A COMPUTER SIMULATION OF BRAITENBERG VEHICLES

THESIS

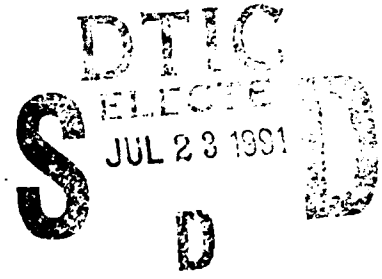
Eric B. Werkowitz

AFIT/GCS/ENG/91M-01

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/91M-01



A COMPUTER SIMULATION OF BRAITENBERG VEHICLES

THESIS

Eric B. Werkowitz

AFIT/GCS/ENG/91M-01

Approved for public release; distribution unlimited

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1991	Masters's Thesis

4. TITLE AND SUBTITLE
A COMPUTER SIMULATION OF BRAITENBERG VEHICLES

5. FUNDING NUMBERS

6. AUTHOR(S)
Eric B. Werkowitz

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6503

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCS/ENG/(91M-05)

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

In his treatise, Vehicles: Experiments in Synthetic Psychology, Valentino Braitenberg used simple, autonomous vehicle designs to illustrate the principles behind animal nervous system organization and operation. The goal of this effort was to produce a computer program to allow a researcher to experiment with these concepts and to analyze their performance using methods employed by experimental psychologists. The resulting program allows the user to design vehicles that respond to changes in their environments and that have the ability to adapt their behavior, using a learning algorithm developed by Teuvo Kohonen. The vehicle designer is free to select sensor attributes, numbers of neurons, learning periods, environments, and starting "knowledge." The vehicles may be analyzed by tracking their success at locating "food" in their environment. The food is located close to stimuli to which the vehicle sensors can respond. Food discovery triggers the learning algorithm, adapting behavior to improve food finding-performance. The initial evaluations failed to provide convincing proof that the simple vehicles tested had succeeded in totally adapting to their environments.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Artificial Intelligence, Self Organizing Systems, Learning Machines, Adaptive Systems, Robots, Autonomous Navigation	132
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

AFIT/GCS/ENG/91M-01

A COMPUTER SIMULATION OF BRAITENBERG VEHICLES

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

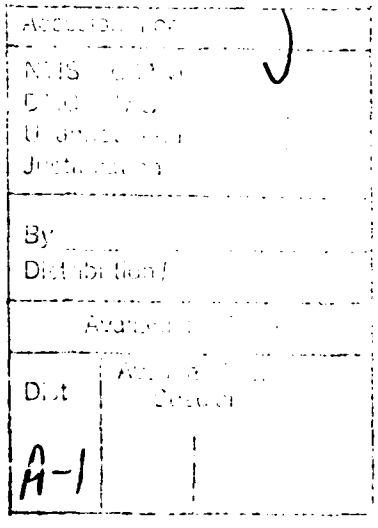
In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Systems

Eric B. Werkowitz, B.S

March 1991



Preface

The purpose of this effort was to simulate simple, biological learning behavior using an artificial neural network to imitate an animal nervous system. The successful application of unsupervised training in this manner could result in a new era of machine control design.

Although the test results were not as promising as I had hoped, the usual thesis time-limits precluded complete evaluation of the resulting design. The Turbo Pascal code presented in the Appendix provides a basic, highly flexible medium for exploring unsupervised training in a manner similar to that used by comparative psychologists.

I owe thanks to many people for assistance in completing this project. First, my daughters, Roxanne and Tiffany, deserve appreciation for their patience with "Dad" for all the times he had to say, "I'm sorry, Sweetie, but I've got to work on my thesis." My wife, Sherry, also deserves much thanks for saying "yes" when I had to say "no."

I also need to thank my thesis committee chairman, Dr Matthew Kabrisky. This thanks is not only for his support and encouragement during the conduct of this project, but also for his patience during the numerous false starts on other topics. Thanks, also, to the other members of the committee, Maj Steve Rogers and Dr Frank Brown, who offered many comments to improve the project research and written thesis.

Others who assisted this project are my co-workers in the Deputy Chief of Staff for Development Planning, Directorate of Mission Area Planning (ASD/XRS). Thanks to Frank Erdman, Dave Webber, and Maj Fred Zietz for their thoughtful comments and critiques. I also need to acknowledge the patience of my boss, LtCol Dave Shoemaker, during this period in which I know I was more than a little distracted from my work duties.

Finally, I thank Julius Becsey who died suddenly and unexpectedly during the course of this study. Julius provided much encouragement and technical support to me during the early stages of this project. He was an excellent chemist, a bright mathematician, an inspired programmer, and a good friend. I miss him every day.

Table of Contents

List of Figures

List of Tables

Abstract

In his treatise, Vehicles: Experiments in Synthetic Psychology, Valentino Braitenberg used **gedanken** experiments based on simple autonomous vehicle designs to illustrate the principles behind animal nervous system organization and operation. The goal of this effort was to produce a computer program to allow a researcher to experiment with these concepts by analyzing their performance in a manner similar to those used by experimental psychologists.

The resulting program allows the user to design vehicles that respond to changes in their environments and that have the ability to adapt their behavior, using a learning algorithm developed by Teuvo Kohonen. The vehicle designer is free to select sensor attributes, numbers of neurons, learning periods, environments, and starting "knowledge."

The vehicles may be analyzed by tracking their success at locating "food" in their environment. The food is located close to stimuli to which the vehicle sensors can respond. Food discovery triggers the learning algorithm, adapting behavior to improve food finding-performance.

The initial evaluations failed to provide convincing proof that the simple vehicles tested had succeeded in totally adapting to their environments. Suggestions for further research are offered.

A COMPUTER SIMULATION OF BRAITENBERG VEHICLES

I. Introduction

Many aspiring young scientists marvel at the varied and complex behaviors evident among the inhabitants of a common ant hill. Close inspection of the bustling, seemingly chaotic, motion of the many colony members reveals individuals responding to unknown instructions for the good of the many. There are obvious specialists in a number of occupational fields ranging from nursemaids, to foragers, to soldiers. Even a single entity from this microdomain is amazing to watch. It may perform a foraging task that necessitates following an established trail over a circuitous route to some source of food found by an earlier explorer. Once there it will isolate a piece of the food small enough to carry and, retracing its steps, return to the colony with the load to be stored and shared with the other inhabitants (Goetsch, 1957:109-115).

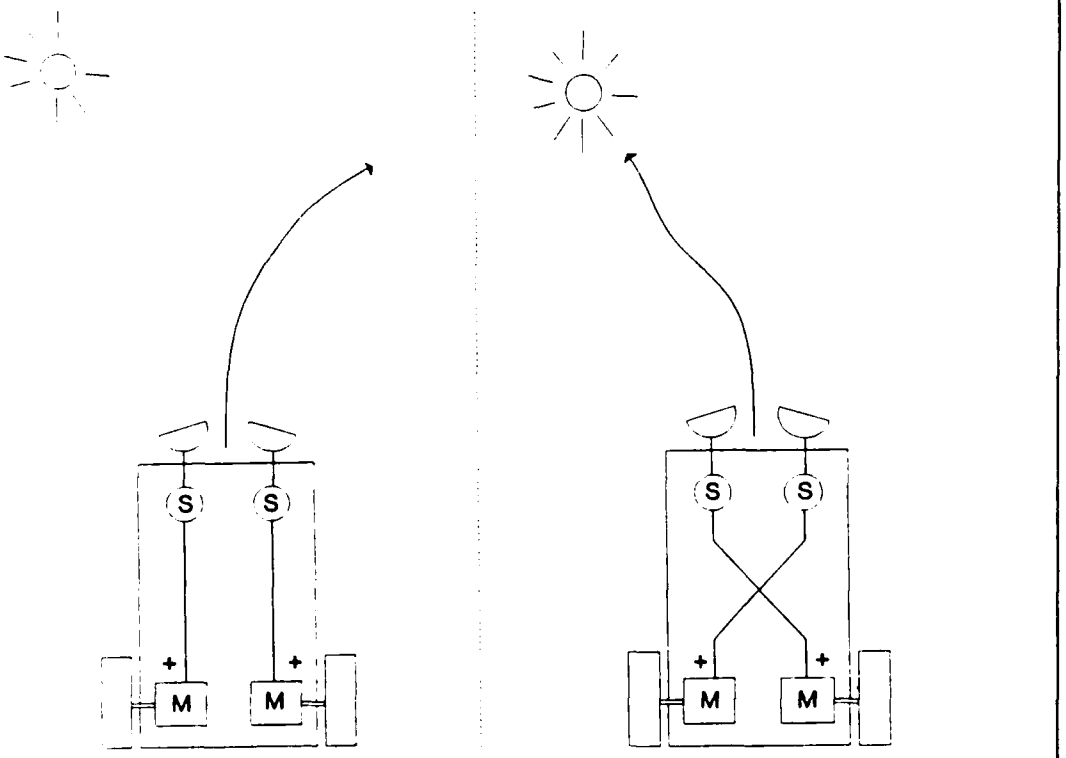
At other times this same worker may become the explorer searching for another source of nourishment to sustain the colony. The search for food and exploitation of a find requires a number of steps that is generally performed by a single searcher. It must perform the sub-tasks of searching for food, identifying acceptable food, navigating back to the colony, establishing a trail back to the source,

communicating its find to other workers, and directing their efforts to begin transportation of the find back to the colony (Goetsch, 1957:118-122; Simon, 1981:63-65). How this tiny forager can perform so seemingly complex a series of tasks is truly amazing. This behavior is even more amazing when one considers that the ant's nervous system is composed of a few hundreds of thousands of neurons. By comparison the human brain contains about 10 billion neurons. How can such a diminutive nervous system direct such complex behavior?

Many mature scientists have also marveled at the rich array of behaviors exhibited by insects and higher forms of animal life. One such researcher, Valentino Braitenberg, has devoted much of his life to studying the nervous-system organization of a variety of fauna. His studies of simple animal forms revealed basic principles of neuronal organization that tend to be retained by organisms higher on the phylogenetic scale. He also noted that even simple arrangements of neurons could produce complex behavior when placed into an environment rich in appropriate stimuli.

Braitenberg's discoveries and observations prompted a dissertation in 1984 he entitled Vehicles: Experiments in Synthetic Psychology. This treatise documents a **gedanken** experiment in which the author synthesizes a series of autonomous vehicles of ever more complicated construction. His first examples illustrate the basic principles of his vehicles (see Figure 1). Vehicle 1A has two sensors that

face forward from the vehicle (toward the top of the page) and are canted outward slightly from the centerline of the vehicle. The outputs of these sensors drive the two motors which drive the independent rear wheels of the vehicle. Although it is not shown explicitly in the figure, a castor supports the front of the vehicle. Unless each motor turns its wheel at exactly the same rate, the vehicle will tend to



Vehicle 1A Vehicle 2A

(After Braitenberg, 1984:8)

Figure 1
Two Basic Braitenberg Vehicles

move in an arc determined by the speed ratio of the wheels and their separation distance. Note that the only difference between Vehicle 1A and Vehicle 1B is the manner in which the sensors connect to the drive motors. In vehicle 1A the sensor on the right drives the right motor and the left sensor drives the left motor. In vehicle 1B the connections cross such that each sensor drives the wheel on the opposite side of the car.

Assume that the sensors are photocells. It is easily seen that the two vehicles will behave quite differently when placed in the vicinity of a light source. Vehicle 1A will find that a light to its front left will provide greater illumination on its left sensor due to its proximity to the source and the sensor's slight orientation to the left. This will result in the left motor receiving proportionately greater stimulation than that which the right motor gets from the right sensor. The vehicle will, therefore, move in an arc to the right away from the light. As the light energy falling on each sensor diminishes, the vehicle will slow and finally stop unless stimulated to avoid another light source that comes into view.

Vehicle 1B, however, behaves in a quite different manner. A light source to its front left causes its right motor and right rear wheel to turn faster than its left. This causes it to turn toward the light until the relative position of the light shifts to the right side. At this point the right sensor and left motor will take control to

steer the vehicle back toward the light. The vehicle will accelerate faster and faster until it crashes into the light source.

Further assume that the wiring and internal workings of the vehicles are enclosed in a "black box." Braitenberg notes that an observer of these vehicles might describe their behavior in terms of certain human emotions or qualities. Vehicle 1A is obviously **afraid** of the light, he might speculate, while Vehicle 1B exhibits **aggression** and must **hate** the light (evidenced by its violent attack).

A simple modification of these vehicles results in new forms of behavior (see Figure 2). The change involves driving the rear wheels at a rate inversely proportional to the sensors' output. The stronger the stimuli, the slower the motors drive the wheels and, conversely, the weaker the stimuli, the faster the wheels are driven. This inhibitory connection between sensor and drive motor (shown as a minus sign near the drive circuitry of the vehicles in Figure 2) causes Vehicle 2A to steer toward the light in a manner reminiscent to that of Vehicle 1B. Its behavior differs, however, in that it slows as it approaches the stimulus, finally stopping close to and facing the source. This vehicle seems to **love** the light (Braitenberg, 1984:12).

Vehicle 2B's behavior is similar to that of 1A's except that its speed, low near the light, increases to its maximum as the vehicle turns away and recedes from the light source. It will continue traveling at top speed until it reaches

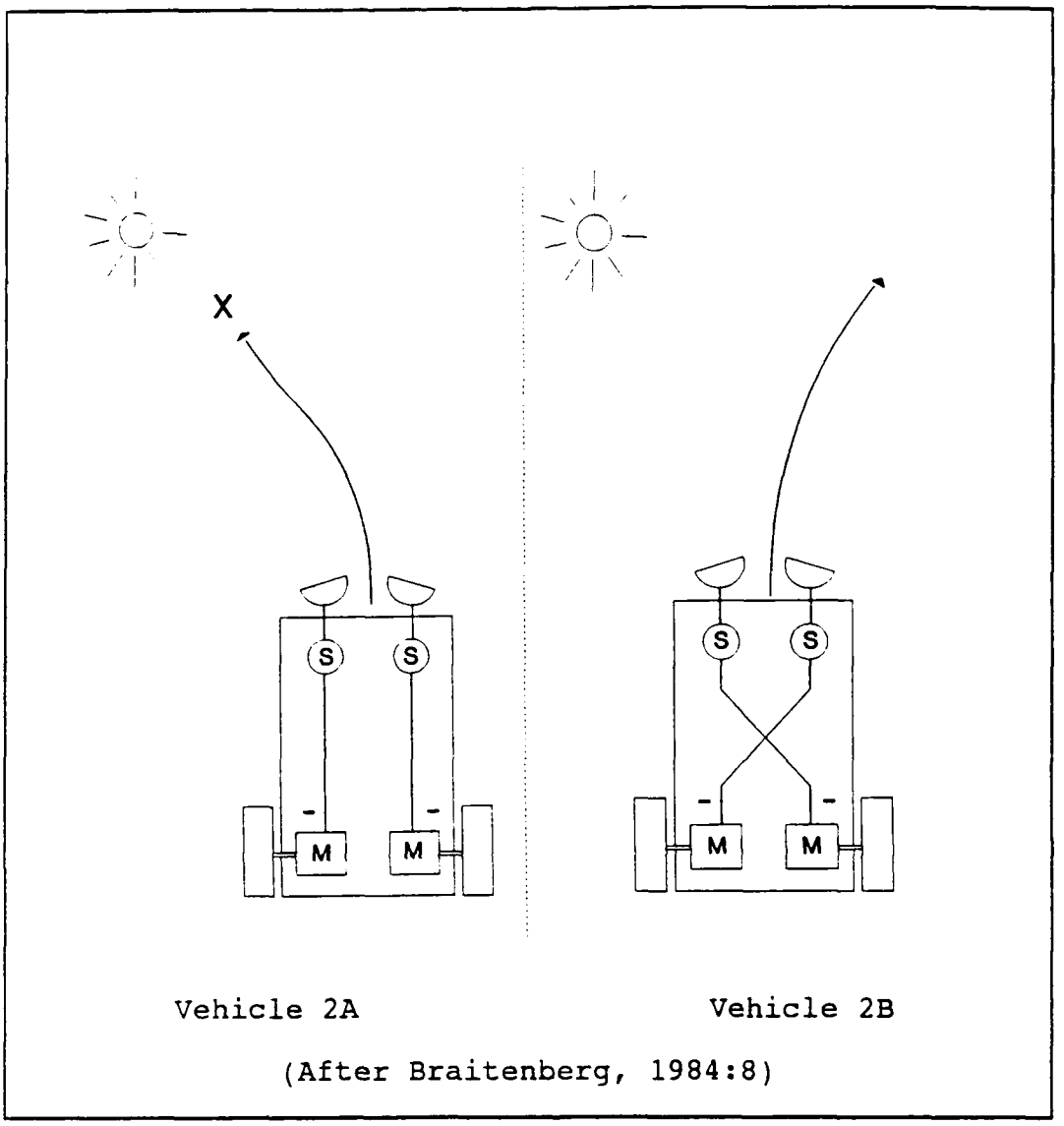


Figure 2
Two More Basic Braitenberg Vehicles

some physical impediment or encounters another light source that will slow and divert its path. Braitenberg describes this as **exploring** behavior (Braitenberg, 1984:12). Vehicle 2B seems to like the light somewhat since it moves slowly in proximity to it, but it soon races off in search of another light to approach and inspect (typical, perhaps, of adolescent romantic involvement?).

Much of the remainder of Braitenberg's book is devoted to discussion of increasingly more sophisticated vehicles. He adds nonlinear control functions for motor speed, additional sensors, computing neurodes between the sensors and the motors, memory devices, and other refinements. As the devices become more complex so do their repertoire of behavioral responses. Soon the observers of these "black box" vehicles (like the young scientists studying ants) are completely at a loss to infer the inner workings of the devices. The reader, however, is constantly reminded that even the complicated vehicles are the result of the assimilation of a number of simple components. Deducing the cause of the behavior is difficult, while designing the vehicles is relatively easy. Braitenberg refers to this phenomena as "the law of uphill analysis and downhill invention" (Braitenberg, 1984:20).

This law is eventually related by Braitenberg to the study of insect nervous systems. In fact, many of the principles used to describe the vehicles' neurodes and the connections between them are based on schemata he knows to exist in nature. It is also likely that Braitenberg has been influenced by the developments in the field of artificial neural networks. Although not adhering tightly to the descriptions of established computing elements found in the current neural network literature, Braitenberg's designs share many common features. Of course it is also possible that both Braitenberg and neural network

researchers were influenced by the existence proofs offered by biological neural networks.

It occurred to this reader of Braitenberg that construction of these sorts of artificial life forms may be an excellent way to investigate properties of both natural and artificial neural networks. The "law of downhill invention" is used in this thesis to investigate possible principles behind the inner workings of animal nervous systems. This investigation was conducted by constructing and evaluating prototypes in the form of Braitenberg's vehicles.

The novel aspect of this approach compared to traditional neural network methods is that the success and failure of various schemes to mimic animal neuron operation and behavior can be evaluated in much the same manner as those used by naturalists and experimental psychologists to study animal behavior. Even principles of evolution such as adaptation and survival of the fittest can be used to evaluate the "goodness" of various network characteristics.

In addition to providing insight into the operation of biological nervous systems, Braitenberg's vehicles can also help us develop autonomous vehicles for a number of uses. Even very simple vehicles could prove militarily useful in roles such as reconnaissance and area denial. Imagine a tactical munition dispenser filled with parachuting mechanical "bugs" equipped with sensors and small burst-transmission data links. After being released by a low

level aircraft these devices would float to earth, discard their parachutes and begin their missions.

Electrically operated and camouflaged to look like rocks, tufts of grass and the like, the devices would move slowly at night and rest during the day while they recharge their batteries with solar cells. Their sensors would note the direction of sounds and odors they had learned to detect during their "basic training." Some would record data about the occurrence of these stimuli for relay to SIGINT satellites to provide intelligence about the passage of large numbers of vehicles, gasoline vapors suggesting refueling points or perhaps the sound of aircraft passing overhead.

More offensive devices may be armed with explosives to destroy passing vehicles. These mobile mines would seek to identify the location of nearby roads from the sound of traffic. Once located, they would crawl to the road and wait for the next vehicle. Since the devices would arrive at the road over a period of many days, enemy mine sweeps of the road would have to be conducted every day. These "cruise mines" could also be used effectively against airfields. Some would be activated immediately while others might "sleep" for some time before beginning their crawl toward the enemy runways.

Of course there are many peaceful roles for such devices too. They could be used as planetary explorers whose low cost and weight could greatly increase the numbers

that could be deployed. They could also perform similar functions on the ocean floor in quests for mineral deposits to exploit. The ability to generate a variety of behavior patterns selected by empirical training and employed in numerous, low cost autonomous vehicles could lead to serendipitous discoveries that would escape a few expensive, preprogrammed exploratory robots.

The balance of this thesis presents four chapters that detail the design considerations, software implementation, and observations on the performance of the simulated Braitenberg vehicles. The next chapter, "Animal Psychological Phenomena," describes the physical/ psychological bases for the neuronal properties of the vehicles to be simulated. Chapter III, "Simulation Design Factors," discusses the manner in which the artificial world of the vehicles is configured and how biological systems' characteristics are emulated in the simulation. Chapter IV, "Vehicle Evaluations," provides a review of the vehicles' testing and observations about the methods used in the simulation. The last chapter, "Summation," is a synopsis of the thesis project that includes proposals for additional follow-on work.

II. Animal Psychological Phenomena

The goal of this project is to develop a flexible, programmable design for a simple autonomous vehicle that exhibits learning and other behavior similar to animals. It is, therefore, imperative to start with an understanding of animal psychological phenomena. Due to the vast amount of research conducted in the learning area in the last one hundred years, the author has relied heavily on reviews of the literature by D'Amato and Mazur (D'Amato, 1970; Mazur, 1986). Many references to other researchers' works are made in these two volumes. The author has chosen to trust the interpretations of the literature offered by these respected reviewers without independent analysis of all the original works. Unless directly cited, assessments of others' works are based on these authors' analyses of the literature.

This chapter is organized into four sections. The first deals with learning or conditioning in animals. The second section briefly addresses aspects of memory. The following section, Animal Psychophysics, presents an overview of the literature relevant to the manner in which animals perceive their environment. The chapter concludes with a brief description of the basic units of animal nervous systems entitled, "Micro-level Aspects of Animal Psychology." This section deals with the basic features of neurons. Included is discussion of their major functional components, their internal operation, and their manner of interaction.

Animal Learning

A key trait governed by animal brains is the ability to learn from experience. Even very simple animals such as the sea slug Aplysia, possessing about 18,000 neurons each, have been shown to learn in controlled situations. For instance, the natural reflex for Aplysia to retract its fragile gill in turbulent waters has been shown to **habituate** (cease responding) after many non-threatening stimulations. This learned response persists for at least several hours (Lazerson, 1988:242-243).

Researchers with animal subjects have identified two primary mechanisms of learning. These mechanisms, classical and operant conditioning, have been extensively studied for the last 100 years. A wealth of information has been gained about the macro-level function of learning. A brief review of the literature on these forms of conditioning is presented in the remainder of this section.

Classical or Pavlovian Conditioning.

The simplest learning paradigms are based on classical or Pavlovian conditioning. Pavlov, a Russian physiologist, discovered this basic learning phenomena while conducting experiments on the digestive juices of dogs. He developed a technique for implanting tubes in the salivary glands of dogs to monitor their secretions while eating. Pavlov noted that some dogs who had experience as subjects in the experiment began to salivate before the food was presented. He realized the dogs behavior indicated they had learned to

anticipate the arrival of the food based on the occurrence of stimuli associated with the experimental procedure. In the ensuing years, Pavlov devoted his life to studying this phenomena.

The basic Pavlovian learning paradigm consists of four parts. An **unconditioned stimulus** (US) was presented to the subject. This US elicited a reflexive response termed the **unconditioned response** (UR). In Pavlov's experiments the UR was usually salivation to the presence of food while the presentation of food formed the US. A **conditioned stimulus** (CS), for instance the ringing of a bell, was then presented prior to further presentations of the US. After a number of trials the CS would begin to evoke the **conditioned response** (CR), salivation to the sound of the bell, even without the original US of food presentation. It was clear to Pavlov that the subject had learned to associate the US with the CS such that the CS elicited a response very similar to that of the US (Mazur, 1986:58-59).

The phenomenon of classical conditioning appears to be universal among animal species. In fact, researchers have demonstrated the principles of classical conditioning at the level of individual neurons (Kandel and Tauc, 1964:145-147; Kandel and Tauc, 1965:1-27). Another researcher has provided evidence that classical conditioning occurs at the synapse level (Alkon, 1989:47). These data imply classical conditioning is fundamental to biological learning.

There are a number of factors concerning the manifestation of classical conditioning that are important to this effort. First, the common metric of CR strength is the percentage of occurrences of the CS that successfully elicit the CR. For a given training condition, this performance tends to reach an **asymptote** below 100 percent. No matter how many training sessions are presented, performance does not improve beyond this level. The level of asymptote and the speed with which it is reached (**acquisition**) can, however, be influenced by the magnitude of the US. The stronger the US the quicker asymptote is reached and the larger its magnitude (within some limit). The strength of the CS can also positively affect the rate of acquisition although it seems to have no effect on the final performance asymptote (Mazur, 1986:67-68).

An important measure of the strength of the CS-US association is the number of presentations of the CS without an US that still occasionally elicit a CR. Psychologists test this linkage by conducting **extinction** trials in which the CS is presented without subsequent US. The frequency of occurrence of the CR during extinction will gradually decrease until the CS no longer elicits the CR. The conditioning's extinction is then said to have occurred. This phenomenon suggests the learning process is reversible and when the subject detects the lack of predictive value of the CS, it forgets the relationship (Mazur, 1986:68-69).

Pavlov, however, discovered a curious effect related to extinction that argues against this simple view of "forgetting" previous conditioning. If a subject is tested to extinction during a set of trials on one day, rested for a day, and retested, it will often show recovery of some portion of the conditioning. In fact, the experimenter may see several periods of this **spontaneous recovery** before the subject ceases to respond to the CS forever. Pavlov believed that during extinction an **inhibitory** association is formed between the CS and US. During the extinction trials this association grows stronger as the previous excitatory relationship grows weaker. Eventually the inhibitory association overcomes the excitatory one and extinction occurs. Over time, however, the newest association (the inhibitory one) weakens faster. When retested, the original association reappears since it has decayed less between the test sessions (Mazur, 1986:69-70).

Not all researchers accept Pavlov's explanation. An equally plausible theory relates to the subject's recognition of successive trials being separate events each with its own set of "rules." Thus, during the original acquisition trials the subject sees the CS reliably predict the US. During the extinction trials the CS never predicts the US. During the next set of trials the subject can not know which set of rules applies and, therefore, responds at some intermediate level between the original asymptotic rate and no responding at all (Mazur, 1986:70). Although the

mechanisms behind spontaneous recovery are unclear, it is a phenomennon that high fidelity simulations of animal learning should have the capability to mimic.

Yet another aspect of classical conditioning is important to this effort. Often a novel stimuli can replace the CS and evoke the US. A subject, for instance, conditioned with a 1200 Hz tone may also respond to a 1000 Hz tone. This phenomena is known as **generalization.** As might be expected, the degree to which the novel stimuli elicits the CR is positively correlated with its similarity to the CS (Mazur, 1986:72-73).

The final features of classical conditioning which seem relevant for this thesis deal with the temporal relationships between the CS and the US. Four temporal-based paradigms for classical conditioning are noted in the literature: simultaneous, delay, trace, and backward. In simultaneous conditioning the CS and US begin at the same time. Extinction studies indicate such conditioning is much weaker than other forms in which there is some delay between the CS and US. It is as if the strength of the link between the CS and US depends on the reliability of the CS to predict the US. If the CS and US appear at the same time, the CS is of no value as a predictor and, therefore, the association is weak (Mazur, 1986:74-75).

In delay conditioning the CS begins before the US and continues during its presentation. When the onset of the US occurs slightly after the CS (short delay conditioning), the

strongest and fastest conditioning occurs. As the delay between onset of the CS and onset of the US increases, the strength of the resulting conditioning decreases (Mazur, 1986:75).

An interesting effect is seen when the time between CS onset and US onset is extended. At first the subject will begin to respond when the CS starts. After a few trials, however, the response is delayed longer and longer until it begins slightly before the US begins (Mazur, 1986:75). Obviously the subject has some capacity to estimate the passage of time and include this information in his conditioning process.

The third temporal paradigm in classical conditioning, trace conditioning, is defined by the onset and termination of the CS before the onset of the US. The subject's association is therefore mediated by the memory "trace." Here again there is an inverse relationship between the interstimulus interval and the strength of the conditioning. As one would expect, the negative effect of the time delay on trace conditioning is greater than that seen in delay conditioning (Mazur, 1986:75). There is also evidence to suggest the effect of time is highly dependent on the particular stimulus-response modalities.

An excellent example of this modality-specific temporal factor occurs in a paradigm known as taste aversion conditioning. In these experiments the CS is a novel tasting food. Following ingestion of the food the subject

is injected with a solution which makes it ill. After recovering, the subject is again presented with the CS. The usual response is little or no ingestion of the food. The aspect of interest is the relatively long delay between the CS (novel food) and the US (illness) which may be hours or even as much as a day (Mazur, 1986:62). The length of this delay is in sharp contrast to the findings Mazur attributes to Schneiderman (1966). Mazur notes that Schneiderman's study of the response of the rabbit's nictating membrane (a translucent, protective "eyelid" found under the actual eyelid) to puffs of air shows no conditioning if the interstimulus interval exceeds a few seconds (Mazur, 1986:75). Intuitively, we should not be surprised by these findings. The benefit to the organism of associating the delayed adverse effects of a toxin to its likely source is great. Conversely, the necessity to predict an irritant entering the eye is limited to a brief period before it arrives (just enough time to blink). We might well expect species to develop specialized association centers with optimized memory duration due to the natural selection advantages these centers would offer.

The final form of conditioning, backward conditioning, presents the CS after the US. As might be expected, this form leads to little, if any, conditioning (Mazur, 1986:75-76). This result is consistent with the principle of conditioning strength being related to the efficacy of the

CS as a predictor of the US. Since the CS is not a predictor, no conditioning strength is formed.

Although classical conditioning may be the fundamental mechanism of learning in animals, few researchers now believe it is the most important. Classical conditioning has given way to operant conditioning as the accepted universal learning mechanism. The next section presents a brief summary of the important features of this companion mechanism of animal learning.

Operant Conditioning. The typical operant conditioning learning paradigm consists of an experimental environment in which the subject is provided some means to perform a task (with rat subjects the usual task is to press a bar while pigeon subjects are generally required to peck a disk). The task corresponds to the conditioned response of the classical conditioning paradigm. As some criteria of task performance is surpassed, the subject is **reinforced** or **punished** with some action that affects its level of satisfaction or level of some internal "drive." Reinforcement actions may either provide something favorable to the subject, positive reinforcement, or withhold something unfavorable, negative reinforcement. Positive punishing actions apply an unpleasant stimulus to the subject while negative punishment removes a favorable stimuli.

Generally, the application of reinforcement, positive or negative, tends to perpetuate the behavior with which it

is associated. Either form of punishment tends to inhibit the occurrence of the behavior (Mazur, 1986:161-162). The dependent variables of interest to the experimenter are generally the number or strength of responses the subject will make between reinforcements or the number of responses it will make before it quits responding (reaches extinction).

Operant conditioning differs from classical conditioning in several important respects. First, classical conditioning pairs an unconditioned stimulus with a conditioned stimulus to study the effect it has on a natural, reflexive response (the unconditioned/conditioned response). The experimenter merely has to present the US to reliably evoke the UR (D'Amato, 1970:263). Operant conditioning, however, examines the principles behind the development of "voluntary" behavior patterns (Mazur, 1986:117). That is, those which require the subject to perform some overt action.

Another difference in the two types of conditioning is the basis on which the US is presented to the subject. With classical conditioning the US presentation is independent of the behavior of the subject. But, in the operant conditioning case the subject's behavior ultimately controls the presentation of the US reinforcer. That is, in the operant conditioning case the US (reinforcement/punishment) is response contingent (D'Amato, 1970:263).

Early experimenters in the investigation of the influence of experience on voluntary behavior used a device termed a puzzle box. The experimenter using a puzzle box paradigm confined a subject animal in a cage until it discovered the correct behavioral response to open the door. The response required to release the door could be one or more specific actions like pulling a string and/or pressing a lever. One experimenter, E.L. Thorndike, proposed a theory of learning he called the **Law of Effect.** This law noted that if a number of responses are made to a given situation those...

> "...which are accompanied or closely followed by satisfaction to the animal will, other things being equal, be more firmly connected to the situation, so that when it recurs they will be more likely to recur; those which are accompanied by or closely followed by discomfort to the animal will, other things being equal, have their connections with that situation weakened, so that, when it recurs, they will be less likely to occur. The greater the satisfaction or discomfort, the greater the strengthening or weakening of the bond (Mazur, 1986:118-119)."

Thorndike defined satisfaction as the state which an animal does not seek to avoid and often acts to achieve. He defined discomfort as a state the animal actively seeks to avoid (Mazur, 1986:119). In terms of the earlier definitions, satisfaction is reinforcement while discomfort is punishment.

An interesting finding by other puzzle box experimenters, E. R. Guthrie and G. P. Horton, demonstrated the association of an animal subject's precise behavioral actions with reinforcement. Their studies involved cats in

a simple puzzle box where any movement of a vertical pole in the box released the door. By connecting a camera to the release mechanism of the door they were able to record the position of the subject when the release response was made. Although the behavior which resulted in release varied considerably during early trials, the later efforts indicated each subject developed particular behavioral responses to achieve release. These responses included pushing the pole with the left paw, rubbing the pole with the nose, lying down and rolling into the pole, biting the pole while standing in a certain position, and rubbing the pole with the body (Mazur, 1986:120,122).

It is obvious the subjects did not "understand" even the simple concept, "door opens when pole is moved." Rather, the animals recreated the movements which they had made just prior to release from the box (reinforcement). Mazur speculates that the animal eventually settles on a particular response due to the formation of secondary reinforcers related to recall of precise body movements, muscle contractions, and limb positions that preceded previous reinforcement. The feedback of these secondary reinforcers lead to the more probable selection of related behaviors until they dominate the learning process (Mazur, 1986:120-122).

Although many variables have been shown to affect operant conditioning, four are particularly relevant to this effort: magnitude of reinforcer, drive level of the subject

relative to the reinforcer, time delay between response and reinforcer, and schedule of reinforcement (D'Amato, 1970:274).

The magnitude of the reinforcer has a positive effect on the performance asymptote. That is, other factors being equal a larger reinforcement will result in stronger response (D'Amato, 1970:275,276). The relationship, however, is not a simple one. Studies in which the reinforcement level has been varied indicate the subjects' response strength quickly adapt to the change. Changes from a large to small reinforcer will see a rapid decrease in response strength that eventually coincides with the normal strength for that reinforcement level. Likewise, changes from small to large reinforcements see rapid increases in the response strength (D'Amato, 1970:277). Due to the rapid adjustment of response strength to reinforcement level, many researchers, according to D'Amato, believe, "...amount of reward affects performance rather than learning." If learning were truly affected, there would be a longer lasting influence of reinforcement magnitude evident (D'Amato, 1970:281).

The effect reinforcement magnitude has on extinction is unclear. D'Amato cites references that indicate both positive and negative relationships. He proposes a theory of a nonmonotonic interaction effect between amount of training and size of reward (D'Amato, 1970:290-291). Due to the lack of subsequent experimental support for this theory

the author has chosen not to consider it as a factor in the vehicle component designs.

The drive level of the subject is an important factor in operant conditioning. Although drive is difficult to define, we can list a number of factors which most people will agree are bases for biological motivation. These include hunger, thirst, sex, and social contact. It is somewhat intuitive that if one is using food as a reinforcer the subject will make more effort to learn if it is hungry than if it has just eaten. In general, the effect of drive level is similar to that of reinforcement magnitude. The higher the drive level, the higher the response asymptote (D'Amato, 1970:282). This fact may also seem intuitive. If reinforcement is sought in order to decrease drive, then larger reinforcements and lower drive should have similar results in that they both reduce the difference between the current and goal states.

Drive level is also believed to have little effect on the rate of eventual extinction of a conditioned response (D'Amato, 1970:289). This finding is consistent with the rapid adaptation of response intensity to reward magnitude. After all, during extinction reinforcement is reduced to zero by definition.

As is the case with classical conditioning, there are important temporal factors evident in operant conditioning. These factors are witnessed by the effect of delay between

the response and reinforcement and by certain phenomena associated with the schedule by which reinforcement is made.

The time delay between the conditioned response and the reinforcement is inversely related to the strength of the association. This relationship is demonstrated both in the asymptote and rate of learning. Delay of reinforcement also affects the subject's resistance to extinction in a positive manner (D'Amato, 1970:283-285, 293-294). Experiments which have investigated the role that delaying reinforcement has on eventual extinction have sometimes been confounded by the timing issue. If a subject is accustomed to receiving a reinforcer some time period following a response, when does the extinction period actually begin? By definition, extinction begins with the last reinforcement. If, however, the subject does not expect the reinforcement for some time interval, then extinction should not commence until expiration of the expected delay. This sort of problem also confounds the study of extinction of behaviors under various temporal schemes for delivery of reinforcement.

The criterion by which reinforcement is administered is an important factor in the overall operant conditioning process. B. F. Skinner pioneered investigations on the effect of differing reinforcement schedules on conditioning. He identified four major classes of these schedules, fixed ratio (FR); fixed interval (FI); variable ratio (VR); and variable interval (VI). The schedules involving a ratio, FR and VR, provide reinforcement after a fixed or variable

number of responses by the subject. The interval schedules, FI and VI, provide reinforcement immediately after the first response following expiration of a fixed or variable time interval (Mazur, 1986:139-144; D'Amato, 1970:389).

Animals trained to respond for some form of reinforcement develop characteristic behavior patterns depending on the type of reinforcement schedule used. Generally these behavior patterns are studied by recording the subjects' cumulative responses over time and the number of responses made before extinction of the conditioning. The shapes of the cumulative response curves provide clues relevant to the associative mechanisms within the animal.

The variable ratio (VR) and interval (VI) schedules both produce generally linear cumulative response records. The subjects respond as if they are aware that there is no discernable pattern to the schedule. The VR response curve usually indicates a steeper slope since the ability to generate more responses will result in more rewards per unit time. This isn't the case with the VI schedule. Reinforcement only occurs after the next response following the expiration of the time interval (which the subject can not predict). Lower level response rates will provide nearly the same amount of reinforcement as faster rates (D'Amato, 1970:392; Mazur, 1986:140-14,143-144).

The fixed ratio (FR) and fixed interval (FI) schedules are another matter. Larger FR schedules (those requiring many response between reinforcements) have a characteristic

response pause after each reinforcement followed by a rapid period of response until the next reward. This gives the cumulative response record a sort of skewed, stair-step appearance. As the ratio of responses to reinforcement decreases so does the post-reinforcement pause (Mazur, 1986:139-140).

FI schedules have a scalloped appearance resulting from a post-reinforcement response pause and an accelerating response rate until the next reinforcement. The subject reacts as if it recognizes that a given amount of time must pass before the next reward is possible (Mazur, 1986:142, D'Amato, 1970:392). This evidence suggests, once again, the existence of some internal time sense within the animal subjects. Also implied is the notion that this time sense can be remembered in relation to events which affect the well being of the animal.

Both classical and operant conditioning have been studied with countless experimental designs and subject species from planaria to human beings. The wealth of data is overwhelming to even the casual observer. Presumably, the preceding section has imparted the reader with a feel for the important features of behavioral adaptation in animals. Assuming that evolution has over the last hundreds of millions of years settled on a near optimal system for controlling its more complex products, the behavior of biological nervous systems provides an excellent model for developing artificial control systems. The features of

operant and classical conditioning form the framework for the performance objectives in this project.

Implied in the concept of learning is the retention of the learned behavior over time. This retention is attributed to a memory trace somehow recorded in the animals' nervous systems. Memory in animals is much harder to study that it is in man. The range of responses for human subjects provides improved opportunities for exploration. The next section provides a very brief assessment of the most relevant memory issues for this project.

Memory

The literature concerning this subject generally identifies two operationally defined types of memory: short term memory (STM) and long term memory (LTM). Short term memory is often viewed as a limited, temporary storage location for sensory inputs or strings of recalled items. Long term memory is much more permanent and has a much larger capacity. In computer terms short term memory might be thought of as a buffer where data is kept between operations. Long term memory in this analogy would be the magnetic disk that permanently stores information.

Short term memory has been shown to decay rapidly when efforts are made to inhibit the rehearsal of the memory items. In fact, an experiment by S. Hellyer indicated that nonsense syllables committed to memory with a single repetition were recalled correctly only 40% of the time

after only 9 seconds (D'amato, 1970:616-617). Increasing the number of repetitions to 8 brought correct retention to the 90% mark, but even then performance dropped to 70% after less than 30 seconds. This rapid fading of the memory trace in STM reinforces the notion that its primary use is as a scratch-pad for various mental functions.

Although there have been a number of researchers who have sought to reject the idea that STM and LTM are two distinct memory processes (D'Amato, 1970:614-615), the question is immaterial for this project. The important aspect of this dual representation is the functional differences between STM and LTM and their roles in learning.

In one sense, short term memory could be thought of as a filter in which incoming sensory information spends a short time while subsequent events determine whether it is important enough to commit to permanent storage. In fact, STM may be the portal through which all memories pass provided they have sufficient reason for so doing. Many researchers believe the human brain's memory capacity is effectively incapable of being overloaded in a normal human life time. In such a model, records of all sensations and everyday events are locked away in our minds. The problem is accessing the memories that are there. This may be true for the relatively enormous brains which differentiate humans from most other species, but those less "blessed" with cerebral cortex may require judicious use of storage capacity.

For this effort it is sufficient to recognize that both STM and LTM have roles in adaptive organisms. Short term memory can serve as a memory store to improve the association between time displaced events. Since most events which are recorded in the STM store will be irrelevant, it must purge itself quickly to make room for possibly more critical information. Once the important data have been recognized, LTM provides the means to store them reliably for extended periods. Such memorized behavior patterns triggered by sensory events are the essence of the basic conditioning mechanisms discussed in the preceeding section of this chapter.

Although the sensors on the Braitenberg vehicles that were discussed in the introduction of this thesis are very simple, there are still important links to be made to the concepts that govern biological sensors. The next section deals with the features of perception that are relevant to the simulated, simple, adaptable organisms that are described in Chapter II.

Animal Psychophysics

Psychophysics deals with the relationships between the physical stimuli that impinge upon an organism and its resulting psychological experience. Physical energy (light, sound pressure, heat, etc.) first must strike a sensory organ in order to be detected by the organism. Within this sense organ some change takes place which ultimately affects the firing rate of one or more neurons. These neurons carry

information about the physical energy (its intensity, rate of change, spatio-temporal organization, etc.) to other neurons which may stimulate behavior changes (Forgus and Melamed, 1966:7-8).

For the design of the vehicles, the important variables are the dynamic range of the various sensor types, their neural response in relation to their energy intake and their directional sensitivity. The goal of this effort is not to model an existing organism, but to model basic animal characteristics. For this reason discussion is limited to general psychophysical phenomena.

Dynamic range is an index of the useful range of stimulus energy levels over which a sensor can work (Geldard, 1972:8). Biological sensors often exhibit dynamic ranges which are impossible to duplicate with current technology. The human eye, for instance, has an operating range of about 1 billion units. That is, the ratio between the minimum light detectable and the intensity at which vision fails is 1 billion to one.

While dynamic range deals with the physical limits of sensors inputs, another important stimulus scaling parameter deals with the sensors' output. Investigations of the relationship between physical stimulus intensity and the resultant sensation level (the perceived stimulus intensity) date to the 1800s. During the 1830s E. H. Weber studied what he termed "just noticeable differences" (JNDs) of weights held in the hand. Weber noted that regardless of

the original amount of weight used the additional weight needed for a human subject to detect a difference was a constant ratio of the beginning weight. This relationship is generally written as

$$D/I = K \qquad (1)$$

where

D = Just noticeable stimulus difference

I = Physical intensity of the stimulus

K = Constant ratio for the sensory modality

(Forgus and Melamed, 1976:46; D'Amato, 1970:154)

A student of Weber's, G. T. Fechner, developed this "law" further by assuming that JNDs represent equal *increments on the sensation scale*. Given this assumption the intensity of any sensation can be expressed as

$$S = k \log E / T \qquad (2)$$

where

S = sensation level

k = the ratio constant

E = the physical stimulus intensity

T = the stimulus threshold or minimum

(Forgus and Melamed, 1976:46-47; D'Amato, 1970:157)

Due to the wide range of intensities of physical stimuli in nature, most biological sensors have a wide dynamic range. The logarithmic relationship in Equation 2 is a mechanism to provide this dynamic range in neural

systems which use firing rate to code strengths of signals. There is much experimental evidence to suggest the law does not perform well at the extremes of the sensory response scale. This relationship will, however, suffice to describe the transfer function for stimulus intensity to sensation level for the vehicles to be modeled.

So far, the discussion has covered animal psychological phenomena by describing system-level responses to various inputs. One final area of discussion for this chapter that needs to be addressed is that of the elemental components of the animal nervous system, the individual neurons.

Micro-level Aspects of Animal Psychology

Neurons are highly specialized cells adapted for the transmission and simple processing of the data and instructions that are required for the operation of an organism. Neurons within an organism are also highly specialized. Some serve as monitors of external and internal conditions (sensors), some process information received from other neurons, while others are adapted to serve as transmission links to various parts of the organism's body. Although the cells that function as sensors differ greatly from those that serve as transmission lines, most neurons are composed of three main, identifiable components: the soma, the dendrites, and the axon.

The soma, or body, of the neuron integrates the input signals it receives from other cells and, if stimulated sufficiently, initiates signals to pass to still other

cells. The soma receives its inputs via appendages called dendrites.

The dendrites receive stimulation from the axons of other neurons. The stimulation is received at specialized areas of the dendrites and soma called receptor sites. The stimulation is propagated across small gaps, termed synapses, between the transmitting neuron and one of the receptor sites on the receiving neuron's dendrites.

Neurons transmit their own signals via another, generally much longer appendage, the axon. Like the rest of the neuron, the axon is electrically active. It maintains an electrical potential through the membrane which forms the cell wall. This resting potential, which is about -70 millivolts from the interior to the exterior of the cell, is maintained by the active transport of potassium, sodium, and chloride ions through the membrane (Cruz, 1988:4).

When the soma receives enough net stimulation to fire, a localized area of the cell wall passes a current that causes a rapid change in the resting potential. The voltage change causes adjacent areas to also begin a current flow resulting in formation of a moving voltage spike along the neuron's soma and down the length of the axon. The propagation of the signals across the synaptic junction between the axon of one neuron and the dendrite of another is accomplished chemically by neurotransmitters. These chemical compounds are released by vesicles at the end of the axon. They diffuse quickly across the narrow synaptic

gap, are absorbed by the receptor site, and induce changes in the dendrite of the receiving neuron. These changes ultimately affect the probability that the neuron will fire. Neurotransmitters can either increase the probability that the receiving cell will fire (excite the neuron) or decrease its probability of firing (inhibit the neuron) depending on their type (Wasserman, 1989:194-198).

Since the resting potential of the neuron requires a small amount of time to recover between discharges, each cell has an upper bound on its firing rate. Its lower bound is zero since a negative firing rate has no physical meaning. In some cases, however, a neuron may have a normal firing rate baseline that can be driven higher or lower by excitatory or inhibitory stimulation. In this case one could view inhibitory inputs as causing a negative firing rate (in relation to the baseline).

The degree to which the neurotransmitters affect the receiving neuron's firing rate appears, at least in some neurons, to be modified by experience. In fact, a number of researchers have reported neuronal changes as a result of classical conditioning. One change noted involves a protein induced decrease in potassium ion flow through the neuron's cell wall. This change in ionic activity results in hyperpolarization of the cell with correspondingly greater sensitivity to excitatory inputs. As viewed by the neuron's soma, the inputs that occur along the sensitized pathways have a greater importance or "weight" than those on less

sensitive routes. Interestingly, the changes noted last for at least a matter of days (Alkon, 1989:44). This phenomena may well represent a fundamental adaptive mechanism in animals.

The firing rate of a neuron is also influenced by the pattern of inputs it receives. The stimulation from one neuron to another is often based on the firing rate of the transmitting neuron (**temporal summation**). If enough input pulses are received in a given period of time (perhaps the time required for the resting potential to recover following the last firing), the firing threshold of the receiving neuron can be exceeded and a discharge impulse generated (Cruz, 1988:4).

Another encoding scheme seen in neural connections is based on the time phasing of arriving pulses. Since not all connections between neurons are strong enough to fire the receiving neuron alone, it is sometimes necessary for pulses to converge at the same time from several transmitters in order to fire the receiver. This **spatial summation** mode provides a means of effecting a wider range of logic gate-like units within the neural system (Cruz, 1988:5).

While the basic neural unit is relatively simple, their organization within an organism becomes increasingly complex as one moves up the phylo-genetic scale. In mammals, for instance, some neurons may receive inputs from hundreds of thousands of other neurons (Alkon, 1989:42). This high degree of connectivity has led some students of neurology to

observe that every neuron is affected, directly or indirectly, by every other neuron. It is this organization of billions of basic units in the central nervous systems of larger animals that provides the robustness we see exhibited in their behavior and survival. The inherent flexibility and fault tolerance exhibited by such a massively redundant and parallel system has led to its evolutionary development in higher life forms from the simple neural building blocks found in lower animals.

III. Simulation Design Factors

Simulation involves the creation of a false "world" in which events occur that are analogous, in their most important respects, to events in the "real" world. The degree with which the laws that govern the simulation events mimic the laws that govern physical events determines the overall fidelity of the simulation. This chapter describes the essence of the simulation world created for this thesis and the elements of the natural world which establish the analog relationships between the two. The first section deals with the environment in which the primary objects of simulation, the vehicles, exist. The second section describes the major components and operating features of the vehicles. The final section traces the operation of the vehicle through a complete cycle of its internal workings.

The Environment

For this thesis project the index of success is the demonstration of the ability of the simulated vehicles to learn. The simulated Braitenberg vehicles must be capable of adapting their behavior to their environment in order to achieve or maintain some desirable state. In the parlance of neural network research, this trait is termed unsupervised training. That is, the network which guides the vehicle must learn without being provided feedback by a human observer. To do so the vehicles must have indices of

their success or current "wellness." It is these indices that the vehicles' control systems will try to maximize.

In order to maintain the analogy between the simulated vehicle and biological systems, the wellness index is modeled as the vehicles' energy states. Movement depletes the vehicles' energy, while discovery of food (and its implicit consumption) increases their energy. The food has no stimulatory signature itself, but it occurs in close proximity to certain environmental features that serve as stimuli. The food location and its relationship to stimulatory features form the key attributes of the environment.

Since there needs to be a large number of encounters with food to provide sufficient adaption opportunities for the vehicles, it is unwieldy to model each item of food explicitly. Instead a probabilistic relationship between the food items and designated stimuli in the environment has been defined. This allowed a great reduction in the number of individual items tracked within the code of the simulation.

A good analogy for the food-stimulus relationship used is mushrooms growing under a street light. The light can be detected by certain electro-magnetically sensitive devices (such as photocells) that are unable to detect mushrooms directly. If the mushrooms require the radiated energy of

the street light for sustenance, then the presence of mushrooms can be indirectly detected by the light sensor.

The mushrooms are distributed about the point directly beneath the light. Their density per unit area is a function of the radiation intensity striking the ground. Therefore, the probability of a mushroom being in a given unit area decreases as a function of the inverse square of the lateral distance from the point directly under the light.

The lights can be any of three types: red, green, and blue. Their intensities fall into the range from 0 to over 2 million units. Not all of the lights can nurture the "mushrooms." Even lights of the same type may not all have food associated with them.

There is another type of stimulus found in the environment in addition to the lights. These are analogous to sound sources like babbling brooks, or rushing water. These features may also be associated with food sources (perhaps cat tails, or lily pads?). Their intensities are selectable through the same range as are the lights. The characteristics in effect for any particular environment can be selected by the environment designer (e.g., number of stimuli, their types, which support food in their vicinity, their intensities, etc.).

The terrain in which the light and sound source features are found is both closed and infinite. That is,

the vehicle can neither exit the world nor encounter its boundary. It is as if the world were laid out on an infinite checkerboard with each square arranged exactly like its neighbors. All the light and sound features are arranged in exactly the same manner in each square. A traveler in the world can move from one square to another by slipping under the black, sound-proof curtain that surrounds each square. Once under, he can then view or hear the stimuli in that square, but no stimuli can penetrate from other squares. The observer monitoring the traveler via the computer screen sees him exit one side of the screen and immediately reappear at the opposite side. The terrain itself is flat and smooth with no obstacles and no additional features.

The software limits the world to no more than eight stimuli. Experience has shown, however, that the screen becomes crowded if there are more than three stimuli present. The number of stimuli can also adversely affect run-speed since the distances from the vehicle to each stimulus must be found during each move cycle. Likewise, the probabilities of foods being found are functions of the distances to each stimulus. These probabilities must also be calculated during each time period. These calculations can noticeably lengthen run-time as the number of stimuli increase. For these reasons, the number of stimuli has generally been limited to three or less.

The Vehicles

The vehicles retain the essential elements of Braitenberg's concepts as presented in Chapter One, Figures 1 and 2. The basic architecture of the thesis vehicle is presented in Figure 3. Several differences are immediately apparent. First, the vehicles contain up to four sensors (labeled S1-S4 in the figure). Second, each sensor has an associated "delta" sensor designated as $S_\Delta 1$-$S_\Delta 4$. The delta sensors indicate the change in the corresponding sensor's output from the last time period. Another difference is the

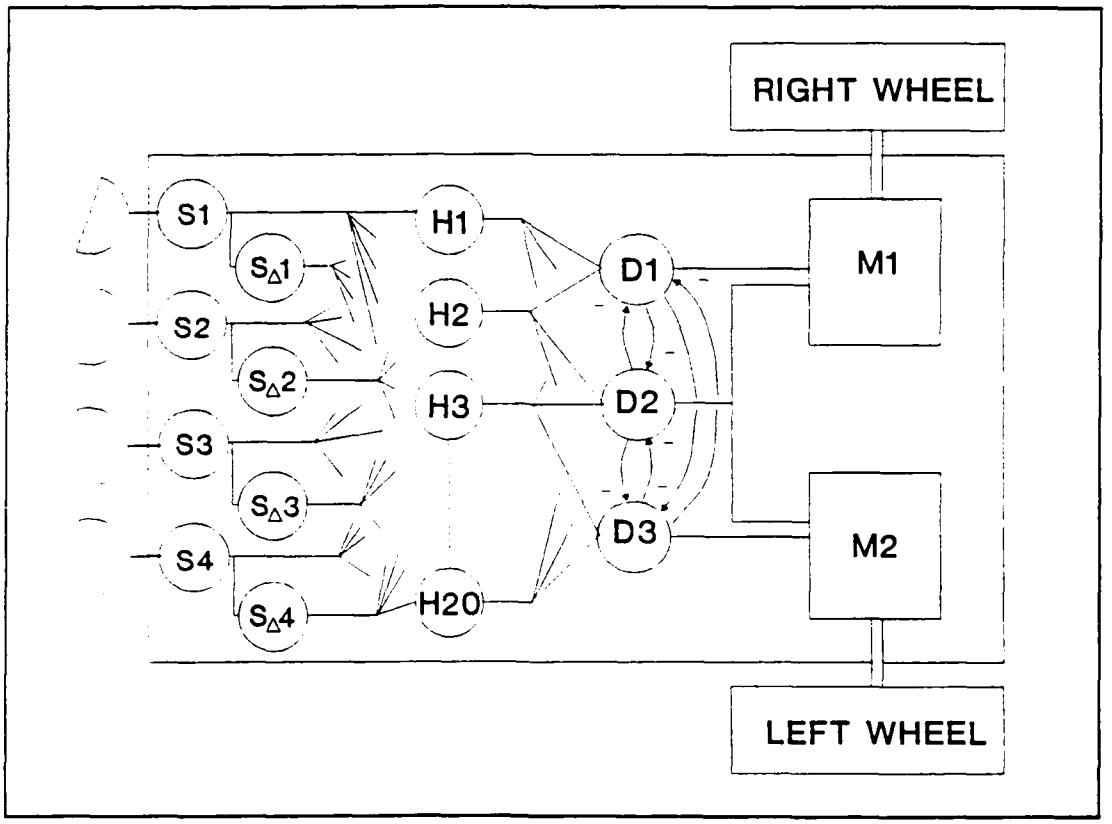


Figure 3
The Complete Simulation Vehicle

three neurons (D1-D3) that drive the motors (M1 and M2). These neurons enable the vehicle to make three distinct movements. D1 and D3 control left and right turns respectively, while D2 allows the vehicle to go straight. The arcing connections among the "D" neurons are mutual inhibition links that result in the neuron with the highest output gaining control of the motors.

The final difference between the thesis vehicle and the earlier Braitenberg concepts lies between the sensor and motor stages. Up to 20 simple processing elements or neurons lie between the sensors and the motor drive units represented in the figure as the column of circles labeled H1 through H20. This "hidden layer" provides additional processing capability for associating sensory inputs with appropriate motor responses. It also provides an architecture more closely aligned with those studied in the branch of artificial intelligence known as artificial neural networks. Since neural network writings have greatly influenced this project (and probably Valentino Braitenberg as well) it is appropriate to digress here to provide the reader with a short history and tutorial on neural networks as they pertain to this thesis. Following this background section, the balance of Chapter III addresses the design, implementation and features of the sensors, hidden neuron layer, and drive neuron stages. The chapter concludes by

tracing the operation of the vehicles' internal workings through a complete cycle.

Artificial Neural Networks. The underlying goal of artificial intelligence (AI) is the development of algorithms and machines that "...exhibit the characteristics we associate with intelligence in human behavior..." (Barr, 1981:3). Much early work in AI was concerned with intelligence at the macro level: symbolic processing, speech understanding, expert systems, etc. A few researchers attacked the problem at the micro level with studies of artificial neurons. These artificial neurons were viewed as the basic building blocks of large, massively parallel computers that would process data in a manner similar to that used by biological nervous systems. After a period of rapid expansion, this research was interrupted when one of its pioneers proved that certain popular architectures were not as robust as first believed. In 1969 Marvin Minsky published his now famous (infamous?) proof of the inability of single layer networks to solve many simple problems such as that of the exclusive-or gate (Minsky and Papert, 1969). Many researchers gave up work in the artificial neuron field based on the impact of the arguments of the highly respected Minsky. The resulting hiatus in neural network developments coincided with continued acceleration in the progress of sequential computers that improved the ability of

researchers to simulate large neuronal nets (Wasserman, 1989:5; Rogers, et al., 1990:13-14).

A small group of researchers continued their efforts at developing neuronal computing elements and architectures. This group, including people like Teuvo Kohonen, Stephen Grossberg, and James Anderson continued their research and eventually developed a theoretical basis for neural networks that rebuked Minsky's assessment as being overly general and based on a narrow set of single-layer architectures (Wasserman, 1989:6).

The period from the late 1970's to the present has witnessed another explosion of research in artificial neural networks. A number of different architectures using multiple layers, various computing elements and a host of learning schemes have been devised. Some have been optimized for pattern recognition, some for resource constraint problems, some for optimized control, and some for their ability to mimic the mental activity of the animal nervous systems at various levels of operation. This project is concerned with this latter area of neural network research.

In general, neural networks consist of multiple layers of individual computing elements. These elements, or artificial neurons, approximate the operation of certain biological neurons. They generally have a number of input ports linked to either the outside world or other neurons.

These ports have varying multiplication factors or "weights" that are applied to the input values. The input ports are analogous to the dendrites of biological neurons. Also like the biological neurons, the input lines may either excite or inhibit the activity of the parent neuron (determined by the sign of their weights).

The body of the artificial neuron, which corresponds to the biological soma, combines all the inputs, excitatory and inhibitory, and passes the resulting sum through a transfer function. The transformed signal is the output of the neuron.

In order to capture the power of additional neuronal layers, the neurons' transfer functions must be non-linear. The reason, alluded to previously, is as follows. It can be shown that any multi-layer network based on linear elements can be represented by an equivalent single layer net (Wasserman, 1989:19). This is not, however, true for networks with non-linear elements. Recall that it was single layer networks that Minsky and Papert demonstrated could not perform some simple functions (like that of the "exclusive or" logic gate). Therefore, the use of non-linear transfer functions results in more powerful networks by avoiding the limitations of the linear activation functions.

An example of a non-linear function is that of a simple step response. Many neural networks rely on this type of

function. The neurons in these architectures are termed threshold devices. In threshold devices the output is zero until the sum of its inputs exceeds its threshold. At that time, its output "steps" o some predetermined value, usually one.

The output of an artificial neuron may connect to the inputs of one or more other neurons. This is analogous to the axon of a biological neuron forming synapses with numerous dendrites of other neurons. As in the biological model, artificial neurons usually transmit the same signal to all connecting neurons. The received signal, however, is modified by the connection weighting values (and their signs) of the receiving neurons.

The critical function of the artificial neural network is its ability to modify the connecting weights between nodes. In fact, unlike a von Neuman-based computer with its centralized memory storage area, neural network data is distributed throughout the network encoded in the pattern of connection weights between the nodes. The node connection weights are analogous to long term memory in biological nervous systems.

The distribution of data throughout a system network of simple, parallel computing elements provides attractive features for both artificial and biological data processors. One feature is speed due to the potentially massive parallelism of the architecture. With many elements working

small parts of the problem at the same time, even large problems of the NP-complete variety can sometimes be solved in less than geological time units (Rogers, et al., 1990:2).

Another important feature of distributed processing/ distributed data storage is graceful degradation. Since individual elements do only small parts of the total job and since there are many paths through the network, loss of a few links or nodes will not result in total incapacitation of the system. Speed and graceful degradation are equally important for the control systems of both organisms governed by the Law of the Jungle and machines governed by the Law of Murphy.

If the utility of neural networks as computational devices is not yet clear I will offer one more analogy to illustrate their functional operation. The usual model used to illustrate the underlying principles behind neural networks is the mapping of activity in a higher dimensional space to that of a lower dimensional one.

To illustrate the concept of mapping, consider a typical three layer neural network consisting of eight input nodes, twenty hidden nodes in the second layer, and a third layer of three nodes. Assume that the input nodes are connected to the environment via sensors. The eight input values can be viewed as an eight dimensional vector. The propagation of the input signals through the second layer results in another vector of twenty dimensions (i.e., the

output of the second layer). At this point the original inputs have been mapped from an eight dimensional space to a twenty dimensional space.

Likewise, after the signals from the second layer propagate through the final layer, the result is a three dimensional mapping of the original eight dimensional input. This reduction of feature space dimensions represents the assimilation and classification/simplification of data. If the nodes used in the network were simple threshold devices with two states ("on" and "off"), the mapping would reduce the number of potential inputs from 256 (two to the eighth power) to 8 (two to the third power). If the mapping is performed with appropriate rules, the reduction of possibilities can make the data much more useful for control and decision support.

Presumably, this brief discussion of artificial neural networks will assist the reader in understanding the design and operation of the basic vehicle developed in this thesis effort. The balance of this chapter deals with these issues beginning with the vehicle sensors.

The Sensor Stage. The sensors provide the interface between the vehicle and its environment. There are three main types of sensors used in the vehicle. Two of these types are exterioceptors (i.e., sensors that monitor external events) that correspond to the two classes of stimuli found in the environment: light and sound. The

remaining type is an interoceptor (i.e., an interior state sensor) that signals changes in the energy state of the vehicle.

The Exterioceptors. The light and sound sensors are treated in identical fashion. Each sensor output is a function of four variables: intensity of the stimulus at the sensor, frequency response (the frequency of the stimulation relative to the response of the sensor to that stimuli), gain of the sensor "antenna", and the sensor transfer function.

The stimulus intensity arriving at the sensor is given by the inverse square law

$$E = I / D^2 \qquad (3)$$

where

E = the stimulus intensity at the sensor

I = the stimulus intensity one unit distance away

D = the distance of the sensor from the stimulus

The second variable affecting sensor output is the frequency response of the sensor to the stimulus. The frequency response of each stimulus to each sensor type is represented in Table 1 below.

For a particular sensor type, the data in Table 1 indicate the proportion of energy from a particular stimulus type that affects the sensor. The notations 1000, 2000, and 3000 refer to the center frequencies of the sound sensors.

The intersection of the stimulus type row and the sensor type column locates the proportion of the stimulus that affects the selected sensor. Thus, only 30% of the energy from a blue stimulus is effectively used by a green sensor. Note that there is no cross sensitivity of light sensors to sound stimuli or vice versa.

TABLE 1

SENSOR/STIMULUS FREQUENCY RESPONSE

		SENSOR TYPE					
		BLUE	GREEN	RED	1000	2000	3000
STIMULUS TYPE	BLUE	1.0	0.3	0.1	0.0	0.0	0.0
	GREEN	0.3	1.0	0.3	0.0	0.0	0.0
	RED	0.1	0.3	1.0	0.0	0.0	0.0
	1000	0.0	0.0	0.0	1.0	0.3	0.1
	2000	0.0	0.0	0.0	0.3	1.0	0.3
	3000	0.0	0.0	0.0	0.1	0.3	1.0

The next important variable, sensor gain, is determined by the sensor field of view response (FOV Response). Each sensor has a gain function which can be selected by the vehicle designer. This function represents the directional sensitivity of the sensor's "antenna." It is assumed to be symmetric about a line normal to the center of the sensor field of view. Four FOV Response curves are defined for the

vehicles: linear, parabolic, decay, and sigmoidal. The linear function is given by

$$G = M(1 - 1/R)A \tag{4}$$

The parabolic function is

$$G = M(1 - 1/R^2)A^2 \tag{5}$$

The decay function is given by

$$G = M(exp(-4A/R)) \tag{6}$$

The sigmoidal function is

$$G = M(exp(-(2A/R)^2)) \tag{7}$$

where

G = Effective sensor gain

M = Maximum gain (always at 0 degrees incidence)

R = FOV / 2 (in degrees, symmetry assumed)

A = Angle of incidence of the stimulus energy

These functions are presented as a graph in Figure 4. As can be seen in the plot, the gain of the linear FOV response curve decreases linearly until it equals zero when the angle of incidence equals the field of view limit. As the angle of incidence increases, the rate of change of the effective gain remains fixed (equal to the negative of max gain divided by half the field of view). The parabolic function features an accelerating negative rate of change as

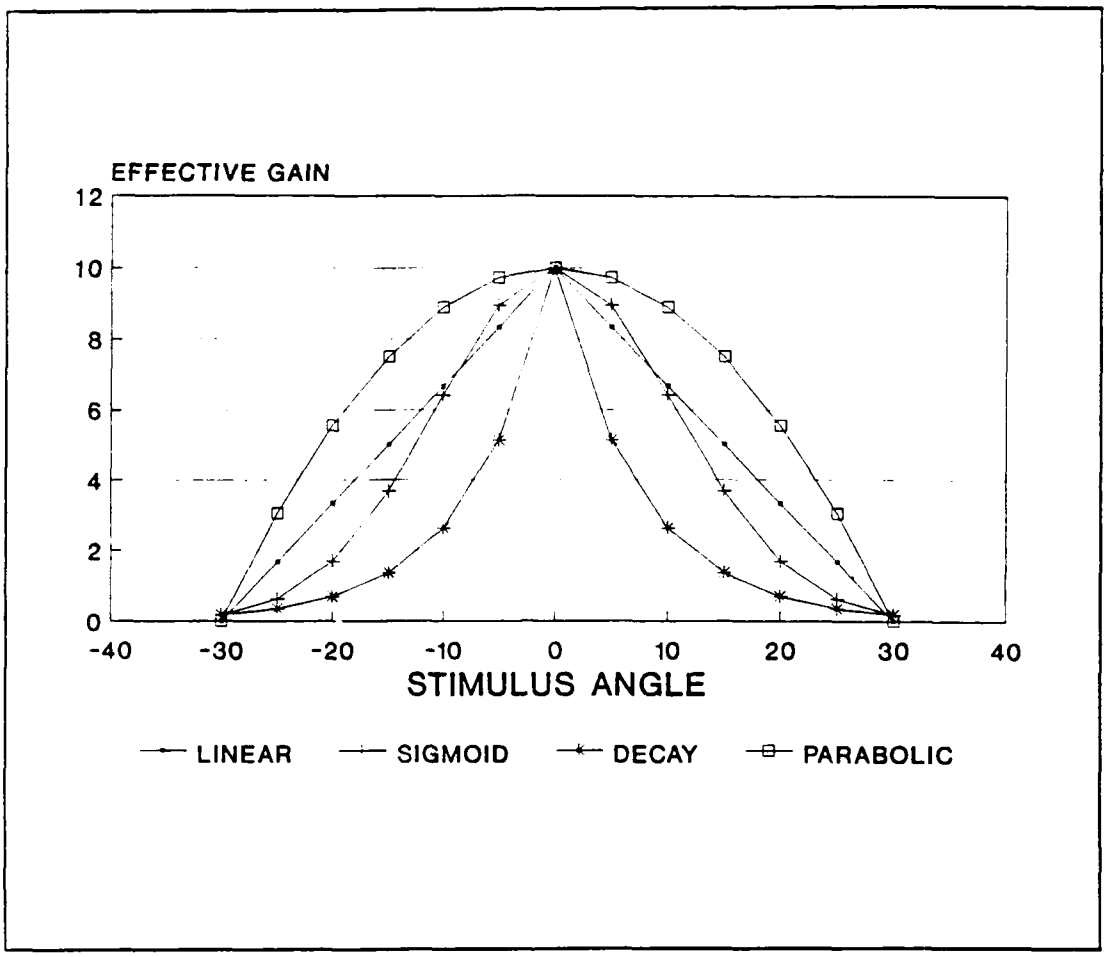


Figure 4
Sensor Field of View Response Curves

angle of incidence increases. The formula for the decay function, Equation 6, reveals a decreasing negative rate of change. The sigmoidal function begins with a negative accelerating rate of change as angle of incidence increases, but concludes with a decelerating rate of change.

This range of FOV response functions provides a vehicle designer with a number of options. Once the function type is chosen, the designer then specifies the maximum gain and the FOV range. An array is then generated that contains the

gain at each angle of incidence from 0 to 180 degrees. Since symmetry about the 0 degree angle is assumed, sensors with 360 degree coverage can be designed (-180 to +180 degrees).

The input to the sensor is calculated by summing the effective energy arriving from each source. The formula is given by Equation 8.

$$E_t = \Sigma_{i=1}^{n}(E_i F_i G(1/D_i^2)) \quad (8)$$

where

E_t = Total effective energy at sensor input

n = Number of stimuli

D_i = Distance from ith stimuli to sensor

E_i = Strength of ith stimulus

F_i = Sensor Frequency response to ith stimulus

G = Gain of sensor

The output of the sensor is then given by

$$S_{out} = E_t^{Fc} / C \quad (9)$$

where

S_{out} = Sensor output

E_t = Total effective energy at sensor

Fc = Fechner Law coefficient

C = Constant to restrict Sout < 1

When Fc is chosen to be between zero and one this function compresses the range of values assumed by the sensor output. As is noted in the Animal Psychophysics section of Chapter II, the range of stimuli intensities extends through many orders of magnitude. In order to provide sensitivity over this wide dynamic range, a compressing function is necessary. A value for Fc of 0.2 provides sufficient compression for stimulus intensities of less than 500,000 units. The constant, C, maintains the output of the sensor within the range of 0 to 1. This makes the sensor response consistent with that of the other neurons in the net.

A total of up to four of the sensors described above can be designed into any vehicle. Each sensor will have a related neuron, also treated as a sensor, that represents its output change since the last time period. This delta sensor provides sensitivity to rapidly changing sensations such as those that occur when the vehicle passes a stimuli in close proximity. Sensor output in this case can quickly go from a very large number to zero as the stimuli moves out of the sensor field of view. Since the detection of food occurs close to stimuli, this feature provides important information to the vehicle network.

Equation 9 is also used to compute the output of the delta sensors. Since the delta sensor outputs become large only under specific circumstances, a larger value for Fc is

usually selected to provide less compression. A typical value for Fc applied to delta sensors is 0.5.

With four possible "real" sensors and a delta sensor for each, the vehicle can have a total of eight sensors. This number seems to be sufficient to demonstrate the adaptive concepts that form the objectives of this research project.

The Interoceptor. The only interoceptor monitors the energy state of the vehicle. Its primary task is to signal the discovery and consumption of food. This signal is used to trigger the adaptive processes. It could be viewed as a two stage receptor like the sensor described in the preceding section. The sensor maintains both an absolute index of the vehicle's energy state as well as a delta index used to note the change in energy due to food consumption.

The energy state changes negatively based on the distance moved in the last time period and is incremented by a fixed positive amount for each food item discovered. The energy increment for food ingestion is much greater than the decrement that can be achieved for movement in a single time period.

The final item relative to both sensors is the propagation of the sensor signals to the hidden-layer neurons. The output of each sensor is connected to each neuron in the hidden layer (most of these connections are

not shown completed in Figure 3 due to the density of the connections). Each of these connections has a weight associated with it. The weights can be positive (excitatory) or negative (inhibitory). The sum of the absolute values of weights on the inputs of any neuron is always equal to one. Note that the sensors are not treated in this way, since their inputs represent links to the outside world (i.e., the sum of their inputs can exceed one).

The vehicles employ matrix multiplication to propagate internal signals between layers. The connection weights are stored in an 8 by 20 matrix in which the rows correspond to the output weights relating the sensors to the hidden-layer neurons. The columns of the matrix correspond to the connection weights of the neuron inputs to the sensors.

The propagation of the signals is performed by calculating the dot product of each element of a row vector containing the output values of the sensors with the column vectors of the matrix containing the sensor-to-hidden-neuron weights. The resulting vector represents the input vector to the hidden layer neurons. At this point the signals are ready to enter the hidden layer neurons for processing.

The Hidden Layer Neurons. The primary purpose of the hidden layer is to associate sensory inputs with events that favorably affect the vehicle (i.e., the discovery of food). Based on each neuron's inputs and the weights on the input

connections, the hidden layer processes sensory inputs and transmits processed outputs to the drive motor neurons. When the energy state delta sensor signals that food has been found, the neurons with the highest outputs adjust the weights on their input lines. This weight adjustment process defines adaptation or "learning" in the vehicle.

The neuron itself is a simple device which sums the products of input signal strengths from each sensor and their corresponding connection weights. This summing process is described in the last section as the propagation of signals from the sensors to the hidden layer neurons. The resulting input signal strength of each neuron is then processed through a transfer function to calculate the neuron's output. The function used is

$$O = 1/(1 + e^{-KI}) \qquad (10)$$

where

O = Neuron's output

I = Neuron's input

K = Constant to control the slope of the curve

There are several important aspects of this formula as a neuronal activation function. First, the formula is defined for all values of I. Second, the output, O, is constrained to the range $0 < O < 1$. This offers protection against large signal surges that can disrupt operation of the net or lock it into undesirable states. But, most

importantly, the shape of the function allows signal strengths in the mid-range (about 0.5) to exhibit maximum sensitivity to small changes while reducing sensitivity beyond this area (Wasserman, 1989:15-16). To illustrate, reference Figure 5. This figure plots the function represented in Equation 10 with the variable K = 1. Note that at the point where the input equals 0.5 the slope of the line is at a maximum. Therefore, small changes in the input strengths of signals near 0.5 exhibit proportionately greater impact on the output than at other

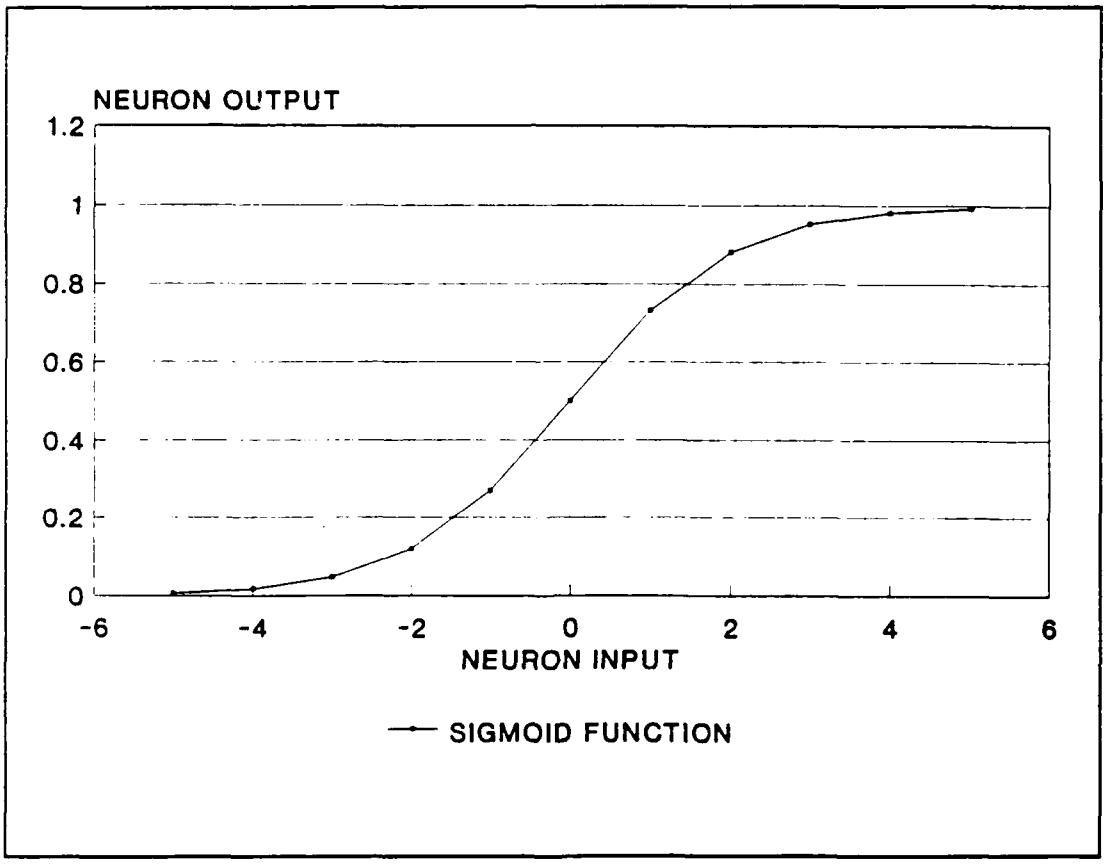


Figure 5
Neuron Transfer Function

strengths. With connection weights and outputs confined between zero and one, the occurrence of signals strengths in the 0.5 range is relatively large. Since the sum of the absolute values of the inputs cannot exceed one, the value for K used in the vehicles has been set to three. This value ensures that neuron outputs can use most of the zero to one range.

The learning algorithm employed by both the hidden layer and the motor drive neurons is attributed to Kohonen (Wasserman, 1989:67; Soucek, 1989:76). It features a strategy in which connection weights are changed based on the magnitude of the inputs when the learning is triggered and the existing connection weights. The actual formula is given by

$$W_n = W_o + C(I - W_o) \tag{11}$$

where

W_n = New connection weight

W_o = Old connection weight

C = Learning rate

I = Input strength on the connection

This algorithm has several important features. First, if the node weights sum to one prior to learning, they also will sum to one after adjustment. This insures that all the node weights will not eventually go to one. The second feature is related to the importance of the signal strength

to the weight adjustment. The training function results in the connection paths with signals greater than their existing weights having their weights adjusted upward. Conversely, the paths with weights larger than the magnitude of their inputs have their weights adjusted downward. The effect is to drive the ratios of the weights to be equal to the ratios of their usual input values. This will tend to maximize the neuron's output for a given input pattern.

To understand the adaptive process in operation, imagine the vehicle chancing upon some food. Since the food is found in the vicinity of a stimuli, some sensors are likely to have high outputs. When the learning algorithm is triggered, the neurons with the highest sensory inputs will have their weights adjusted upward. In similar future circumstances these neurons will have higher outputs that are sent to the motor drive neurons. These higher outputs will exert more influence on the drive neurons than those of less adapted hidden layer neurons.

Note that many food finding events may occur with small sensor outputs. If, for instance, the vehicle has passed the stimulus with the food and is in the process of increasing its distance from the source, the stimulus may be completely out of the field of view of the sensor. In actuality this is a common occurrence. To improve the vehicles' ability to deal with such instances, they need

short term memories (STMs). These memories will allow the neurons to train based on input states from the recent past.

The STM scheme used in this project is based on a characteristic of many biological neurons. Once stimulated, neurons tend to fire at an accelerated, but decaying, rate even after the input is removed. This "firing rate" memory encoding provides some ability to correctly associate rapidly changing signals with favorable, recently-past events. The "ringing" of the recently excited neuron provides the input values to ensure additional opportunities to form the "right" associations.

While this memory scheme can help the formation of some associations, it can also hinder performance. When applied to sensory information, for instance, the result is less responsiveness when the vehicle is close to a stimulus. This is due to the affect of the "after image" from the sensor firing rate decay. The vehicle requires more time to sense the rapidly changing environmental situation. The software contains a switch to enable this feature to be turned "on" for training the hidden layer. It can then be turned "off" for the hidden layer neurons when the drive layer is being trained.

The output of the hidden layer neurons is propagated to the motor drive neurons in the same manner as is done from the sensor to the hidden layer. The dot product is calculated for an array containing the hidden layer outputs

and a matrix containing the hidden layer-to-motor drive neuron connection weights. The resulting three element vector becomes the input to the motor drive neurons.

The Motor Drive Neurons. As is indicated in Figure 3, the purpose of the three drive neurons is to provide three possible movements: turn left, go straight, turn right. The three motor drive neurons are identical to the hidden layer neurons. However, where as the outputs of all neurons in the hidden layer are propagated to the drive neurons, the drive neurons work in a somewhat winner-take-all fashion. Only the neuron with the highest output is trained when food is discovered. The reason for this is related to another difference between motor drive neurons and hidden layer neurons.

This second difference is related to the manner in which the drive neurons' outputs are used. Among the drive neurons only one neuron affects the operation of the vehicle directly. The drive neuron with the highest output determines what direction the vehicle will go. When food is found, it is assumed that the last move made was favorable to the discovery. Therefore, the probability the vehicle will make the same movement with similar patterns of stimuli will be enhanced by training the neuron which controlled the last movement.

The actual movement made by the vehicle is determined by the outputs of the drive neuron. When the "go straight"

neuron has the highest output, the vehicle moves on the same heading a distance proportional to the output of the neuron. The "turn" neurons (D1 and D3 in Figure 3) also cause the vehicle to move a distance proportional to their output, but the radius of the turn made is based on the ratio of the winner neuron's output to the output of the other "turn" neuron. This maintains the Braitenberg vehicle drive concept in which the relative speeds of the drive motors determine the course of the vehicle. A heuristically derived formula for calculating the turn radius was chosen. This formula is given as

$$R = 5(10 - 10(O_w - O_l)) \qquad (12)$$

where

R = radius of the turn

O_w = output of winner turn neuron

O_l = output of loser turn neuron

When the simulation was first run, the test vehicle had 15 hidden layer neurons and 4 true sensors (8 including the delta sensors). The vehicle tended to adopt a single behavior pattern despite the input stimuli. If the first movement made was a turn, it continued to turn. This seemed to be due to the randomization of the initial node connection weights and the number of nodes. Apparently, one output neuron always had the highest output, thereby, taking control of the vehicles movement. This created a serious

problem for the vehicle. If learning did occur it could only strengthen the control of the winning neuron. This would perpetuate the single-move condition.

The solution was to begin the vehicle's "education" with near-random motion. Based on an exponential decay function, the contribution of the neurons' connection weights to the output of the neurons is kept small. The rest of the output is made up of random inputs injected every 20 time periods. This results in the vehicle taking a pseudo-random walk. As the vehicle matures, less and less randomness is injected until finally the behavior is based solely on the connection weights.

This "childhood" feature is also used to force the vehicle to experiment with other behavior patterns when the energy state is either too high (satiated) or too low (starving). The idea is to provide opportunities for the vehicle to learn superior behavior patterns even if experiencing some success with the current connection weights.

The "age" of the vehicle is used in another way to modify learning. Young vehicles experience faster learning based on a higher values for the learning rate variable in Equation 11. This added learning rate gradually decays until the rate reaches a small steady state value. At this time the "old dog" can still learn new tricks, but it takes longer than does a young pup.

This concludes the description of the general design of the vehicles. To provide a better understanding of the simulation program the next section steps through the operation of the code during a complete cycle of movement. The steps equate to the main section of the simulation code.

The Simulation Operation

The simulation program begins by initializing the data structures and retrieving the records which contain the definitions of the vehicle and environment. Following these operations the program retrieves a file of distance measures that are stored in an array. This array provides rapid access to distances between points in the simulation world. The data is referenced by entering the differences in the X and Y coordinates of the two points. This array structure eliminates the need to perform squaring and square root operations to calculate the distance values. This speeds the program execution.

The program next generates a report header containing information about the vehicle run conditions: date, environment name, vehicle name, beginning connection weights, etc. The video display is then switched to graphics mode. The environment and vehicle are displayed on the screen. Small rectangles represent the stimuli while a diamond figure with a protruding line represents the vehicle. The line indicates the vehicle's current heading.

The program then calculates the sensor stage input based on the location of the stimuli relevant to the starting location of the vehicle and the sensor/stimuli characteristics found in the vehicle and environment definition records. Equation 8 defines the relationship between the variables and the total input to each sensor. The results of the calculation are stored in an array.

As the movement calculations begin, a timer is set to record the number of iterations made through the movement loop. The loop terminates when the user entered value for stop time is met.

The loop begins by calculating the output of the sensor stage using the array containing the sensor input data. Equation 9 is used to generate these values. The outputs are stored in another array for use by the matrix multiplication routine that propagates the signals to the hidden layer neurons. This routine uses the sensor output array and the sensor-to-hidden layer array of connection weights to generate the inputs to the hidden layer. These inputs are also stored in an array.

The hidden layer input array is used along with Equation 10 to calculate the output of the hidden-layer. This output is propagated to the motor drive neurons in the same manner as was the sensor to hidden layer signals. The propagated signals become the input to the motor drive neurons.

The drive neuron outputs are calculated as are the hidden layer outputs with one exception. To implement the short term memory, the output value of each neuron is compared to its value at the last time period multiplied by a decay factor. The output becomes the greater of these two values. The result of this operation is that the neuron exhibits a retention of output for a few time periods even after all inputs go to zero. The intention is to improve the association of sensory patterns from the hidden layer to the discovery of food even when the vehicle has passed the stimuli and no actual stimulation is at work. In this case, the neurons' outputs will continue to "ring" even while the sensors show no input. This ringing can provide the training algorithm (Equation 11) with a more accurate assessment of the cause and effect relationships when adjusting the connection weights.

Once the motor drive neuron outputs are calculated, the actual movement can be generated. Equation 12 is used to determine the radius of the arc traveled (if a turn neuron has the greatest output). The actual distance is proportional to the output of the most active neuron.

Following the move, the program determines whether food was found. The probability is based on the inverse square of the average distance of the vehicle from each stimuli multiplied times the distance moved. More than one food item can be found during a move. If the number found is

greater than zero, the node connection weights are adjusted using the learning algorithm represented by Equation 11. The value used for the rate-of-learning constant is proportional to the number of food items found. At this point, the stimuli energy at the sensors is recalculated based on the new location of the vehicle and the loop begins again.

Note that new signals entering the neural network are propagated through the entire network during each time cycle. In some networks, feedback loops and propagating signals are calculated iteratively until the network assumes a stable state. This approach, while more faithful to simulating hardware implementations of neural networks, was impractical for a microcomputer-based simulation intended to run in real time.

IV. Vehicle Evaluations

As was mentioned earlier, the acid test for success in this project was the demonstration of adaptation or learning by the vehicles. A number of methods might be used to verify whether the vehicles were successful at improving their performance over time (the operational definition of learning for this effort). First, one must select a credible measure of merit for performance. A number of metrics are available from the vehicles and their behavior including average distance to food-related stimuli, number of learning events (these are triggered by discovery of food), amount of food found, and vehicle energy states over time.

Performance Measures

Average distance to food-bearing stimuli lacks sensitivity since the vehicles are confined to a one-screen world. Some behavior pattern also emerge where the vehicles orbit stimuli, but never get close enough to find food. These vehicles will get relatively small average distance scores, but without actually having learned to find food.

Although the number of training events will be positively correlated with energy state, it will not be a sensitive measure due to the possibility of multiple food items being found in one time period. Therefore, two vehicles with identical numbers of training events may

differ considerably in their actual success rates of finding food.

The number-of-food-items-found provides more information on vehicle success than the other measures. Its only short coming is a lack of sensitivity to food finding efficiency. Since the vehicles expend energy by moving, more successful vehicles should find more food with less movement.

For this reason, energy state over time provides the most accurate performance measurement. Since multiple food items can be found in a single move, it will deliver more precision than analyzing the number of training events alone. It also contains an element of food finding efficiency not included in the number-of-food-items-found measure since it takes into consideration the distance moved as well as the food found.

The Evaluation Vehicles

The initial intent was to design a controlled experiment in which vehicles with randomized starting connection weights would be run over a period of time. The energy scores would then be analyzed for significant trends. This would require a number of vehicles to be tested to ensure that results were not an artifact of the beginning connection weights.

During the validation of the simulation code, another, more compelling, test paradigm became apparent. In order to

determine whether the program was performing as intended, several of the basic Braitenberg designs were built and tested (see Figure 1). Vehicle 1A is intended to avoid the stimuli. What would happen if this vehicle were provided opportunities to learn to find food? Would it develop stimuli-seeking behavior like that of Vehicle 1B? This suggests a true acid test for the learning algorithm.

This concept was chosen for the test paradigm. The beginning vehicle was configured as shown in Figure 6. Two symmetrically-placed sensors provide inputs to two hidden layer neurons. The sensor-to-hidden layer connection weight values between components on the same side of the vehicle are fixed at one. Cross-side connections weights are fixed at zero. The "delta sensors" were effectively disabled by setting their connection weights to the hidden layer to zero. Since the hidden layer is omitted in the "avoider" vehicle (see Figure 1, Vehicle 1A), training for this layer is disabled.

The hidden layer neurons feed the drive layer with starting connection weights shown in Table 2. Note that the middle drive neuron has both of its input connection weights set to zero. Since it will never see inputs greater than zero it can never "win" against the other neurons. Therefore, it will never receive training and its weights will never change. This architecture effectively duplicates the "avoider" vehicle (Vehicle 1A) from Figure 1.

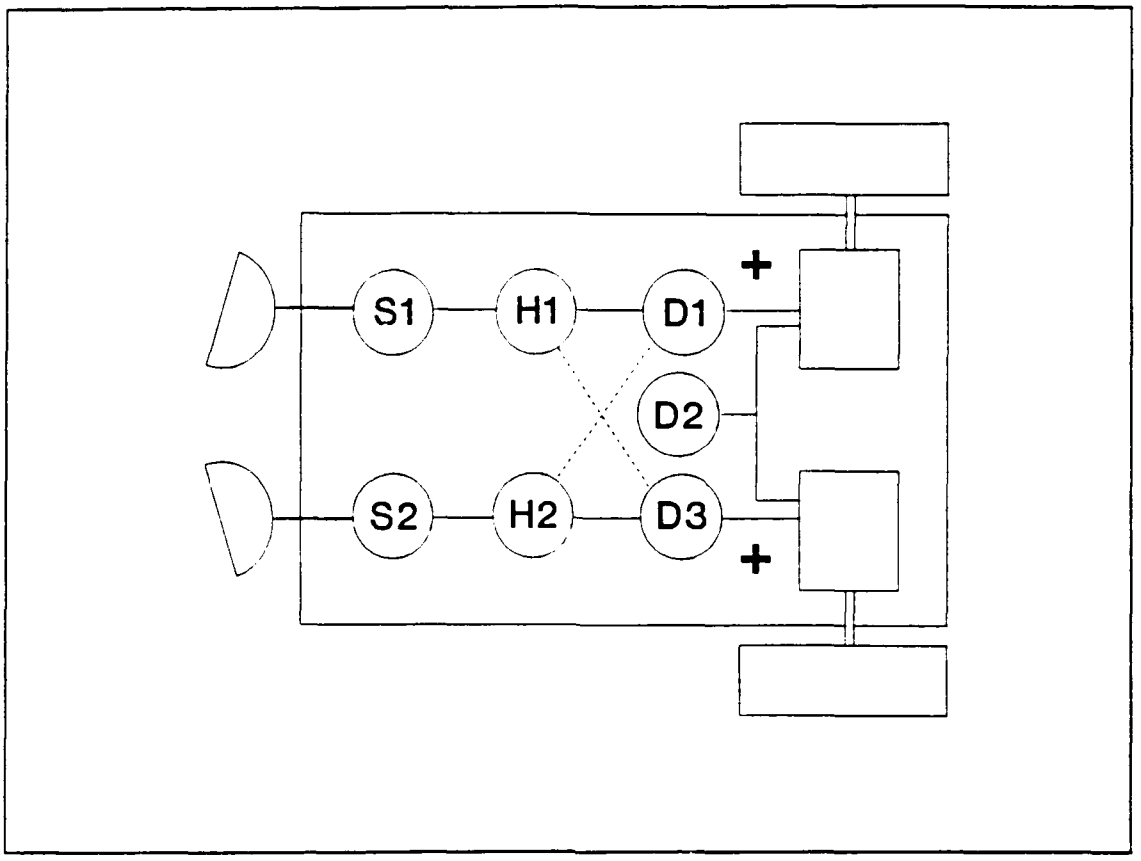


Figure 6
Evaluation Vehicle

The environment used for the evaluations contained three food-associated stimuli whose type corresponded to the sensor type on the vehicle. The sensors were chosen to have parabolic field of view responses with a maximum gain of 10 and a cut-off of 30 degrees (see Figure 4). The sensors' look-angles were set symmetrically at 30 degrees from the vehicle centerline. This resulted in Sensor 1 having a -30° look-angle while Sensor 2's look-angle was set to 30°. During training, the learning rate coefficient (see Equation 11) was made a function of success rate by setting

it to 0.2 times the number of food items found in the last time period.

Table 2

Hidden Layer-to-Drive Layer Connection Weights for Evaluation Vehicle

Hidden Layer Neuron	Drive Layer Neuron	Connection Weight
1	1	1.0
2	1	0.0
1	2	0.0
2	2	0.0
1	3	0.0
2	3	1.0

All runs lasted for 8000 time periods. For the first 6000 iterations, the "childhood" random movement feature was enabled. This feature provided random behavior to the drive layer in linearly diminishing amounts until, at time equal to 6000, all behavior was controlled by the connection weights. As explained in Chapter Three, the childhood stage provides the necessary opportunity for desired learning by temporarily forcing the vehicle out of its "natural" behavior pattern.

Even after the childhood stage ends, if the vehicle experiences an extended interval without a learning event a short interval of randomness is inserted. This can kick a vehicle out of an unprofitable orbit about a stimuli or

provide opportunities for new learning experiences to a partially successful vehicle.

Evaluation Results.

The nature of this evaluation suggested another success metric in addition to energy state statistics. Since the aim was to demonstrate the reversal of behavior patterns from avoiding stimuli to seeking them, the node weights for the simple test vehicles provide important information. Of interest are the node weights between the hidden layer neurons and drive neurons one and two. When the weights between both hidden layer neurons and their opposite-side drive neurons exceed those to their same-side drive neurons the vehicle behavior will reverse.

This behavior reversal will occur when connection weights H1-D3 and H2-D1 both exceed 0.5 (recall that all node weights into a neuron sum to 1.0). For this evaluation, a graph indicating the change in these connection weights can effectively show the progress of the learning process. Such a graph is presented in Figure 7 for a typical evaluation run. Note that the scale for connection weights H1-D3 is located on the left ordinate and the scale for the weights H2-D1 is on the right ordinate. Also note the reversal of the direction of increasing connection weight on the two scales. This provides maximum clarity of the learning artifacts.

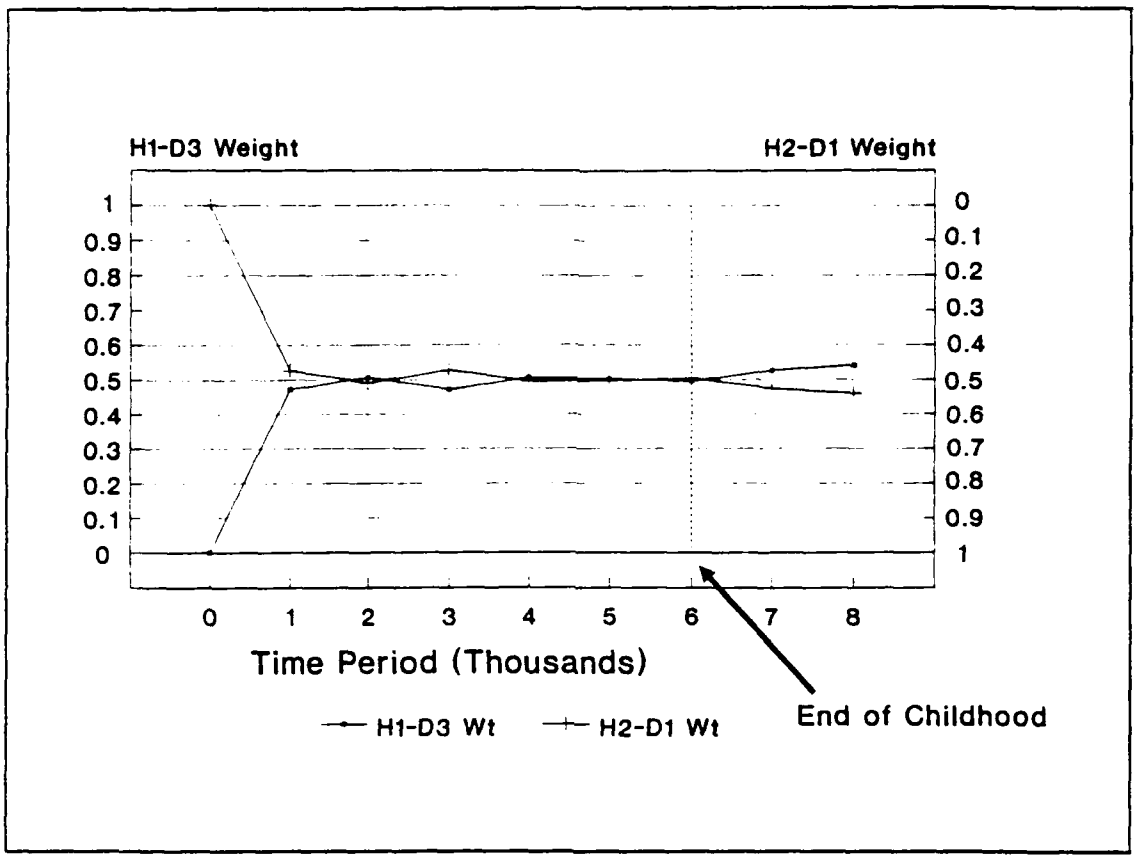


Figure 7
Hidden Layer-to-Drive Layer Connection Weights

The data in Figure 7 indicate a rapid initial convergence toward the behavior changing point. As the connection weights approach the break point, however, the monotinicity disappears and the values tend to waver just above and below the 0.5 mark. The rapid convergence is attributable to the nearly 50-50 odds that a particular reinforced move will be "right" or "wrong." Recall from Chapter III that the drive neuron which is trained following a learning event is the one with the highest output during the last time period. The assumption made is that the

vehicles will be rewarded more often when they turn toward the stimuli than when they turn away.

This effect is not great, however, since the vehicles do not tend to continue a particular movement for long during the latter stages of the childhood phase. In fact, the vehicles often exhibit a servo-like hunting response in which the heading changes rapidly from left to right and back again. If the trajectory carries a vehicle close to a stimuli while exhibiting this behavior it is almost as likely to be wrong (turning away) as it is to be right (turning toward the stimuli) when it encounters food.

It is often the last training event before the childhood expiration that dictates the fortune of the vehicle. Correct learning results in a vehicle which begins to be rewarded and has its connection weights tipped permanently toward the "attractor" configuration. If the last training is incorrect, the vehicle loops across the corners of the screen in an attempt to maintain maximum distance from the stimuli. In this case the node weights are not changed since the vehicle never approaches the stimuli. Training can, however, occur in this case if the temporary childhood feature senses extended periods without food discovery. If the resulting random movement results in food discovery, the weights can be "tipped" toward the attractor scheme. They will then be quickly reinforced by additional "correct" learning events.

Inspection of the vehicle-energy-over-time plot, Figure 8, along with the connection weights chart, Figure 7, demonstrates the sensitivity of vehicle success to the node weights. The vehicle energy is relatively steady-state during the random motion of its childhood phase. During this period food discovery is largely a matter of chance. At childhood's expiration (time = 6000), however, the reoriented connection weights take control and result in numerous food discoveries. Vehicle energy quickly soars to the maximum of 30,000 units. This energy glut is precipitated by a connection weight separation of only a few percent on each side of the 0.5 threshold.

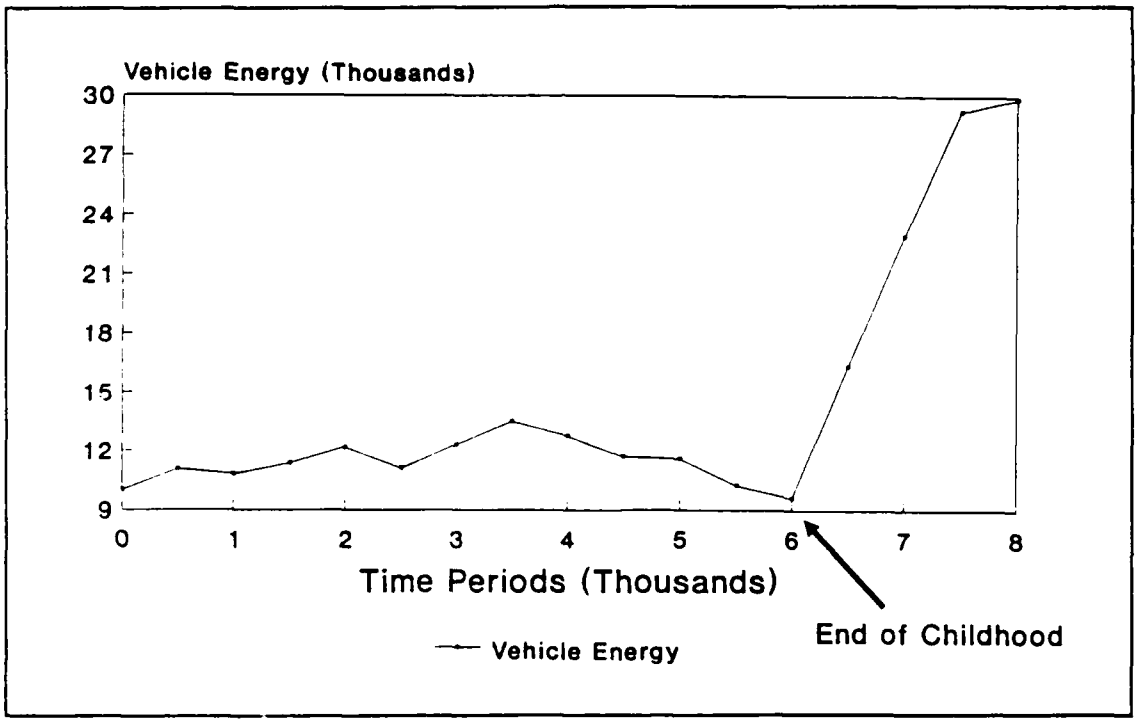


Figure 8
Vehicle Energy Over Time

As was mentioned earlier, the results shown here are not necessarily the norm. On some runs the connection weights do not make the reversal state, only one link does, or the weights reverse several times, but are not in the correct state when childhood ends. In order to investigate possible reasons for the lack of learning robustness, a switch was added to the simulation to allow an experimenter to control reinforcement of behavior. The result is a sort of electronic "Skinner box" in which behavior can be "shaped" (see Chapter II, Operant Conditioning). By providing a high frequency of correct learning events, the connection weights can be made to rapidly reverse their strengths. This suggests problems exist in the vehicles with the association of movement and sensor patterns. Most likely the problem lies with learning events occurring after the sensors have lost sight of the stimuli (as the vehicle passes close by). Training may often be occurring to a blank field of view in reduction of weight for desirable input patterns. Coupled with the tendency for movements to change rapidly, the opportunity for incorrect learning is pronounced.

This evaluation resulted in a mixed review. While the vehicles sometimes manage to achieve the goal state, they also sometimes fail. The normal time restrictions on thesis projects precluded a test program involving the full range of parameters that can be "tweaked" in the simulation

software. It is quite possible that there is a pony in there somewhere waiting for the right young man with a shovel (and high-topped shoes)!

V. Summation

This chapter presents three concluding discussions. First, is a project synopsis concentrating on the simulation design features and how they correspond to animal models of learning. The second section, "Conclusions," presents the author's assessment of the project. The final section presents recommendations for further study and analysis.

Synopsis

The objective of this effort was to develop a computer simulation that enables an experimenter to design and evaluate simple autonomous vehicles capable of exhibiting unsupervised learning. The model for the vehicle architecture was taken from the treatise by Valentino Braitenberg, Vehicles: Experiments in Synthetic Psychology (Braitenberg, 1984). The implementation of the simulation was heavily influenced by artificial neural network research and investigations of animal learning from the experimental psychology literature.

The simulation software, written in Turbo Pascal (see Appendix), provides a great deal of flexibility in the design of vehicles and their environments. Six types of stimuli and corresponding sensors are available to the experimenter as well as control of sensor fields of view, look-angles, and antenna gains. Each vehicle can have a maximum of four primary sensors and four associated "delta"

sensors which produce first derivatives of the primary sensor outputs.

The vehicle "brains" are made up of two layers of artificial neurons employing the customary sigmoidal logistics function to provide non-linear outputs. The two layers are interconnected by adjustable weighted links that propagate signals through the neuronal network. A software switch allows the hidden layer of up to 20 neurons to imitate mutual inhibition by adopting a "winner-take-all" output scheme.

The output layer of three neurons controls the vehicles' movement. The neuron with the highest output dictates left turn, straight, or right turn moves. Distance moved is proportional to the output of the winning neuron. Turn radii are determined by the output ratios of the two "turn" neurons.

The neurons have a primitive short term memory based on a feature of natural neurons. Once stimulated, neurons tend to reduce their output as an exponential decay even if input to the neuron falls to zero. This provides a simple means of relating present events to past ones.

The learning scheme is one attributed to Teuvo Kohonen (Wasserman, 1989:67; Rogers, et al., 1990:68). The implementation ignores certain features of Kohonen's scheme (establishment of "neighborhoods" of similarly responding neurons) due to the small size of the network.

Neuronal learning is triggered by the discovery of "food" in the environment. The food is distributed about designated stimuli to which the vehicles' sensors respond. The learning event results in the vehicle adjusting its connection weights between neurons. Adjustments are intended to increase the probability of responding to sensory patterns in the future in the same way that led to the past discovery of food. The assumption is that due to the increasing density of food as one gets closer to the stimuli, turning toward stimuli will result in more behavioral reinforcement than turning away.

The simulation software runs sufficiently quickly to operate in real-time on an Intel 80286 processor-based microcomputer. The program records statistics during evaluation runs including vehicle energy states, inter-neuron connection weights, initial conditions, and numbers of learning events. Vehicles can be "saved" at the end of runs to retain the connection weights they have learned.

Conclusions

The simulation design incorporates features to enable the vehicles to imitate some aspects of the learning behavior of animals. Animal learning phenomena such as the weakening of associations between events displaced in time is simulated using the decaying short term memory trace discussed in the preceding section. The role of reinforcement magnitude in learning is also mimicked by

multiplying the number of food items found times the base learning coefficient before adjusting connection weights.

The overlap of the frequency responses of some sensor-types provides opportunities to imitate generalization of learning to similar stimuli. Since the sensor output is reduced for non-matching sensor-stimuli pairings, the response magnitude will be reduced as it is for animal subjects (see Chapter III, Classical Conditioning).

The simulation demonstrates operant conditioning in the same manner as is done with animals: the actions of the vehicle are, in a sense, voluntary. The incorporation of provisions for "shaping" vehicle behavior provides an additional analogy to animal learning. A final feature of the simulation provides the vehicle with the potential for learning as was noted by puzzle-box experimenters. The phenomenon of cats repeating specific movements to gain release from confinement when any number of means were effective at opening the door (Mazur, 1986:120,122) is imitated by the learning of the drive layer. Movements which lead to reinforcement are more likely to reoccur in future circumstances.

The vehicles, like animals, also have two-stage memories. The decay function of active neurons serves as short term memory while long term memory is effected by the connection weights between neurons. Short term memory fades quickly while the long term memory is retained until supplanted by new learning.

The vehicles' sensors are given a large dynamic range like natural sensors. They also use a physical magnitude-to-sensation level mapping that is close to that found in human subjects (Equation 2).

At the neuronal level, basic functions imitate those found in some biological units. Inputs can stimulate or inhibit receiving cells. The vehicles' neurons also provide analogies to both spatial and temporal summation. Finally, the connection-weight changes are analogous to the manifestations of learning found by a number of researchers at the synaptic level (Alkon, 1989:44).

Evaluations of a few vehicle designs indicate that the vehicles are not always successful at learning the desired associations. The problem appears to center on the unreliability of reinforcement. Many times incorrect behavior can be reinforced, thus negating previous learning. The success of the test vehicles was very sensitive to small changes in the connection weights. This effect was exacerbated by the sparsity of the neural network in the test vehicles.

Recommendations

Due to time limitations, the simulation received only cursory evaluation. There are a numerous options for designing vehicles that have not yet been explored. For instance, changes to the "avoider" vehicle giving it wider fields of view, overlapping forward coverage by both

sensors, and better side visibility may improve its ability to learn to turn toward stimuli as opposed to turning away.

There may also be reasons to improve the short term memory beyond the simple version implemented in the simulation. A better scheme for short term memory might be a bucket brigade of delay elements which all feed back to the neuron (Figure 9). In this arrangement, output from the primary neuron passes through the bucket brigade, being delayed at each bucket. The feedback link from the first bucket presents the "memory" of the neuron's output at the last time period. The second element feeds back the output from two time periods ago, and so on (MacDonald, 1988:510). The feedback signals would be subjected to a decay function to preserve the relative importance of the most recently occurring events over earlier events.

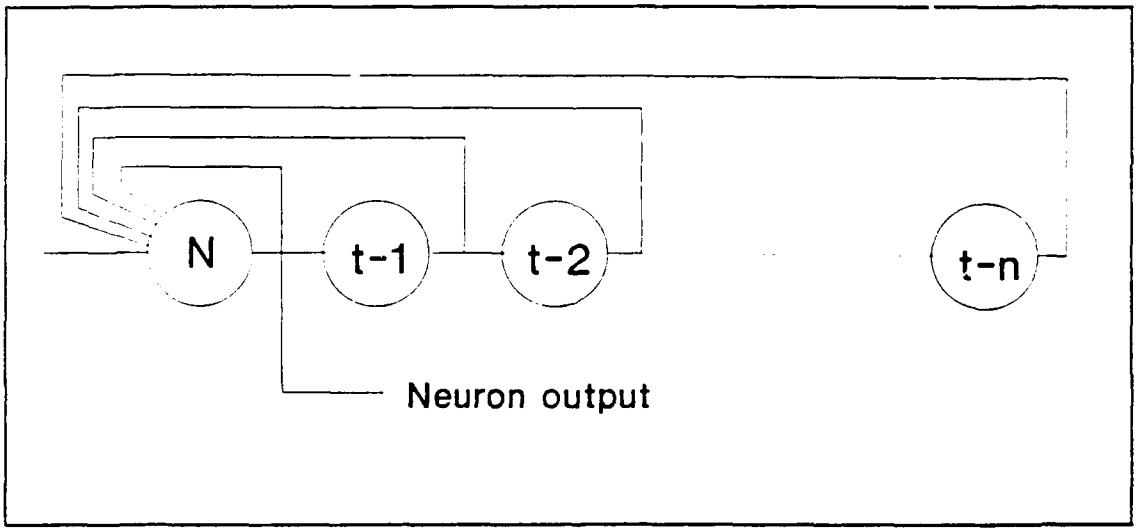


Figure 9
Bucket Brigade Short-Term Memory

This short term memory scheme combined with the learning algorithm could result in formation of stronger associations between time-separated events that occur with relatively uniform delays. Such a short term memory scheme may allow associations and conditioning behavior to form that are similar to those seen in classical and operant conditioning under the various temporal experimental paradigms (see "Operant Conditioning," Chapter II).

In conclusion, the product of this thesis effort is a significant tool for experimenting with artificial neural networks. It allows the user to evaluate neural network behavior in much the same manner as that used by psychologists in studying the learning behavior of animals. Although the network is somewhat sparse, the visibility associated with its working provides greater opportunities for identifying relevant relationships.

Appendix: Simulation Source Code


```
(*************************************************************************)
{  Turbo Pascal 5.5 code to implement the simulation described in this }
{  document.  The code is fairly well commented, but is presented here }
{  for illustrative purposes only.  All rights outside of Government   }
{  use are specifically reserved.  Inquires should be posted to the    }
{  address found in the Vita at the end of the thesis document.        }
(*************************************************************************)


{$IFDEF CPU87}                                  {Is 8087 in this machine?}
{$N+}                                           {Compile 8087 code       }
{$ELSE}                                         {else}
{$N-}                                           {Emulate 8087 operation  }
{$ENDIF}

{$R+}
{$M 32000,0,655360}

Program RunTest;

USES DOS, CRT, GRAPH, VCOORD, PRINTER, KYBD;

TYPE

   FType          = (Linear, Sigmoid, Decay, Parabolic);

   StimType       = (R {ed} , G {reen} , B {lue} ,
                           Hz1 {000} , Hz2 {000} , Hz3 {000} );

   SenResp        = array[0..5, 0..5] of Real;

   FileName       = string[20];                 {File name w/o ext.       }

   RunRec = record                              {Control rec for run      }
               RunFileName    : FileName;       {Assume .run extension    }
               VehName        : FileName;       {Assume .veh extension    }
               EnvFileName    : FileName;       {Environment file name .env}
               TrackFile      : FileName;       {Vehicle movement data .trk}
               StatFile       : FileName;       {Statistics on run     .sta}
               RunStart       : Integer;        {Starting time            }
               RunLength      : Integer;        {Number of time periods, t }
               BeginX         : Integer;        {starting x,y locations   }
               BeginY         : Integer;
               BeginHdng      : Integer;        {Vehicle heading in degrees}
               Decay          : Real;           {Exponent FOR decay FORmula}
            END;
```

```
StimRec = record                              {Defines stim attributes }
            Stimulus       : StimType;  {Stim type (R,G,B,Hz1..3)}
            Nutrient       : Boolean;   {Nutrient associated?    }
            Pos            : Boolean;   {ReinFORce/punish        }
            StimX          : Integer;   {Stimulus location       }
            StimY          : Integer;
            StimEnrg       : LongInt;   {Stimulus energy         }
          END;

StimuliData  = Array[1..8] of StimRec;

EnvRec = record                               {Defines world          }
           MaxXScrn        : Byte;      {World dimensions       }
           MaxYScrn        : Byte;
           TX,TY           : Integer;   {Top lft/bottom rt corner}
           BX,BY           : Integer;   { virtual screen coords  }
           StimNum         : 1..8;      {How many stimuli? Max 8 }
           Stimuli         : StimuliData;{Stimuli data          }
         END;

Sen_Fov_Resp   = array[0..180] of Real; {Angular gain of sensors }
                                        {[ang of stim]          }

Sensor = record                               {Defines sensor attribute}
           SResp           : StimType;  {Sensor response        }
           CurveType       : FType;     {Sensor FOV response    }
           SAngle          : -180..180; {Sensor angle to vehicle }
           MaxGain         : Real;      {Peak sensor gain       }
           CutOff          : 1..180;    {Max sensor off bore resp}
           Curve           : Sen_Fov_Resp;{Gain vs angle of stim }
           SenK            : Real;      {Stevens law CONSTant   }
           Sen2K           : Real;      {Stevens law exp FOR delta}
         END;

TOut    = Record                              {Energy decay data (STM)  }
   Time   : Word;
   Out    : Real;
END;

SenArray    = Array[1..4] of Sensor;

SenOutArray = Array[1..8] of Real;

S_O_A       = Array[1..8] of TOut;

SenT1Wts    = Array[1..8,1..20] of Real;

T1Out       = Array[1..20] of Real;

T_O_A       = Array[1..20] of TOut;

T1T2Wts     = Array[1..20, 1..3] of Real;
```

```
T2Out       = Array[1..3] of Real;

T2_O_A      = Array[1..3] of TOut;

VehicleRec  = record                        {Vehicle definition      }
                VehName          : FileName;
                VehX             : Integer; {Veh coords rel to scrn  }
                VehY             : Integer; {  center.  Max= 100,100 }
                VehHeading       : Integer; {Heading in degrees      }
                VehEnergy        : Integer;
                Age              : Word;    {Previous age of vehicle }
                MaxEnergy        : Integer;
                MotResp          : Byte;    {Motor response function }
                FstMutInh        : Boolean; {1st layer mutual inhib? }
                SndMutInh        : Boolean; {2nd          "          }
                Sen_Num          : Byte;    {Number of sensors (1-4) }
                SenRecs          : SenArray;{Sensor definitions      }
                Sen_Hz_Resp      : SenResp; {Freq resp of sensor/type}
                Sen_Out          : SenOutArray; {Sensor[i] output    }
                SnOutArray       : S_O_A;   {Sensor energy decay data}
                Sen_T1_Wts       : SenT1Wts;{Sensor-1st T-cell wts   }
                NSum             : Byte;    {Sum hidden layer nodes  }
                T1_Out           : T1Out;   {1st T-cell outputs      }
                TOutArray        : T_O_A;   {T1 energy decay data    }
                T1_T2_Wts        : T1T2Wts; {1st T-cell-2nd Tcell wts}
                T2_Out           : T2Out;   {2nd T-cell outputs      }
                T2OutArray       : T2_O_A;  {T2 energy decay data    }
              END;

StimDist = record                           {Data to compute avg dist}
             DistSum             : LongInt; {Sum of distances to Stim}
             S_Type              : StimType;{Stimulus type           }
           END;

AvgStDist  = array[1..8] of StimDist;       {Data on Veh dist to Stim}
RunStatRec = record                         {Vehicle statistics      }
               AvgStimDist       : AvgStDist;{Avg dist to 4 stimuli   }
               AvgVehEnergy      : Real;    {Veh energy (Running avg)}
               NumPts            : word;    {Number of data point (T)}
             END;

TrackRec   = Record
               NewX                  : Integer;
               NewY                  : Integer;
               NewHdg            : Integer;
               Dist              : Real;
               Time              : Word;
               END;

TrackRecArray  = Array[1..100] of TrackRec;

TrackFile      = file of TrackRec;
```

```
    RunStatFile    = file of RunStatRec;      {Holds run statistics    }

    RunFile        = file of RunRec;

    VehFile        = file of VehicleRec;

    EnvFile        = file of EnvRec;

    Txt         = Text;

    DistAng     = record
       Dist        : Byte;
       Ang         : 0..90;
       END;

    StimuliDat  = record
       Dist        : Integer;
       LDist       : Integer;
       Ang         : 0..360;
       END;

    StimArray   = Array[1..8] of StimuliDat;

    StmDatArray = Array[1..5151] of DistAng;

    SumSqr      = Array[1..8] of Real;

    Buffer      = Array[1..100] of RunStatRec;

    MDecay      = Array[0..10] of Real;   {ST memory decay factor/time  }

    SDecay      = Array[0..5] of Real;    {ST memory decay FOR sensors  }

    Layer2Train = (Hidden, Both, Drive); {Layers that can be trained   }

VAR
    Answer         : Char;
    RRec           : RunRec;
    VRec           : VehicleRec;
    ERec           : EnvRec;
    FuncType       : FType;
    FileStr        : String;
    X, Y, T        : Integer;            {Current position, Time}
    OldX, OldY     : Integer;            {Last position }
    Hdg, OldHdg    : Integer;            {Vehicle heading in degrees}
    Loop, i        : Integer;            {Loop control VARiables  }
    Start, Stop    : Integer;            {Time FOR run start, stop}
    StimuliArray   : StimArray;          {Data FOR stim rel to veh}
    Dat            : DistAng;            {Temp record }
    DatFile        : File of DistAng;    {File handle}
    RFile          : File of RunRec;
    EFile          : File of EnvRec;
```

```
    VFile           : File of VehicleRec;
    TFile           : TrackFile;
    RnFi            : Text;
    EFName,VFName   : FileName;             {Env/Veh File names     }
    RnFlName        : FileName;
    StimDatArray    : StmDatArray;          {Look-up table of dist/ang}
    SumSqrs         : SumSqr;               {Array of stim dist sum sqr}
    SFV             : Sen_Fov_Resp;         {Sensor FOV gain [0..180]}
    SNum,NodeSum    : byte;                 {Num sensors, hidden nodes}
    A               : byte;
    StatRec         : RunStatRec;           {Run statistics record    }
    StatBuff        : Buffer;               {Temp buffer FOR stat data}
    Buff            : Array[1..1024] of Byte;{Run data text buffer     }
    SBIndex         : Byte;
    TrkRecArray     : TrackRecArray;        {Buffer FOR move data     }
    TrkIndex        : Byte;                 {Index to TrkRecArray     }
    MoveDist        : Integer;              {Distance last moved      }
    Test,Video      : Boolean;              {Switchs FOR debug/video  }
    PosTrainNum     : Word;                 {Cum num pos learning events}
    NegTrainNum     : Word;                 {Cum neg learning events  }
    DeltaPosTrnNum  : Word;                 {Pos Trn events this period}
    DeltaNegTrnNum  : Word;                 {Neg trn evevts this period}
    DLay            : Word;                 {Wait betwn loops (sharing)}
    ChildHood       : Integer;              {Period of ranDOm movement}
    Reward          : Integer;              {Nutrient found           }
    Punish          : Integer;              {Punishment incurred      }
    NodeTrain       : Byte;                 {Number of nodes to train }
    PrntRep, Sign   : Boolean;              {Print report switch      }
    Resp,LRtF       : Char;                 {Query Resp/ learn rate fn}
    LastReward      : Integer;
    OpCond          : Boolean;              {Operant Conditioning Flag}
    LRtC            : Real;                 {Learning rate CONSTant   }
    Cease           : Char;
    L2T             : Layer2Train;          {IDs layer(s) to train    }
    WrtRep          : Boolean;
    WnTkAll         : Boolean;              {Implement winner take all}
                                            { neurons in hidden layer }
    SenSTM          : Boolean;              {Sensor STM switch        }

CONST
    S_H_R     : SenResp = (                 {Freq resp of sensor/type}
                        (1.0, 0.3, 0.1, 0.0, 0.0, 0.0),
                        (0.3, 1.0, 0.3, 0.0, 0.0, 0.0),
                        (0.1, 0.3, 1.0, 0.0, 0.0, 0.0),
                        (0.0, 0.0, 0.0, 1.0, 0.3, 0.1),
                        (0.0, 0.0, 0.0, 0.3, 1.0, 0.3),
                        (0.0, 0.0, 0.0, 0.1, 0.3, 1.0));

    MemDecay : MDecay = (1.0, 0.86, 0.74, 0.64, 0.55, 0.47, 0.41, 0.35,
                                                       0.30, 0.26, 0.22);

    SENDecay : SDecay = (1.0, 0.67, 0.44, 0.30, 0.20, 0.13);
```

```
   Erase     : Word =  0;
   Drw       : Word = 15;


{******************** PROCEDURES AND FUNCTIONS ************************}


Function Greater(A, B : Integer) : Integer;
      BEGIN
         IF A >= B THEN
            Greater := A
         ELSE
            Greater := B;
      END;

{-------------------------------------------------------------------------}

Function Lesser(A, B : Integer) : Integer;
      BEGIN
         IF A < B THEN
            Lesser := A
         ELSE
            Lesser := B;
      END;

{-------------------------------------------------------------------------}

Function GetAngle(Dx, Dy, Ang : Integer) : Integer;

      BEGIN
         IF Dx >= Dy THEN
            IF Abs(Dx) >= Abs(Dy) THEN
               BEGIN
                  IF Dy >= 0 THEN
                     GetAngle := Ang
                  ELSE
                     GetAngle := 360 - Ang
               END
            ELSE
               BEGIN
                  IF Dx >= 0 THEN
                     GetAngle := 270 + Ang
                  ELSE
                     GetAngle := 270 - Ang;
               END
         ELSE
            IF (Abs(Dx) >= Abs(Dy)) THEN
               BEGIN
                  IF Dy < 0 THEN
                     GetAngle := 180 + Ang
                  ELSE
                     GetAngle := 180 - Ang
```

```
            END
         ELSE
            BEGIN
               IF Dx < 0 THEN
                  GetAngle := 90 + Ang
               ELSE
                  GetAngle := 90 - Ang
            END
      END;                    {Function GetAngle}

{-------------------------------------------------------------------------}

Function X_to_the_Y(X,Y : Real) : Real;

VAR                           {Raise X to the Yth Power}
   R    : Real;

BEGIN
   IF x > 0 THEN
      BEGIN
      R := Y * Ln(X);
      X_to_the_Y := Exp(R)
      END
   ELSE
      X_to_the_Y := 0
END;

{-------------------------------------------------------------------------}

Procedure LoadSENData(Range : Byte ;Gain : Real; CType : FType;
                                                  VAR FOV : Sen_FOV_Resp);
                  {Calc FOV response, put in FOV}
VAR
   Loop           : Byte;

BEGIN
   FOR Loop := 0 to 180 DO
      BEGIN
         IF Range   = 0 THEN
            FOV[i] := 0

         ELSE
            IF Loop > Range THEN
               FOV[Loop] := 0
            ELSE
               CASE CType OF

               Linear    : FOV[Loop] := Gain - (Gain / Range) * Loop;

               Sigmoid   : FOV[Loop] := Gain * (Exp(- Sqr(2 *
                                                   Loop / Range)));
               Decay     : FOV[Loop] := Gain * (Exp((-Loop * 4)/Range));
```

```
                Parabolic : FOV[Loop] := Gain - (Gain / (Sqr(Range))) *
                                                        (Sqr(Loop))
                END       {CASE}
        END             {FOR Loop}
END;                            {Procedure LoadSENData}

{-------------------------------------------------------------------}


Procedure SetUp(VAR Strt, Stp, CHood : Integer; VAR NdTrn, NSm : Byte;
 VAR PRep : Boolean; VAR VName, EName : FileName; VAR VRc : VehicleRec;
 VAR ERc : EnvRec; VAR OC : Boolean; VAR LRC : Real; VAR LRF : Char;
 VAR LtT : Layer2Train; VAR WTA : Boolean; VAR RFName : FileName;
 VAR Xc, Yc, Hdg : Integer; VAR RF : Text; VAR WR, SnSTM : Boolean);

VAR
   Resp  : Char;
   Temp  : Char;

{:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::}

Procedure Initialize(VAR ER : EnvRec; VAR VR : VehicleRec;
                ERF, VRF, RFN : FileName; VAR NS : Byte; VAR RF : Text);

VAR
   FileStr  : String;
   Runfile  : File of RunRec;
   Envfile  : File of EnvRec;
   VehFile  : File of VehicleRec;
   Resp     : Char;
   RespStr  : String;

BEGIN
   ClrScr;                                          {Prep screen}
   Assign(EnvFile,ERF + '.env');
   Reset(EnvFile);
   Read(EnvFile,ER);
   Close(EnvFile);
   Assign(VehFile,VRF + '.veh');
   Reset(VehFile);
   Read(VehFile,VR);
   NS := VR.NSum;
   VR.VehName := VRF;                  {Ensure VRec filename is correct}
   Close(VehFile);
   Assign(RF,RFN + '.run');
   SetTextBuf(RF,Buff);
   ReWrite(RF);
END;                {Procedure initialize}

{:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::}
```

```
BEGIN
   WriteLn('ENTER VEHICLE FILE NAME');
   ReadLn(VName);
   WriteLn('ENTER ENVIRONMENT FILE NAME');
   ReadLn(EName);
   REPEAT
      WriteLn('WRITE RUN DATA FILE? (Y/N)');
      ReadLn(Resp);
   UNTIL Resp In ['Y','y','N','n'];
   IF Resp IN ['Y','y'] THEN
      BEGIN
      WriteLn('ENTER RUN DATA FILE NAME');
      REadLn(RFName);
      END
   ELSE
      WR := False;
   Initialize(ERc,VRc,EName,VName,RFName,NSm,RF);
   REPEAT
      WriteLn('ENTER START TIME  (1-8000)');
      ReadLn(Strt);
   UNTIL (Strt >= 1) AND (Strt <= 8000);
   REPEAT
      WriteLn('ENTER STOP TIME (',Strt:4,'-8000)');
      ReadLn(Stp);
   UNTIL (Stp >= Strt) AND (Stp <= 8000);
   ClrScr;
   REPEAT
      WriteLn('SET INITIAL CONDITIONS - VEHICLE X, Y, & HEADING (Y/N)?');
      ReadLn(Resp);
   UNTIL Resp IN ['Y','y','N','n'];
   IF Resp IN ['Y','y'] THEN
      BEGIN
      REPEAT
         Write('ENTER STARTING X COORDINATE (1-500)');
         ReadLn(Xc);
      UNTIL (Xc >= 1) AND (Xc <= 500);
      REPEAT
         Write('ENTER STARTING Y COORDINATE (1-300)');
         ReadLn(Yc);
      UNTIL (Yc >= 1) AND (Yc <= 300);
      REPEAT
         Write('ENTER STARTING HEADING (1-360)');
         ReadLn(Hdg);
      UNTIL (Hdg >= 1) AND (Hdg <= 360);
      END;
   REPEAT
      WriteLn('ENABLE SENSOR SHORT TERM MEMORY? (Y/N)');
      ReadLn(Resp);
   UNTIL Resp IN ['Y','y','N','n'];
   IF Resp IN ['Y','y'] THEN
      SnSTM := True;
   REPEAT
```

```
   WriteLn('ENTER LAYERS TO TRAIN: (H)IDDEN, (D)RIVE, OR (B)OTH');
   ReadLn(Resp);
UNTIL Resp IN ['H','h','D','d','B','b'];
CASE Resp OF
   'H','h' : LtT := Hidden;
   'D','d' : LtT := Drive;
   'B','b' : LtT := Both;
END;                {CASE Resp...}
REPEAT
   WriteLn('ENABLE WINNER-TAKE-ALL HIDDEN LAYER? (Y/N)');
   ReadLn(Temp);
UNTIL Temp IN ['Y','y','N','n'];
IF Temp IN ['Y','y'] THEN
   WTA := True;
IF (Resp In ['H','h','B','b']) AND (WTA = False) THEN
   REPEAT
      WriteLn('ENTER NUMBER OF HIDDEN NODES TO TRAIN');
      WriteLn('DURING EACH TRAINING EVENT (<= ',NSm:2,')');
      ReadLn(NdTrn);
   UNTIL (NdTrn <= NSm) AND (NdTrn >= 1)
ELSE
   IF WTA THEN
      NdTrn := 1;
WriteLn('CURRENT AGE = ',Vrc.Age:6,'.  ENTER LENGTH OF CHILDHOOD');
ReadLn(CHood);
REPEAT
   ClrScr;
   WriteLn('ENTER LEARNING RATE FUNCTION TYPE: (L)INEAR, (D)ECAY, ',
           '(C)ONSTANT');
   ReadLn(LRF)
UNTIL LRF IN ['L','l','D','d','C','c'];
LRC := 0.0;
IF LRF In ['C','c'] THEN
   REPEAT
      WriteLn('ENTER LEARNING RATE CONSTANT (0.0 - 1.0)');
      ReadLn(LRC);
   UNTIL (0.0 <= LRC) And (LRC <= 1.0);
REPEAT
   ClrScr;
   WriteLn('ENABLE BEHAVIOR SHAPING? (Y/N)');
   ReadLn(Resp);
UNTIL Resp IN ['Y','y','N','n'];
IF Resp IN ['Y','y'] THEN
   BEGIN
   OpCond  := True;
   DLay := 100;
   END
ELSE
   OPCond := False;
REPEAT
   ClrScr;
   WriteLn('WRITE DETAILED REPORTS? (Y/N)');
```

```
      ReadLn(Resp);
   UNTIL Resp IN ['Y','y','N','n'];
   IF Resp IN ['Y','y'] THEN
      PRep := True
   ELSE
      PRep := False;

END;                        {Procedure SetUp}

{-------------------------------------------------------------------------}

Procedure DrawStims(StimData : StimuliData; Clr : Word);

VAR
   Loop     : Byte;
   OldColor : Word;

{:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::}

Procedure DrawStim(XLoc, YLoc : Integer);
                    {Draw a cross centered at XLoc, YLoc}

CONST
   L    : Integer = 3;

Begin
   Line(Dx(XLoc-L), Dy(YLoc), Dx(XLoc+L), Dy(YLoc));
   Line(Dx(XLoc), Dy(YLoc-L), Dx(XLoc), Dy(YLoc+L));
END;                        {Procedure DrawStim}

{:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::}

Procedure DrawNutrientStim(XLoc, YLoc : Integer; Rwrd : Boolean);
                                    {Draws a small square}
CONST
   D     : Integer = 3;

Begin
   Rectangle(Dx(XLoc-D),Dy(YLoc+D),Dx(XLoc+D),Dy(YLoc-D));
   Line(Dx(Xloc-D),Dy(YLoc),Dx(XLoc+D),Dy(YLoc));
   IF Rwrd THEN
      Line(Dx(XLoc),Dy(YLoc-D),Dx(XLoc),Dy(YLoc+D));
END;

{:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::}

Begin
   OldColor := GetColor;
   SetColor(Clr);
   FOR Loop := 1 to Erec.StimNum DO
      WITH StimData[Loop] DO
         IF Nutrient THEN
```

```
            IF Pos THEN
               DrawNutrientStim(StimX,StimY,True)
            ELSE
               DrawNutrientStim(StimX,StimY,False)
         ELSE
            DrawStim(StimX, StimY);
   SetColor(OldColor);
END;                        {Procedure DrawStims}

{-------------------------------------------------------------------------}

Procedure StartGraph;

VAR
   Driver, Mode, Error  : Integer;

Begin
   Driver := Detect;
   Initgraph(Driver, Mode, 'C:\TP');
   Error := GraphResult;
   IF Error <> GrOK THEN
      Begin
      ClrScr;
      WriteLn('CALL TO GRAPHICS FAILED! CHECK IT OUT.');
      WriteLn('REASON: ',GraphErrorMsg(Error));
      END;
   SetIntCoords(ERec.TX, ERec.TY, ERec.BX, ERec.BY);
   DrawStims(ERec.Stimuli,Drw);
END;

{-------------------------------------------------------------------------}

Procedure BuildStimLookUp(VAR StDatArray : StmDatArray);

VAR
   DatFile    : File of DistAng;
   i          : Integer;

BEGIN
   Assign(DatFile, 'StimDat.dat');        {Build stimulus data look-up }
                                          { table. Assume StimDat in   }
   Reset(DatFile);                        { current directory.         }
   i := 1;                                {StDatArray holds data       }
   WHILE i <= 5151 DO
      BEGIN
      Read(DatFile, Dat);
      StDatArray[i].Dist := Dat.Dist;
      StDatArray[i].Ang  := Dat.Ang;
      Inc(i);
      END;                                {WHILE i}
   Close(DatFile);
END;                                      {Procedure BuildStimLookUp   }
```

```
Procedure UpDateStimDat(VAR StmArray : StimArray);

VAR
   Dx,Dxs,Dy,Dys     : Integer;
   DistScaler        : Real;
   Loop              : Byte;
   ArrayIndex,Angle  : Integer;
   ScaleX            : Boolean;

BEGIN
   WITH ERec DO
      BEGIN
      FOR Loop := 1 TO StimNum DO        {Up date stimulus data since   }
         BEGIN                           { last move                    }
         Dx    := Stimuli[Loop].StimX - X;{Find dIF from current location}
         Dy    := Stimuli[Loop].StimY - Y;{ to stimulus.                 }
         DistScaler := 1.0;               {Set scale value to default   }
         IF (Abs(Dx) > 100) OR (Abs(Dy) > 100) THEN {Scale to Dx/Dy value}
         DistScaler := Greater(Abs(Dx),Abs(Dy))/100;{Compute scale value}

         DxS := Abs(ROUND(Dx/DistScaler));  {Scale dist to 100 or less  }
         DyS := Abs(ROUND(Dy/DistScaler));
         ArrayIndex := (Greater(DxS,DyS)*(Greater(DxS,DyS)+1))
                                        DIV 2 + 1 + (Lesser(DxS,DyS));
         StimuliArray[Loop].Ldist := StimuliArray[Loop].Dist;
         StimuliArray[Loop].Dist  := ROUND(Abs(DistScaler *
                                      StimDatArray[ArrayIndex].Dist));
         Angle := StimDatArray[ArrayIndex].Ang;
               {Convert stored angle to actual relative to horz line}
         StimuliArray[Loop].Ang  := GetAngle(Dx,Dy,Angle);

      END;                                          {FOR Loop}
   END;                                             {WITH ERec}
END;                                                {Procedure UpDateStimDat}

{-------------------------------------------------------------------------}

Function SigmoidResp2(InSum : Real) : Real;
               { 0 < SigmoidResp < 1 FOR 0 < Insum < +Big}
BEGIN
   IF InSum > 10 THEN
      InSum := 10;
   SigmoidResp2 := 2 * (1 / (1 + Exp(- InSum)/10)) - 1;
END;

{-------------------------------------------------------------------------}

Procedure SensorOut(Tm : Integer; VAR SnOut : SenOutArray;
                    VAR SOut : S_O_A; SnSTM : Boolean);
VAR
   AngIndex  : Integer;       {Angle of sensor to stimulus              }
   InvSqr    : Real;          {Inverse square of distance               }
```

```
   StimInt   : Longint;       {Stimulus intensity at one unit distance}
   SenGain   : Real;          {Sensor gain based on FOV Response      }
   SenFrqRes : Real;          {Frequency response of sensor to stim   }
   StevensX  : Real;          {Stevens law exponent                   }
   Stevens2X : Real;          {Stevens law exponent FOR deltas        }
   EnrgIn    : Real;          {Effective energy input to sensor       }
   Temp      : LongInt;
   Count     : Byte;
   TempArray : Array[1..4] of Real;
   TempReal  : Real;

BEGIN
   WITH VRec, ERec DO
      BEGIN
      FOR count := 1 to Sen_Num DO          {Put Sensor output at t-1 }
         TempArray[Count]:= SnOut[Count];   {in the TempArray         }
      FOR Loop := 1 TO Sen_Num DO
         BEGIN
         FOR i := 1 TO StimNum DO
            BEGIN
               AngIndex := StimuliArray[i].Ang - Hdg -
                                    SenRecs[Loop].SAngle;
               REPEAT
                  IF (AngIndex >= 360) OR (AngIndex < 0) THEN
                     IF AngIndex >= 360 THEN {Make 0 <= AngIndex < 360}
                        AngIndex := AngIndex - 360
                     ELSE
                        AngIndex := 360 + AngIndex;
               UNTIL (AngIndex < 360) And (AngIndex >= 0);
               IF AngIndex > 180 THEN
                  AngIndex := -(360 - AngIndex);
               Temp        := StimuliArray[i].dist;
               IF Temp      = 0 THEN
                  Temp     := 1;
               InvSqr      := 1 / Sqr(Temp);
               StimInt     := Stimuli[i].StimEnrg;
               SenGain     := SenRecs[Loop].Curve[Abs(AngIndex)];
               SenFrqRes   := S_H_R[Ord(Stimuli[i].Stimulus),
                                    Ord(SenRecs[Loop].SResp)];
               StevensX    := SenRecs[Loop].SenK;
               Stevens2X   := SenRecs[Loop].Sen2K;
               EnrgIn      := InvSqr * Stimuli[i].StimEnrg * SenGain *
                              SenFrqres;
               SnOut[Loop] := SnOut[Loop] + EnrgIn;


            END;                    {FOR i}
         END;                      {FOR Loop}
      FOR i := 1 to (2 * Sen_Num) DO
         BEGIN
         IF i <= Sen_Num THEN
```

```
            BEGIN
            SnOut[i] := X_to_the_Y(SnOut[i],StevensX) / 8;
                                                    {Normal response}
            IF SnOut[i] > 1 THEN
               SnOut[i] := 1                  {CONSTrain output 0<Out<1}
            END
         ELSE
            BEGIN
            TempReal := SnOut[(i - Sen_Num)] - TempArray[(i - Sen_Num)];
            SnOut[i] := X_to_the_Y(TempReal,Stevens2X) / 8;
                                                    {Sensor delta resp}
            IF SnOut[i] > 1 THEN
               SnOut[i] := 1                  {CONSTrain output 0<Out<1}
            END;
         END;           {FOR i}

      IF SnSTM THEN                             {DO Sensor Shrt Trm Mem  }
         FOR Loop := 1 to (2 * Sen_Num) DO      {Decay outputs IF rapid  }
            WITH SOut[Loop] DO                  { negative change occured}
               BEGIN                            { Simulates ST memory    }
               IF (Tm - Time) <= 5 THEN
                  TempReal := Out * SENDecay[(Tm - Time)];
                                                {Decayed previous outp}
               IF TempReal > SnOut[Loop] THEN
                  SnOut[Loop] := TempReal
               ELSE
                  BEGIN
                  Time := T;
                  Out  := SnOut[Loop];
                  END;              {ELSE}
               END;                  {WITH SOut[Loop]..}

   END;                 {WITH}
END;                            {Procedure SensorOut}
{-------------------------------------------------------------------------}

Function SigmoidResp1(InSum : Real) : Real;
             { 0 < SigmoidResp2 < 1  FOR   -Big < InSum < +Big}
VAR
   Temp : real;

BEGIN
   IF InSum > 10 THEN
      Insum := 10
   ELSE
      IF InSum < -10 THEN
         InSum := -10;
   Temp     := Exp(- Insum);       {Attempt to avoid 8087 stack overflow}
   Temp     := Temp + 1;
   Temp     := 1/Temp;
   SigmoidResp1 := Temp;
END;
```

```
Procedure Matrix1(SO : SenOutArray; ST1 : SenT1Wts; VAR TOneOut : T1Out);

{Assumes calling program has defined T1_Out : array[1..4,1..20] of Single}

VAR
   Sen, T1, loop  : Byte;
   Temp           : Real;

BEGIN
   WITH VRec DO
   BEGIN
   FOR T1 := 1 to Nodesum DO                {Initialize T1          }
      T1_Out[T1] := 0;
   FOR T1 := 1 to NodeSum DO                {Matrix multiply        }
      BEGIN
      Temp := 0;
      FOR  Sen := 1 to (2 * Sen_Num) DO
         Temp := Temp + (SO[Sen] * St1[Sen,T1]);
      TOneOut[T1] := Temp
      END;
   END;
END;                    {Procedure Matrix1}

{-------------------------------------------------------------------------}

Procedure T1OutResp( NS : Byte; VAR T_Out : T1Out; VAR T1O : T_O_A;
                                                        WTA : Boolean);

VAR
   Loop          : Byte;
   Temp,Temp2    : Real;

BEGIN
{>}Temp2     := 0.0;
   FOR Loop := 1 to NS DO
      BEGIN
      Temp  := T_Out[Loop];
      T_Out[Loop]    := SigmoidResp1(Temp);
{>>}  IF T_Out[Loop] > Temp2 THEN
         Temp2 := T_Out[Loop];
      WITH T1O[Loop] DO    {Check T1 output FOR ST memory decay firing}
         BEGIN
         IF (T - Time) <= 10 THEN
            Temp := Out * MemDecay[(T - Time)];
                                        {Decayed previous output    }
         IF Temp > T_Out[Loop] THEN     {Put Decay from T-1 in T-Out}
            T_Out[Loop] := Temp
         ELSE
            BEGIN
            Time := T;                  {Reset Time to current T    }
            Out  := T_Out[Loop];
            END;
```

```
         END;                          {WITH TO[Loop]  }
      END;                             {FOR Loop = 1...}

   IF WTA THEN
      FOR Loop := 1 to NS DO                          {Winner take all!}
         IF Temp2 > T_Out[Loop] THEN
            BEGIN
            T_Out[Loop] := 0.2;
            T1O[Loop].Time := T;
            T1O[Loop].Out  := 0.2;
            END;

IF Test = true THEN
   ReadLn;
END;                               {Procedure T1Out}

{-------------------------------------------------------------------------}

Procedure Matrix2(TOneOut : T1Out; T1T2 : T1T2Wts; VAR TTwoOut : T2Out);

VAR
   T1,T2       : Byte;
   Temp        : Real;
BEGIN
   WITH Vrec DO
   BEGIN
   FOR T2 := 1 to 3 DO
      T2_Out[T2] := 0.0;
   FOR T2 :=1 to 3 DO
      BEGIN
      Temp := 0.0;
      FOR T1 := 1 to NodeSum DO
         Temp := Temp + TOneOut[T1] * T1_T2_Wts[T1,T2];
      TTwoOut[T2] := Temp;
      END;                 {FOR T2    }
   TTwoOut[T2] := Temp;
   END;                    {WITH VRec}
END;                       {Procedure Matrix2}

{-------------------------------------------------------------------------}

Procedure T2OutResp(VAR TTwoOut : T2Out; VAR T2O : T2_O_A);

VAR
   Loop           : Byte;
   WinNum         : Word;
   ETime          : Integer;
   Temp, TempRem  : Real;
   AnnealFac      : Real;

{:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::}
```

```
Function Pick(Total : Word): Word;
                {RanDOmly selects a number from 1 to Total}

BEGIN
RanDOmize;
Pick := RanDOm(Total) + 1;                    {ShIFt range to 1..Total}
END;                                          { (Not 0..(Total-1))    }

{::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::}

Function Decay(ET, ChldHood : Integer) : Real;
               {Exponential decay from 1 to 0 over elapsed time ET}
BEGIN
   IF ChildHood = 0 THEN
      Decay := 0
   ELSE
      Decay := (Exp((-ET * 4)/ChldHood));{At ET = chldhood, decay = 0.018}
END;

{::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::}

BEGIN
   ETime    := T - Start;                     {Elapsed time      }
   AnnealFac:= Decay(ETime, Childhood);       {Calc ranDOm move factor}

   FOR Loop := 1 to 3 DO
      BEGIN
      Temp  := TTwoOut[Loop];
      Temp  := (1 - AnnealFac) * SigmoidResp1(Temp);
      IF ChildHood > ETime THEN
         TTwoOut[Loop] := Temp + RanDOm * (1 - Temp) / 2
      ELSE                    {FOR Loop..}
         TTwoOut[Loop] := SigmoidResp1(TTwoOut[Loop]);
      END;          {FOR Loop...}

   Temp      := Frac((ETime) / 20);          {Detect 20 unit time intvl}
   IF (Temp = 0.0) AND (ETime <= ChildHood) THEN
      BEGIN
      WinNum          := Pick(3);            {Select winner T2 node}
      TempRem         := (1 - TTwoOut[WinNum]) * RanDOm;
                                             {Compute FORcing func}
      TTwoOut[WinNum] := TTwoOut[WinNum] + TempRem;
      END;               {IF Temp = ..}
      FOR Loop := 1 to 3 DO
         WITH T2O[Loop] DO   {Check T1 output FOR ST memory decay firing}
            BEGIN
            IF T = 1 THEN
               BEGIN
               Out  := 0.0;
               Time := 1;
               END;
            IF ((T - Time) <= 10) AND (Time <> 1) THEN
```

```
            Temp := Out * MemDecay[(T - Time)]{Decayed previous output}
         ELSE
            Temp := 0.0;
         IF Temp > TTwoOut[Loop] THEN        {Store decay from T-1    }
            TTwoOut[Loop] := Temp
         ELSE
            BEGIN
            Time := T;                       {Reset Time to current T}
            Out  := TTwoOut[Loop];
            END;

         END;    {WITH T2O[Loop]}
END;             {Procedure T2OutResp}

{-------------------------------------------------------------------}

Procedure Move(T2O : T2Out; VAR NewX, NewY, LastX, LastY,
                           NewHdg, LastHdg, MDist, Enrg : Integer);

VAR
   D, MRadn    : Real;                   {Distance moved             }
   NX, NY ,NH  : Integer;                {New coordinates and heading}
   CurvRad     : Real;                   {Move arc radius            }
   AspRatio    : Real;
   LX, LY, LH  : Integer;
   Temp        : Real;
   ETime       : Word;                   {Elapsed time from start}

{:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::}

Procedure PlotPts(Radius : Real; VAR XPos, YPos, LastX, LastY : Integer;
            Aspect, MovRadn : Real; VAR Head, LastHdg : Integer);

VAR
   MoveDir              : (Straight, Turn);
   AspRatio             : Real;
   TRec                 : TrackRec;
   TFile                : TrackFile;
   Radian,RadDir        : Real;
   Loop                 : Byte;
   CenX, CenY           : Integer;
   MaxX,MaxY,MinX,MinY  : Integer;

BEGIN
   Radian    := 180 / (Pi);
   LastX     := XPos;              {Store old X,Y, Hdg FOR erase    }
   LastY     := YPos;
   LastHdg   := Head;
   MaxX      := ERec.BX;
   MinX      := ERec.TX;
   MaxY      := ERec.TY;
   MinY      := ERec.BY;
```

```
MinY       := ERec.BY;

IF Radius   = 0 THEN
   MoveDir := Straight
ELSE
   MoveDir := Turn;
RadDir     := Pi / 2;              {Direction of radius FOR lft turn}
IF MovRadn < 0 THEN
   RadDir  := -RadDir;             {Direction of radius FOR rt turn}

WITH TRec DO
   BEGIN
   CASE MoveDir OF

   Straight :
      BEGIN
      Dist    := MovRadn;
      NewX    := Round(Dist * Cos(Head / Radian) + LastX);
      NewY    := Round(Dist * Sin(Head / Radian) + LastY);
      Time    := T;
      NewHdg := Head;
      END;

   Turn     :
      BEGIN
      Time    := T;                {Store time      }
      Dist    := MovRadn * Radius;
      CenX    := Round(Radius *
                 Cos(Head/Radian + RadDir) + LastX); {Find center FOR }
      CenY    := Round(Radius *
                 Sin(Head/Radian + RadDir) + LastY); {track arc       }
      NewX    := Round(Radius * Cos(-RadDir + Head/Radian +MovRadn)
                               + CenX);      {Calc new coords }
      NewY    := Round(Radius * Sin(-RadDir + Head/Radian + MovRadn)
                                          + CenY);
      NewHdg  := Round(Head + Movradn * Radian);  {New Heading deg }
      IF NewHdg >= 360 THEN                       {0 <= Head <= 360}
         NewHdg := NewHdg - 360
      ELSE
         IF NewHdg <= -360 THEN
            NewHdg := NewHdg + 360;
      END;                {CASE OF Turn}
   END;                   {CASE}
   IF NewX > MaxX THEN                  {Keep Vehicle on screen}
      NewX := NewX - MaxX
   ELSE
      IF NewX < MinX THEN
         NewX := NewX + MaxX;
   IF NewY > MaxY THEN
      NewY := NewY - MaxY
   ELSE
      IF NewY < MinY THEN
```

```
            NewY := NewY + MaxY;
        XPos  := NewX;                          {Pass new veh coords back}
        YPos  := NewY;
        Head  := NewHdg;                        {Pass new heading back    }
        END;                        {WITH TRec}

END;                    {Procedure PlotPts}

{:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::}

BEGIN
   NX       := NewX;
   NY       := NewY;
   NH       := NewHdg;
   LX       := LastX;
   LY       := LastY;
   LH       := LastHdg;
   AspRatio := 1.0;

   Temp           := 0.0;

   FOR Loop       := 1 to 3 DO                    {Find largest output}
      IF T2O[Loop] > Temp THEN
         Temp     := T2O[Loop];

   IF T2O[2] = Temp THEN
      BEGIN                                                {Go Straight}
      D       := Round(10 * T2O[2]);
      MRadn   := D;                                        {Move distance}
      CurvRad := 0;
      END
   ELSE
      IF T2O[1] = Temp THEN                                {Turn left}
         BEGIN
         D       := 10 * T2O[1];
         CurvRad :=  5 * (10 - 10 * (T2O[1] - T2O[3]));{Turn radius}
         IF CurvRad = 0 THEN
            CurvRad := 0.1;
         MRadn   := D / CurvRad;                    {Move angle in radian}
         END
      ELSE
         BEGIN                                             {Turn right}
         D       := 10 * T2O[3];
         CurvRad :=  5 * (10 - 10 * (T2O[3] - T2O[1]));{Turn radius}
         IF CurvRad = 0 THEN
            CurvRad := 0.1;
         MRadn   := -D / CurvRad;                          {Move angle }
         END;

   PlotPts(CurvRad,NX,NY,LX,LY,AspRatio,MRadn,NH,LH);
   IF NH >= 360 THEN
      NH := Hdg - 360
```

```
    ELSE
       IF NH <= -360 THEN
          NH := Hdg + 360;
    NewHdg  := NH;
    NewX    := NX;
    NewY    := NY;
    LastX   := LX;
    LastY   := LY;
    LastHdg := LH;
    MDist   := Round(D) Div 2;
    Enrg    := Enrg - MDist;

END;                {Procedure Move}

{-------------------------------------------- ---------------------}

Procedure DrawMov(VX, VY, VH : Integer);

{::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::}

Procedure DrawVeh(PX,PY,PH : Integer; Clr : Word);

CONST
   Size    : Integer = 3;

VAR
   NumPts  : Word;                              {Number of PolyPoints}
   OldClr  : Word;
   VehPoly : Array[1..5] of PointType;

{::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::}

Procedure DrawVector(PX,PY,PH : Integer);
                       {Draw a vector at PX,PY pointing PH degrees}
CONST
   D = 15;

VAR
   ENDX, ENDY   : Integer;
   Radian       : Real;

Begin
   Radian    := 180 / (Pi);
   ENDX      := Dx(Round(D * Cos(PH/Radian) + PX));
   ENDY      := Dy(Round(D * Sin(PH/Radian) + PY));
   Line(Dx(PX),Dy(PY),ENDX, ENDY);
END;                    {Procedure DrawVector}

{::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::}
```

```
Begin
   VehPoly[1].X := (Dx(PX+Size));
   VehPoly[1].Y := (Dy(PY));
   VehPoly[2].X := (Dx(PX));
   VehPoly[2].Y := (Dy(PY+Size));
   VehPoly[3].X := (Dx(PX-Size));
   VehPoly[3].Y := (Dy(PY));
   VehPoly[4].X := (Dx(PX));
   VehPoly[4].Y := (Dy(PY-Size));
   VehPoly[5].X := (Dx(PX+Size));
   VehPoly[5].Y := (Dy(PY));
   NumPts       := 5;
   OldClr       := GetColor;
   SetColor(Clr);
   DrawPoly(NumPts,VehPoly);
   DrawVector(PX,PY,PH);
   SetColor(OldClr);
END;                          {Procedure DrawVeh}

{::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::}

Begin
   IF T <> Start THEN
      DrawVeh(OldX,OldY,OldHdg,Erase);
   DrawVeh(VX,VY,VH,Drw);
   DrawStims(ERec.Stimuli,Drw);                    {Redraw stimuli}
END;                           {Procedure DrawMov}

{------------------------------------------------------------------------------}

Procedure NodeWtReport(VAR Dev : Text; ST1Wts : SenT1Wts; T1T2Wt :T1T2Wts;
                                       Time : Word);
VAR
   Loop, i : Byte;

BEGIN                                          {lst, print sensor to T1 wts}
   FOR Loop := 1 to 80 DO
      Write(Dev,'-');
   WriteLn(Dev);
   WriteLn(Dev,'   NODE CONNECTION WEIGHTS AT TIME = ',Time:8);
   WriteLn(Dev);
   WriteLn(Dev,'                          SENSOR TO HIDDEN LAYER WEIGHTS');
   WriteLn(Dev);
   FOR i := 0 to NodeSum DO
      BEGIN
      IF i > 0 THEN
         Write(Dev,i:2,' : ')
      ELSE
         BEGIN
         Write(Dev,'    ');
         FOR Loop := 1 to 2*VRec.Sen_Num DO
            BEGIN
```

```
            Write(Dev,Loop:8);
            END;            {FOR Loop ...}
         WriteLn(Dev);
         END;               {ELSE..}
      IF i > 0 THEN
         BEGIN
         FOR Loop := 1 to 2*VRec.Sen_Num DO
            Write(Dev,ST1Wts[Loop,i]:8:3);
         WriteLn(Dev);
         END;               {FOR i...}
      END;                  {FOR i :=...}
   WriteLn(Dev);
                                          {Now print T1 to T2 weights}
   WriteLn;
   WriteLn(Dev,'  HIDDEN LAYER TO DRIVE WEIGHTS');
   WriteLn(Dev);
   FOR i := 0 to NodeSum DO
      BEGIN
      IF i > 0 THEN
         Write(Dev,i:2,' : ')
      ELSE
         BEGIN
         Write(Dev,'    ');
         FOR Loop := 1 to 3 DO
            BEGIN
            Write(Dev,Loop:8);
            END;            {FOR Loop ...}
         WriteLn(Dev);
         END;               {ELSE..}
      IF i > 0 THEN
         BEGIN
         FOR Loop := 1 to 3 DO
            Write(Dev,T1T2Wt[i,Loop]:8:3);
         WriteLn(Dev);
         END;               {FOR i...}
      END;                  {FOR i :=...}
   FOR Loop := 1 to 80 DO
      Write(Dev,'-');
   WriteLn(Dev);
   WriteLn(Dev);
END;                        {Procedure NodeWtReport}

{-------------------------------------------------------------------}

Procedure Report;

VAR
   Resp     : Char;
   Loop     : Byte;
```

```
BEGIN
      CloseGraph;
      WriteLn('Enter: "A" FOR all data');
      WriteLn('"S" FOR sensor output data');
      WriteLn('"1" FOR T1 output data');
      WriteLn('"2" FOR T2 output data');
      ReadLn(Resp);
      WITH VRec DO
         BEGIN
         IF (Resp in ['A','a','S','s']) THEN
            BEGIN
            ClrScr;
            WriteLn('Time = ',T:5,'  Last Train Event = ',LastReward:5,
                                    '  Childhood = ',Childhood:5);
            WriteLn('Energy = ',VRec.VehEnergy:5);
            FOR Loop := 1 to  (2 * Sen_Num) DO
               WriteLn('Output of Sensor # ',Loop:1,' = ',
                                                   Sen_Out[Loop]:2:4);
            ReadLn;
            END;              {IF S}
         IF (Resp in ['A','a','1']) THEN
            BEGIN
            ClrScr;
            Loop := 1;
            WHILE Loop <= (2 * (NodeSum div 2)) DO
               BEGIN
               WriteLn('Output of T1 node ',Loop:2,' = ',T1_Out[Loop]:2:4,
                '   ','Output of T1 node ',(Loop+1):2,' = ',
                                                   T1_Out[Loop+1]:2:4);
               Loop := Loop + 2;
               END;
            IF NodeSum > (2 * Nodesum Div 2) THEN
               WriteLn('Output of T1 node ',NodeSum:2,
                                          ' = ',T1_Out[NodeSum]:2:4);
            ReadLn;
            END;              {IF Resp 1}
         IF (Resp in ['A','a','2']) THEN
            BEGIN
            ClrScr;
            FOR Loop := 1 to 3 DO
               WriteLn('Output of T2 node ',Loop:1,' = ',
                                                   T2_Out[Loop]:2:4);
            NodeWtReport(OutPut,Sen_T1_Wts,T1_T2_Wts,T);
            ReadLn;
            END;             {IF Resp 2}
         END;                {WITH VRec}
      StartGraph;
END;                          {Procedure Report}

{-------------------------------------------------------------------}
```

```
Procedure Beep(Hz : Word);

BEGIN
   Sound(Hz);
   Delay(50);
   NoSound;
END;

{-------------------------------------------------------------------}

Procedure FORage(SDA : StimArray; MDist, Time, MxEnrg : Integer;
                              VAR Enrg, Rwrd, Pnish, LReward : Integer);

VAR
   PosOdds     : Real;
   NegOdds     : Real;
   Dst         : Integer;
   Temp        : Real;
   Sigma       : Real;
   Loop        : Byte;
   Temp2       : Integer;

{:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::}

Function Probability(D : Integer) : Real;

CONST
   Prob : Array[0..25] of real = (1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
   1.0,0.755,0.538,0.365,0.238,0.152,0.095,0.059,0.036,0.022,0.013,0.008,
   0.005,0.003,0.002,0.001);

BEGIN
      Probability := Prob[D Div 2]/2;
END;
{:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::}

BEGIN
   PosOdds  := 0;
   NegOdds  := 0;
   Sigma    := 2.0;            {Distribution parameter (not true SD)}
   Rwrd     := 0;
   FOR Loop := 1 to ERec.StimNum DO
      WITH SDA[Loop] DO
         BEGIN
         Dst  := Dist;
         IF ERec.Stimuli[Loop].Nutrient THEN
            IF Dst <= 50 THEN
               IF ERec.Stimuli[Loop].Pos THEN
                  PosOdds := PosOdds + Probability(Dst)
               ELSE
                  NegOdds := NegOdds + Probability(Dst);
         END;                    {WITH SDA ...}
```

```
   FOR i := 1 to MDist DO
      BEGIN
      RanDOmize;
      Temp := RanDOm;
      IF (Temp < PosOdds) THEN
         BEGIN
         Inc(Rwrd);
         LReward := Time
         END;                        {IF Odds..}
      IF (Temp < NegOdds) THEN
         BEGIN
         Inc(Pnish);
         LReward := Time;
         END;
      END;                           {FOR i ...}

   Temp2  := Enrg + (50 * Rwrd) - (50 * Pnish);
   IF Temp2 > MxEnrg THEN
      Temp2 := MxEnrg
   ELSE
      IF Temp2 < -30000 THEN
         Temp2 := -30000;
   Enrg  := Temp2;
END;                        {Procedure FORage}

{-------------------------------------------------------------------}

Procedure Learn(NodeTrainNum,CHood,Tm,Rwrd,Pnish : Integer;T1O : T1Out;
SO : SenOutArray; T2O : T2Out; VAR STWts : SenT1Wts; VAR TTWts : T1T2Wts;
VAR PosTrnNum, NegTrnNum : Word; PR : Boolean; FnCd : Char; CnR : Real;
                                                    LtT : Layer2Train);
              {Implements non-supervised learning algorithm}
VAR
   Delta, Deltal : Real;
   Node, Factor  : Integer;
   Buff, i, j, k : Integer;
   Temp, Temp2   : Real;
   Pick,TempPick : Set of Byte;
   Loop, Index   : Byte;
   Temp3         : Real;

{:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::}

Function LearnRate(FnCode :  Char; Time, CH : Integer; CR : Real): Real;

CONST
   k = 10;

VAR
   Temp : Real;

{:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::}
```

```
Function Decay(ET, ChldHood : Integer) : Real;
                {Exponential decay from 1 to 0 over elapsed time ET}
VAR
Temp  : LongInt;

BEGIN
   Temp := -ET * 4;
   IF ET > ChldHood THEN
      Decay := 0.1
   ELSE
      Decay := (Exp(Temp/ChldHood));{At ET = chldhood, decay = 0.018}
END;

{::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::}

Function Linear(ET, Chldhood : Integer) : Real;

BEGIN
   IF ET > Chldhood THEN
      Linear := 0.1
   ELSE
      Linear := 1 - Et/Chldhood;
END;

{::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::}

Function CONSTant(Con : Real): Real;

BEGIN
   CONSTant := Con;
END;

{::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::}

BEGIN
   CASE FnCode OF
      'L','l' :  Temp := Linear(Time, CH)/k;
      'D','d' :  Temp := Decay(Time, CH)/k;
      'C','c' :  Temp := CONSTant(CR);
   END;            {CASE FnCode}
   LearnRate := Temp;
END;               {Function LearnRate}

{::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::}

BEGIN
   IF (Rwrd <> 0) OR (Pnish <> 0) THEN
      BEGIN
      Pick     := [];
      TempPick := [];
      i        := 1;
      Temp3    := 0;
```

```
IF (LtT = Hidden) OR (LtT = Both) THEN
   BEGIN
   FOR i := 1 to NodeTrainNum DO
      BEGIN
      Index := 1;
      FOR Loop := 1 to NodeSum DO {Find largest outputs of T1}
         IF (T1O[Loop] >= T1O[Index]) AND (Not (Loop In Pick)) THEN
            Index := Loop;
      TempPick := [Index];          {Put largest num IDs in Pick}
      Pick := Pick + TempPick;
      END;           {FOR i :=..}
   i := 1;
   Temp     := 0.0;
   FOR k    := 1 to 2 * VRec.Sen_Num DO
      Temp  := Temp + SO[k];{Calculate total input to Hidden layer}
   IF Temp = 0 THEN
      Temp := 0.01;

      FOR Loop := 1 to NodeSum DO            { train nodes in pick}
         IF Loop IN Pick THEN
            BEGIN
            Temp3 := 0;
            FOR j := 1 to (2 * VRec.Sen_Num) DO
               Temp3 := Temp3 + Abs(STWts[j,Loop]);
            IF Temp3 <> 0 THEN       {DOn't train IF ALL wts = 0}
               FOR i := 1 to (2 * VRec.Sen_Num) DO
                  BEGIN
                  Delta    := ((LearnRate(FnCd,Tm,CHood,CnR) *
                  (Rwrd+Pnish)) * ((SO[i] / Temp) -
                                                  (STWts[i,Loop])));
                  STWts[i,Loop] := STWts[i,Loop] + Delta;
                  END;                        {FOR Loop..}
            END;          {IF Loop..}
   END;                   {IF LtT..}

IF (LtT = Both) OR (LtT = Drive) THEN
   BEGIN
   Temp := 0;
   FOR Loop := 1 to NodeSum DO
      Temp  := Temp + T1O[Loop];       {Total input to drive layer}
   k     := 0;
   Temp2 := 0;
   FOR Loop := 1 to 3 DO
      IF T2O[Loop] > Temp2 THEN
         BEGIN
         Temp2 := T2O[Loop];
         k     := Loop;            {Node WITH greatest output}
         END;                  {IF T2O..}
   Temp2    := 0;
   FOR Loop := 1 to NodeSum DO
      Temp2 := Temp2 + TTwts[Loop,k];
   IF Temp2 <> 0 THEN             {DOn't Train IF ALL wts = 0}
```

```
            BEGIN
            FOR i := 1 to NodeSum DO
               BEGIN
               Delta1 := ((LearnRate(FnCd,Tm,CHood,CnR) * (Rwrd-Pnish))
                                 * ((T1O[i] / Temp) - (TIWts[i,k])));
               TIWts[i,k] := TIWts[i,k] + Delta1;
               END;                        {FOR i..}
            END;          {IF Temp2}
         END;             {IF LtT =..}
      END;                {IF Reward ..}
      IF Rwrd > 0 THEN
         BEGIN
         Inc(PosTrnNum);
         Beep(2500);
         END;
      IF Pnish > 0 THEN
         BEGIN
         Inc(NegTrnNum);
         Beep(400);
         END;
      IF (Frac(Tm/15) = 0.0) AND (PR) THEN
         NodeWtReport(RnFl,STWts,TIWts,Tm);        {Print new node wts}
END;                    {Procedure Learn}

{-------------------------------------------------------------------}

Procedure Stats(VAR Dev : Text; VR : VehicleRec;
          PosTrnNum, NegTrnNum : Word; VAR DPTrnNum, DNTrnNum : Word);
CONST
   SType    : Array[0..5] of String[3] = ('R','G','B','Hz1','Hz2','Hz3');

VAR
   Loop     : Byte;
   ETime    : LongInt;                    {Elapsed time in run }
   Count    : Word;

BEGIN
   Etime := T;
   WITH VR DO
      BEGIN
         IF (Frac((ETime-1) / 1000) = 0) THEN
            BEGIN
            WriteLn(Dev);
            WriteLn(Dev,'  Time  TPTE  NPTE  TNTE  NNTE   Energy');
            WriteLn(Dev);
            END;
         IF (Frac(ETime / 100) = 0) OR (T = Stop) THEN
            BEGIN
            WriteLn(Dev,ETime:6,'  ',PosTrnNum:3,'    ',(PosTrnNum -
              DPTrnNum):3,'    ',NegTrnNum:3,'    ',(NegTrnNum -
                                    DNTrnNum):3,'    ',VehEnergy:7);
            DPTrnNum := PosTrnNum;
```

```
            DNTrnNum := NegTrnNum;
            END;      {IF (Frac(...}
         IF (Frac((ETime)/1000) = 0.0) THEN
            NodeWtReport(Dev,Sen_T1_Wts,T1_T2_Wts,ETime);
                                    {Print Node wts every 1000 loops}

      END;            {WITH VR..}
END;                  {Procedure Stats}

{-------------------------------------------------------------------}

Procedure Experiment(Time, LRwrd : Word; VR : VehicleRec;
                                                  VAR CHood : Integer);

VAR
   Temp, EDelta  : Real;

BEGIN
   Temp := 0;
   WITH VR DO
      BEGIN
      RanDOmize;
      EDelta := MaxEnergy / 4;
      IF (Time > Chood) AND (Frac(Time/300) = 0.0) THEN
         BEGIN
            IF (Time - LRwrd) > 500 THEN
               CHood := Time + 50;
            Beep(50);
            END;            {IF Time..}
      END;                   {WITH..}
END;              {Procedure Experiment}

{-------------------------------------------------------------------}

Procedure WriteHeader(VAR Dev : Text;ER : EnvRec;VR : VehicleRec;LRF :
Char; LRC : Real; Start,Stop,TrnNode,CHood: Integer; LtT: Layer;
                                                  WTA: Boolean);
Type
   Str3    = String[3];
   MoArray = Array[1..12] of Str3;

CONST
   Mo  : MoArray  = ('Jan','Feb','Mar','Apr','May','Jun','Jul','Aug',
                     'Sep','Oct','Nov','Dec');
VAR
   Month, Year, Day, DWk  : Word;
   Hr, Min, Sec, S100     : Word;

BEGIN
   WITH ER,VR DO
      BEGIN
      GetDate(Year,Month, Day, Dwk);
```

```
      Write(Dev,'DATE : ',Day:2,' ',Mo[Month],' ',Year:4);
      GetTime(Hr,Min,Sec,S100);
      WriteLn(Dev,'                          TIME : ',Hr:2,':',Min:2);
      WriteLn(Dev);
      Write(Dev,'ENVIRONMENT FILE : ',EFName + '.env');
      WriteLn(Dev,'                          VEHICLE FILE : ',VFName +
                                                          '.veh');
      WriteLn(Dev);
      WriteLn(Dev,'START: ',Start:6,'  STOP: ',Stop:6,
                                              '  CHILDHOOD: ',CHood:6);
      WriteLn(Dev);
      WriteLn(Dev,'LEARNING FUNCTION = ',LRF);
      IF LRF In ['C','c'] THEN
         Write(Dev,'  LEARNING RATE CONSTANT = ',LRC:0:3);
      WriteLn(Dev);
      Write(Dev,'LAYER(S) TRAINED = ');
      IF (LtT = Hidden) THEN
         Write(Dev,'HIDDEN')
      ELSE
         IF (LtT = DRIVE) THEN
            Write(Dev,'DRIVE')
         ELSE
            Write(Dev,'BOTH');
      IF (LtT = Hidden) OR (LtT = Both) THEN
         BEGIN
         WriteLn(Dev,'    # OF TRAINING NODES: ',TrnNode:2);
         WriteLn(Dev);
         IF WTA THEN
            WriteLn(Dev,'WINNER-TAKE-ALL HIDDEN LAYER');
         END;
      WriteLn(Dev,'VEHICLE AGE = ',Age:6,'  VEHICLE ENERGY = ',
                                                   VehEnergy:5);
      WriteLn(Dev);
   END;              {WITH ER,VR...}
END;                    {Procedure WriteHeader}

{-------------------------------------------------------------------}

Procedure InitArrays( SN : Byte; VAR VR : VehicleRec;
                                  VAR StmArray : StimArray);

VAR
   Loop    : Byte;

BEGIN
   WITH VR DO
      BEGIN
      FOR Loop := 1 to 8 DO
         BEGIN
         SnOutArray[Loop].Out  := 0.0;
         SnOutArray[Loop].Time := 0;
         END;
```

```
      FOR Loop := 1 to 20 DO
         BEGIN
         TOutArray[Loop].Out  := 0.0;
         TOutArray[Loop].Time := 0;
         END;
      FOR Loop := 1 to 3 DO
         BEGIN
         T2OutArray[Loop].Time := 0;
         T2OutArray[Loop].Out  := 0.0;
         END;
      FOR Loop := 1 to SN DO
         BEGIN
         StmArray[Loop].Dist  := 0;
         StmArray[Loop].Ldist := 0;
         END;
      FOR i := 1 to 2*Sen_Num DO   {Initialize Sensor output array     }
         Sen_Out[i] := 0;
      FOR Loop := 1 to Sen_Num DO  {Determine Sensor FOV response      }
         WITH SenRecs[Loop] DO
            LoadSENData(Cutoff,MaxGain,CurveType,Curve);
           {Calc FOV response FOR each sensor, put in SenRecs[Loop].curve}
   END;              {WITH VR DO..}
END;                {InitArrays}

{-------------------------------------------------------------------------}

Procedure  WriteVFile(VAR  VhRec:  VehicleRec;  VAR  RF:  Text;  VName:
FileName);

VAR
   VhFile  : VehFile;
BEGIN
   REPEAT
      WriteLn('SAVE CURRENT VEHICLE? (Y/N)');
      ReadLn(Resp);
   UNTIL Resp IN ['Y','y','N','n'];
   IF Resp IN ['Y','y'] THEN
      BEGIN
      WriteLn('ENTER VEHICLE FILE NAME.  OLD NAME = ',VhRec.VehName);
      Readln(VName);
      Assign(VhFile,VName + '.Veh');
      Rewrite(VhFile);
      Write(VhFile,VhRec);
      Close(VhFile);
      END;            {IF Resp..}
END;                 {Procedure WriteVFile}

{-------------------------------------------------------------------------}
```

```
                {********** MAIN PROGRAM ********}
BEGIN
   REPEAT
      ClrScr;
      RanDOmize;
      Test := False; Video := True; WnTkAll := False; WrtRep := True;
      Start := 0; Stop := 0; ChildHood := 0; DLay := 0; NodeTrain := 0;
      SenSTM := False;
      X          := RanDOm(500);                  {BEGINX}
      Y          := RanDOm(300);                  {BEGINY}
      Hdg        := RanDOm(360);                 {BEGINHdng}
     SetUp(Start,Stop,ChildHood,NodeTrain,NodeSum,PrntRep,VFName,EFName,
     VRec,ERec,OpCond,LRtC,LRtF,L2T,WnTkAll,RnFlName,X,Y,Hdg,RnFl,WrtRep,
                                                                  SenSTM);
      OldX       := X;
      OldY       := Y;
      OldHdg     := Hdg;
      InitArrays(ERec.StimNum, VRec, StimuliArray); {Init VRec arrays}

      WITH ERec, VRec DO
         BEGIN
         BuildStimLookUp(StimDatArray); {Build look-up table FOR
                                                          dist/angles}
         PosTrainNum := 0; DeltaPosTrnNum := 0; {Init training recorder}
         NegTrainNum := 0; DeltaNegTrnNum := 0;
         TrkIndex := 1;                          {Init Track buffer index}
         Reward := 0; Punish := 0;
         LastReward := 0;
         Sign := False;
         UpDateStimDat(StimuliArray);     {Calc sensor inputs from Stim}
         IF WrtRep THEN
            BEGIN
            WriteHeader(RnFl,ERec,VRec,LRtF,LRtC,Start,Stop,NodeTrain,
              ChildHood,L2T,WnTkAll);         {Print report header info}
            NodeWtReport(RnFl,Sen_T1_Wts,T1_T2_Wts,0);{PrintNodeweights}
            END;
         IF Video THEN                              {Display movement}
            StartGraph;
         {*********** START VEHICLE MOVEMENT *************}
         FOR  T := Start TO Stop DO
            BEGIN
            SensorOut(T,Sen_out,SnOutArray,SenSTM);{Calc sensor output}
            Matrix1(Sen_Out,Sen_T1_Wts,T1_Out); {Propagate signals to T1}
            T1OutResp(NodeSum, T1_Out, TOutArray, WnTkAll);
                                          {Calculate T1 output    }
            Matrix2(T1_Out,T1_T2_Wts,T2_Out);{Propogate signals to T2 }
            T2OutResp(T2_Out,T2OutArray);    {Calculate T2 output     }
            Move(T2_Out,X,Y,OldX,OldY,Hdg,OldHdg,MoveDist,VehEnergy);
                                       {Calculate and save move   }
            IF Video THEN
               DrawMov(X,Y,Hdg);
            IF Keypressed THEN
```

```
                BEGIN
                Resp := GetKey;
                IF (Resp = '8') AND OpCond THEN      {Operant condition}
                   Inc(Reward)
                ELSE
                   IF (Resp = '2') AND OpCond THEN
                      Inc(Punish)
                   ELSE
                      Report;
                END;
             UpDateStimDat(StimuliArray);  {Calc sensor inputs from Stim}
             IF Not OpCond THEN
               FORage(StimuliArray,MoveDist,T,MaxEnergy,VehEnergy,Reward,
                           Punish,LastReward);        {Was food found?}
             Experiment(T,LastReward,VRec,ChildHood);
                                        {Not successful? Change behavior}
             IF (Reward > 0) OR (Punish > 0) THEN
                Learn(NodeTrain,ChildHood,T,Reward,Punish,T1_Out,Sen_Out,
                 T2_Out,Sen_T1_Wts, T1_T2_Wts,PosTrainNum,NegTrainNum,
                                              PrntRep,LRtF,LRtC,T2T);
             IF WrtRep THEN
                Stats(RnFl,VRec,PosTrainNum,NegTrainNum,DeltaPosTrnNum,
                                           DeltaNegTrnNum);
                Reward := 0;                          {Reset reward & punish}
                Punish := 0;
                Delay(DLay);                          {Give time FOR shaping}
                END;                        {FOR T = Start ...}
          Age := Age + stop;

          T := Stop;
          IF Video THEN                               {Shut DOwn graph mode}
          BEGIN
             RestoreCrtMode;
             CloseGraph;
          END;
       END;                                    {WITH RECs}
       WITH VRec DO
          IF (Frac(T/1000) <> 0) AND WrtRep Then
             NodeWtReport(RnFl,Sen_T1_Wts,T1_T2_Wts,Stop);
                                             {Print Node weights}
       WriteVFile(VRec,RnFl,VFName);         {Write final node wts}
       WriteLn(RnFl,'LAST VEHICLE NAME = ',VFName + '.veh');
       Write(RnFl,Chr(12));                          {Flush printer buffer}
       Close(RnFl);
       REPEAT
          WriteLn('MAKE ANOTHER RUN? (Y/N)');       {Continue?}
          ReadLn(Cease);
       UNTIL Cease IN ['Y','y','N','n'];
    UNTIL Cease in ['N','n'];                        {Quit!}
END.
                          {PROGRAM MAKERUN}
```

Bibliography

Alkon, Daniel L. "Memory Storage and Neural Systems," Scientific American, 260: 42-50 (April 1989).

Barr, Avron and Edward A. Feigenbaum (Editors). The Handbook of Artificial Intelligence. Los Altos: William Kaufman, 1981.

Bloom, Floyd E. and Arlyne Lazerson. Brain, Mind and Behavior. New York: W. H. Freeman and Company, 1988.

Braitenberg, Valentino. Vehicles: Experiments in Synthetic Psychology. Cambridge, MA: MIT Press, 1984.

Churchland, Paul M. A Neurocomputational Perspective: The Nature of Mind and the Structure of Science. Cambridge MA: MIT Press, 1989.

Cruz, Claude A. Understanding Neural Networks: A Primer. Amherst NH: Graeme Publishing Corp, 1988.

D'Amato, M. R. Experimental Psychology: Methodology, Psychophysics, and Learning. New York: McGraw-Hill, 1970.

Dumpert, Klaus. The Social Biology of Ants. Boston: Pitman Publishing Inc, 1972.

Forgus, Ronald H. and Lawrence E. Melamed. Perception: A Cognitive Stage Approach. New York: McGraw-Hill, 1976.

Geldard, Frank A. The Human Senses. New York: John Wiley and Sons, 1972.

Goetsch, Wilhelm. The Ants. Ann Arbor, Michigan: University of Michigan Press, 1957.

Kandel, E. R. and L. Tauc. "Mechanism of Prolonged Heterosynaptic Facilitation," Nature, 202: 145-147 (April 11, 1964).

Kandel, E. R. and L. Tauc. "Heterosynaptic Facilitation in Neurons of the Abdominal Ganglion of **Aplysia depilans**," Journal of Physiology, 181: 1-27. (1965).

MacDonald, Bruce A. "Connecting to the Past," Neural Information Processing Systems. 505-514 edited by Dana Z. Anderson. New York: American Institute of Physics, 1988.

Mazur, James E. Learning and Behavior. Englewood, NJ: Prentice-Hall, 1986.

Minsky, Marvin and S. Papert. Perceptrons. Cambridge, MA, MIT Press, 1969.

Porter, Kent and Mike Floyd. Stretching Turbo Pascal Version 5.5. New York: Simon & Schuster, Inc, 1990.

Rietman, Ed. Experiments in Artificial Neural Networks. Blue Ridge Summit, PA, Tab Books Inc, 1988.

Rogers, Steven K., Matthew Kabrisky, Dennis W. Ruck, and Gregory L. Tarr. An Introduction to Biological and Neural Networks, Wright-Patterson Air Force Base, OH: Air Force Institute of Technology, 1990.

Simon, Herbert A. The Sciences of the Artificial. Cambridge MA: MIT Press, 1981.

Soucek, Branko. Neural and Concurrent Real-Time Systems: The Sixth Generation. New York: John Wiley and Sons, 1989.

Wasserman, Philip D. Neural Computing: Theory and Practice. New Yory: Van Nostrand Rheinhold, 1989.

Vita

Eric B. Werkowitz was born 8 May 1949 in Dayton, Ohio. After graduating from high school in 1967 he served with the United States Marine Corps in Vietnam. Following his discharge, he attended Wright State University where he earned a Bachelors of Science degree in Human Factors Engineering. He was hired as an engineer by the Air Force in 1977 and began work in the Flight Dynamics Laboratory. While employed there he developed an in-house speech technology facility that was used to research applications of speech recognition/synthesis in aircraft cockpits. Mr Werkowitz attended the Air Force Institute of Technology on long-term, full-time training from September 1982 to August 1983. In 1985 he recieved a merit promotion to work in the Deputy Chief of Staff for Development Planning (ASD/XR) where he performs long-range acquisition planning for the Aeronautical Systems Division of the Air Force Systems Command.

Permanent Address:

ASD/XRS

WPAFB, OH 45433-6503