract #DASG60-99-C-0081. Approved for Public Release; distribution unlimited.
[‡] Presently at Carnegie Mellon University.

while temporal features are computed based on the robot's performance. The adaptation step in 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. Such a method permits automatic selection of optimal parameters at run-time while the mission specification process no longer requires manual configuration of these values.

While the library of cases that the CBR module uses can be manually supplied for a given robot, this paper also proposes an extension to the CBR module that allows it to create and optimize cases in the library automatically as the result of either experiments or actual mission executions. In the learning mode, the module can start with either a completely empty, partially specified, or fully specified library. It will then add new cases as necessary and optimize the new and existing cases by performing a gradient descent search in the space of behavioral parameters for each of the cases in the library. Once the training is over and the robot exhibits good performance in the training session, the library can be "frozen".

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 environmental similarities, low-level action-selection based on local sensor information, place learning, acceleration of complex problem solving based on past problem solutions and other problems [4, 5, 6, 7, 8, 14, 21]. Previous work has also been performed on the incorporation of Case-Based Reasoning in the selection of behavior parameters by our group [1, 2], on which this present research is partially based, and a few others (e.g., [13]). The approach described in this paper, however, differs significantly from these previous algorithms by introducing: a novel feature identification mechanism that produces spatial and temporal vectors describing the current environment; a notion of *traversability vectors* that measure the degree of traversability around a robot in a configurable number of directions; a randomization in the case selection process to allow for the exploration of cases; and a case switching decision tree to adaptively control case switching based on case performance. This novel methodology results in very robust performance of the robot while allowing for an easy input and output vector space representation, straightforward extensions and modifications, and simple control of computational complexity depending on the available computational resources and precision of sensor data. Additionally, the work presented in this paper extends our previous work [1, 2] by incorporating the Case-Based Reasoning within a hybrid robot architecture and extensively evaluating the performance on both real and simulated robots.

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 experiential knowledge

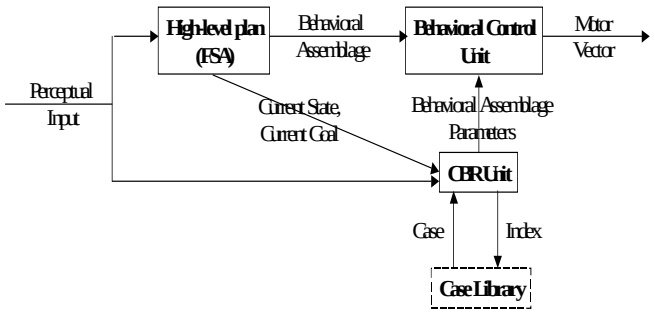


Figure 2. Integration of the case-base reasoning module within the AuRA architecture.

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 the use of a library of cases.

II. ARCHITECTURAL CONSIDERATIONS

The framework chosen for the integration of Case-Based Reasoning for behavioral selection is the *MissionLab* system [9], which is a version of AuRA (Autonomous Robot Architecture) [10]. This hybrid architecture 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] that are grouped into sets called behavioral assemblages. Each individual primitive behavior is driven by its perceptual input(s), a perceptual schema(s), producing its own motor response. The vectorial responses from each of the active schemas are added together in the behavioral control module as shown in figure 3, resulting in the overall behavioral output. The weighted sum of the vectors, after normalization, defines the final vector that is sent to the actuators. Hence, each motor schema affects the overall behavior of the robot.

Within *MissionLab*, a finite state automaton defines the high-level plan of a robot's mission. Each state in the plan is one of the predefined behavioral assemblages chosen to achieve the goal of a robot at this state. The transitions between states are triggered by perceptual inputs called triggers.

Every behavioral assemblage (a state in a high-level plan) is controlled by a set of parameters. Normally, these parameters would be carefully chosen by a user to correspond to the task-environment the robot is expected to inhabit. If optimal behavior for the robot is desired, states could be split into multiple alternative states with the same behavioral assemblage but with different sets of internal parameters, where each state is attuned to some particular environmental characteristics. This method would also require designing special perceptual triggers to detect relevant changes in environment conditions. The methods described in this article avoid this complexity by introducing a Case-Based Reasoning module that for a currently chosen behavioral assemblage, selects in real-time the set of parameters that is best suited for the current environment. As the type of environment might change

unexpectedly, the CBR module continually monitors and re-selects and re-adapts the assemblage parameters as necessary, according to current conditions.

A diagram of how the CBR module is integrated within *MissionLab* is shown in figure 2. The sensor readings enter into both the high-level FSA-based planner and the CBR module. Based on the perceptual input, the same or a new state is selected. The chosen state defines a behavioral assemblage that is then passed into the behavioral control module. The chosen state identifier is also passed into the CBR module along with relevant information about the current robot's goal, such as the goal's position. If the CBR module supports the current state, then based on the perceptual input, goal information and the state, a set of parameters for the behavioral assemblage is selected from the case library and adapted to better fit the environment. These parameters are passed into the behavioral control module, which applies them to the current behavioral assemblage. After evaluating this assemblage, the motor vector is produced and supplied to the actuators for execution. If the chosen state, however, is not supported by the CBR module, then the behavioral control module evaluates the behavioral assemblage with its default parameter values as defined in the finite state machine.

Currently, the CBR module supports navigational states of type **GOTO**, which are used for goal-directed navigation. This particular assemblage contains the following four primitive motor schemas, as shown in figure 3: **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 active 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 the **BiasMove** schema. Thus, a case in the library consists of a set of values for the above parameters.

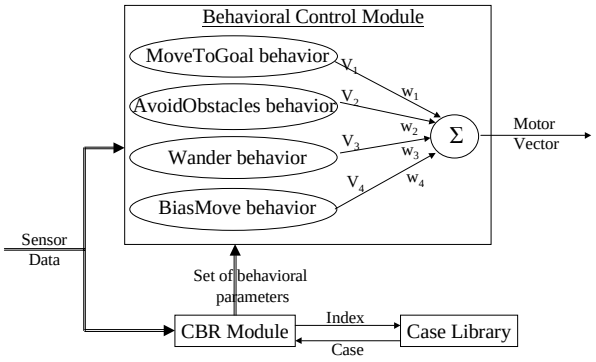


Figure 3. Interaction between the behavioral control module running a GOTO behavioral assemblage and the CBR module.

III. SPATIO-TEMPORAL CASE-BASED REASONING

This section describes a CBR module that uses a hand-coded library of cases. Additional details on this method can be found in [3]. The next section provides a description of the changes required to make the CBR module learn new cases and its implementation. Additional details on these issues and the learning approach in general can be found in [19].

A. Overview

The overall structure of the CBR module, shown in figure 4, is similar to a traditional non-learning Case-Based Reasoning system [12]. The sensor data and goal information is supplied to the Feature Identification sub-module, which computes a spatial features vector representing the relevant spatial characteristics of the environment and a temporal features vector representing the relevant temporal characteristics. Both vectors are passed forward for to the best matching case selection process.

During the first stage of case selection, all the cases in the library are searched, and the distances between their spatial feature vectors and the current environmental spatial feature vector are computed. These distances define the 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

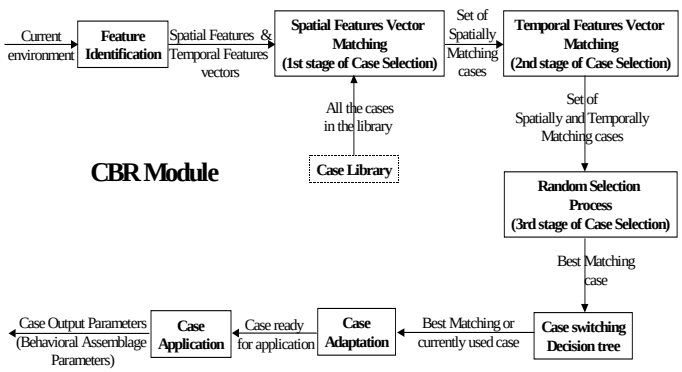


Figure 4. High-level structure of the CBR module.

spatially matching case are also selected for the next stage selection process. These cases are called *spatially matching* cases. At the second stage of selection, all the spatially matching cases are searched, and the distances between their temporal feature vectors and the computed environmental temporal feature vector are generated. These distances define the temporal similarities of these cases with the environment. The case with the highest temporal similarity is *the best temporally matching case*. Again, all the cases with a temporal similarity within some delta from the similarity of the best temporally matching case are selected for the next stage of the selection process. These cases are *spatially and temporally matching* cases and constitute 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. The set, however, can never be empty as the case most similar to the environment is always selected, independently of how dissimilar this case might be.

The last phase of the selection stage is a uniformly random selection from the set of spatially and temporally matching cases. The idea is that these cases are all close enough to the current environment. Their output parameter vectors, however, might be very different. One specific pair of temporal and spatial feature vectors does not necessarily map onto an optimal solution due to possible aliasing. As a result, all of the cases found to be sufficiently similar to the current environment deserve at least a chance to be tried.

The case switching decision tree is then used to decide whether the currently applied case should still be applied or should be switched to the new case selected as the best matching one. This protects against thrashing and overuse of cases. If a new case is chosen to be applied, then it goes through the case 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 on to the behavioral control module outside of the CBR module for execution.

B. Technical Details

1) Feature Identification Step

In this step spatial and temporal feature vectors are produced based on current environment data. This data includes sensor readings and the goal position. The sensor data are distances to obstacles measured by the robot's sonar or laser sensors.

The spatial feature vector consists of two elements: a distance D from the robot to the goal and a sub-vector which represents an approximation of the obstacle density function around the robot and is computed as follows. The space around the robot is divided into K angular regions, as shown in figure 5. The regions are always taken in such a way that the bisector of the 0^{th} region is directed toward the goal of the robot. Within each region the cluster of obstacles that obstructs the region most of all is found. This can be done, for example, by sorting all obstacles within each region in the order of their obstruction angles and finding a single largest sequence of obstacles such that no space in between these obstacles is large enough for the robot to traverse through. An obstacle density function approximation vector is then represented by K pairs $<\sigma, r>$ where σ is the degree of obstruction of a region by the most obstructing cluster in this region, and r is the distance to this cluster.

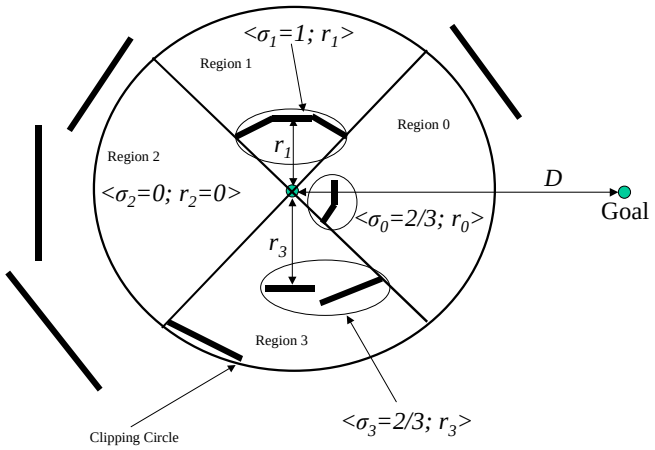


Figure 5. Computation of the spatial feature vector for K=4 regions. The robot is in the center of the circle. Thick lines are obstacles as detected by 12 sensors evenly placed around the robot. The circled clusters of obstacles within each region are the most obstructing clusters.

Figure 5 demonstrates an example computation of the obstacle density. There are 12 sensors in this example, evenly spaced around the robot. The large circle is centered on the robot and describes a clipping circle, beyond which all detected obstacles are ignored in the computation of the density function. The encircled obstacles within each region define the most obstructing clusters within that region. Their corresponding degree of obstruction σ is then computed as the ratio of the angle that they obstruct within the region over the angle of the whole region. Thus, σ is equal to 1.0 for region 1, indicating that the obstacles obstruct the region completely, whereas σ is equal to 2/3 for the 0^{th} and 3^{rd} regions, as the obstacles leave 1/3 of the angle in those regions free for traversing. Region 2 has σ equal to 0.0 since there are no obstacles detected within the region's clipping circle. Thus, this whole region is available for traversing.

Figure 5 also shows the distance of the robot to the goal, *D*, which is the first element of the spatial feature vector. The number of regions K is determined based on the desired computational complexity and the resolution of the sensor data. If K is equal to the number of sensors on the robot, then the obstacle density function is the actual raw sensor data clipped by the clipping region. Thus, in the above example setting K to more than four might not yield any benefit as there are only three sensors per region anyway. On the other hand, setting K to a larger value might make sense if a robot uses a high resolution sensor such as a laser range finder.

The temporal feature vector contains two scalar elements: a short-term relative motion R_s and a long-term

relative motion R_l. The short- and long-term relative motion measures represent short- and long-term velocities of the robot, respectively, relative to the maximum possible velocity of the robot, and are computed as shown in formula (1). The same formula is used for the computation of both relative motion measures. However, the time window lengths used to compute average robot positions differ between long- and short-term relative motion computations as shown in the formula (2).

$$R_i = \frac{\left\| Pos_{i,longterm} - Pos_{i,shortterm} \right\|}{N * MaxVel}, \text{ for } i = s, l \quad (1)$$

where N is the normalization constant, $MaxVel$ is the maximum robot velocity, and $Pos_{i,longterm}$ and $Pos_{i,shortterm}$ are average positions of the robot over long- and short-term time windows, respectively, and are updated according to formula (2) every time the CBR module is called.

$$Pos_{i,j} = a_{i,j} * Pos_{i,j}^{old} + (1 - a_{i,j}) * NewPos \quad (2)$$

$$\text{for } i = s, l \text{ and } j = shortterm, longterm$$

where $NewPos$ is a new current position of the robot, and the filter coefficient a is dependent on whether $Pos_{s,shortterm}$, $Pos_{s,longterm}$, $Pos_{l,shortterm}$ or $Pos_{l,longterm}$ is computed. Thus, for example, $a_{s,shortterm}$ is set to a coefficient with decay time of five time cycles, whereas $a_{l,longterm}$ is set to a coefficient with decay time of 600 time cycles.

Formula (3) summarizes the form of the spatial and temporal vectors.

$$V_{spatial} = \begin{bmatrix} D \\ \sigma_0, r_0 \\ \vdots \\ \sigma_{K-1}, r_{K-1} \end{bmatrix} \quad V_{temporal} = \begin{bmatrix} R_s \\ R_l \end{bmatrix} \quad (3)$$

These two vectors define input features (indices) for cases and are passed in this form into the best matching case selection described next.

2) Case Selection Process

The best matching case selection is broken into three steps. In the first step, a set of spatially matching cases is found. All the cases in the library contain their own spatial and temporal feature vectors. The similarity between the spatial feature vector for a case and an environment is used to assess the degree to which the case matches the environment spatially. In order for spatial feature vectors to be comparable, however, they are first transformed into *traversability vectors*. The *traversability vector F* eliminates actual distances by representing the degree to which each region can be traversed. Formula (4) presents the transformation from a spatial vector into the traversability vector.

$$F = \begin{bmatrix} f_0 \\ \vdots \\ f_{k-1} \end{bmatrix}, \quad f_i = \min(1, 1 - \sigma_i * \frac{D_f - r_i}{D_f}), \quad (4)$$

$$D_f = \max(D_{\min}, \min(D_{\max}, D))$$

where D is the distance to the goal from equation (3), D_{min} and D_{max} are the minimum and maximum thresholds, respectively, for considering traversability in a region, and $<\sigma_i, r_i>$ are elements of $V_{spatial}$ as defined in equation (3). The idea is that D_f represents the circle of interest for traversability computation. The goal distance D limits it on one hand, while D_{max} also limits it if the goal is overly distant. D_{min} is just a minimum threshold to prevent zero radius circles. The traversability measure f_i ranges from 0 to 1. It is proportional to the degree of obstruction σ_i of the most obstructing cluster in the region and the distance r_i at which this cluster is present in the region. Thus, if a cluster of obstacles is extremely close to the robot and it obstructs the whole region, then the region's traversability measure f becomes 0. If, however, obstacles obstruct the region minimally, or they are all beyond the circle of interest with radius of D_f, then the traversability measure f approaches 1.

To avoid large changes in the traversability vector for environment F^{env} due to noise in sensor data, the vector is passed through the smoothing filter given in formula (5). The coefficient b is chosen such as to have a decay time on the order of 5 to 10 sensor readings.

$$f_i^{env} = b * f_i + (1 - b) * f_i^{env,old} \quad (5)$$

where f_i is computed according to formula (4) based on the spatial vector for the current environment and $f_i^{env,old}$ is f_i^{env} from the previous execution of the CBR module.

As every case in the library is represented by a traversability vector F and the current environment is represented by a traversability vector F^{env}, these vectors can be used to assess the spatial similarities between the cases and the environment. The spatial similarity is computed as the weighted sum of squared errors between a case and the environment traversability vectors. There is significantly more weight given to the regions directed more towards the goal. This assures that, for example, if a case and an environment have clear-to-goal situations in the 0th region, then the environment is more similar to this case than to any other case that might have other very similar regions, but does not have the clear-to-goal situation in the 0th region. Formula (6) shows the computation of spatial similarity S.

$$S = 1 - \frac{\sum_{i=0}^{K-1} w_i * (f_i - f_i^{env})^2}{\sum_{i=0}^{K-1} w_i} \quad (6)$$

where W is the vector of weights for each region, F is the traversability vector of a case, and F^{env} is the traversability vector of the current environment. Thus, the perfect match is represented by S equal to 1, and the maximum difference by S equal to 0.

After the spatially based case selection, the set of spatially matched cases contains all the cases with spatial similarity S within some delta from the spatial similarity of the best spatially matching case. The best spatially matching case is defined as the case with the highest spatial matching similarity with the current environment.

Similarly, at the second selection step the temporal similarity with the current environment is computed for all the cases in the set of spatially matched cases according to formula (7).

$$S = 1 - \frac{w_l * (R_l - R_l^{env})^2 + w_s * (R_s - R_s^{env})^2}{w_l + w_s} \quad (7)$$

where w_l and w_s are long- and short-term relative motion measure weights, $< R_s, R_l >$ is a temporal vector for a case, and $< R_s^{env}, R_l^{env} >$ is a temporal vector for the current environment. The long-term relative motion measure is given more weight indicating its greater importance in the assessment of temporal similarities.

The best temporally matching case is the case that has the highest temporal similarity with the environment. All cases with a temporal similarity within some delta from the temporal similarity of the best temporal matching case are selected from the set of spatially matched cases for the next selection stage. Thus, after the temporal-based selection process, the set of matched cases contains the cases that are both spatially and temporally similar to the environment.

Finally, at the third and the last step of the case selection process, randomness is added to the selection process. Namely, one case from the set of matched cases is selected uniformly at random. This selected case is declared to be the best matching case with the current environment.

3) Case Switching Decision Tree

At this step the decision is made as to whether the best matching case or the currently applied case should be used until the next call to the CBR module. This decision is based upon a number of characteristics describing the potential capabilities of the best matching case and the current case. The decision tree is shown in figure 6.

At the root of the tree, the time the current case was applied is checked against some threshold *CaseTime* that is specific to each case in the library. If the current case was applied for less time than the threshold, then the spatial similarity of the current case is checked against threshold S_{low}, and the difference between the new case's spatial

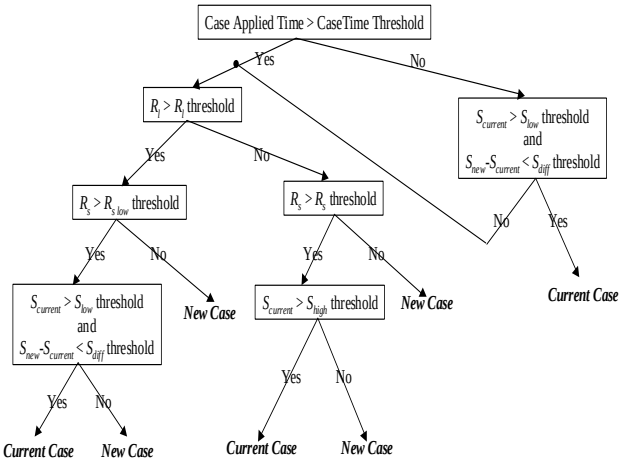


Figure 6. Case switching decision tree.

similarity and the current case's spatial similarity is checked against some threshold S_{diff}. If the two conditions are satisfied, then the current case continues to be used. The intent is that the current case should not be thrown away too soon, unless the environment became significantly different from what it was when the current case was initially selected. If one or both of the conditions are unsatisfied, or if the case was applied for longer than the suggested threshold, the decision-making proceeds to check the long-term relative motion measure R_l. If it is larger than some threshold, then the case is more likely to be performing well and the short-term relative motion measure R_s should be compared against a low threshold $R_{s\ low}$. If the short-term relative measure is also higher than the low threshold, it suggests that the current case performs well and it is exchanged for the new one only if its spatial similarity is very different from the environment or a much more similar case is found. Otherwise, the current case in use remains unchanged. If, on the other hand, the short-term relative motion measure is less than the low threshold, then the case is switched to the new one. Going back to the long-term relative measure check, if it is smaller than the R_l threshold, then the case might not be performing well and, therefore, the short-term relative measure is compared against a stricter threshold R_s *threshold*. If it falls below this threshold, then the new case is selected. Otherwise, the case spatial similarity is compared against a strict threshold S_{high} *threshold*. If the similarity is less, then the new case is selected, otherwise the current case is given more time to exert itself.

4) Case Adaptation

If it is decided at the previous step to keep the current case, then this step (case adaptation) is not executed. If it is decided, however, to apply a new case, then the new case needs to be fine-tuned to the current environment.

The adaptation algorithm is very simple:

X = ($R_{l\ adapt}$threshold + $R_{s\ adapt}$threshold) / (R_l + R_s);
Y = $R_{l\ adapt}$threshold / R_l ;
Z = $R_{s\ adapt}$threshold) / R_s;
If ($R_l < R_{l\ adapt}$threshold and $R_s < R_{s\ adapt}$threshold)
    Increase *Noise_Gain* proportionally to X;
    Increase *CaseTime* proportionally to X;
    Limit *Noise_Gain* and *CaseTime* from above;
Else if ($R_l < R_{l\ adapt}$threshold)
    Increase *Noise_Gain* proportionally to Y;
    Increase *CaseTime* proportionally to X;
    Limit *Noise_Gain* and *CaseTime* from above;
Else if ($R_s < R_{s\ adapt}$threshold)
    Increase *Noise_Gain* proportionally to Z;
    Limit *Noise_Gain* from above;
End;

The adaptation algorithm looks at both the long-term and short-term motion measures of the robot and increases the level of noise (random motion) in the robot's behavior if any of the measures fall below the corresponding threshold. The amount to which the noise is increased is proportional to how long the robot's progress

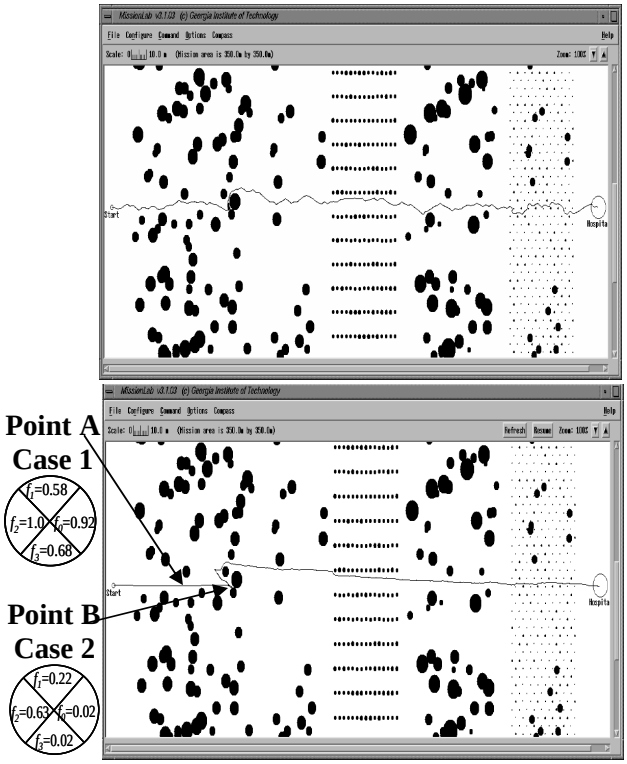


Figure 7. Robot runs in a simulated environment. Top: without CBR module; Bottom: with CBR module. The circles on the left show the values of the traversability vectors that correspond to the indicated points in the environment.

was impeded as determined by the long- and short-term motion measures. If the robot lacks progress for a sufficiently long time period enough, then the long-term motion measure R_l falls below its threshold and the *CaseTime* threshold is also increased to ensure that the new case is applied long enough given the current environment.

The adaptation of the case is followed by its application, which simply extracts the behavioral assemblage parameters from the case and passes them to the actively executing behavioral control module within the *MissionLab* system.

C. An Example of Operation

Figure 7 shows two runs of a simulated robot with and without the CBR module within *MissionLab,* which provides a simulator as well as logging capabilities, making the collection of the required statistical data easy. Black dots of various sizes represent obstacles and the curved line across the picture depicts the trajectory of the robot. The mission area is 350 by 350 meters.

During the entire run the same behavioral assemblage is used. However, as the environment changes from one type to another, the CBR module re-selects the set of parameters that control the behavioral assemblage. As a result, a robot that does not use the CBR module requires a higher level of noise in its behavior in order to complete the mission (figure 7, top). If, however, the CBR module is enabled, then the **Wander** behavior is rarely used, and the distance traveled by the robot is 23.7% shorter, whereas the

a)

Environment characteristics at A:	**Environment characteristics at B:**
Spatial Vector:	**Spatial Vector:**
D (goal distance) = 300	D (goal distance) = 275
density distance	density distance
Region 0: σ_0 = 0.31; r_0 = 5.13	Region 0: σ_0 = 1.00; r_0 = 0.11
Region 1: σ_1 = 0.71; r_1 = 2.83	Region 1: σ_1 = 0.79; r_1 = 0.11
Region 2: σ_2 = 0.36; r_2 = 7.03	Region 2: σ_2 = 0.38; r_2 = 0.12
Region 3: σ_3 = 0.54; r_3 = 2.80	Region 3: σ_3 = 1.00; r_3 = 0.11
Temporal Vector:	**Temporal Vector:**
(0 - min, 1 - max)	(0 - min, 1 - max)
ShortTerm_Motion R_s = 1.000	ShortTerm_Motion R_s = 0.010
LongTerm_Motion R_l = 0.931	LongTerm_Motion R_l = 1.000
Traversability Vector:	**Traversability Vector:**
(0-untraversable, 1- excellent)	(0-untraversable, 1- excellent)
f_0=0.92 f_1=0.58 f_2=1.00 f_3=0.68	f_0=0.02 f_1=0.22 f_2=0.63 f_3=0.02

b)

Case 1 used at A:	**Case 2 used at B:**
CLEARGOAL	**FRONTOBSTRUCTED_SHORTTERM**
Spatial Vector:	**Spatial Vector:**
D (goal distance) = 5	D (goal distance) = 5
density distance	density distance
Region 0: σ_0 = 0.00; r_0 = 0.00	Region 0: σ_0 = 1.00; r_0 = 1.00
Region 1: σ_1 = 0.00; r_1 = 0.00	Region 1: σ_1 = 0.80; r_1 = 1.00
Region 2: σ_2 = 0.00; r_2 = 0.00	Region 2: σ_2 = 0.00; r_2 = 1.00
Region 3: σ_3 = 0.00; r_3 = 0.00	Region 3: σ_3 = 0.80; r_3 = 1.00
Temporal Vector:	**Temporal Vector:**
(0 - min, 1 - max)	(0 - min, 1 - max)
ShortTerm_Motion R_s = 1.000	ShortTerm_Motion R_s = 0.000
LongTerm_Motion R_l = 0.700	LongTerm_Motion R_l = 0.600
Traversability Vector:	**Traversability Vector:**
(0-untraversable, 1- excellent)	(0-untraversable, 1- excellent)
f_0=1.00 f_1=1.00 f_2=1.00 f_3=1.00	f_0=0.14 f_1=0.32 f_2=1.00 f_3=0.32
Case Output Parameters:	**Case Output Parameters:**
MoveToGoal_Gain = 2.00	*MoveToGoal_Gain* = 0.10
Noise_Gain = 0.00	*Noise_Gain* = 0.02
Noise_Persistence = 10	*Noise_Persistence* = 10
Obstacle_Gain = 2.00	*Obstacle_Gain* = 0.80
Obstacle_Sphere = 0.50	*Obstacle_Sphere* = 1.50
Bias_Vector_X = 0.00	*Bias_Vector_X* = -1.00
Bias_Vector_Y = 0.00	*Bias_Vector_Y* = 0.70
Bias_Vector_Gain = 0.00	*Bias_Vector_Gain* = 0.70
CaseTime = 3.0	*CaseTime* = 2.0

Figure 8. a) Environment features at points A (left) and B (right); b) Cases used at point A (left) and point B (right).

mission completion time is 23.4% less (figure 7, bottom). For example, during the part of the run before the local minimum produced by two obstacles is encountered (point A in figure 7, bottom) the robot uses case 1, called CLEARGOAL case (figure 8b, left). In this case no noise is present in the robot behavior making the trajectory a straight line. When the robot approaches the two obstacles (point B in figure 7, bottom), it switches to case 2 called FRONTOBSTRUCTED_SHORTTERM (figure 8b, right). In this case, the gains of the **Wander** and **BiasMove** schemas and *Obstacle_Sphere* are increased. This ensures that the robot quickly gets out of the local minima and then proceeds toward the goal, switching back to the CLEARGOAL case.

IV. LEARNING BEHAVIORAL PARAMETERIZATION

A. Technical Details

This section provides an overview of the learning CBR module, as shown in figure 9, emphasizing the extensions that were made to the non-learning CBR algorithm described previously. First, as before, the sensor

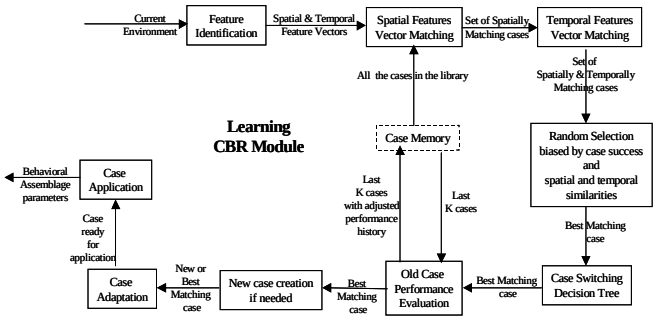


Figure 9. High-level structure of the learning CBR module.

data and goal information are provided to the Feature Identification sub-module that operates identically to its non-learning CBR module counterpart. The resulting spatial and temporal feature vectors are then passed to the best matching case selection process.

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 an exponential function) to the difference between their spatial similarity and the spatial similarity of the best spatially matching case. Figure 10 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 process 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 10, 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 10, C_1 has a higher weighted sum of spatial and temporal similarities and success, and therefore has a higher chance

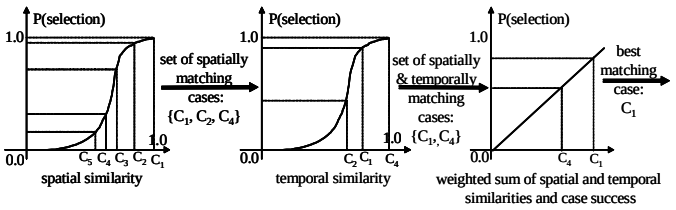


Figure 10. Case selection process.

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 module. Otherwise, the CBR module continues its execution with the evaluation of the currently applied case performance as described below.

2) Old Case Performance Evaluation

The velocity of the robot relative to the goal, that is the speed with which the robot is approaching its goal, is used as the main criteria for the evaluation of case performance. The pseudo code for the performance evaluation of a case C follows:

```
Compute velocity V(C) according to Equation (8)
If (V(C) ≤ 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)> μ·Vmax(C) and V(C)>0)  //μ=0.9
        I(C) = max(1, I(C) + 1);
    else
        I(C) = I(C) – 1;
    end
    I(C)=min(Imax, I(C));  //limit I(C);Imax=100
    Update Vmax(C) according to Equation (2)
    if(C was applied last)
        if(V(C)> μ·Vmax(C) and V(C)>0)
            Increase S(C) by Δ proportional to I(C);
        else
            Decrease S(C) by Δ;
        end
    else
        if (Robot advanced towards its goal)
            Increase S(C) by Δ proportional to I(C);
        else
            Decrease S(C) by Δ;
        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 because the evaluation was postponed; and the case that was applied some time previously that was not yet evaluated and is the 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, then the case performance evaluation is postponed. The robot did not advance if its average velocity V(C) relative to its goal from the time just before case C was applied up until the current time is not positive. 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 V(C) of the robot relative to the goal for case C is computed as follows:

$$V(C) = \frac{g_{t_b(C)} - g_{t_{curr}}}{t_{curr} - t_b(C)} \qquad (8)$$

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 9 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):

$$V_{\max}(C) = \max(V(C),\ \eta \cdot V_{\max}(C) + (1-\eta) \cdot V(C)) \qquad (9)$$

where η is a large time constant, here chosen to be 0.99.

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, where 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 of 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 needs to 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 need 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 newly created, the case still goes through the adaptation process. Every case C in the library also maintains an adaptation vector $\boldsymbol{A}(C)$ 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 $\boldsymbol{O}(C)$ as follows:

```
if (I(C) ≤ 0)
    //change the adaptation direction
    A(C) = – λ· A(C) + ν· R;
end
//adapt
O(C) = O(C) + 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,

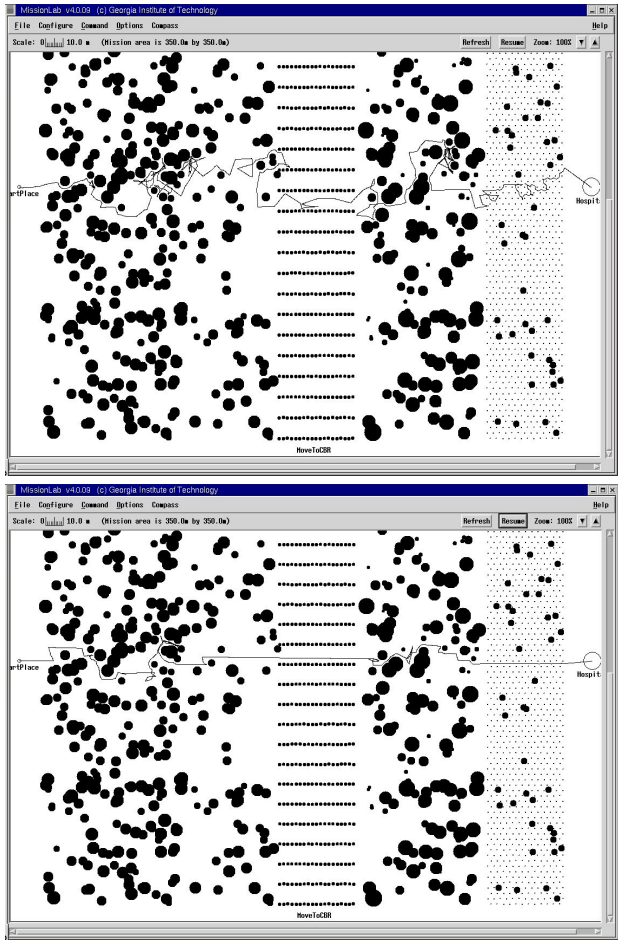


Figure 11. Screenshots of training runs in a heterogeneous environment. Top: initial run that starts off with an empty library; Bottom: a run after fifty-four training runs.

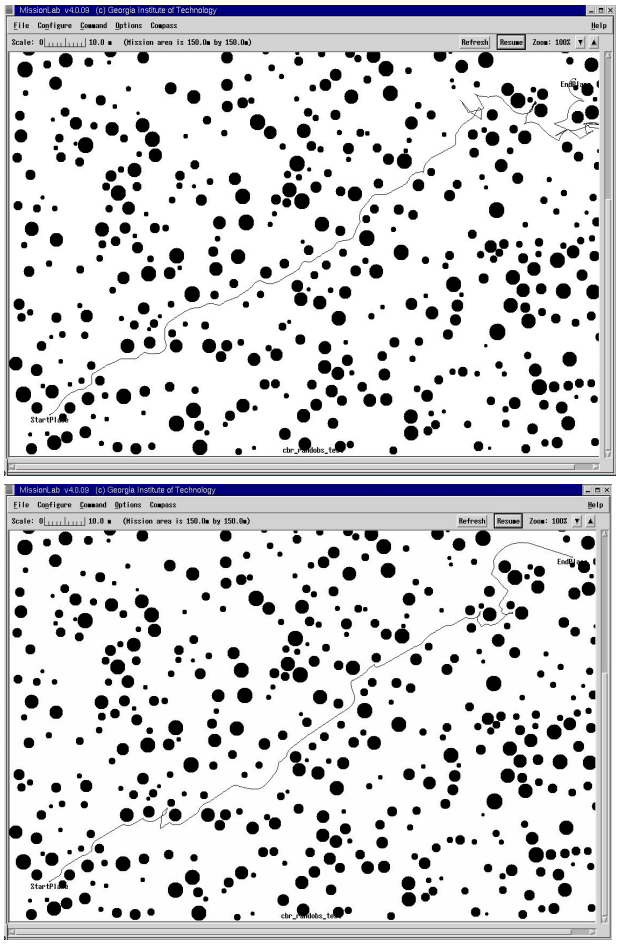


Figure 12. Screenshots of training runs in a homogeneous environment. Top: initial run that starts off with an empty library; Bottom: a run after fifty training runs.

decreased by a constant λ and a randomly generated normal vector $\boldsymbol{R}$ scaled by a constant ν 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- and long-term relative velocities of the robot, which are elements of the temporal features vector. This adaptation step is similar to the adaptation step performed in the non-learning CBR module (section III.B.4) and, in short, increases the *Noise_Gain* and *Noise_Persistence* behavioral parameters inverse proportionally to the short- 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, as can be the case with difficult "box canyons" when using purely reactive methods.

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.

The adaptation of the case is followed by its application, which simply extracts the behavioral assemblage parameters from the case and passes them to the behavioral control module within the *MissionLab* system for execution.

B. An Example of Operation

Figures 11 and 12 demonstrate the runs of a simulated robot that uses the learning CBR module to learn a library of cases. In figure 11 the training is done on heterogeneous environments, where the obstacle density and pattern change within the same robot mission, whereas in figure 12 the training is done on homogeneous environments, where the obstacle density and pattern remain constant throughout a mission. These were two separate training instances resulting in two different learned libraries. The top figures in figures 11 and 12 show screenshots of *MissionLab* during the very 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. In figure 11 the mission area is 350 by 350 meters, whereas in figure 12 it is 150 by 150 meters. Since the library is being created from scratch, the performance of the robot in these initial runs is very poor. The search for an optimal parameterization has just started, and thus the

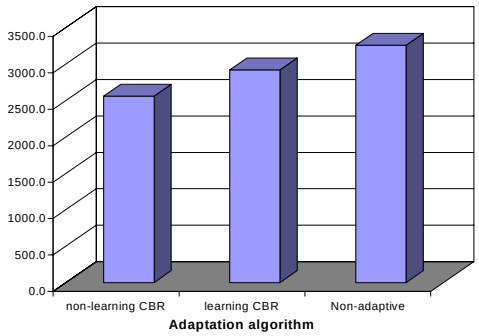


Figure 13. Average number of time steps of a simulated robot in heterogeneous environments.

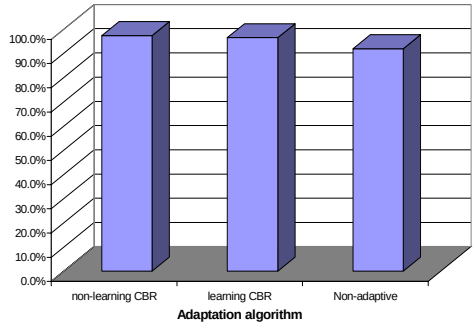


Figure 14. Mission completion rate of a simulated robot in heterogeneous environments.

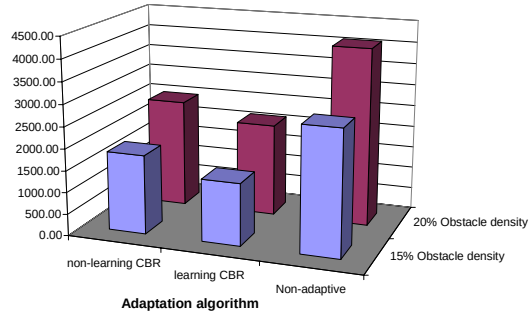


Figure 15. Average number of time steps of a simulated robot in homogeneous environments.

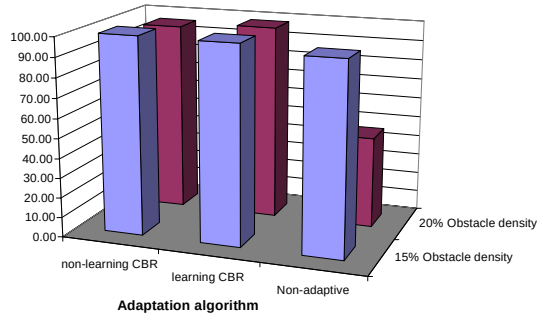


Figure 16. Mission completion rate of a simulated robot in homogeneous environments.

robot behavior is very noisy. In contrast, after about fifty training runs for both heterogeneous and homogeneous environments, the robot successfully learned improved parameterizations and therefore the robot trajectory in the final runs (figures 11 and 12 bottom) is far better.

A good example of learning an efficient parameterization can be found 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 traverse this area. By contrast, in figure 11 bottom, the robot successfully "squeezes" through this area along a straight line. The log files indicate 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.

It is important to note, however, that not all of the cases converge. In particular, the cases created for large "box canyon" environments do not usually converge. Instead, they seem to maintain aggressive adaptation vectors together with the large *Noise_Gain* values. In most cases this appears to be sufficient for good performance of the robot. On the other hand, most of the cases that correspond to the "squeezing" strategy, for example, converge, showing that the robot successfully learns to discriminate between these two scenarios based on the spatial and temporal features of the environment.

V. SIMULATION RESULTS

Figures 13 through 16 depict statistical data gathered from the simulations. Three systems were evaluated: (1) a *non-adaptive* system which did not use the CBR module but instead used manually hard-coded behavioral parameters; a *non-learning CBR* system that employed the non-learning CBR module for the selection and adaptation of behavioral parameters given a manually specified library of cases; and a *learning CBR* system that started out with an empty library and trained it using the learning CBR module. Cases for the non-learning CBR module were created manually by running preliminary experiments on a few heterogeneous environments. Three libraries for the learning CBR system were created automatically by running about 250 training runs on training data consisting of homogeneous, heterogeneous and even empty environments. All three libraries were evaluated and the average values are shown in the graph. For the runs with the non-adaptive system, the optimal set of parameters was chosen for a given average obstacle density, which is equivalent to a user specifying the parameters for a given mission.

Figures 13 and 14 illustrate the performance of a simulated robot on a navigational task in heterogeneous environments, such as the one shown in figure 11. Overall, the test results for 37 missions in different heterogeneous environments were gathered. The performance of a robot is represented by the time steps it takes the robot to complete

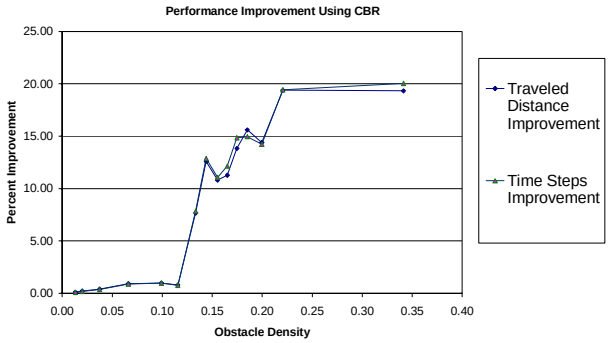


Figure 17. The performance improvement of a simulated robot induced by the non-learning CBR module over the non-adaptive system as a function of obstacle density.

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 optimized on the set of heterogeneous environments used for training. The mission completion rate shown in figure 14 is approximately equal for both non-learning and learning CBR systems. The non-adaptive system has the lowest mission success rate.

Figures 15 and 16 report the results of tests in homogeneous environments such as the one shown in figure 12. In each of the figures, the front row represents an environment with a 15% obstacle density and the back row 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 heterogeneous environments as training data. 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 16, 20% density).

Figure 18. Left: Nomad 150 used for indoor experiments; Right: ATRV-Jr used for outdoor experiments.

Figure 17 presents the results of a separate experiment that demonstrates how the performance of the non-learning CBR system depends on the obstacle density of an environment. The performance represented by traveled distance and time steps was measured as a function of obstacle density. Just as in figure 11, the robot had to travel through different types of environments, but the average density varied between trials. Note that for the runs without the CBR module, 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 particular mission. Even larger improvement could be expected if the parameters were chosen to be constant throughout all the trials. As shown in figure 17, if the average obstacle density is very small (below 12%), then the improvement is insignificant. This is due to the fact that in an environment that is almost obstacle-free, where really only one case is applied all the time. The same set of parameters can be easily chosen manually for the robot without the CBR module. As the obstacle density increases, however, the cases are switched more and more often leading to a significant improvement in performance due to the use of multiple cases in different situations.

VI. ROBOT EXPERIMENTS

We performed both indoor experiments using a Nomad 150 robot (figure 18 left) and outdoor experiments using an ATRV-Jr robot (figure 18 right). The first section below describes the evaluation of the CBR-based navigation system without learning (non-learning CBR). The next section describes extensive evaluation of CBR-based navigation with learning (learning CBR) using robotic hardware.

A. Spatio-Temporal CBR without learning

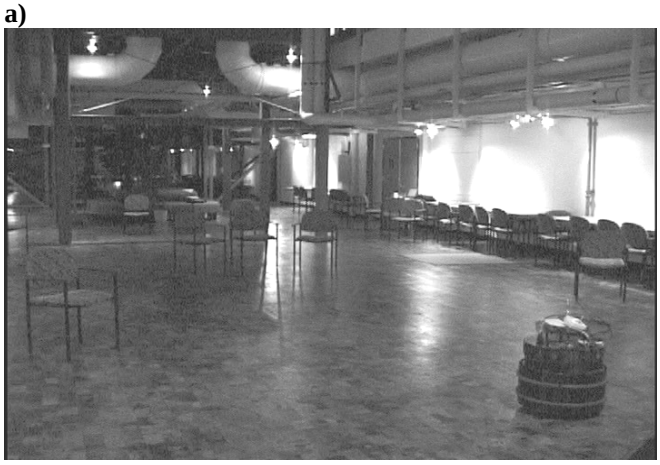
1) Indoor Navigation

The system was first tested on a Nomad 150 series mobile robot performing indoor navigation. It used 12 sonar sensors evenly distributed around it. The data from these sensors was the only perceptual input driving the behavior of the robot in this experiment. The *MissionLab* system provides support for real robotic systems including the Nomad 150 and ATRV-Jr robots. Thus, for the real robot experiments the exact same experimental framework as for the simulations was utilized.

The environment for the real robot experiments is shown in figure 19. The chairs were used to introduce additional obstacles in the environment. The plant in the white vase by the couch shown in the back of the picture represents the goal location for the robot.

Figure 19a shows the start of a robot run. The path is clear and the traversability vector indicates that the 0^{th} region directed toward the goal is fully traversable. This corresponds to the CLEARGOAL case (figure 8b left) with a **Wander** schema gain of zero, and the robot therefore moves straight towards its goal. As it reaches the small box canyon constructed by the three chairs (figure 19b), the

a)



b)

c)

Figure 19. Indoor experiments: chairs are used as obstacles; the plant in the rear near the couch is the goal of the robot.

traversability vector now indicates very low traversability in the 0^{th} region and a high traversability in other regions. The new case FRONTOBSTRUCTED_SHORTTERM (figure 8b right) is applied with a greater gain for the **Wander** schema, a larger sphere of influence for obstacle avoidance and some gain for the **BiasMove** schema directing the robot away from the chairs. As a result, the robot comes quickly out of the canyon. In figure 19c the robot is again clear to its goal, and a case with no **Wander** behavior is selected that makes the robot go straight to the goal.

Improvement(%)		
Obstacle Density	Traveled Distance	Time Steps
Low	6.2	3.3
Medium	17.8	17.6
High	26.4	28.6

Figure 20. Improvement in the efficiency of Nomad 150 indoor navigation when using the non-learning CBR module.

Ten runs were conducted with the non-learning CBR module and ten without the module. Each pair of runs was done on exactly the same environment. Just as in simulations, the trials ranged from very low obstacle density environment to a quite large obstacle density. The collected data is shown in figure 20. The numbers correlate well with the simulation-based data, also showing that as the average obstacle density increases, the benefits from the CBR module also increase.

2) Outdoor Navigation

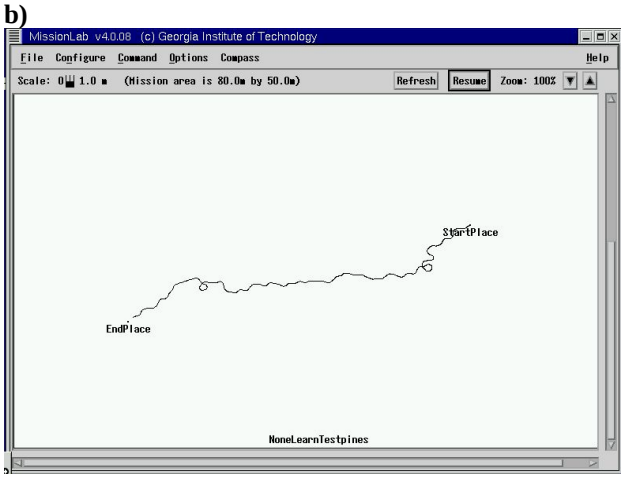
Outdoor experiments were done with an ATRV-Jr robot (figure 18, right), using two SICK LMS200 laser scanners as sensors, one in the front and one in the rear, allowing for a 360 degree view. The behavioral parameters for the system without the CBR module (the non-adaptive system) were chosen manually to produce the best results for the given environment. The library of cases for the CBR module was also manually hard-coded. The library had to be different from the one used in simulations since there are important differences in size and movement capabilities of simulated and physical robots. Therefore, while the library of cases used for simulations was optimized as a result of numerous experiments, the library of cases for the real robot was only based on a few robot experiments and the simulated robot experiments due to the larger time scale for collecting data on real robots versus simulated one. As a result, the library of cases was not necessarily close to optimal.

The robot's mission during the outdoor experiments was to navigate first a small area filled with trees and a few artificial obstacles to increase the difficulty of the environment (figure 21a), creating larger local minima, and then to traverse a relatively clear area until it reaches a goal. The straight-line distance from the start position to the goal was about 47 meters.

Figure 21b shows a trajectory of a typical run by the non-adaptive system. In order to be able to efficiently navigate the area with trees, the robot needs to employ a sufficiently large amount of noise and a small sphere of influence for obstacle avoidance. However, since the parameterization is constant (i.e., non-adaptive), the robot's trajectory continues to be noisy even after the tree region is passed and the environment is clear of obstacles. In contrast, the non-learning CBR system first chooses a case that permitted the robot to squeeze between the trees and to quickly extricate itself from local minima of this type, and then switched to the CLEARGOAL case that resulted in a straight trajectory directly towards the goal (figure 21c). The trajectory of the robot that used the non-learning CBR

a)

b)



c)

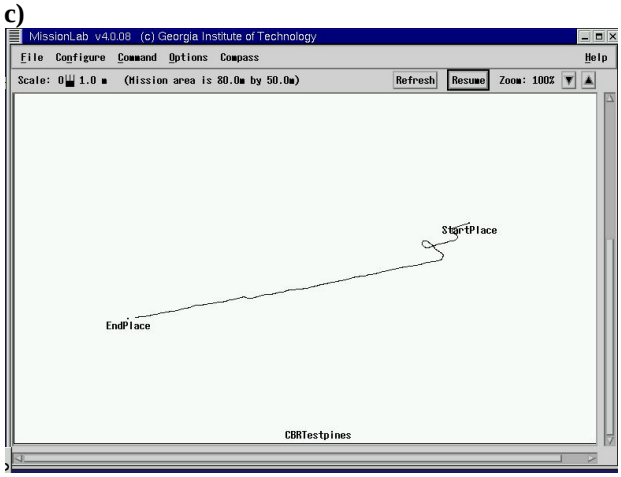


Figure 21. Outdoor experiments: a) ATRV-Jr during one of its test runs. b) The traveled trajectory of the robot using the non-adaptive system. c) The traveled trajectory of the robot using the non-learning CBR system (11% percent shorter).

module is 11% shorter than the one with the non-adaptive system.

Figure 22 summarizes the results from these experiments. It shows the traveled distance and the number of time steps averaged over ten runs for each of the systems for both the non-adaptive system and the system that used

	Traveled Distance	Time Steps
Non-adaptive	76	1348
Non-learning CBR	68	991
Improvement (%)	10.5	26.5

Figure 22. Comparison of ATRV-Jr outdoor navigation efficiency with and without the non-learning CBR module.

the non-learning CBR module. An individual run was considered a failure if the robot ran for ten minutes without reaching its goal. Runs during which the robot became disoriented, that is the robot thought it was facing a different direction than it really was, were discarded, justly isolating and eliminating data points resulting from hardware failures. The non-adaptive system had more of such failures than the non-learning CBR system since the former had a tendency to get stuck for short periods of time in box-canyons, producing greater odometric error, and thus having a longer mission runtime on average.

The results illustrate that the average time for mission completion by the robot that used a non-learning CBR module was 26.5% smaller than the average time of the non-adaptive system. Similarly, the traveled distance for the non-learning CBR system was also 10.5% shorter.

B. Spatio-Temporal CBR with learning

The Learning CBR system was also tested on an ATRV-Jr in outdoor experiments. Just as in the previous experiments, the robot was equipped with two SICK laser range finders allowing for a full 360 degree laser profile (figure 23b). Placing them on the top of the robot eliminated blind spots and problems with uneven terrain and was well suited for environments with large scale obstacles such as buildings and trees. These experiments, however, were conducted in a larger environment making each run substantially longer. To deal with the localization errors the robot was equipped with a three-axis gyro, odometry sensors and an electronic compass. The position of the robot was tracked by fusing sensor information from these sensors, which was automatically taken care of by MissionLab.

The outdoor environment, seen in figure 23a, consisted of a grassy field with surrounding buildings, trees, and bushes. In addition, a large box canyon was placed near the beginning of the course and various obstacles in the form of large trash cans were placed throughout the environment. The layout of the testing and training environments can be seen in figure 24. Both environments were of the same size: 52 meters from start to goal, with the largest obstacle field being 20 meters wide and 8 meters long.

Three different systems were compared: a purely reactive navigation (NAdapt), a purely reactive navigation with swirl behavior (Swirl), and learning CBR, also using swirl. Swirl is a modified GOTO behavior, where obstacle avoidance vectors circulate around the obstacles. This usually leads to a better performance than a purely reactive

a)

b)

Figure 23. Outdoor experiments with the learning CBR: a) the field used for experimental testing of Learning CBR. b) the ATRV-Jr robot navigating within the field.

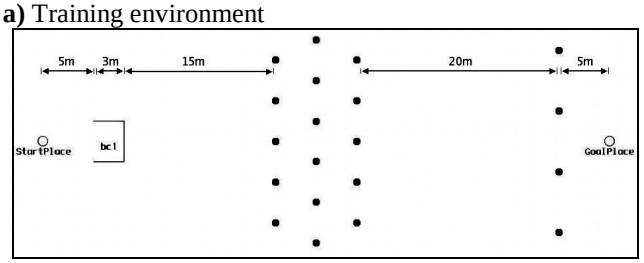
system. The Learning CBR module controls the following parameters:

<Noise_Gain, Noise_Persistence, Obstacle_Sphere, MoveToGoal_Gain>

The training phase consisted of 131 runs during which a full case library was learned. Training started with an empty library, with the size restricted to a maximum of ten cases. The library reached the maximum size towards the end of the training, with two of the cases converged to a high success rate (figure 25). The first one is a "clear to goal" case representing the situation where no obstacles in the direction of the goal are present. The second converged case represents the situation where obstacles appear ahead of the robot. This case increases the obstacle sphere slightly. As in the previous experiments, some training was conducted in an empty environment. A few runs were omitted due to communications or robot firmware failures; it was verified that the failures were not caused by our software and hence should not have affected the results.

During the testing phase, each system was tested 18 times on a modified novel environment. As can be seen in figure 24b, the box canyon has a different shape, the sequence of the two obstacle fields is reversed, and their relative positions as well as their densities are changed. The results, as measured by distance and time traveled, are shown in figure 26. All differences are statistically

a) Training environment



b) Testing environment

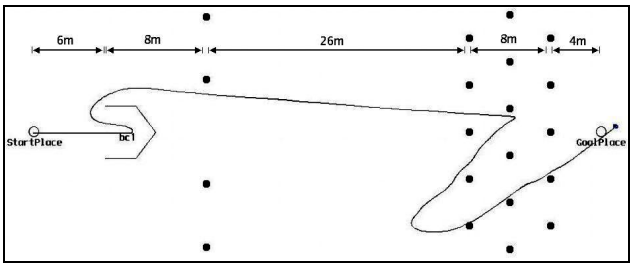


Figure 24. The training (a) and testing (b) environments for outdoor experiments with learning CBR. The figure in (a) also shows the path of the robot that uses the learning CBR module during one of the testing trials (note that the path is uncorrected odometry). In this trial, the robot escaped from the box canyon easily and went around the larger obstacle field.

significant except for the difference between the distance traveled by Swirl and NAdapt. Using learning CBR, the robot completed the missions much faster than when using the simple reactive avoidance system with hand-tuned parameters. The CBR version traveled further because it tended to go around the obstacle field, allowing it to drive faster and hence perform well in terms of time taken to complete the missions. This behavior can be seen in figure 24, which portrays a typical path of the robot during testing. The robot escaped the box canyon easily, passed through the middle of the first line of obstacles, and circumnavigated the obstacle field. CBR can also achieve performance that is close to the hand-tuned Swirl system, which was tuned to the *test* environment, in terms of running time. This demonstrates that the cases learned in the training environment were general enough to be useful in the different testing environment. Also, the CBR and

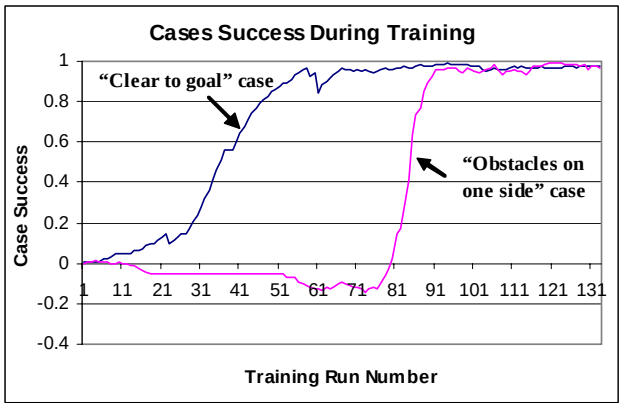


Figure 25. Evolution of the case success for the two converged cases during the training process.

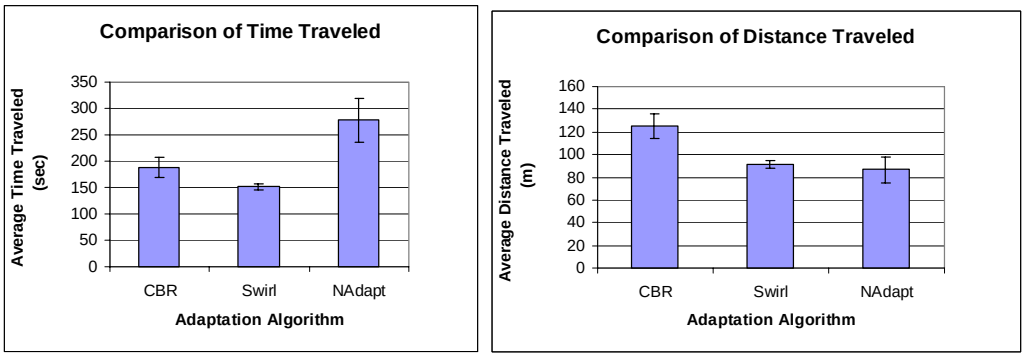


Figure 26. Comparison of outdoor navigation efficiency with the ATRV-Jr in three configurations: CBR – using the learning CBR; Swirl – non-adaptive with the swirl behavior; NAdapt – non-adaptive without the swirl behavior. All differences, with the exception of the one between the distance traveled by Swirl and Nadapt, are statistically significant.

Swirl systems both completed 100% of the test missions, while the simple avoidance system only completed approximately 44% of the test missions.

VII. DISCUSSION AND CONCLUSIONS

This paper presented a method for automatic selection and modification of behavioral assemblage parameters for autonomous navigation tasks. This method can make obsolete the manual configuration of behavioral parameters, making the process of mission specification easier and more robust and making the robot perform its tasks more efficiently. The method was extensively tested both in simulations and indoor and outdoor experiments on real robots, and the results supported the claims. Furthermore, the results have been shown to hold for multiple environments and multiple sensor types.

One of the current limitations of the work presented in this paper is that cases can only be learned and are never forgotten. In particular, this may present a problem if a training environment does not capture all the characteristics of the actual environment used for mission executions. The work in [20] investigates this issue and proposes few easy-to-implement, yet successful metrics for forgetting of cases. Additionally, there has also been some work on integrating the CBR module within a larger framework of learning algorithms such as Learning Momentum within *MissionLab* [18].

VIII. ACKNOWLEDGMENTS

The authors of the paper would like to thank the following people who were invaluable in the work with *MissionLab* and in conducting the experiments: Dr. Douglas MacKenzie, Yoichiro Endo, William C. Halliburton, Mike Cramer, J. Brian Lee, 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*, 2, 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. C. Arkin, and J. Cameron, "Multiagent Mission Specification and Execution," *Autonomous Robots*, 4(1), pp. 29-57, 1997.

[10] R. C. 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. C. 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. C. 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.

[18] J. B. Lee, M. Likhachev, and R.C. Arkin, "Selection of Behavioral Parameters: Integration of Discontinuous Switching via

Case-Based Reasoning with Continuous Adaptation via Learning Momentum," *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, 2, pp. 1275-1281, 2002.

[19] M. Likhachev, M. Kaess and R. C. Arkin, "Learning Behavioral Parameterization Using Spatio-Temporal Case-Based Reasoning," *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, 2, pp. 1282-1289, 2002.

[20] Z. Kira and R. C. Arkin, "Forgetting Bad Behavior: Memory Management for Case-Based Navigation," *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3145-3152, 2004.

[21] C. Urdiales, J. Vázquez-Salceda, E.J. Perez, M. Sànchez-Marrè and F. Sandoval, "A CBR based pure reactive layer for autonomous robot navigation," *Proceedings of the 7th IASTED International Conference on Artificial Intelligence and Soft Computing*, pp. 99-104, 2003.