Selection of Behavioral Parameters: Integration of Discontinuous Switching via Case-Based Reasoning with Continuous Adaptation via Learning Momentum

J. Brian Lee, Maxim Likhachev, Ronald C. Arkin

Mobile Robot Laboratory
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280

Email: blee@cc.gatech.edu, maxim@cs.cmu.edu, arkin@cc.gatech.edu

Abstract

This paper studies the effects of the integration of two learning algorithms, Case-Based Reasoning (CBR) and Learning Momentum (LM), for the selection of behavioral parameters in real-time for robotic navigational tasks. Use of CBR methodology in the selection of behavioral parameters has already shown significant improvement in robot performance [3, 6, 7, 14] as measured by mission completion time and success rate. It has also made unnecessary the manual configuration of behavioral parameters from a user. However, the choice of the library of CBR cases does affect the robot's performance, and choosing the right library sometimes is a difficult task especially when working with a real robot. In contrast, Learning Momentum does not depend on any prior information such as cases and searches for the "right" parameters in real-time. This results in high mission success rates and requires no manual configuration of parameters, but it shows no improvement in mission completion time [2]. This work combines the two approaches so that CBR discontinuously switches behavioral parameters based on given cases whereas LM uses these parameters as a starting point for the real-time search for the "right" parameters. The integrated system was extensively evaluated on both simulated and physical robots. The tests showed that on simulated robots the integrated system performed as well as the CBR only system and outperformed the LM only system, whereas on real robots it significantly outperformed both CBR only and LM only systems.

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

1. Introduction

This research is being conducted as part of a larger robot learning effort funded under DARPA's Mobile Autonomous Robotic Software (MARS) program. In our project, five different variations of learning, including learning momentum, case-based reasoning, and reinforcement learning, are being integrated into a well-established software architecture, *MissionLab* [4]. These learning mechanisms are not only studied in isolation, but the interplay between these methods is also being investigated.

This paper focuses on the interaction between two such learning methods: case-based reasoning (CBR) and learning momentum (LM). Both methodologies were successfully used in robotic systems in different contexts [2, 3, 8, 9, 10, 11, 12, 13, 14]. In this work these methods are used to change behavioral parameters of a behavior-based robotic system at run-time. Both algorithms have already been shown, in isolation, to increase performance in a robotic system in relation to navigating unknown obstacle fields while trying to reach a goal position [2, 3, 6, 7, 14]. Learning momentum was shown to increase the probability that a robot would successfully reach the goal, while case-based reasoning was shown to improve both the robot's probability of reaching the goal as well as the average time it takes for the robot to do so. Both algorithms, however, have their drawbacks, and the hypothesis of this research is that both algorithms could complement each other and reduce these drawbacks by running simultaneously and interacting with each other.

Learning momentum as a stand-alone algorithm is capable of executing only one strategy, and it therefore has a problem when using a strategy in situations for which a different strategy is better suited (see section 2.2 for an explanation on strategies.) Also, searching for the "right" behavioral parameters usually takes too long. Both of these problems result in long mission completion times even though LM mission success rate is very high. CBR can solve both of these problems by changing these learning momentum strategies and by setting the behavioral parameters in the right ballpark using the library of cases in real-time.

CBR by itself also has its own drawbacks. First, it allows for parameter changes only when the environment has changed enough to warrant a case switch. Thus, in between the case changes, the parameters stay constant even though the environment may change to some extent. Second, and more importantly, the behavioral parameters as defined by each case in the CBR library may not be the best parameters for a particular environment the robot operates in. This may happen either because the environment sufficiently differs from the closest match in the library or because the library itself is not particularly well optimized for the robot architecture it targets. In order to avoid such a situation, the library size should be

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

large, and a large number of experiments should be conducted to establish an optimal set of parameters for each case in the library. Even though this needs to be done only once, this solution still may be infeasible and is almost always impossible when working with a real robot since conducting such experiments is costly and time-consuming. As an alternative, learning momentum can provide the continuous search for the best set of parameters in-between case switches. Thus, the hypothesis is that by integrating the learning momentum and case-based reasoning methodologies together for the selection of behavioral parameters, the best of both algorithms can be achieved.

Additionally, this work is meant to be a foundation for future work. Currently, the CBR algorithm uses a static library of cases. In the future, CBR and LM will interact together to both learn new cases and optimize existing cases for the CBR library.

2. Overview of CBR and LM Algorithms

2.1. Framework

Both CBR and LM work as parameter adjusters to a behavior-based system. The adjustment happens by changing the parameters of individual behaviors that contribute to the overall robot behavior. Since the system is developed within a schema-based control architecture, each individual behavior is called a motor schema. Each active motor schema produces a velocity vector. A set of active motor schemas is called a behavioral assemblage. At any point in time, the robot executes a particular behavioral assemblage by summing together weighted vectors from all of the active schemas in the assemblage and uses the resulting vector to provide the desired speed and direction of the robot. The combined learning system was tested on the behavioral assemblage that contains four motor schemas: **MoveToGoal**, **Wander**, **AvoidObstacles** and **BiasMove** schemas. The **MoveToGoal** schema produces a vector directed toward a 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.

For this assemblage the following parameters are changed by CBR module:

<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.

Learning Momentum has control over the same parameters except for the parameters related to the **BiasMove** schema: *Bias_Vector_Gain, Bias_Vector_X* and *Bias_Vector_Y* .

2.2. Overview of Case-Based Reasoning

The detailed description of the CBR module used for behavioral selection can be found in [3]. In this section, only a high level overview of the module is given. The overall structure of the CBR unit is similar to a traditional non-learning case-based reasoning system [5] (figure 1). The sensor data and goal information is 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. Both vectors are passed forward for a best matching case selection.

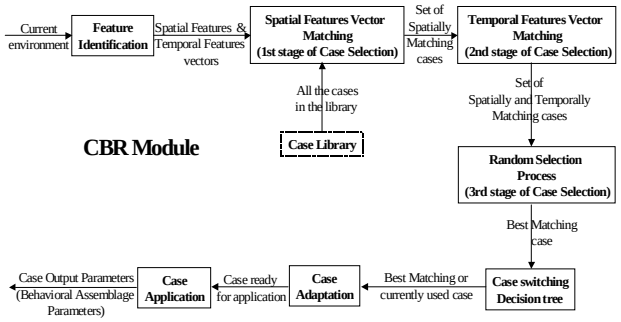

Figure 1. High-level structure of the CBR Module

Case selection is done in three steps. During the first stage of case selection, all the cases from the library are searched, and weighted Euclidean distances between their spatial feature vectors and the environmental spatial feature 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 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 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 *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 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 on 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 should be switched to the case selected as the best matching one. This protects against thrashing and overuse of cases. If a new case is 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, which uses these parameters in the evaluation of the current behavioral assemblage.

2.3. Overview of Learning Momentum

The detailed description of the learning momentum module used for behavioral parameter adjustment can be found in [2]. In this section, only a high-level overview of the module is given. LM is basically a crude form of reinforcement learning. Currently, LM is used in behavior-based systems as a means to alter a robot's behavioral parameters at run time instead of keeping hard-coded values throughout the duration of its mission. Different values for these parameters are appropriate for different environments; LM provides a way for the values to change in response to what the robot senses and the progress it makes.

To work, a LM-enabled system first keeps a short history of pertinent information, such as the number of obstacles encountered and the distance to the goal. This information is used to determine which one of four pre-defined situations the robot is in: *no movement*, *progress*, *no progress with obstacles*, or *no progress without obstacles*. The robot has a two-dimensional table, where one dimension's size is equal to the number of possible situations, and the other is equal to the number of changeable behavioral parameters. For each parameter, the parameter type and situation is used to index into the table to get a value, or delta, that is added to that particular parameter. In this way, the robot may alter its controller to more appropriately deal with the current situation. For example, if the robot were making progress, then the *move-to-goal* behavior would be weighted more heavily. If, however, obstacles were impeding the robot, then the *wander* and *avoid-obstacles* behaviors would be weighted more heavily.

There are currently two LM strategies: ballooning and squeezing. These strategies, which did not change dynamically in the previous work, define how the robot deals with obstacles. When a ballooning robot is impeded by obstacles, it increases the obstacle's sphere of influence (the radius around the robot inside of which obstacles affect the robot's behavior). This pushes the robot out of and around box canyon situations. A squeezing robot, on the other hand, decreases the sphere of influence, allowing itself to move between closely spaced obstacles.

Learning momentum was shown to increase a robot's probability of successfully navigating an obstacle field, but there was an accompanying increase in the time it took to do so. Most of this time increase came from the usage of one strategy (ballooning or squeezing) in situations better suited for another strategy.

3. Implementation

The CBR and LM algorithms themselves were not changed for the integration. Rather they remain the exact same algorithms as reported previously [2, 3]. Since this previous work was already performed within the *Missionlab* mission specification system developed at the Georgia Tech Mobile Robot Lab, the process of integrating both algorithms to work together in the context of *Missionlab* was relatively simple. Existing versions that already had these stand-alone algorithms incorporated into them were easily merged to create a single system with both algorithms incorporated into it.

The parameters that are controlled by LM remain the same as described in Section 2.1. CBR, on the other hand, now controls not only the parameters described in Section 2.1 but also some of the values (i.e., search deltas and bounds) used by the LM algorithm. This in effect controls LM strategies such as ballooning versus squeezing in run-time (a capability LM did not have on its own). For example, if the robot finds itself in a situation where the front is totally blocked, the CBR module may change the deltas in the LM module so that a ballooning strategy is used instead. Conversely, if the robot finds itself in a situation where the environment is traversable but the obstacle density is high, the CBR module may change the deltas in the LM module so that a squeezing strategy is used.

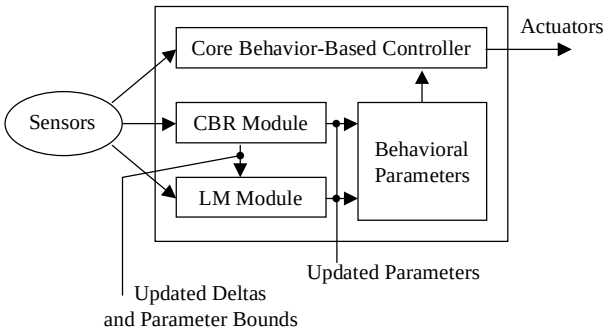

Figure 2. A high level diagram of Core/CBR/LM controller interaction.

Both algorithms utilize the robot's global "blackboard" space to store the behavioral parameters that they control. Thus every time CBR decides to switch a case, it overwrites the parameters stored in the "blackboard" space with the behavioral parameters suggested by the selected and adapted case and specifies what strategy the LM should use to fine-tune those parameters. Afterwards, every few robot cycles, learning momentum retrieves the behavioral parameters from the "blackboard" space, adapts them based on the sensor data and robot progress and stores the parameters back. At the same time, the behavioral control module (Core

Behavior-Based Controller) also reads the behavioral parameters from the "blackboard" space and uses them for the evaluation of the behavioral assemblage every robot cycle. Figure 2 depicts this architecture.

4. Simulation Tests

The system was first evaluated in simulated environments. *MissionLab* provides a simulator as well as data logging capabilities, allowing an easy collection of the required statistical data.

The system was evaluated on two different types of environments. First, the tests were conducted on heterogeneous environments such as the one shown in Figure 3, which shows a screenshot of the *MissionLab* simulator after the robot completed 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 these environments the obstacle pattern and density changes as the robot traverses the test course toward its goal. The size of the mission area is 350 by 350 meters. The tests were also conducted on a set of homogeneous environments such as the one shown in figure 4. In these environments the obstacle density is constant throughout the whole area. The size of the mission area shown is 150 by 150 meters.

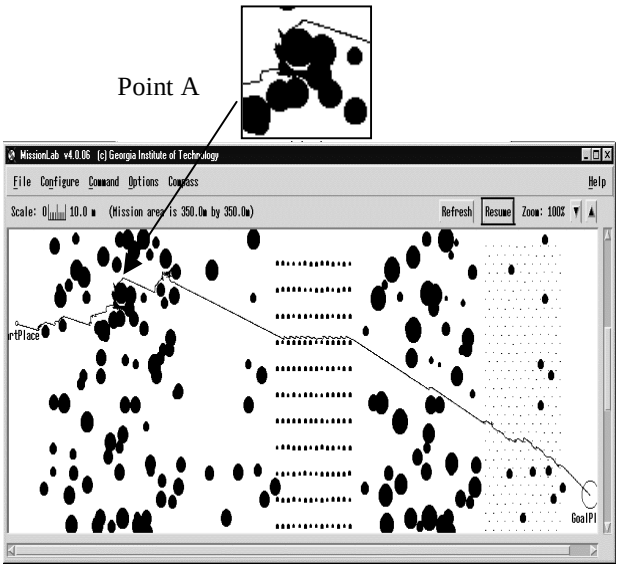

Figure 3. Robot run with CBR integrated with LM algorithm in a heterogeneous simulated environment. Point A is magnified at the top of the figure.

Point A in Figure 3 is magnified to show the robot's behavior in a rather large and narrow box canyon created by obstacles. Here the CBR module recognizes that the robot is stuck for some period of time and the area around the robot is fully obstructed by obstacles. Therefore, it selects a case called FULLOBSTRUCTION_LONG-TERM_BALLOONING. The case sets the *Noise_Gain* and *Noise_Persistence* to large values. It also sets the learning momentum module to use the ballooning strategy. As the robot gets out, the CBR module switches the case to SEMICLEARGOAL, for which the *Noise_Gain* is set to a very small value. The

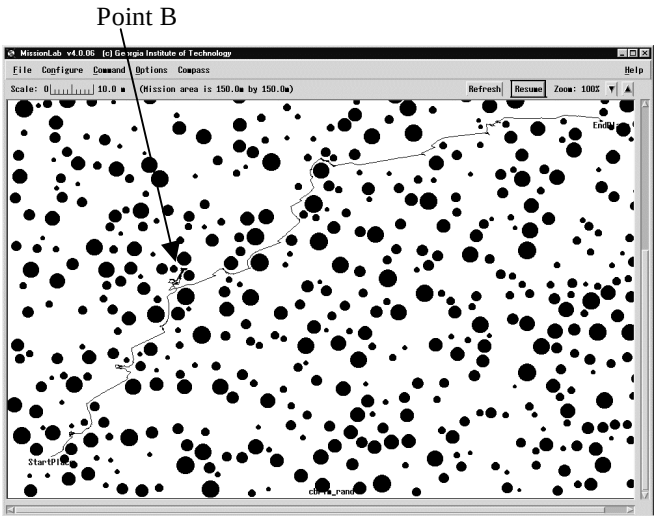

Figure 4. Robot run with CBR integrated with LM algorithm in a homogeneous simulated environment.

Obstacle_Sphere is reduced as well. As a result, once it is out of the box canyon, the robot proceeds along a relatively straight line toward its goal, as can be seen on the top picture of point A in Figure 3.

Figure 4 shows a test run of a simulated robot that employs both CBR and LM modules within a homogeneous environment. The obstacle density in this environment is twenty percent. As before, point B shows the place where the robot becomes stuck and searches for a set of behavioral parameters that would allow it to proceed. The increase in *Obstacle_Gain*, *Obstacle_Sphere*, *Noise_Gain* and *Noise_Persistence* allows the robot to escape the local minimum. Otherwise, the rest of the robot trajectory is a smooth curve with a very good travel distance.

Figures 5 through 8 show the results of tests conducted on the integrated CBR with LM, CBR only, LM only and a system without any adaptation algorithms (non-adaptive). Figures 5 and 6 show the performance of a simulated robot on a navigational task in heterogeneous environments. Overall, the results for 37 missions in heterogeneous environments were gathered. The performance of a robot is represented by the time steps that it takes for a robot to complete its mission as well as the percent of completed missions. Thus, in Figure 5 the least amount of time on average for mission completion is required for systems that use either CBR module or CBR and LM modules together to adapt the behavioral parameters. These systems also have a very high probability of mission completion as shown in Figure 6, and therefore present the best performance. A robot that employs only the LM algorithm, on the other hand, has the longest average time of mission completion but is also very good in terms of mission completion rate. It correlates with the results reported in [2] on the performance of a system with LM adaptation only. Finally, the non-adaptive system takes longer to complete its mission than the system with both LM and CBR together and also fails to complete more missions than any of the adaptive systems.

Figures 7 and 8 report the results of tests in homogeneous environments such as the one shown in

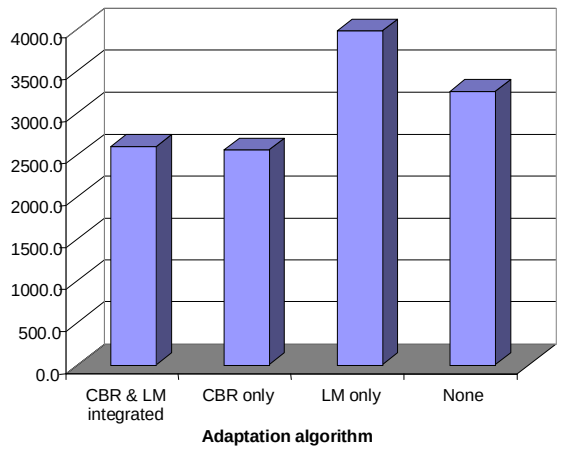

Figure 5. Average number of steps of a simulated robot in heterogeneous environments

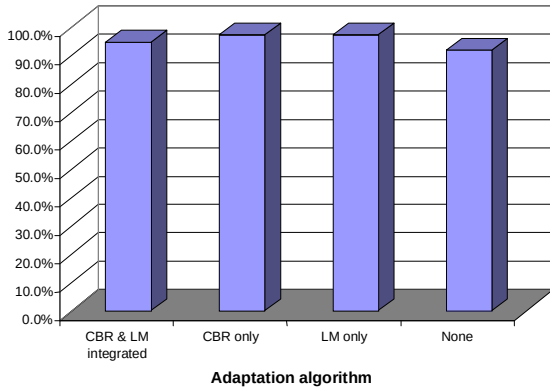

Figure 6. Mission completion rate of a simulated robot in heterogeneous environments

Figure 4. In each of the figures, the first row is for an environment with a 15% obstacle density and the second (farther) 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 both CBR and LM on average completes its missions in the shortest time (Fig. 7) as well as having an almost 100 percent completion rate (Fig. 8). As before, a system with only the LM algorithm has the best completion rate but on average takes a very long time to complete a mission. A non-adaptive system takes longer to complete its mission than either the integrated LM-CBR or CBR-only systems. More importantly, a non-adaptive system exhibits only 46 percent mission completion rate for denser environments (Fig. 8).

According to these results, a robot that uses both CBR and LM algorithms shows a significant improvement over non-adaptive or the LM-only approach. However, it shows just a slight improvement over a system that uses the CBR-only approach for the selection of behavioral parameters. The reason for this is that in simulated environments it is relatively easy to find the best set of parameters for each case in the library as in these tests. What LM provides, on the other hand, is a

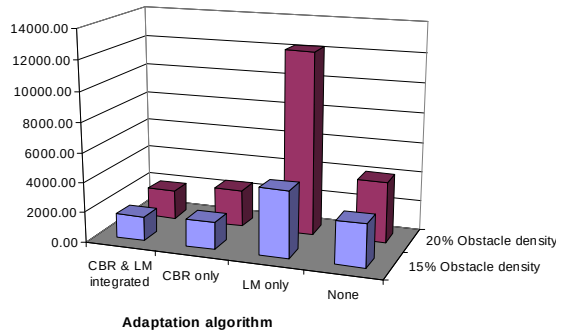

Figure 7. Average number of steps of a simulated robot in homogeneous environments

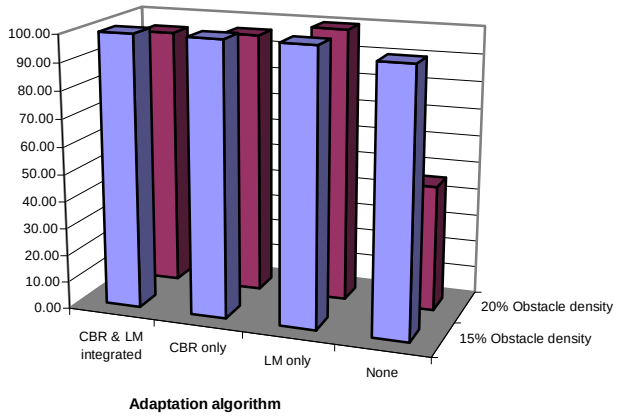

Figure 8. Mission completion rate of a simulated robot in homogeneous environments

search for a best set of parameters for a particular environment in real-time, and therefore is most beneficial when manually establishing an optimal library of cases is difficult. Such is the case when one works with real robots. Conducting experiments on a real robot in order to establish a best library of cases is usually unreasonable due to the number of experiments required. Instead, cases are chosen based on a limited number of experiments coupled with the knowledge derived from extensive simulation studies. Then the real-time adaptation of parameters as provided by the LM algorithm can be beneficial. This point is seen in the next section where the real robot experiments are presented.

5. Robotic Tests

This section describes the methods and results of experimentation on a physical robot.

5.1. Experiment Setup

After concluding experiments on a simulated robot, the system was moved to an ATRV-Jr robot for experimentation on a physical robot. Some behavioral parameters on the non-integrated systems (non-adaptive, LM only, and CBR only) were hand-adjusted to improve the robot performance so that the CBR-LM integrated system could be tested against systems that were believed

to be near-optimal for their respective algorithms. Because the ballooning strategy of LM performed so poorly in preliminary runs, only the squeezing strategy was used on LM-only system. Also, for the systems with CBR enabled (both stand-alone and integrated with LM), a different case library was used for the real robot than was used in simulation. Since there are important differences in size and movement capabilities of simulation and physical robots, the library of cases had to be changed. Therefore, whereas the library of cases used for simulation robot was well 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. As a result, the library of cases was not necessarily optimal, stressing the advantage of having Learning Momentum to optimize the parameters online.

The robot's mission during the outdoor experiments

Figure 9. ATRV-Jr during one of its test runs

was to navigate first a small area filled with trees (some artificial obstacles were also used to increase difficulty), and then to traverse a relatively clear area to finally reach a goal. The straight-line distance from the start position to the goal was about 47 meters.

Data was gathered from ten runs for each of the four types of systems: non-adaptive, LM enabled, CBR enabled, and both LM and CBR-enabled. An individual run was considered a failure if the robot ran for ten minutes without reaching its goal. Runs where the robot became disoriented (i.e., the robot thought it was facing a different direction than it really was) were discarded and redone, isolating and removing data points resulting from hardware failures.

5.2. Robotic Results

The results summarized in figure 10 show that there is an increase in the performance of the integrated system over both the non-integrated and non-adaptive ones. In particular, the non-adaptive system took the longest to complete the mission. These results are inconsistent with the simulation results in that, in simulations, LM-only took the longest time.

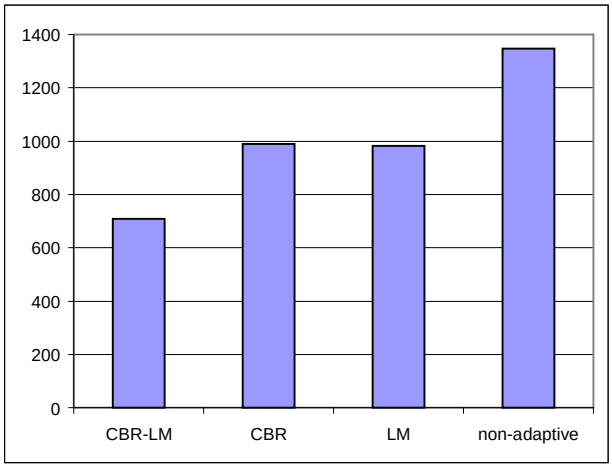

Figure 10. Average steps to completion of a real robot using different learning strategies.

One of the probable explanations is that usually non-adaptive systems would either find a good path to the goal, or they would not reach the goal at all. This meant that the average number of steps to completion for the successful runs is relatively low, but so is the case for success rates. In these experiments, however, there were no failures. All valid runs got to the goal. That fact, coupled with the fact that the non-adaptive robots usually got stuck for short periods of time in box-canyon areas, would drive up the average time to completion for the series of non-adaptive runs. On the other hand, the average time to completion for the LM-only runs was driven down because only the squeezing strategy was used in an environment where ballooning really wasn't needed. (Using one LM strategy in places where the other was more appropriate was found to be a major cause of delay in a learning momentum system [2].) The test environment was not large enough to significantly suffer from not being able to switch strategies for the LM-only system.

Another observation is that the robot using both CBR and LM performed significantly better than the robot using only CBR. This observation again differs from the simulation results, which showed that the addition of LM to CBR provided only small performance increase over CBR-only systems. As mentioned previously, in simulation experiments the CBR library for simulations was well optimized manually before the experiments, whereas for the physical robot experiments the library was not as optimal since case optimization is a very costly and time-consuming operation. Instead, whenever the CBR module set up the behavioral parameters after selecting a new case, the LM module fine-tunes them in run-time until the set of "right" parameters is found.

6. Conclusion

Both case-based reasoning and learning momentum have separately been shown to increase performance when applied to behavior-based systems [2,3]. Those algorithms have now been shown also to further improve performance when used in tandem in a behavior-based control system. Still while the integration of CBR and

LM improves the performance over that of either algorithm when used alone, a significant performance increase is by no means guaranteed. While physical robot experiments indeed show a significant improvement, the simulation results must not be overlooked. Simulation results seem to indicate that if a robot is using CBR with a case library that is well tuned for the robot characteristics, the addition of LM does not necessarily result in improvement. Instead one of the conclusions is that LM is most beneficial when the CBR case library is *not* near optimal. Thus the main benefit from having LM integrated with CBR is that the library no longer requires careful optimization. As the manual optimization requires numerous experiments and therefore is very often impossible when dealing with real robots, the addition of the LM algorithm proves to be important.

Other conclusions that can be drawn from this work are the potential benefits of LM in the process of dynamically updating the CBR case library. Currently we are working on adding such capabilities to CBR as learning new cases, optimizing existing cases, and forgetting old ones. However, because LM already performs the parameter search at run-time, the results of these searches could be valuable for the optimization of cases. As LM finds new sets of the "right" parameters, they could be used to update the existing cases in the library for retrieval whenever the robot encounters a similar environment later. This cooperation would both optimize the library of cases and speed up the search performed by LM. This possibility provides fertile ground for future work.

Acknowledgments

This research is supported under DARPA's Mobile Autonomous Robotic Software Program under contract #DASG60-99-C-0081. The authors would also like to thank Dr. Douglas MacKenzie, Yoichiro Endo, Alex Stoytchev, William Halliburton, and Dr. Tom Collins for their role in the development of the *MissionLab* software system. In addition, the authors would also like to thank Amin Atrash, Jonathan Diaz, Yoichiro Endo, Michael Kaess, Eric Martinson, and Alex Stoytchev for their help with real robot experiments.

References

[1] Arkin, R.C., Clark, R.J., and Ram, A., "Learning Momentum: On-line Performance Enhancement for Reactive Systems," Proceedings of the 1992 IEEE International Conference on Robotics and Automation, May 1992, pp. 111-116.

[2] Lee, J. B., Arkin, R. C., "Learning Momentum: Integration and Experimentation," Proceedings of the 2001 IEEE International Conference on Robotics and Automation, May 2001, pp. 1975-1980.

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

[4] MacKenzie, D., Arkin, R.C., and Cameron, R., "Multiagent Mission Specification and Execution," Autonomous Robots, Vol. 4, No. 1, Jan 1997, pp. 29-52.

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

[6] Ram, A., Arkin, R. C., Moorman, K., and Clark, R. J., "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, Vol. 27, No. 30, 1997, pp. 376-394.

[7] Ram, A., Santamaria, J. C., Michalski, R. S., and Tecuci, G., "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, 1993, pp. 259-275.

[8] Vasudevan, C., Ganesan, K., "Case-based Path Planning for Autonomous Underwater Vehicles," Autonomous Robots, Vol. 3, No. 2, 1996, pp. 79-89.

[9] Kruusmaa, M., Svensson, B., "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, Vol. 2, 1998, pp. 132-141.

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

[11] Veloso, M. M., Carbonell, J. G., "Derivational Analogy in PRODIGY: Automating Case Acquisition, Storage, and Utilization," Machine Learning, Vol. 10, No. 3, 1993, pp. 249-278.

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

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

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