Chapter 8
Robot Control in Dynamic Environments using Memory-Based Learning

Patrick M. McDowell,[1] Brian S. Bourgeois[2] and Frederick E. Petry[2]

Abstract Learning systems used on robots typically require a-priori knowledge in the form of environmental models or trial-and-error approaches requiring a robot to physically execute multitudes of trial solutions. Neither approach is very suitable for dynamic unstructured environments in which a robot is sent to explore and gather information prior to human entry. This chapter presents a new approach, 'memory-based learning', in which a robot is provided with an initial baseline behavior whose performance is linked with a metric explicitly defined by a function whose arguments are sensory inputs and resulting robot actions. A neural network, using sensor inputs and action outputs, has been chosen as the basic controller building block for behaviors as it is very amendable to rapid in-situ generation and adaptation. The use of a neural network controller for a maneuvering task using electronic 'ears' for sensory input is demonstrated in this work, and genetic algorithms are shown to be an effective method for rapidly developing the network weights and transfer functions. The concept of memory based learning is introduced and a construct for representing coupled sense/action information is presented and demonstrated using both simple reactive and memory based controllers, whose performance is demonstrated through simulation studies and on land robots.

8.1 Introduction

For a robot to work autonomously in a complex, changing environment it must be able to adapt dynamically. The work discussed in this paper develops an approach to this complex problem using a sensor input based metric for success and recently collected memories - a temporal series of sensor/action pairs. In essence this scheme monitors the progress of a robot while it carries out a particular task. If the robot is not achieving its goal based on the metric for success, then a learning process

[1]Southeastern Louisiana University, Computer Science Department, Hammond, LA 70402 .[2]Naval Research Laboratory, Stennis Space Center, MS, USA 39529

will be triggered that uses recently collected sensor/action pairs to train a new controller. The learning is achieved with a neural network controller utilizing a genetic algorithm to develop both the network weights and node transfer functions. This approach is applied to a formation maneuvering problem [3,28] where robots with 'ears' listen for a leader robot and attempt to follow. For this application the sensor input is the intensity of the sound in the left and right ear and the action is either turn left, go straight or turn right; the metric of success is to keep the intensity within a certain range and equal in both ears -corresponding to the situation where the robot is successfully following the leader.

Conventional robotic control follows basically a functional approach [14,18]. A key aspect is the use of a world model which can be updated by sensor inputs. To develop a set of actions to achieve the desired goals the functional approach utilizes a planning module. The planner uses the world model and current sensor inputs to produce the steps toward a goal that constitute the plan of actions. This approach is very brittle especially in its reliance on a pre-developed world model and is very limited in controlling multiple robots. Learning approaches such as those described in this chapter can avoid the drawbacks of a functional approach.

In this chapter we first provide some background on Remote Operated Vehicles (ROVs) and Untethered Unmanned Underwater Vehicles (UUVs) by describing their potential applications and issues involved in their operation. The following section discuses the paradigm of recent memories to generate an adequate training set and collecting memories needed to train an initially 'blank' controller. Next we describe the neural network controller which is trained using genetic algorithms. In the following sections we explain the design and training of reactive and memory based controllers using recent memories. Results from simulation tests of these controllers are shown in the experimental results section.

8.2 ROV/UUV Background

Remote Operated Vehicles (ROVs) are used extensively for salvage operations, ocean floor surveying and numerous inspection activities that support a wide range of underwater commercial activities. In deep water (greater than 1000 ft) an ROV is the platform of choice because of the depth and endurance limitations for human divers. The key disadvantage to an ROV is the requirement for the long tether.

Untethered Unmanned Underwater Vehicles (UUVs) have demonstrated the ability to perform deep-water surveys faster and cheaper than towed vessels [9,10]. UUVs have the potential for supplementing and even replacing ROVs for many deep-water operations because of the cost and problems associated with the tether. One promising scenario for the near future is to use a ROV to control several UUVs in a local work area.

Table 1 compares key factors of tethered vessels to single and multiple UUVs. A tethered vessel clearly has the advantage of power and endurance, but this advantage can be lost when multiple UUVs are deployed from the same host ship.

The factors of deck gear size, launch and recovery time, maneuverability and coverage rate are distinct disadvantages of tethered vessels and directly impact their cost-effectiveness.

Table 8.1 Comparison of UUVs and Tethered Vessels for Deep Water Work

Comparison Factors	ROV or TOW	Single UUV	Multiple UUVs
Power/endurance	+	-	+
Deck gear size	-	+	+
Launch/recovery time	-	+	+
Maneuverability	-	+	+
Coverage Rate	-	+	++
Positioning	-	-	-
Communications bandwidth	+	-	-
Intelligence	+	-	-

Accurate positioning of deep-water vessels remains a difficult and expensive issue. At present this can be accomplished with ultra-short baseline systems on GPS equipped ships that remain directly over the deployed vessels [9]. A tether offers a distinct advantage in the areas of communications and intelligence since it provides high-bandwidth instantaneous communications and allows human intelligence to be in the loop for vehicle operation. The communications, intelligence and multi-vessel navigation capabilities required for deploying multiple UUVs are still in active development and of particular interest for our research approach.

Figure 8.1 illustrates a UUV task force that could be used to conduct an oceanographic survey [5]. Data of interest in such an operation would include bathymetry, acoustic imagery, wave characterization, sediment type, water salinity and temperature, currents, bioluminescence, water clarity, etc.

An important aspect to note is that a survey of this sort could cover large areas (many square miles) and need to be executed in a fairly short period (a few days). Due to the characteristics of the vessels and the sensors involved, high vessel speeds cannot typically be used. Consequently, a large number of vessels are required to accomplish the mission as is shown in the figure.

As with most operations employing a large number of assets, it is not economically feasible or desirable for all vessels to have capabilities of the same type and grade. Bathymetry and acoustic imagery for example, require a vessel that can cruise smoothly at fairly low speeds. Sound velocity profiles (salinity and temperature) need a vessel that is good at performing vertical excursions. Obstacle sensors are not necessarily required for every vessel; a maximum safe depth can be set based on a priori knowledge of the operation area and a few vessels equipped with look-ahead sonar can be deployed ahead of the task force to ensure safe navigation for all vessels. Close inspection of an object of interest would require a vessel with a hover capability and a high-resolution imaging system. This will in general require an active coordination of a robot team [4].

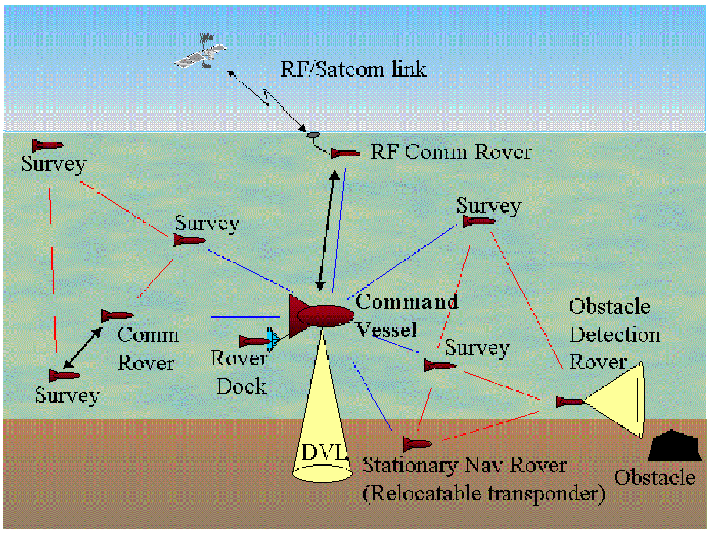

Fig. 8.1 A UUV task force concept for a notional oceanographic mission. The blue lines indicate range/bearing positioning links and the red links are range only positioning links. The black lines indicate communication links.

A command link to the outside world is required for status monitoring and for task force redirection. Instead of equipping every vessel with this capability, this can be served by a vessel that provides a radio/acoustic communications interface to the task force's Command Vessel (CV) as shown in Figure 8.1.

Communications for status monitoring and asset redirection is also required at the task force level, so a communication capability is required between the CV and the rest of the task force. Asset redirection can typically be handled using low-bandwidth communications and the CV could be equipped with a low frequency long-range broadcast acoustic communications system for this task [6,20,26].

Military and commercial applications that would use multiple UUVs share the common problem of positioning and communications. While a commercial application typically has greater flexibility and could put a fixed infrastructure into place or use more ships, these approaches are not always economically desirable. Here a team concept can be used with lower cost UUVs that are co-coordinated to the higher capability UUV leader with an advanced positioning capability. This allows a group travel formation to and from the mission site where the individual UUVs will then carry out their mission specific goals [16,24].

8.3 Memory Based Learning

Relevant related work for this topic includes Q-learning, anytime learning and value-dependent learning. Q-learning [17,25,27] is an algorithm that learns optimal con-

trol strategies through the exploration of its environment. The main strength of Q-learning is that it allows a robot or agent to operate in its environment without a priori knowledge. While it has been applied successfully to specific game playing tasks, it is best suited for well defined problems with finite state spaces and not for dynamic environments since it must exhaustively test alternatives. Anytime learning [13] allows a robot to learn without requiring it to physically execute many trial solutions. However, anytime learning's functionality is derived through extensive simulation that requires significant processing power and a priori knowledge of the environment in which the robot will work. The research effort by Rucci et al. [21] developed 'value-dependent learning' that uses a sensor based approach similar to this research, but requires significant physical repetition by the robot to converge on a solution.

The approach described in this chapter is intended to control autonomous robots in unstructured dynamic environments, such as the undersea environment or in a desert. Environments such as these motivate this approach, which focuses on creating a method for the robot to explore its environment in a non-goal-oriented manner so it can learn what actions lead to success. The environmental exploration routines are intended to provide the robot with memories from which it can learn to achieve its goals, using the reactive and sensor history driven (memory based) learning methods, without having to physically repeat each movement until it reaches a criterion for success.

Learning from recent memories is used so that the robot or agent can avoid regimes of directed testing of trial solutions. The approach records a memory consisting of sensor data and associated action values and learns from this which actions helped to optimize goals. By learning from previous experiences the robot can create a "draft" controller that can be iteratively improved as the robot operates in its environment in fulfillment of its goals. Of primary importance in this strategy is for an agent to be able to identify where in the recent memory the robot's goals were being realized. So the memory must contain sufficient usable examples to learn all the functions needed to realize its goals. If this information is not available, or cannot be generated from what is available, the robot will learn an incomplete strategy. Thus, for this method to be successful, some assumptions are necessary:

1. A goal must be specifiable in terms of sensor values.
2. The system must be able to observe its progress in order to detect when goals are not being met.
3. The data in the recent memory that the learning system is utilizing must contain sufficient information to form a complete strategy for the situation at hand.
4. The learning system must be able to generate a new controller before the environment changes enough to make it irrelevant.

As discussed, the formation following goal can be specified in terms of sensor values in such a way that progress to the goal can be observed. Several approaches have been used to develop a baseline controller (non-adaptive) that would collect an adequate memory for training; the most successful was a random but purposeful

algorithm that utilized slightly modified goal criteria to direct a robot during a memory collection process. This controller uses sensor feedback to continue actions that move it closer to its goal state, where the goal is for example an increasing sound intensity. As long as the feedback indicates that the metric is being met it continues the current action, but as soon as this is no longer valid, it randomly chooses another action. Since the selection of the new action is random, examples of correct actions are distributed relatively evenly. That is, there will be about an equal number of examples in which a right turn was the correct action as there will be for going straight or left.

The structure of the process for a following robot is described next. Since we specifically have an underwater application context, the sensors that are being utilized are some sort of microphones. Acoustic sensing is typically the only feasible choice for such an environment, but the following discussion is not specific to this. The variables used in the subsequent discussions are:

$ML(t)$ - Left Microphone (Sensor) value;
$MR(t)$ - Right Microphone (Sensor) value
$A(t)$ - the current action of the robot: left, straight, right

Now first, for the following robot, if both sensor values are increasing in intensity and close to each other, the robot will go to a neutral state, that is, it goes straight ($A(t)$ = straight). We can express this case by the condition:

$$C_{Sraight} = [(ML(t) > ML(t-1)) \wedge (MR(t) > MR(t-1)) \wedge (Little * f \geq Big)] \quad (8.1)$$

The first two parts of condition represent the gains in intensity. The last term captures the concept of closeness of the sensor values. The values of Big and Little are the larger and smaller of the sensor values respectively and f is a user specifiable parameter (e.g. 1.05).

$$C_{Nochange} = [(ML(t) > ML(t-1)) \vee (MR(t) > MR(t-1))] \quad (8.2)$$

Lastly if neither of these cases apply then the approach will be to attempt a different action. The new action $A(t)$ is a random choice from the set of possible actions {left, straight, right}.

Figure 8.2 shows a graphic example of the robot trajectory taken using the *random but purposeful* training controller. It is evident that this controller will generate a generous set of both good (moving towards the source) and bad (moving away from the source) examples for training a new controller. As noted earlier, a sufficient memory is required to generate a properly functioning controller. In the context of this application we could envision a situation in which the good examples predominately consist of left turns; as a result, the generated neural network controller may not know how to turn right to move towards the source. To contend with this situation, and due to the symmetry in this particular application, a mirroring technique was created that generated 'extrapolated' memories to complete the training set. Essentially this means that for every example where a left turn was executed to get closer to the source, a right turn example was manufactured from this memory and

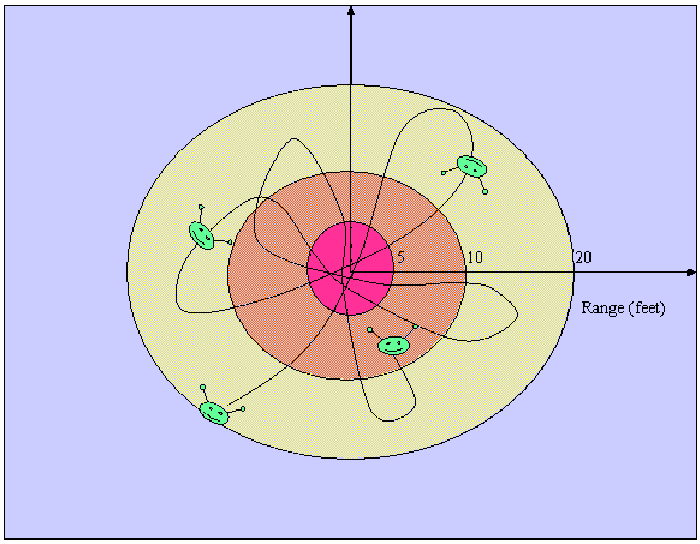

Fig. 8.2 An example path generated by the random but purposeful controller. The sound source is at the center of the plot and the smiley faces with antenna denote the robot and its orientation as it follows the path.

vice versa. Also the positive straight example set takes advantage of the strategy, providing more variety in that example set. A high level description of the algorithm is illustrated below, where a positive example is one in which an action results in the situation in which sensor intensity increases the most:

For each positive example

Swap right and left sensor values
if (action = left) then
    New action ← right;
if (action = right) then
    New action ← left
else
    New action ← straight

In our approach the methods of collecting the memories and using that information to create controllers is addressed. Another equally important consideration is the method in which the memories and controllers developed from them are maintained. For instance the goal may remain constant but the environment may change rapidly, necessitating the use of a previously generated controller, or consider the case where the goal changes but the environment remains unchanged. The storage and management of these behaviors and memories is a robot architectural issue and is not addressed in this paper.

8.4 Genetic Algorithm Training of Neural Network Controllers

The controllers used in this research are based on feed forward neural networks (FFNNs) with one hidden layer, since their structure readily lends itself to rapid generation on an autonomous robot. We utilize a genetic algorithm (GA) to train the neural network in which the GA is used to find the neural network's transfer functions along with the weights. A GA is used to train the network for two main reasons:

GAs provide an efficient and robust means of training neural networks[2], while avoiding many of the problems with becoming trapped in local minima as experienced with gradient techniques[23].

Since the best transfer function for a particular environment may not be known ahead of time, having it be GA selectable decreases the user's need for a priori knowledge.

The GA was selected over more traditional methods such as Back Propagation so that the need for explicit desired values for each training example could be avoided. In our current experiments involving formation maneuvering, this data could be acquired from a simulator, but for other applications the correct motions at each moment are often not known. In these instances a GA can use a fitness function that promotes overall system performance by rewarding controllers that best fulfill a programmed goal.

There are a variety of issues that are involved in the effective and efficient execution of a genetic algorithm. The selection of suitable representation is a crucial component of GA design. Often the particular fitness function which carries out the evaluation of individuals in a population can be the most costly aspect of a GA's execution. Also other parameters such as selection, crossover and mutation schemes must often be adjusted for efficient execution [8].

Although the inclusion of network transfer functions as part of the optimization algorithm has not been widely reported, there is at least one paper describing the use of a GA to optimize the number and placement of radial basis nodes in a modular neural network [15]. That research differs from our effort in that the GA worked with node placement and topology, while the weights were found with more conventional back propagation techniques.

In our approach we use an Elite genetic algorithm to train the neural networks. For an Elite GA, the breeding portion of the population (best scoring individuals) is carried through to the next generation with the remainder of the population being derived using individuals from the breeding portion. A high level outline of the Elite GA algorithm used to find the weights and transfer functions of the feed forward neural network controller is described next.

The population $P = \{p_i\}$ for the GA consists of n individuals (organisms) pi that represent feed forward neural networks. Each $p_i = (w,t)$ where w represents the weight and t the transfer functions of the FFNN respectively:

$$w = (w_1, \cdots w_q) \quad -1.0 \leq w_j \leq 1.0$$

where q = [(number of inputs * number of hidden nodes)+(number of hidden nodes * number of outputs)]

$t = (t_1, \cdots t_r)$ r =number of hidden nodes

where $t_j = 0$ for discrete transfer function: $t_j = 1$ for linear transfer function; $t_j = 2$ for sigmoid transfer function.

The fitness function of the GA is $Fit(p_i)$ which measures the cumulative distance between the follower robot and a lead robot. The follower robot is controlled by the neural network described by individual pi, while the simulation system controls the lead robot. We use a notation topPercent(P) to represent the elite percent of the population P that is directly transferred to the next generation's population. This percentage is a user adjustable parameter with a typical range of 10 to 33 percent.

The Elite GA can be outlined as follows:

Generate an initial population P0 by randomly initializing for each $p_i \in P0$ the values of w and t.
Run the simulation for M generations

Evaluate the population $P0$ using the simulation in the fitness function $Fit(pi)$
Order the evaluated individuals p_i in $P0$ from best to worst $P1 \leftarrow$ topPercent $(P0)$
Use crossover and mutation on the topPercent portion of $P1$ to provide individuals for the remaining portion of the new generation's population $P1$
$P0 \leftarrow P1$

Return the individual p_i from $P0$ with the highest fitness value after M generations

The benefits of using this method are its stable operation and straightforward implementation. This method was originally developed for generating feed-forward neural networks for use in a formation maneuvering simulator [16]. In a typical simulation session, the user will use this algorithm to grow a controller. Usually the GA will build a working controller within 200 generations. For this research it was integrated into an online learning process that operates on our laboratory robots. In this capacity it trains neural networks based on training sets generated by reactive and memory based training processes. Unlike the brute force methods used in the formation maneuvering simulator, these methods take advantage of positive and negative example sets developed using the goal focused exploration described earlier. The next sections will discuss how the memory captured by the focused exploration is processed so that system can avoid excessive use of a priori knowledge and still use the GA to generate controllers.

8.5 Reactive Controller

For a reactive controller, the actions taken at time t are based solely upon the current sensor readings. Consequently, the reactive learning algorithm uses a training set comprised strictly of individual sensor-action values that are marked as good if the action results in an increased sensor value and bad for a decreased value. The recent memory is examined for occurrences of these examples, where the sensor values at time $t+1$ are examined to determine if the action at time t was a good or bad example. One known drawback of this type of controller is that it cannot react to trends that occur over time. A classic example for the formation maneuvering task is the forward-backward ambiguity: a following robot with a reactive controller cannot discern if it is heading directly away or towards the leader if the amplitude in both sensors remains the same; without an additional observation to detect this trend a robot could wander away from its leader in the wrong direction. Figure 8.3 and the steps below illustrate the process for creating the reactive controller using recent memories.

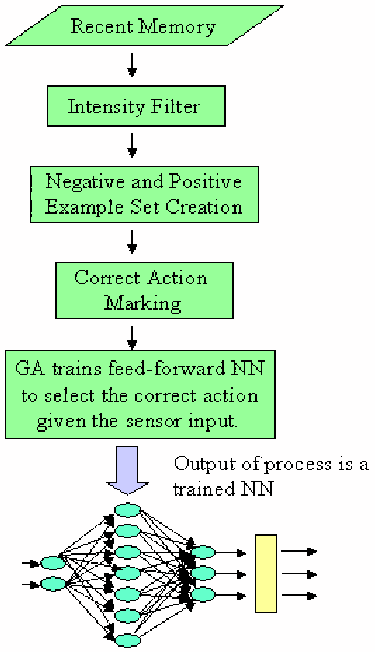

Fig. 8.3 The reactive learning process. Key to the process is the intensity filter. It ensures that the memory is composed of examples for which actions at time t had a direct influence on sensor values at $t+1$. As a part of a robot's architecture, this process would be called when an observer function notices goals are not being met.

A robot 'trainee' records a memory $\boldsymbol{M}$ consisting of sensor values and actions taken while seeking either a fixed or moving sound source. The robot could be guided by an operator or an algorithm such as the classic logic method, another neural network, or the *random but purposeful* controller. The sensor values and actions at a given time, denoted $SA[t]$, is a 3-tuple (ML, MR, A) where ML / MR are the left / right microphone values respectively and A the action taken. A recent memory $\boldsymbol{M}$ will be defined as a sequence of sensor action values $SA[0] \cdots SA[n-1]$.

Now we can organize training sets of positive examples, P, and negative examples, N, from M by using the changes in the total sound energy $E[t]$ where

$$E[t] = ML[t] + MR[t] \tag{8.3}$$

The classification of the $SA[t]$ values will be based on the total energy at times t and $t+1$. So

$$P = P \cup SA[t] \quad if\ E[t+1] \geq E[t] \tag{8.4}$$

or

$$N = N \cup SA[t] \quad if\ E[t+1] < E[t] \tag{8.5}$$

Using the set of positive examples as a new training set, the GA based technique can generate a new neural network. In order to increase the efficiency of the GA, the current best neural network can be used to seed the initial population.

We have described only the use of positive examples for which we present experimental results in section 7. However, it was also demonstrated in our laboratory simulations that negative examples could also be used by rewarding a neural network that is being trained if it takes a different action than the one associated with the example.

8.6 Memory based Controller

Figure 8.4 below shows the neural network configuration used for demonstrating learning with a memory based controller. The memory based controller behavior is achieved by using past sensor inputs in addition to the current input. The choice of the number of past inputs dictates the span of time over which the controller will be able to observe trends and also its ability to react quickly to sudden changes. The motivation for testing learning with this type of controller was to ensure proper handling of the forward-backward ambiguity mentioned earlier.

Creating a training set for this type of controller proved to be significantly more challenging and was accomplished by splitting the recent memory into four categories with respect to the average sensor intensity: rising, dropping, stable and fluctuating. These cases are illustrated in Figure 8.5, with the 4 categories referred to as the 'principal parts' of the memory.

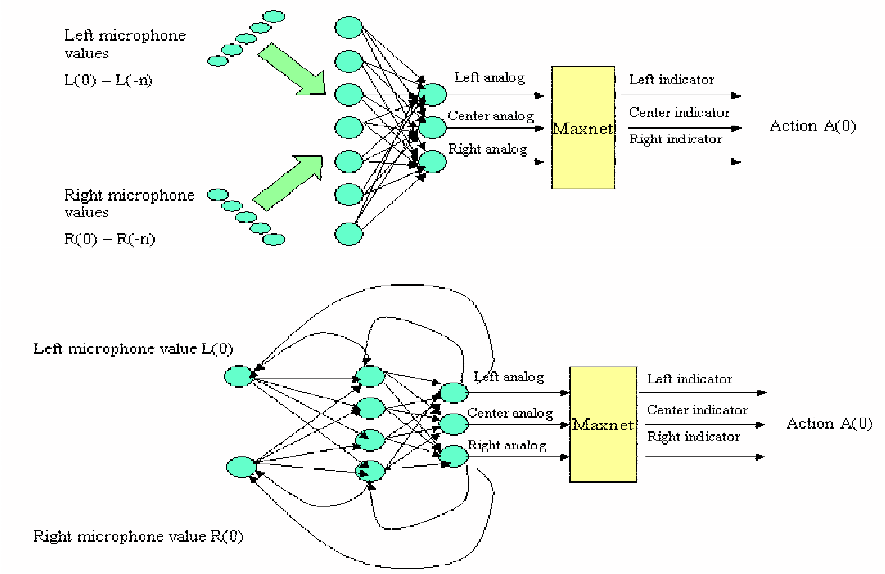

Fig. 8.4 A feed forward neural network with past inputs.

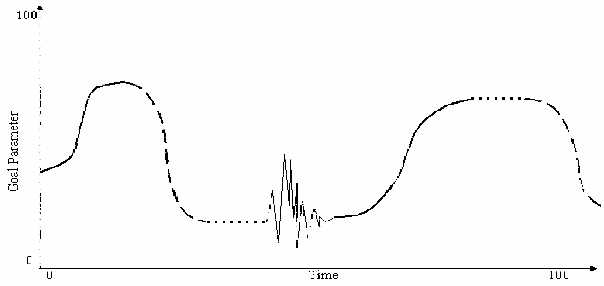

Fig. 8.5 A time series of the goal parameter. The line is drawn in different line styles to illustrate the principal parts that it is made of. The bold part of the line fits in the category of rising parts, the dashed part in the dropping category, the dotted part in remaining stable category, and the thin part near the middle of the time series is the fluctuating category.

Since the controller uses 4 consecutive time inputs, the training set is composed of groups of 4 consecutive sets of sensor-action values obtained from the recent memory M. The classification of the memory into its principal parts will be based on the sound intensity value I (the average of ML and MR). So now $SA[t]$ becomes a 4-tuple including I : (I, ML, MR, A) and a recent memory M of length n is again a sequence: $SA[0]..SA[n-1]$.

Next we will describe the characteristics of the principal parts categories:

RC: rising category where $I(t) > I(t-1)$. This is further subdivided into subsets in which the action is consistent.

RC_left - all actions are left turns
RC_straight - all actions are straight
RC_right - all actions are right turns

Whether any given series of 4 consecutive SA values (referred to as a 'run') in the $\boldsymbol{M}$ belong to these categories is determined by applying the following rules time sequentially throughout the memory:

$SA[t] \in$ RC_left if $I(t) > I(t-1) \wedge A(t) =$left

Membership in RC_straight and RC_right are defined similarly.

SC: stable category where $I(t) = I(t-1) \pm \varepsilon$; ε is a designer specified small value. This is further subdivided into subsets in which the action is consistent: SC_left, SC_straight and SC_right as above except $=$ is used instead of $>$, relative to I in the rules.

DC: dropping category where $I(t) < I(t-1)$. Again we have subsets in which the action is consistent: DC_left, DC_straight and DC_right and for which $<$ is used instead of $>$ for the rules.

FC: fluctuating category. All series of 4 SA values in the memory that do not fall into the RC, SC, or DC categories are placed in the fluctuating category.

The training sets are formed from the RC, SC and DC groups, with a slight variation of the DC components. For the RC and SC groups the action taken during each particular run is assumed to be correct, so the neural networks in training are rewarded when they take this same action. However, when the goal parameter is dropping the correct action is unknown. To determine the action to be used for a run that falls into the dropping category, an adjacent or overlapping rising run must be found. Once the overlapping example is found, its action is used as the correct action for the dropping run. It is used because it is the first correct action that occurs since the negative energy gain. If there is no adjacent or overlapping rising run, the dropping run is not used. Fluctuating runs are not used to create training examples at this time because there is no programmatic method of determining what a correct action is for these states.

8.7 Experimental Results

Quantitative testing was conducted with a LabView [19] simulation that measured the distance and the heading differences between the leader and follower robot in a simple maneuvering task. In the test, illustrated in Figure 8.6, the lead robot traverses two laps around a preprogrammed course. The tests are started with the robots in formation, by maneuvering the leader-follower pair to the starting position and then engaging the automatic navigation control for the leader robot [11,12,22]. The robots' turning rates and speeds were set the same for all tests. The tests are started in the upper left hand corner of the track, when the leader/follower combination is in formation and tracking. Tests were also conducted using ActivMedia robots [1] equipped with microphones and speakers. The reactive feed forward neural nets (FFNNs) generated were able to control the robots in these tests although the sensor history network has not yet been tested with the robots.

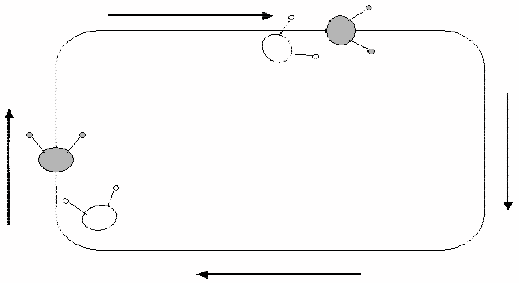

Fig. 8.6 Test course for the following robots. The lead robot (gray robot) is controlled by the simulator. The follower robot (clear robot) is controlled by the current controller being tested. Over two laps the inter-robot distance and course differences are recorded.

Figure 8.7 shows the average distance between the leader and a follower controlled by a typical reactive neural network trained as previously discussed. The distance between the two robots remains very stable except during the turns. The turns are the eight places in the curve where the distance fluctuates. Figure 8.8 next shows the heading differences measured during the same test. The heading difference grows large after the leader turns since it takes the follower a finite time to respond to the changing relative position of the leader and effect its own course change to follow.

For comparison, Figures 8.9 and 8.10 show a similar test run using a classic logic controller based on Braitenberg's robots [7,17]. The FFNN controller is seen to hold its distance almost as consistently as the classical controller. While the classic con-

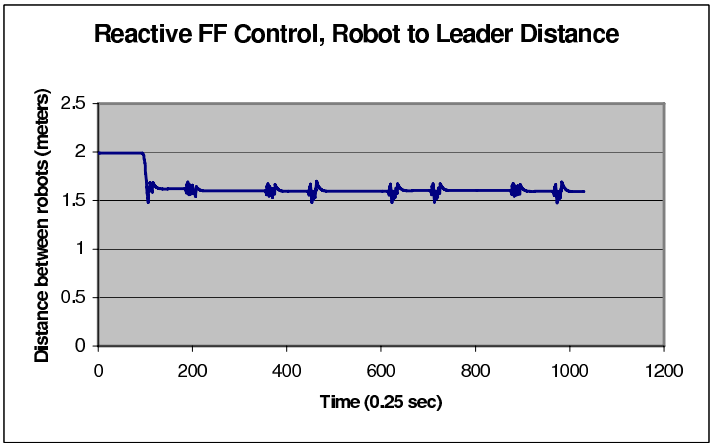

Fig. 8.7 The leader/follower distance over two laps. The follower robot is controlled by a reactive FFNN. As can be seen in the figure, the robot closed the distance to the leader and maintained it throughout the run except during the turns.

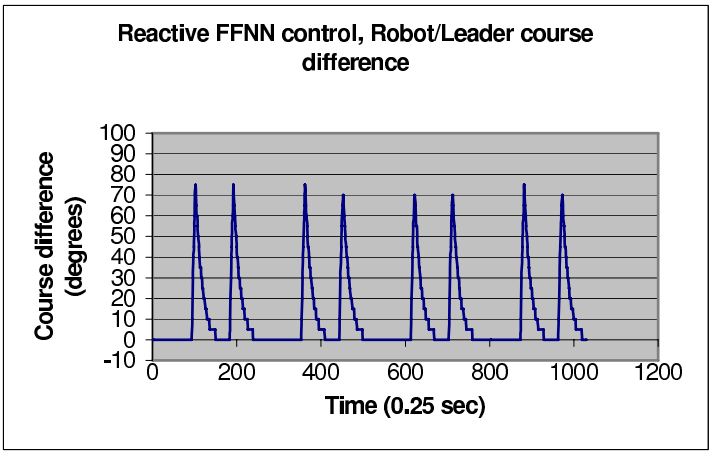

Fig. 8.8 The heading difference between the leader and the follower robot controlled by a reactive FFNN during a two lap test in the simulator. Notice that each of the 8 peaks (the peak denotes a course change by the leader) occur at about the same time as the fluctuations occur in the curve in distance plot.

troller has a slight advantage on the robot-to-robot distance, the FFNN handles the turns a little more smoothly, as can be seen by examining the oscillations in the robot-to-robot distance near the turns in Figures 8.7 and 8.9. Also, it was noted that the classic controller tends to oscillate more about the proper heading, depending upon the operator-specified 'dead-zone'. This is observed in the thick lines in the course difference figure at the end of each turn, where the following robot oscil-

lated about the desired course. One of the advantages of a learning system is the elimination of heuristically set parameters such as the dead zone size.

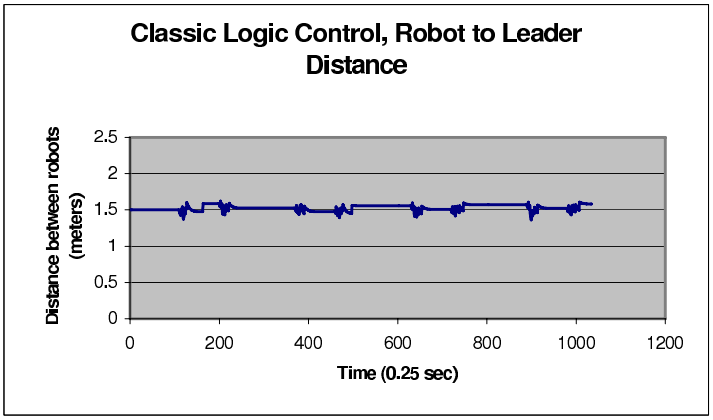

Fig. 8.9 The leader/follower distance for the two lap test where the follower is using the classic logic controller.

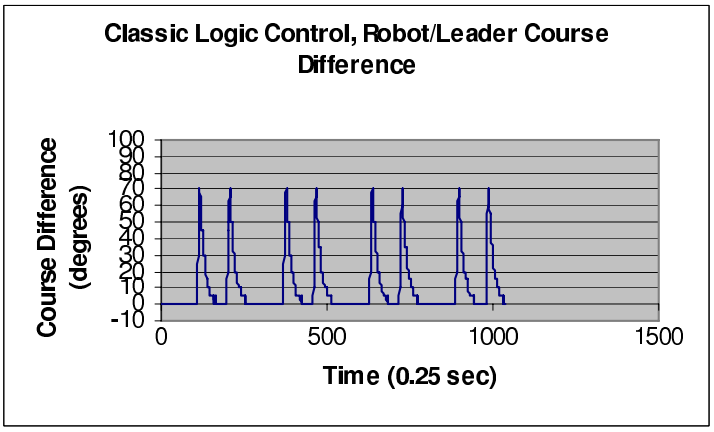

Fig. 8.10 This figure shows the course difference between the leader and follower where the follower is using the classic logic controller. Each peak corresponds to when the leader is making a turn.

Results for the principle parts (memory based) controller are shown in Figures 8.11 and 8.12. As can be seen in these graphs the simulated robot was able to follow successfully, although it took much longer than the reactive controller to steady onto a heading. This result is anticipated since the reactive controller responds only to the last set of sensors values while the principle parts controller looks for a trend in the last n values and will typically be slower to respond.

Tests were also conducted to demonstrate that the principle parts controller could properly deal with the forward/backward ambiguity problem, which is a known problem for the reactive controller. The outcomes of the tests were mixed. Holding the memory-based robot still (zero velocity), but letting it control its direction and moving the source directly at it quick enough for there to be a sensor gradient present in its input, allowed the robot to track the source. Slowing the source's approach to where the gradient in the sensor history did not exist caused it to behave as the reactive system.

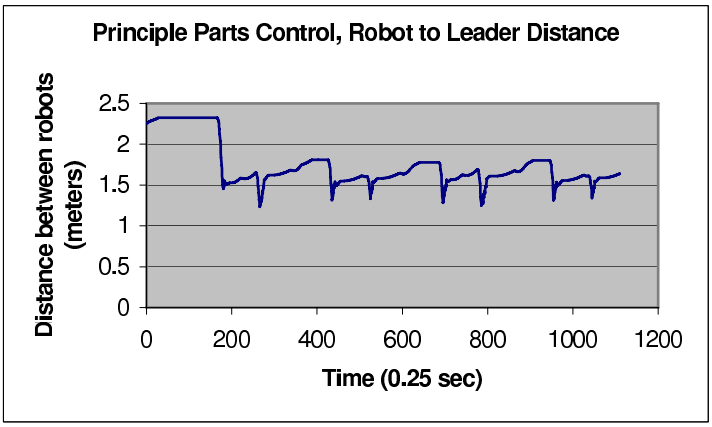

Fig. 8.11 The leader/follower distance for the two lap test when the follower is using the principle parts controller.

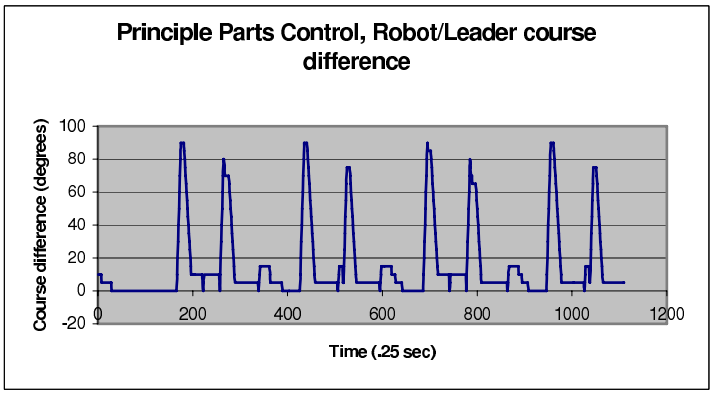

Fig. 8.12 This figure shows the course difference between the leader and follower for the principle parts controller. Each peak corresponds to when the leader is making a turn. Notice that this controller takes more time to complete turns than the reactive controllers.

8.8 Conclusions

Learning using recent memories was demonstrated successfully with both a reactive and memory based controller. Simulation testing with the formation following application showed promising results, and the reactive controller was also tested successfully on land robots. The use of the genetic algorithm to train a neural network controller based on selected recent memories gathered from the *random but purposeful* exploration was observed to be efficient and suitable for implementation in a real-time learning system. This work represents a new approach to in situ learning for an autonomous robot based on experiences (sensor/action pairs), without requiring exhaustive testing of all possible actions or the use of a simulator requiring a substantial amount of a priori knowledge about the operating environment.

Future work may involve the development of a graph learning algorithm which can create a controller whose purpose is to achieve a specific goal. A directed graph can be built and updated from memories recorded as the robot explores and operates in the environment. The graph's structure can represent the robot's experiences in a topological manner, preserving the temporal sequencing information. This could be used to train a neural network or by traversing the graph in real time, directly controlling the robot.

Acknowledgement

We would like to thank the Office of Naval Research through the Naval Research Laboratory for sponsoring this research.

References

1. ActivMedia Robotics (2002) Pioneer 2 Operations Manual, Version 1-1, Petersborough, NH.
2. Angeline P, Saunders G, Pollack J (1993) An Evolutionary Algorithm That Constructs Recurrent Neural Networks, IEEE Transactions on Neural Networks, 5: 54-65.
3. Balch T, Arkin R (1998) Behavior-Based Formation Control for Multirobot Teams, IEEE Trans. on Robotics and Automation, 14: 926-939.
4. Balch T, Parker L (eds) (2002) Robot Teams: From Diversity to Polymorphism. AK Peters. Wellesley, MA.
5. Bourgeois B, Petry F, Harris M, Alleman P (1999) A GIS Integration Approach for Dynamically Reconfigurable Surveys. Hydrographic Journal, 91: 3-11.
6. Bradley M (1996) Environmental Acoustics Handbook 2nd Edition.
7. Braitenberg V (1986) Vehicles: Experiments in Synthetic Psychology, MIT Press, Boston MA.
8. Buckles B, Petry F (1992) Genetic Algorithms: Introduction and Applications, IEEE Computer Society Press, Los Alamitos, CA.
9. Chance T, Kleiner A (2002) The Autonomous Underwater Vehicle (AUV): A Cost Effective Alternative to Deep-Towed Technology. Integrated Coastal Zone Management, 8: 65-69.
10. Church R, Warren D (2002) New Technologies Rewrite History. Hydro International, 6: 56-59.

11. Desai J, Ostrowski J,. Kumar V (2001) Modeling and Control of Formations of Nonholonomic Mobile Robots. IEEE Trans. on Robotics and Automation, 17: 905-908.
12. Fierro R, A. Das A, Kumar V, Ostrowski J (2001) Hybrid Control of Formations of Robots In: Proc. IEEE Int. Conf. on Robotics and Automation, Seoul, Korea, pp. 157-162
13. Grefenstette J, Schultz A (1994). An evolutionary approach to learning in robots In: Proc Learning Workshop on Robot Learning, New Brunswick, NJ, pp. 47-56.
14. Holland J (2004) Designing Autonomous Mobile Robots, Elsevier.
15. Jiang N, Zhao Z, Ren L (2003) Design of Structural Modular Neural Networks Genetic Algorithm. Advances in Engineering Software, 34:17-24.
16. McDowell P, Chen B, Bourgeois B (2002) UUV Teams, Control From A Biological Perspective. In: Proceedings of the Oceans 2002 MTS/IEEE Conference, Biloxi MS, pp 331-337
17. Mitchell T (1997) Machine Learning, McGraw-Hill, New York.
18. Nehmzow U (2003) Mobile Robots: A Practical Introduction 2nd Ed., Springer.
19. NI LabVIEW Manual (2000), National Instruments, Austin TX.
20. Proakis, J., Sozer, E, Rice J, Stojanovic M (2001) Shallow Water Acoustic Networks, IEEE Communications Magazine, 39: 114-119.
21. Rucci M, Wray J, Edelman G (2000) Robust localization of auditory and visual targets in a robotic barn owl, Robots and Autonomous Systems 30: 181-193
22. Sgorbissa A, Arkin R (2003) Local Navigation Strategies for a Team of Robots. Robotica, 21: 461-473.
23. Sofge D, White D (1992) Applied Learning: Optimal Control for Manufacturing, In: Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches, White D, Sofge D, (Eds.), Van Nostrand Reinhold, pp 272-273.
24. Stroupe A, Balch T (2005) Value-Based Action Selection for Observation with Robot Teams Using Probabilistic Techniques Journal of Robotics and Autonomous Systems. 50: 85-97.
25. Touzet, C (2004) Distributed Lazy Q-Learning for Cooperative Mobile Robots, Int. Jour. Of Advanced Robotic Systems, 1: 5-13.
26. Urick R (1983) Principles of Underwater Sound, 3rd Edition, McGraw-Hill.
27. Watkins C, Dayan P (1992) Q-Learning, Machine Learning, 8:279-292.
28. Yamaguchi H, T. Arai T (1994), Distributed and Autonomous Control Method for Generating Shape of Multiple Robot Group. In: Proceedings of IEEE Int. Conf. on Intelligent Robots and Systems, pp. 800-807