Learning Coordinated Behaviors for Control of a Simulated Race Car*

Larry D. Pyeatt **Adele E. Howe** **Charles W. Anderson**
Computer Science Dept, Colorado State Univ., Ft. Collins, CO 80523
email: {pyeatt,howe,anderson}@cs.colostate.edu
category: Control, navigation and planning
This work is not in submission or press to any other conference.

Abstract

The demands of rapid response and the complexity of many environments make it difficult to decompose, tune and coordinate reactive behaviors while ensuring consistency. We hypothesize that complex behaviors should be decomposed into separate behaviors resident in separate networks, coordinated through a higher level controller. To explore these issues, we have implemented a neural network architecture as the reactive component of a two layer control system for a simulated race car. By varying the architecture, we tested whether decomposing reactivity into separate behaviors leads to superior overall performance and learning convergence. Based on these results, we further modified the architecture to produce a race car that is competitive with publicly available solutions.

1 Introduction

Reactive systems have been favored for applications requiring quick responses, such as robotic or process control applications. Ensuring consistency or coordination between reactions is necessary in order to discourage self-defeating behaviors and encourage progress towards achieving goals. This research examines the questions of whether reactivity should be divided into specific behaviors or incorporated in a monolithic structure, and how reactivity can be coordinated in a two layer control system.

*This research was supported in payment by ARPA-AFOSR contract F30602-93-C-0100 and by NSF Research Initiation Award #RIA IRI-930857.

To explore these questions, we have implemented and tested three two-layer control systems for a simulated robot race car. The control systems consist of a high level coordinating mechanism and a set of low level reactive components. The reactive components may be implemented with neural networks or by using heuristics. The heuristic reactions are set up by the designer and do not change. The neural network reactions use a form of reinforcement learning to improve performance. The high level coordination mechanism heuristically switches control between the low level behaviors and assigns reinforcements to the reinforcement learning components.

Neural network approaches have been used for controlling autonomous robots for several years. Krishnaswamy [5] demonstrated a structured neural network approach to controlling a robotic manipulator. Liu [7] used neural networks to control grasping of a robotic hand. Comoglio [4] presented results of simulations for the control of an autonomous underwater vehicle using neural networks. The adaptability and noise rejection of a neural based robot controller was studied by Poo [8], who found that the neural based controller performed better than a standard model based control algorithm. Bullock [3] demonstrated a neural network system for positioning a robotic manipulator with various tools attached and under various hardware failure conditions.

Although previous work has demonstrated that it is possible to use neural networks for robotic control, the question remains as to *how* to build a neural network controller. The remainder of this paper describes what we have learned about building a simulated automobile racer. In section 2 we introduce the task and the control architecture. In section 3 we present the results from experiments. In section 4 we outline some of the improvements that we have made to the control system and present the results of these improvements. Finally, we conclude in section 3 with a summary of our findings and directions for future work.

2 Control Architecture

Our testbed is the Robot Automobile Racing Simulator (RARS)[10], which is designed to allow control systems designers to compete in simulated automobile races. The official RARS races are held regularly to allow competition between various heuristic and adaptive control systems. RARS can simulate 1–16 different agents simultaneously at faster than real time speed. Each agent is implemented as a RARS subroutine. Each agent routine receives information about its position, current velocity, distance to the end of the current track segment, curvature of the track, and relative position and velocity of nearby cars. The agent routine calculates signals for acceleration and steering, which are returned to RARS for updating the simulation.

Our basic architecture consists of two low level controllers and a high level coordinating mechanism. One low level controller is heuristic and the other is implemented as an actor-critic reinforcement learning architecture similar to those used by Barto, Sutton and Watkins [2], Anderson [1], and Lin [6]. The high level coordinating mechanism switches vehicle control between the heuristic strategy and the reinforcement learning mechanism as appropriate. If the car leaves the track (i.e., crashes) or is going in the wrong direction around the track, the reinforcement learning mechanism is disabled temporarily while the heuristic control strategy pilots the car back onto the track or turns it around.

Two distinct control signals are required for driving the car. This naturally subdivides the required low level controllers into two primary reactive behaviors: accelerating and steering. This arrangement requires the control system to make decisions about two complex interacting behaviors for each time step. Performance is directly influenced by the interaction of acceleration and steering. Thus, for the control system to perform well, it must coordinate both behaviors. To facilitate the learning of coordination, each network is provided with the set of actions chosen at the previous time step.

To simplify the design, we divided each control signal into three possible actions. The three actions for setting the acceleration are to accelerate by 5 ft/sec, decelerate by 5 ft/sec, or remain at the same speed. RARS limits the amount of acceleration to the physical constraints of the simulated system. The possible steering commands are to adjust steering to the left by 0.05 radians, do not adjust steering, or adjust steering to the right by 0.05 radians.

The reinforcement learning network is an actor-critic architecture that uses two feed-forward neural networks, both trained with the back-propagation temporal difference algorithm [1, 9]. A stochastic action selector forces the action network to explore the space of possible actions by occasionally choosing an action that is not the one selected by the action network. The probability of choosing a random action is set to 0.01 for our experiments.

The inputs to the network describe the current state of the car. The inputs consist of speed, steering angle, command speed, speed in the direction normal to the direction of the track, curvature of the current section of track, curvature of the next section of track, the distance to the end of the current track segment, and the actions chosen on the previous time step. The inputs are scaled so that the values are between ± 1.

We tested three alternative network architectures. The SS approach uses a single critic network and a single action network to control both acceleration and steering. SM uses a single critic network but separate action networks for acceleration and steering. MM uses separate critic and action networks for acceleration and steering.

The reinforcement signal is constructed from the output of the environment simulator. After some experimentation to check that the signal would provide the correct response, we settled on one signal for the first two approaches (SS and SM) and two signals for the third (MM). For SS and SM, the reinforcement signal is -1 if the car is off the track, moving too slowly or just completed a lap more slowly than the time before and 0 otherwise. For the third approach (MM), the same rule is used to calculate the reinforcement signal for the acceleration network, but the reinforcement signal for the steering network does not include a penalty for moving too slowly because the steering behavior is not directly responsible for the speed of the car.

3 RARS Results

Our primary design hypothesis was that implementing multiple behaviors in *separate* control mechanisms simplifies generalization and expedites scale up. We tested this hypothesis by implementing three reinforcement learning networks for the low level behaviors. One network is monolithic (SS), one is totally modular (MM),

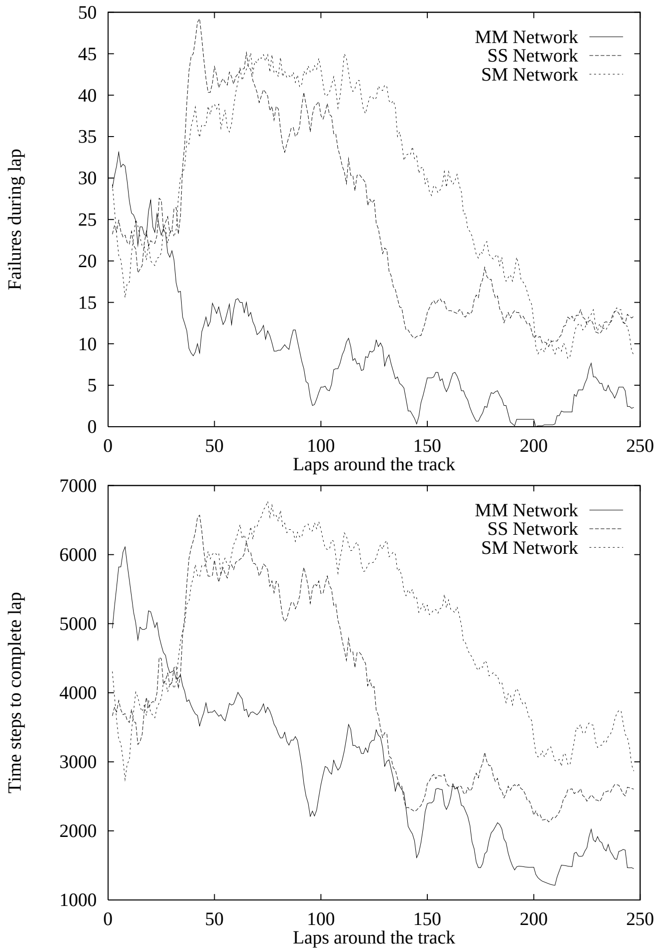

Figure 1: Learning performance for the three reinforcement learning algorithms.

and one is in between (SM). The question is whether or not the modular approach results in the best overall performance.

Each network implementation was tuned, trained, and tested on a single track in RARS in order to establish a set of parameters that yields good performance. Each network was then loaded with random weights and trained for 250 laps around the track with no other cars on the track. Performance was measured in length of time needed to learn and the failure rate after a certain number of races. A failure is considered to have occurred whenever the car leaves the track for any reason.

Each network architecture was run with various parameter settings to find those that resulted in the best performance. In all of the networks, a learning rate of 0.0001 and a momentum term of 0.25 resulted in good performance. We also found that adding direct connections between the input and output nodes improved performance in all cases. The best size of the network was architecture dependent: 6 hidden nodes in each network for SS, 5 for SM and 4 for MM.

Figure 1 compares the number of failures for each reinforcement learning network during training and the time to complete each lap. The MM approach finds a good solution with less training than the other two approaches. We believe that the MM approach exhibits superior performance because the two utility networks are trained with slightly different reinforcements, to reflect their differences in behavior. Since credit is assigned to actions more accurately, it takes less time to learn appropriate actions for each situation.

Additionally, although the *total* number of weights in the MM approach is actually higher than in the other two approaches, the modular structure expedites training. The two graphs are very similar because fewer failures means less time spent recovering from failure. The interaction is reflected in a slowly improving cycle of 3 steps forward and 2 backward. The MM network finds a solution fails rarely or not at all, but then the time to complete a lap is reduced until failures begin again. The network then tries to reduce the number of failures. Over time, this behavior minimizes both the time to complete a lap and the rate of failure.

4 Improvements

After settling on an overall configuration, we began a series of improvements to the reinforcement learning modules in our MM approach. We made changes to the input representation, learning algorithm and reinforcement protocol. First, we changed the input representation. Instead of simply feeding the inputs directly to the network, the inputs are quantized into bins. For example, the steering input is represented by 21 inputs, one of which is set to one while the remainder are set to zero. The second change was to use Q learning instead of the actor-critic approach to reinforcement learning. The third change was to adopt a backwards view of reinforcement learning. During training, the states visited by the car are saved in a stack structure but no learning is performed until a positive reward is received. When a positive reward is received, credit is assigned and the network is trained on all states in the stack, working backwards in time until the stack is emptied. The stack then begins to grow again until another positive reward is received.

These modifications have resulted in significantly increased performance. Figure 2 shows the learning performance of the new network compared to our original MM network. The new approach learns to avoid failure much more rapidly and the cyclical behavior found in the earlier approach is less apparent. The time to complete a lap is much lower with the new approach and is competitive with the heuristic control strategies which are supplied with RARS. Some of the best heuristic strategies can complete a lap in under 600 time steps on average while the worst heuristic strategies take more than 800 time steps. After sufficient training, the Q learning approach can complete a lap in about 700 time steps on average.

5 Future Work and Conclusions

The improvements to the network configuration produced a competitive race car. Unfortunately, because we made several modifications simultaneously, we cannot at this point attribute credit to any one of the changes, although we believe Q learning is the primary contributor. We are currently performing a more controlled comparison of each modification to the original MM reinforcement learning approach.

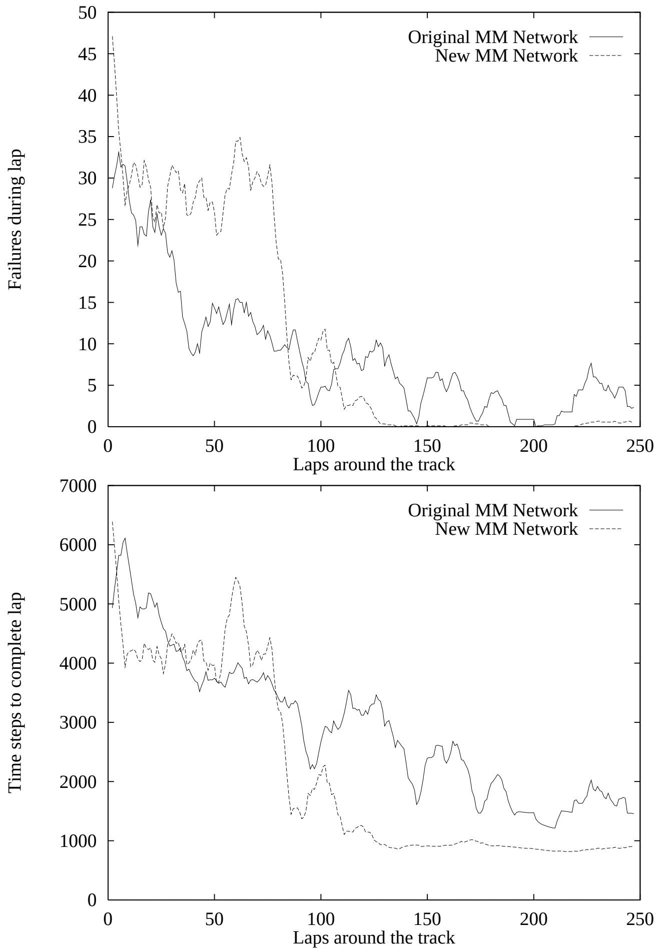

Figure 2: Learning performance for the improved learning strategy.

We have also begun experiments with adding two additional low level behaviors for passing and performing a pit stop. The new behaviors are both implemented as reinforcement learning strategies. The high level control mechanism has been modified to switch control to these behaviors when appropriate. As expected, we found it easy to implement the new behaviors in the MM configuration and results are encouraging. However, it is still too early to report them with confidence.

Our results strongly support the idea that implementing multiple behaviors in separate control mechanisms expedites coordination. The MM approach is more modular than the other two approaches and as new behaviors are required, they can be added. More promising still, the Q learning approach was able to perform comparably to the heuristic control strategies available with RARS.

References

[1] Charles W. Anderson. Strategy learning with multilayer connectionist representations. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 103–114, 1989.

[2] A.G. Barto, R.S. Sutton, and C.J.C.H. Watkins. Learning and sequential decision making. Technical report, COINS Technical Report 89-95, Dept. of Computer and Information Science, University of Massachusetts, 1989.

[3] Daniel Bullock, Stephen Grossberg, and Frank H. Guenther. A self-organizing neural network model for redundant sensory-motor control, motor equivalence, and tool use. In *International Joint Conference on Neural Networks*, pages 91–96, Baltimore, 1992. IEEE.

[4] R. F. Comoglio and A. S. Pandya. Using a cerebellar model arithmetic computer (cmac) neural network to control an autonomous underwater vehicle. In *International Joint Conference on Neural Networks*, pages 781–786, Baltimore, 1992. IEEE.

[5] Gita Drishnaswamy, Marcelo H. Ang Jr., and Gerry B. Andeen. Structured neural-network approach to robot motion control. In *International Joint Conference on Neural Networks*, pages 1059–1066. IEEE, 1991.

[6] Long-H Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3/4):69–97, 1992.

[7] Huan Liu, Thea Iberall, and George A. Beckey. Neural network architecture for robot hand control. In *Proceedings of IEEE International Conference on Neural Networks*. IEEE, July 1989.

[8] A.N. Poo, M.H. Ang Jr., C.L. Teo, and Qing Li. Performance of a neuro-model-based robot controller: adaptability and noise rejection. *Intelligent Systems Engineering*, 1(1):50–62, 1992.

[9] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[10] Mitchell E. Timin. Robot Auto Racing Simulator. Anonymous ftp from `ftp.ijs.com:/rars`, 1995.