INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
A connectionist approach to autonomous robotic navigation

Weiland, Peter Lawrence, Ph.D.
Rice University, 1991
RICE UNIVERSITY

A CONNECTIONIST APPROACH
TO AUTONOMOUS ROBOTIC NAVIGATION

by

PETER L. WEILAND

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE
DOCTOR OF PHILOSOPHY

APPROVED, THESIS COMMITTEE

John B. Cheatham, Jr., Director
Professor of Mechanical Engineering

J. Ed Akin, Professor and Chair
Department of Mechanical Engineering

Angelo Miele, Professor
Department of Mechanical Engineering

Ian D. Walker, Assistant Professor
Department of Electrical Engineering

Houston, Texas

January 1991
ABSTRACT

Robotic navigation has been an area of intense research since the onset of mobile robot development. The usefulness of mobile robots ultimately reside in their ability to move and interact with the environment. Current approaches to robotic navigation are primarily based on simulating intelligent, human-like behavior through the intelligent system model processing cycle; sense, perceive, reason, act. Unlike these methods, this thesis presents a navigation system based on biological and behavioral principles which functions in a stimulus-response manner. Using connectionist architectures, a relationship between stimulus and response is acquired through the learning of conceptual information pertaining to navigation. In this research, the mammalian visual system provides a guide for the processing of environmental stimulus. Simulated laser range data are processed in retinal patch size elements by a cellular neural network. This network is designed to detect obstacle existence for each patch segment based on an invariant feature of range discontinuity. Obstacle information is then mapped in binary format, indicating the traversable state of the patch, to the system's visual cortex. Response to this mapping is derived from a hierarchical structure of back error propagation neural networks in which each network has learned a particular navigational behavior; obstacle avoidance, wander, and goal seeking. Output from these networks indicate an appropriate motor response for the environmental stimulus. A simulation was developed to evaluate the performance of this system by having a robot traverse an environment. The connectionist approach was verified through system display of human-like navigational behavior for the simulation's environment. Advantages of the neural network approach were also demonstrated by its processing speed and adaptability. Procedures are discussed for actual system implementation in which cycle times of under one second are completely feasible. Proposals for unsupervised learning of navigational responses for environmental stimulus are also made. From the research presented in this thesis, a foundation is established for continuing the study of the connectionist approach to the problem of autonomous navigation.
ACKNOWLEDGEMENTS

Although only one name is listed on the front of this work, it does not accurately reflect the number of people who contributed to its final completion. I wish to express my deepest gratitude to all of those who have supported me during this endeavor. I especially want to thank my thesis advisor, Dr. John B. Cheatham who gave me invaluable insight and direction. His advice and guidance contributed significantly to the quality of the final product. I wish to thank my peers, John Norwood, Yu-Che (Jack) Chen and especially Sarmad Adnan. I am deeply indebted to them for their time, consultation, and encouragement.

I wish to thank those responsible for providing me with this opportunity to pursue this degree. This includes the U.S. Army, Dr. Ed Akin, Chair of the Mechanical Engineering Department and Dr. Hardy Borland, Assistant Dean of Engineering. Their support truly provided me with the opportunity to continue my studies at Rice University.

I wish to make a special acknowledgment to the assistance provided by Professor Robert O. Shelton of NASA/JSC. His expertise in artificial intelligence and neural networks has been extremely beneficial. He has been a true inspiration throughout my research.

Most important of all, I wish to thank my wife, Lisa. It is doubtful that this work would have been possible without her support, encouragement, and patience. She provided an additional reason for pursuing works such as this and has made my life worthwhile.
DEDICATION

To Lisa and P.C.
# TABLE OF CONTENTS

Chapter One -- Introduction ........................................................................................................ 1
  1.1 Problem Statement ............................................................................................................. 1
  1.2 Current Approaches and State of the Art ......................................................................... 4
    1.2.1 Traditional Approaches to Navigation ...................................................................... 5
    1.2.2 Why a Neural Network Approach ......................................................................... 7
    1.2.3 Current Neural Network Approaches ...................................................................... 9
  1.3 Development of a Neural Network Navigation System .................................................. 11
    1.3.1 Network Configuration .......................................................................................... 12
  1.4 Specific Contributions ..................................................................................................... 14
  1.5 Measuring Performance and Assumptions ...................................................................... 16
  1.6 Organization of Text ....................................................................................................... 17

Chapter Two -- Developing the Neural Approach .................................................................. 19
  2.1 Intelligent System Model ............................................................................................... 19
    2.1.1 Traditional Navigation Sensors ............................................................................ 20
    2.1.2 Traditional Perception .......................................................................................... 22
    2.1.3 Traditional Navigation Reasoning ........................................................................ 22
    2.1.4 Concluding Remarks of Traditional Approaches .................................................. 25
  2.2 Navigation Hierarchy ....................................................................................................... 25
  2.3 Stimulus-Response Cycle ............................................................................................... 28
  2.4 Neural Network Application .......................................................................................... 31
    2.4.1 Structure ............................................................................................................... 33
    2.4.2 Training ................................................................................................................ 35
      2.4.2.1 Generalization versus Convergence ............................................................... 36
      2.4.2.2 Development of Training Patterns ................................................................. 38
  2.5 Executive Summary and Conclusion ............................................................................... 39

Chapter Three -- Sensing and Association ......................................................................... 42
  3.0 Extraction and Mapping of Environmental Information ............................................... 42
  3.1 Biological Considerations .............................................................................................. 43
    3.1.1 Human Visual System ......................................................................................... 44
    3.1.2 Visual Cortex ......................................................................................................... 46
      3.1.2.1 Invariant Feature Extraction ....................................................................... 49
    3.1.3 Functional Characteristics of Visual System ......................................................... 49
  3.2 Component Development ............................................................................................... 50
    3.2.1 Laser Scanner ........................................................................................................ 51
      3.2.1.1 LASER IMAGING SIMULATION ALGORITHM (LISA) ................................ 54
    3.2.2 Retinal Patches ...................................................................................................... 56
      3.2.2.1 Foot Placement-Robot Dimensions ............................................................... 58
  3.3 Implementation of Sensing-Association System .............................................................. 59
    3.3.1 Shunting Equations ............................................................................................... 63
    3.3.2 Cellular Neural Networks ...................................................................................... 71
3.3.2.1 Edge Detection .......................................................... 77
3.3.3 Cellular Neural Network Performance .............................. 78
3.4 Classifying Regions ........................................................ 81
  3.4.1 Classifying Network .................................................. 82
3.5 Summary of Sensing Issues .............................................. 84
  3.5.1 Executive Summary of Specific and Significant Results .... 86

Chapter Four -- Navigational Behavior .................................. 88
  4.1 Artificial Neural Network Approach to Navigation ............... 88
  4.2 Biological Considerations ........................................... 89
    4.2.1 Implementation of Biological and Behavioral
         Considerations ................................................... 93
  4.3 Computer Simulation .................................................. 94
    4.3.1 Simulation Display and Representation ....................... 94
    4.3.2 Generation of Training Patterns .............................. 98
    4.3.3 Training ........................................................ 99
      4.3.3.1 Counter Prop Networks ................................ 100
      4.3.3.2 Back Prop ............................................... 101
    4.3.4 Evaluation ..................................................... 101
  4.4 Obstacle Avoidance Network ......................................... 103
    4.4.1 Obstacle Network Training .................................. 105
    4.4.2 Obstacle Network Performance ............................... 106
  4.5 Wander Network ....................................................... 108
    4.5.1 Training Pattern Development ............................... 108
    4.5.2 Network Structure and Training ............................. 111
    4.5.3 Wander Network Performance ................................. 114
  4.6 Goal Directed Movement Network .................................. 117
    4.6.1 Goal Directed Network Training ............................. 120
    4.6.2 Goal Network Structure ..................................... 123
    4.6.3 Development of Exploration Process ......................... 124
    4.6.4 Evaluating Goal Directed Behavior ......................... 127
  4.7 Summary of Navigation Network ................................... 131
    4.7.1 Executive Summary ............................................ 132

Chapter Five -- Application .............................................. 134
  5.1 System Simulation ................................................... 135
  5.2 System Performance ................................................ 141
  5.3 System Implementation ............................................. 144
  5.4 Extension of Current Work ....................................... 147
  5.5 Summary of System Performance .................................. 151

Chapter Six -- Summary and Conclusions .............................. 152
  6.1 Summary of Development Processes ............................... 152
  6.2 System Performance ............................................... 155
  6.3 Conclusions ....................................................... 156
  6.4 Closing Remarks .................................................. 158
Chapter One -- Introduction

Robotic navigation, or more simply put, the movement of a robot through its environment has been an area of intense investigation since the onset of mobile robot development. Usefulness of mobile robots ultimately resides in their ability to move and interact with the environment. Consequently, the mobile robot’s future depends on the effectiveness of the navigation system. The research described in this thesis is concerned with the development of a neurally inspired approach to the robotic navigation problem.

1.1 Problem Statement

Robotic navigation systems must be capable of directing movement in collaboration with the desired task and current environment. In developing these navigation systems, there is a proportional relationship between the degree of environmental interaction and the sophistication of the navigation system. Navigation systems can be classified by this degree of interaction. At one end of the spectrum are automatic guided vehicles (AGV’s) which exhibit limited interaction with the environment.

AGV’s are used primarily in the work place for transporting items between manufacturing processes. According to Ronald Arkin [4], "These vehicles simplify the problem of navigation by restricting their paths to predetermined routes, which are typically demarcated by striping the floor in some manner or by using buried cables. A major issue is just how 'flexible' such systems are". Increasing flexibility requires
increasing the interaction and responsiveness of the system to environmental conditions. The most flexible systems, which are at the other end of the spectrum, are designated autonomous navigation systems.

Autonomous navigation systems are characterized by their ability to exhibit intelligent movement for environmental conditions independent of external guidance. Intelligent responses for these systems are typically based on human navigational performance. The benefits of obtaining a robotic vehicle that exhibits human-like performance cannot be overstated as this would achieve the greatest environmental interaction conceivable. Obtaining the necessary sophistication for autonomous behavior is an extensively pursued goal in robotics research.

The concept of navigation includes many aspects of mobility. In its most elementary form, it is the simply facet of moving along a collision free route from point A to point B. In the research literature this process usually comes under one of the following headings: path-planning, find-path, or mobile robot obstacle avoidance (as opposed to manipulator obstacle avoidance). While the movement between two points is a humanly intuitive task, attaining human-like performance in robotic vehicles has been elusive.

The problem of robotic navigation pursued in this work is to develop principles and concepts for an autonomous system capable of exhibiting human-like performance for
moving in a collision free manner between two points in the environment. System autonomy should be demonstrable by its ability to interact with the environment in a human-like manner with no external assistance. A system of this sort will advance the applicability of mobile robots to a wide variety of non-specific, generic, tasks. While there is currently an extensive amount of research directed toward achieving autonomous mobility, the resurgence of neural network research provides the impetus for seeking a neural network solution. The neural approach espoused in this thesis represents a significant departure from traditional approaches.

In this research a neurally based navigation system was developed by designing the system along anatomies and behaviors of biological systems. A sensory-association system, based on the mammalian visual system, processes laser sensor data to extract obstacle positions within the environment. These locations provide input to a response processes that is developed in a hierarchical fashion along lines of navigational behavior. Output of the response system dictates an appropriate motor command which updates the robot's position in the environment. This process is modelled in Figure 1.1. Issues concerning both this approach and current traditional approaches to navigation are discussed in this chapter. For a discussion of neural network technology, a neural network tutorial is
provided in Appendix A.

1.2 Current Approaches and State of the Art

![Intelligent System Model]

Figure 1.2 Intelligent System Model

Autonomous navigation is an intelligent process. Obtaining comparable intelligent performance in a mobile robot is the objective of the navigation system. From observations of human action, NASA/Johnson Space Center [22] uses a model that describes an intelligence process as consisting of four components: sensing, perceiving, reasoning, and action. Current approaches to autonomous navigation are predominately based on this model. With a limited choice in both sensor selection and motor drives which actually cause the action, most researchers focus on the development of perception and reasoning processes. Algorithms, models and representations for these processes are developed to achieve the desired human-like performance. The emphasis of these approaches is in obtaining external human-like appearances from any applicable internal process. In contrast, the strategy employed in our research is to achieve human performance by basing the system on an artificial implementation of the substance and structure of human intelligence; neural networks. A review of the current approaches to navigation will now be presented followed by a discussion of the applicability of the neural network approach. Current neural network approaches and related neural network research areas will then be discussed to provide the reader with an understanding of the
foundation and background for the neural approach.

1.2.1 Traditional Approaches to Navigation

There is wealth of literature pertaining to robotic navigation. It is a principal topic at major robotic conferences and is communicated in most every journal associated with robotics. For this discussion, only the most common systems, based on the intelligent system model and using algorithmic or AI reasoning schemes, are being reviewed. These procedures will be called the traditional approach to autonomous navigation.

Perception in traditional navigation systems is the explicit representation of sensory data in a form that permits cognition. This representation is usually a top down view of the environment that either represents obstacles of the scene as polygons, examples being Lozano-Perez [55][56], Brooks [8] or as occupying grid positions, Oliver [72], Metea [61]. From this representation some algorithmic reasoning process is applied to derive an appropriate response from the perceived environment.

The reasoning procedure must be in concert with the representational form of the perception phase. Usually, polygon representation results in a computer graphic type analysis of the scene. Lozano-Perez [55][56] and Brooks [8] provide some typical examples of this technique. Computer graphic approaches usually utilize a generate and
test\textsuperscript{1} analysis. Potential routes or path segments are generated and then tested for feasibility of travel. Grid representations are often more algorithmic or mathematical in their analysis. Typical examples of this approach are by Metea [61] and Oliver [72]. Grid position values are usually associated with some cost for traversal of that particular position in space. Procedures are then applied to establish a path of grid positions that obtain a minimum cost of traversal across the representation.

The approaches discussed above are representative of a major portion of navigational systems. Numerous other perception and reasoning processes have also been developed and used in the intelligent system model. Pomerlau [73] and Jorgensen [44] have even used neural networks to provide for the representational processes of the intelligent system model. Norwood [67] has also utilized a neural network for the reasoning processes. However, the majority of systems are similar in concept to those presented above with some particular nuance added by the researcher.

Several major autonomous navigation programs, Turk [82][83], Dunlay [21], Gex [25], Herbert [41], process information along the guidelines of the intelligent system model. While complete success, as measured by comparable human performance, has yet to be achieved, the intelligent system model approach has permitted some semblance of

\textsuperscript{1} Winston [89] provides an excellent description of the generate and test approach used for Artificial Intelligence (AI) problems such as path selection. He also presents an overview of many AI search techniques used for selecting acceptable paths.
human-like performance. Typical examples are the road following vehicles of Carnegie
Mellon University (CMU) [49] [30] [31] [80], Martin Marietta [83] [21] [76], and FMC
[64]. FMC's vehicle, described by Kuan [50], is capable of real-time obstacle avoidance
at 8 km/hr and real-time road following at 19 km/hr.

Common to these sophisticated navigation systems is a hierarchical structure of
navigation tasks, Nita [64], Quek [74], Keirsey [45]. The higher level of this hierarchy
is characterized by the processes of task planning and mission analysis. At the opposite
end of the hierarchy, the more basic movement decisions are made. Brooks [7] has
formalized this concept into a hierarchical structure of eight navigation processes. Brooks
also provides a theory for developing procedures for each level of the hierarchy
independently. This procedure proves quite useful in reducing the complexity of
perceptual and cognitive tasks.

1.2.2 Why a Neural Network Approach

Determining a path through a scene seems like a rather intuitive task for humans.
It is doubtful that humans confront this problem by connecting path segments with
obstacle positions, calculating cost functions, or through some other application of
mathematical algorithms. To accomplish a task that is intuitive to humans, why not
approach the problem from how the human accomplishes the task? Neural networks
provide a computational process applicable to problems of this nature, Martinez [57].
Figure 1.3 Neural Network Approach

Human action can be described as a response to the application of either an internal or external stimulus, Bourne [6]. For physical responses, this Stimulus-Response (S-R) process can be referred to as sensory motor response. Neural networks function in a S-R fashion where input provokes a learned response through network propagation. Applied to navigation, environmental stimulus can be used to prompt an appropriate learned motor response to provide for collision free locomotion through the scene, see Figure 1.2. According to Ogmen [71], the stimulus-response cycle of neural networks consists of three phases: sensing, association, response.

*Association* can be compared to the processes of perception in the intelligent
system model. Network propagation of signals, which results in a response, can be considered analogous to reasoning. Instead of explicitly representing environmental information and applying reasoning algorithms, these processes are innately bred in the internal representation and dynamics of neural networks. Furthermore, this internal representation gives the network a generalization capability, Rumelhart [75]. When similar, incomplete or noisy environmental information is provided to the network, the network is able to infer a proper response in many instances where a traditional approach may fail. Thus, because of its ability to generalize and develop an internal understanding of the problem through training, neural networks may be more applicable to the problem of navigation than traditional approaches.

1.2.3 Current Neural Network Approaches

Neural networks are basically pattern classifiers. Input patterns are classified into different responses by the network based on training or network dynamics. Network dynamics and training extract a conceptual basis from training data and store this abstraction in the internal representation of the network, Rumelhart [75], Kohonen [48]. To develop a network for navigation requires that the concept of navigation be extracted during the training process and stored in the network. Subsequent application of environmental stimulus will then evoke an appropriate behavioral response based on the extracted concept. This process describes the approach taken at Fujitsu Laboratories [77] in developing a neural based navigation system.
At Fujitsu Laboratories [77], they have imparted navigational behavior to two toy robots using neural networks. Using a back error propagation network with supervised learning, the robots were able to properly respond to environmental input by avoiding obstacles. Further, additional behavior was bestowed upon the robots by teaching one robot the behavior of pursuit and the other the behavior of avoidance. Being able to sense one another, the two robots were capable of playing a game of cops and robbers. A significant attribute of this work is the use of two neural networks in a hierarchical design to simplify the training process. As with the navigational hierarchy espoused by Brooks [7], neural networks can be devised for a specific aspect of mobility planning. This hierarchical approach is extensively utilized in the research effort presented in this thesis.

Advancements in neural network technology have permitted researchers to apply networks to many traditional problems. Studies of network dynamics and internal representations, Grossberg [37], McClelland and Rumelhart [58], Wasserman [85], Lippman [54], explain the essence of the neural approach. These works provide a firm understanding of networks which is pivotal for developing proper training schemes and applications. While the field of navigation has yet to receive much specific attention, work is progressing in other areas which have an application to navigation. Research in manipulator movement, Kuperstein [53], and manipulator obstacle avoidance, Mel [60], provide methods for unsupervised learning of robotic movement and control. Some of the principles developed in these works can be applied to mobile robots. There are some
tangential areas, less related to robotics and neural networks, that also offer insight into developing structures for intelligent navigation. The areas of physiology, Carlson [11], and psychology, Bourne [6], explain the structures and behaviors that comprise the processes of human intelligent action.

1.3 Development of a Neural Network Navigation System

It is an objective of this research to determine appropriate network structures and training procedures to acquire human-like navigational response from neural networks. To accomplish this requires the selection of appropriate network dynamics, configurations and training procedures. In developing this system a decision was made to retain the biological analogies of networks as much as possible. Structures and processes were designed based on established physiological, Carlson [11], and psychological principles, Bourne [6]. For example, humans are not bound by a rigid configuration of head position and body angle when interacting with an environment. Thus, structures and processes were developed that sought to minimize the reliance of rigid sensor configurations in the robotic system.

While there is no contention made that these artificial systems are equivalent to those of their biological origins, it is interesting to explore the depths of this type of implementation. In this way, the expanse of the neural network, biological, psychological approach to navigation can be determined. Mel [60] has even gone as far a describing the personality of his neural robot. This may appear as a wasted, inane effort at first, but
in seeking *human-like* intelligent performance, some aspects of intelligence may be byproducts of structure. Since the original objective of this research was only to obtain autonomous *human-like* navigational performance, it should not matter how the system is structured as long as it achieves the desired outcome. Approaching the problem along a biological and psychological lines, even if there is no correlation between living and artificial systems, provides, at a minimum, a model on which to develop the navigation system.

### 1.3.1 Network Configuration

There are numerous existing neural network paradigms from which to choose in designing a neurally based system. Wasserman [85] provides a good overview of the more common and significant networks. While it is possible to develop a new paradigm, it is more convenient to apply an existing paradigm since the dynamics and stability of these networks are well documented. In this work, several network paradigms were examined for various aspects or applications within the system. Two networks were determined to be especially suited for the navigation task. For learning the behavior of navigation, the back error propagation network of Rumelhart [75] was selected. A short term memory network, based on a cellular structure of Chua [13], was used for the extraction of obstacles from sensory data.

Neural networks were constructed in a biologically feasible anatomy, operating in a *stimulus-response (S-R)* manner, for navigation. This process consists of two distinct
phases. First, simulated scanning laser range data provided sensory input to the **eye** of the navigation system. The structure and processes of the sensing system were based on established physiological principles described in Carlson [11]. Propagation of sensory data in the **eye** was through a cellular neural network developed by Chua [13]. Output from this network was processed through a classifying network which resulted in an internal mapping of obstacle information.

The second phase of the process uses this mapping as input to a hierarchical structure of back error propagation networks. Each layer of the hierarchy, pertaining to a different aspect of navigation, was taught an appropriate behavioral response for the incoming environmental information. While the work of Sekiguchi [77] also utilized back error propagation networks in a hierarchical structure, the procedures developed in this research differ. More complex environmental data is used in this research and responses are learned by the network through **watching** human responses to similar data. Further, the two layer hierarchy of Sekiguchi was implemented for sequential behavior and not separate behaviors as was implemented here.

From the concept of the S-R cycle, a feasible robotic navigation system was developed based on the biological and psychological principles discussed above. The validity of this approach was demonstrated through a computer simulation of the two major functional areas: sense-association (perception) and supervised learning of navigation behavior. Separate software packages were developed for each area and are
discussed in Chapters Three and Four respectively. Chapter Five combines these two processes for a system application in a simulated robot environment.

In developing the software packages, over 4,500 lines of "C" code were written. The packages were developed to operate on an IBM compatible PC. There is extensive use of graphics in the packages requiring at least an EGA configured system.

While there are no claims made that the final network design of this research is the definitive structure for navigation, a positive confirmation as to the applicability of the neural network approach to navigation was derived. *Human-like* navigational performance was exhibited from the network structures developed. The procedures developed in this research validated the concept of an artificial neural network based S-R approach to navigation.

1.4 Specific Contributions

While neural network technology cannot be considered new, its application to the field of robotic navigation is a relatively new development. This research shows that by formulating the problem correctly, the general principles of neural networks are applicable to navigation. In developing procedures and structures, this work establishes and defines issues germane to the application of neural networks for navigation. Of specific interest is the analysis of procedures used in developing training for networks and the hierarchical organization of networks. Different methods were used to develop training patterns for
each of three different levels of the navigational hierarchy. In one approach, the system was able to watch a human respond to input. Procedures had to be developed to account for the many inconsistencies of human input. The hierarchical organization allowed individual networks to be developed for a specific behavior. These individual responses were then combined to exhibit more complex behavior. This research also presents methods and architectures that may enable a robotic system to learn these responses autonomously. Methods are proposed for autonomous learning through internal genetic design. The developments described here clearly extend the field of study of neural networks in the area of robotic navigation.

The main focus of this work has been to demonstrate the application of neural network technology to the field of navigation. By basing network architecture on a physiological and psychological foundation additional developments and contributions are made to the field. One specific contribution is applying a cellular neural network, originally designed by Chua and Yang [13] for vision applications, to the processing of range data for navigation applications. This cellular network was selected because it exhibits many of the biologically observed phenomena of human retinal cells. By continuing the integration of biologically based processes and behaviors into the automated world of robotics, this work should encourage continued research in this area. From this work, it can be seen that neural networks are quite applicable in obtaining human-like navigational performance. Through the groundwork of previous neural network researchers and the information presented here, neural networks may someday
provide the means of obtaining truly human-like navigational capabilities.

1.5 Measuring Performance and Assumptions

In evaluating the performance of an intelligent system, it is often most useful to compare the artificial system's response with that of an intelligent system's response. This rationale is the basis for the evaluation standards developed here. Networks were trained to exhibit human-like navigational responses to environmental conditions. An easy and direct measurement of network performance would then be a comparison of human performance with that of the network. In evaluating performance in this manner, one must completely rely on a subjective evaluation of the robot's response. As in most artificial intelligence applications, no effort has been made at obtaining the optimal solution. It is sufficient in mimicking human movement to choose acceptable routes or movement. This sufficiency criteria is the basis for the subjective evaluation.

A major assumption of this research is that it is expandable and portable to real world usage. Research cost limitations, discussed in the text, precluded the actual employment of the system on a mobile robot. A computer simulation of an environment and mobile robot were produced in order to develop the principles and concept of neural navigation. In developing the simulation there were naturally numerous assumptions and simplifications. Every effort was made to limit the artificialness as much as possible. A point in case is that while the simulation's environment is a flat, almost interior office type setting, the system, as configured, could operate in uneven open terrain. This is
accomplished primarily through the use of binary declarations as to a region's traversability and removal of rigid sensor configurations for developing the internal representation.

Navigation is a broad classification of many facets of locomotion. For brevity and in keeping with the goal of simply proving the concept of navigation networks, attaining all aspects of navigation was not attempted. This research concentrates on the lower, simple operations of moving through an environment. It is assumed that some, higher level system can plan the current task and determine the goal location within the scene. It is also assumed that concentrating on merely forward movement of the robotic vehicle is sufficient enough to prove the concept of neural navigation.

1.6 Organization of Text

Organization of this text parallels the development of the robotic navigation system. After this introductory chapter, a more detailed understanding of navigation, neural network and S-R principles are developed in Chapter Two. Subsequent chapters then sequentially form the sub-sets of the navigation system development. Chapter Three discusses the processes of sensing the environment and developing internal representations. Chapter Four provides methods for responding to the sensory information through supervised training. These two systems are combined in a simulated application discussed in Chapter Five. Extensions of this work and methods for autonomously learning navigation are also presented in Chapter Five. Chapter Six then summarizes this
research and highlights some specific conclusions. Appendix A contains a brief tutorial on neural networks. For the uninitiated, it is highly recommended that the tutorial section be read first in order to comprehend the neural network terminology and discussions of the text. The remaining appendices contain the source code for simulation developed in this thesis.
Chapter One provided an introduction to the neural approach to navigation developed in this research. This chapter seeks to expound on some of the principles that have motivated this research. Specifically, issues concerning both the applicability and the implementation of neural networks for navigation are presented here. Traditional navigation systems, based on the intelligent system model, provide an understanding of the issues concerning the process of navigation. Neural networks differ from these systems by basing responses on how they are developed in intelligent beings. Conceptual encoding of information permits systems to form a relationship between input and output for a myriad of environmental conditions. Networks, being able to produce intelligent, conceptual responses, are applicable to the problem of navigation. Developing structures and architectures for navigation networks enters into the realm of the artistry of neural networks. While biological systems provide some fundamental anatomies on which to model networks, network parameters and training are best developed through a sound, fundamental understanding of network dynamics. The issues confronting network employment and development are discussed in this chapter.

2.1 Intelligent System Model

Early artificial intelligence (AI) research sought to define a process as intelligent by analyzing actions naturally considered intelligent, Winston [89]. Because of the intangible properties of intelligence, a model was devised to describe the components
constituting an intelligent action. Through the observation of intelligent actions, researchers were able to discern four functional elements: sensing, perceiving, reasoning and action, NASA [22], see also Quek [74]. These four processes comprise the intelligent system model that was presented in the previous chapter, see Figure 1.1.

In applying the intelligent system model to navigation, the four processes, while separate, are clearly dependent upon one another. The environment is first sensed to extract pertinent environmental information. The perception phase then represents the sensory data in a form to permit cognition. Cognition, or reasoning, produces an appropriate movement command based on the perception of the environment. This movement command is then decoded and used to drive the actuators of the vehicle which results in an action that was ultimately based on sensory data.

Since traditional navigation systems are all based on this model, differences in systems reside in their use of sensors, representations and reasoning procedures. Sensors are selected based on their ability to meet the representational needs of the perception and reasoning modules. The type of sensor will also dictate the degree to which environmental information may be extracted.

### 2.1.1 Traditional Navigation Sensors

Navigation decisions will be predicated on the spatial relationship of objects within a scene. Therefore navigational sensors must be able to extract the three
dimensional relationships of environmental objects. There are three sensors primarily used for this purpose: sonar, laser range and video or vision systems. Each of these sensor has a different capability of extracting environmental information. Consequently, the selection of a particular sensor will be based on application. There is a plethora of background information concerning these three sensors which exceeds both the discussion space and discussion necessities of this section. Nitzan [66] offers an excellent review of the internal workings of these sensors and of their ability to extract three dimensional information. The characteristics and primary applications in navigation for these sensors can be quickly summarized as follows:

<table>
<thead>
<tr>
<th>SENSOR</th>
<th>Description of Environment</th>
<th>Image Processing</th>
<th>Use in Navigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sonar</td>
<td>Single Reading</td>
<td>Simple</td>
<td>Blob Detection</td>
</tr>
<tr>
<td>Laser</td>
<td>Range Values</td>
<td>Medium</td>
<td>3-D Relations</td>
</tr>
<tr>
<td>Vision</td>
<td>Rich</td>
<td>Difficult</td>
<td>Object Recognition</td>
</tr>
</tbody>
</table>

In selecting a sensor for navigation, factors of use, cost, environment, and most importantly, representational needs of the reasoning system must be considered. It is possible to improved sensing capability by fusing multiple sensor outputs. For example, the NAVLAB at Carnegie Mellon University [41] fuses the output of vision and laser range sensors in some applications. Research described in this thesis is based on the use of simulated laser range data. Generation of these data will be discussed in Chapter Three.
2.1.2 Traditional Perception

As mentioned in the previous chapter, traditional navigational perception tasks represent sensory data in a top down representation in one of two ways. The first method is to model objects of the scene as polygons. This approach explicitly depicts the spatial relationship amongst objects. The second form is the use of a grid-like structure. Grid positions are assigned numeric values pertaining to the traversal of corresponding physical positions within the scene. From both of these representations, an algorithmic or AI reasoning procedure is usually applied to find an appropriate path through the scene.

2.1.3 Traditional Navigation Reasoning

The number of different perceptual representations of the environment is somewhat limited because of the format of sensory data. In contrast, the number of different reasoning systems is only limited by the imagination of the researcher. There is a profusion of navigational reasoning systems which makes it difficult to categorize the procedures easily. For this discussion, a characteristic approach for both of the representations discussed above will be presented.

One of the earliest approaches, and still one of the more prominent which utilizes the polygonal representation, is the configuration space approach developed by Lozano-Perez [55][56]. In the configuration space approach, robotic vehicle dimensions are taken into account by shrinking the robot to a point and expanding the polygonal obstacles in accordance with the vehicle dimensions. Line segments then connect the vertices of the
expanded polygons. These line segments then represent all acceptable routes of travel. Some search technique, typically A* (see Winsom [89]) is then used to select a route, see Figure 2.1.

**Figure 2.1 Configuration Space**

Most systems that utilize a polygonal representation can be appropriately classified as computer graphic techniques. Intersections between potential paths and obstacles are calculated directly from the graphical representation. There are several reasoning schemes that may be applied to the selection of an appropriate path besides traditional AI search routines. Heuristic [12], analytic [10], and numerous other algorithmic procedures have been used to determine appropriate paths from the candidate route segments.

Unlike computer graphic approaches which evaluate pathways, grid representation techniques evaluate the traversibility of individual positions. Each position of the scene is evaluated as to its traversable state. Differences in grid approaches reside in the manner in which values are assigned to the positions of the grid. Elevations, costs of traversal, or potential fields are typical approaches. Once values are assigned to the grid, a search procedure, such as A* for instance, is used to determine an acceptable path through the grid.
A typical usage of a grid representation is illustrated in a previous work by the author and John D. Norwood of Rice University [86] [67]. The perception process extracts obstacle locations from simulated laser sensory data and maps these locations to the grid. Obstacle positions then emit a repulsive force across the grid while the goal position emits an attractive force. Effects of these forces decreases linearly with distance and are summed for each grid position. A path to the goal is obtained by using a neighborhood search technique that seeks the route of highest potential.

A difference in this neural network approach and the one developed in this thesis, is that the previous approach was based on the intelligent system model and used the neural network only for the reasoning process. The current research is devoted to developing a complete neural network system based on physiological and psychological models. Furthermore, the previous network did not actually learn how to navigate. The network structure offered a convenient method for computing the effects of the emitted potentials. Network weights were adjusted by the position of obstacles and goals and not through a learning algorithm. While this previous network provided the reasoning capability for the system it lacked the ability to actually learn the concepts of navigation.
2.1.4 Concluding Remarks of Traditional Approaches

The preceding discussion was not designed to be a thorough analysis of traditional navigation approaches nor was it design to dispel these approaches. This endeavor was to merely show how the problem is currently being approached. While there are other navigation techniques that do not specifically fall into the general categories discussed above, most are similar in some regard. Common to almost all of the current, traditional approaches to navigation is the reliance on the intelligent system model processing scheme. In using the model it is permissible to base the perception and reasoning processes on any desired technique or concept. Whatever achieves the desired representation or cognitive process is appropriate. The approach espoused in this work is to build the navigation system more along the lines of the biological processes instead of some mathematical representation. A concept that is useful to traditional approaches and also to the neural network approach is that of the navigation hierarchy.

2.2 Navigation Hierarchy

Brooks' [7] noted that early navigation systems were often hampered by large, complex and often burdensome perception and reasoning processes. He has suggested that the navigation problem should be segmented according to the desired external operation of the system and not along the internal workings of the solution. In other words, the problem should be decomposed along behavioral lines and not the functional lines of the intelligent system model. Approaching the problem in this manner alleviates the requirement of having large, bulky, all encompassing perception and reasoning
processes. Separate procedures can be developed for specific behaviors. The result is that the individual processes, delineated in the hierarchy, will be less complex, see Figure 2.3.

Each level of the hierarchy can be considered a separate intelligent process. While individual levels would operate sequentially, the hierarchy as a whole would processes information in parallel. This structure not only reduces the complexity of each level's processing, it also permits pursuing several goals at different levels. Each level receives specific sensory data and develops its own perception and reasoning process for a distinctive task. Different sensors can be used at different levels according to their ability to develop the representational needs of a given level. Output from each level is then combined as part of the bigger system.

The navigational hierarchy not only simplifies the processes at each level, it also permits the use of a building block approach in developing the navigation system. Succeeding levels of navigational behavior can be constructed on top of existing proven layers. This not only enables separate development of components, but also allows the

<table>
<thead>
<tr>
<th>Reason About Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plan Changes</td>
</tr>
<tr>
<td>Recognize Objects</td>
</tr>
<tr>
<td>Monitor Changes</td>
</tr>
<tr>
<td>Build Maps</td>
</tr>
<tr>
<td>Explore - Seek Goals</td>
</tr>
<tr>
<td>Wander</td>
</tr>
</tbody>
</table>

Figure 2.3 Navigation Hierarchy after Brooks[7]
utilization of proven methods for various applications. In effect, it permits the designer to select the appropriate tool for the job.

As discussed in Chapter One, this research concentrates on only the low level navigation tasks of Figure 2.3: obstacle avoidance, wander and goal seeking (exploration). Since representation and reasoning routines are developed only to the degree required for a particular level, the use of the polygonal representation may be improper for these lower levels. For instance, at the obstacle avoidance level it is not necessary to identify an object in order to avoid a collision with the object. Knowing that something exists, often referred to in the literature as a *blob*, is sufficient enough information to avoid a collision. In perceiving the *blob* there is no need to extract any features other than the knowledge of it physical presence.

In developing a polygonal representation from sensory data, features of objects within the scene must be extracted. When edges or features are extracted to model the obstacle as a polygon, the obstacle is no longer merely a *blob*. It begins to take shape which is one of the rudimentary processes of object recognition. It is questionable whether the detail of the polygonal representation is necessary. A grid representation is immune to the above feature extraction problem. Only physical presence and location of objects are being extracted from sensory data. Features and characteristics are of no concern. Thus, a grid representation simplifies the modelling of the scene to a minimal level for obstacle avoidance. Furthermore, polygons may be useful models for the
computer graphics world, but the real world does not consist of pure polygonal shapes. The grid representations permits the representation of any shaped obstacle.

The hierarchical concept has been widely implemented for navigational purposes. The sophisticated autonomous navigation systems cited in the previous chapter all use some form of hierarchical organization of navigational tasks. The development of the navigation system of this research will also be based on a hierarchical structure. However, instead of traditional processing at each level of the hierarchy, a neurally based system will be employed. In addition, it is also noted that the grid-type representation will be used in the neural approach.

2.3 Stimulus-Response Cycle

Human movement is the result of stimuli applied either externally through the environment or internally through what are commonly called mediational factors, Bourne [6]. For navigation, external stimulus can be considered the environment and mediational factors, internal stimulus, describes the type of interaction desired with the environment. Together, these stimuli provoke an appropriate movement or motor response. The selection of this motor response is the result of learning how to relate stimuli to an appropriate response. Learning processes garner a concept of how stimuli are related to responses. Learning a concept "... is not only an easy way but also a necessary way of dealing with the tremendous diversity one encounters in everyday life. Concepts code things into a smaller number of categories and thus simplify the environment to some
degree", Bourne [6]. Intelligent robotic navigational behavior is sought in this research through the learning of navigation concepts. Once these concepts are learned, they can then be applicable to a myriad of environmental conditions. An issue arises as to the network’s ability to learn a concept.

It is interesting to note that the process espoused by some behaviorist psychologists for the learning of concepts, see Bourne [6], are extremely similar to the learning processes of supervised neural networks. The behavioral psychologist will present a subject with a data set of which some concept is to be extracted. As a subject categorizes or classifies an input, he is informed as to the correctness of his response. Data are repeatedly presented until the number of misclassifications diminish. This process is analogous to neural network supervised learning schemes for networks such as back error propagation.

In concept learning, the desire is to extract from the input data the relevant features which clearly define and categorizes the input. Application of training patterns that enhance the extraction of conceptual information will facilitate the learning process. The selection of these patterns is a pivotal issue confronting the development of the navigation system described in this thesis. Through the learning process, internal representations of the neural networks effectively encode the relationship stimulus and response, Rumelhart [75]. Upon subsequent application of similar data, the generalization feature of neural networks enable an appropriate classification. "When behavior comes
to be controlled by a few relevant stimulus features of a variety of otherwise dissimilar patterns, that behavior represents an abstraction and is called conceptual"; Bourne [6]. These relevant features can be considered invariant across the input space and characterize a particular input pattern.

Conceptual encoding is related to the generalization capability of neural networks. Through learning and network dynamics, neural networks extract distinguishing features of input patterns of the training set. Upon subsequent presentation of new patterns, network processes will determine a degree of similarity between these new patterns and previously learned patterns. If they are reasonably similar, the new input is classified as belonging to the previous category. An example of this process would be that of a network taught to distinguish polygonal shapes. Assume this network is trained on the shapes of the triangle, square, and octagon. The network develops an internal representation of the problem by determining some feature of the input space which distinguish between the inputs. If a pentagon shape is then presented, it will be seen as being most similar to the square. This is the generalization capability of the network and in this case, the network has developed some concept of polygonal forms.

By being capable of encoding conceptual relationships between stimulus and response, neural networks are applicable to problems such as navigation. This research seeks to impart this concept into network structures through training. Frequently there is not a lone, single concept applicable to a particular input. There may be several
appropriate responses for the same input. Mediating factors will dictate the application of an appropriate concept. This will be shown in later examples where different movement responses are generated for identical environmental conditions depending on the type of interaction desired.

Since conceptual information is encoded in the internal representation of the network, there is a mapping or associating of sensory data to this internal representation. Responses, or output, are then generated from this internal representation. Thus, the neural network approach operates in a sense-associate-response cycle. In noting the similarity in the stimulus-response cycle and that of the intelligent system cycle there should be little surprise since the biological system provided the basis for the intelligent model. The sense and association process forms the perception of the environment and the response can be considered as the cognitive process. For this research, the sense-associate-response process is divided into two systems. The first sub-system consists of the sense and association process. This entails the conversion of sensory data into an internal mapping and is discussed in Chapter Three. Responses will then be made from this mapping. This response sub-system, which will also use an internal mapping of its own, is discussed in Chapter Four.

2.4 Neural Network Application

Neural networks are extremely useful in areas where the system must be adaptive to partial or incomplete input, Martinez [57]. These areas are often characterized by the
difficulty of explaining or defining these phenomenon mathematically and can generally be considered problems which humans solve intuitively. Neural networks applicability to intuitive problems may be innately acquired because of their foundation in biological principles. Artificial neural networks are based on the operation of neurons of the brain. These neurons in essence provide the biological being with its intelligence and hence its ability to handle problems of an intuitive nature. Applying neural networks to achieve intelligent behavior is therefore essentially basing the solution to intelligence on the processes of intelligence. Instead of developing the system by mimicking or simulating behavior as with the intelligent system model, the network approach seeks to develop the system by emulating the processes of intelligence that attains the behavior. If human intelligent performance is desired, why not approach the problem by analyzing and building the solution on how the human actually performs the task instead of basing the solution merely on external observations.

Neural networks, because of their ability to codify conceptual data and their applicability to humanly intuitive problems, are appropriately being employed in this research for the purpose of navigation. A complete neural network solution is also being sought to determine the extent of network application. Pivotal aspects concerning the use of neural networks is the development of proper network structures and training. These are two key research issues addressed in this text.
2.4.1 Structure

In developing a network structure it was found to be extremely beneficial to appraise the processes of biological systems in terms of both physiology and psychology. Psychology provides for an understanding of processes at the macro or behavioral level where the physiology understanding explains phenomenon at the micro or individual cellular level. The stimulus-response and sensory motor response cycles have their roots in psychological studies. Neural network technology originates in the physiological arena. Using knowledge of these two fields, the approach of this research was to develop cellular structures into functional groups of behavior for navigation.

In developing this system, the physiological structure of the mammalian visual system provides a model for implementing an artificial neural network for the sense-association process. This sub-system, discussed in Chapter Three, maps sensory data to an internal mapping. Appropriate navigation decisions are then made by the response sub-system. For the response, navigation is being considered a behavior. A model of a hierarchy structure of navigational behaviors provides the framework for developing tiered networks within the response sub-system. Final system output is the integration of responses from the network hierarchy. This output dictates the proper motor command for the scene.

A disclaimer is necessary to clarify the limits of the biological analogy. While neural networks are biologically inspired, the extent that they actually replicate the
definitive mechanism of human intelligence is highly debatable. Some of their processes may be identical to that of the biological organism or they may be purely coincidental. For instance, neural networks are said to be able to generalize. This generalization capability is an extremely powerful characteristic and has undoubtedly contributed to the increased interest neural networks are now receiving. But the exhibition of a characteristic does not necessarily mean the processes of neural networks in generalization are identical to the processes of a biological being in generalizing. The intelligent system model provides a means for exhibiting intelligent behavior, but in ways totally unrelated to the processes used by intelligent beings. Likewise, it has yet to be shown, nor may it ever be, that the neural network generalization process is anyhow related to the generalization process of a human. The mathematically proven concept of content addressable memory in neural networks may or may not be how the brain generalizes.

Throughout this text, there will be references to biological systems, structures and observed phenomenon. An attempt will be made to model the artificial system after the biological system whenever possible. There is no attempt made to state unequivocally or even remotely that the structures or systems derived are biological equivalents. It is sometimes difficult to ignore the intriguing similarities between biological systems and neural networks. However, neural networks should not be considered as anything magical but rather a computing tool with interesting exploitable characteristics. Furthermore, there is a plethora of behavioristic research concerning sensor motor response issues and an abundance of physiological information concerning neural processes. For the purpose of
this research, only the basic, well established principles of these areas are used as guidelines in developing the system.

Finally, while the biological and behavioristic models provide a framework network structure, the actual dimensions and parameters for the network must be determined by the researcher. It is in this area that the *art* of neural networks appears. While there are some basic principles for developing network structures, experience and a thorough understanding of network dynamics serves the researcher best. Some of the issues concerning the development of network structure are intimately related to the training of the network.

2.4.2 Training

Neural networks are trained to respond in a desired manner to input. The primary issue concerning training is that the network learns the desired concept. How well a network learns this concept is governed by the quality of both training patterns and training session. In judging a training session, one necessary condition of learning is that the error between network response and desired response converges to an acceptable level. This condition alone is not sufficient to ensure a proper future response. The network must also be able to generalize on the input data. The selection of training patterns in accordance with network specifications is crucial in permitting proper generalization while obtaining convergence.
2.4.2.1 Generalization versus Convergence

Generalization describes a network’s accuracy to properly classifying input. Convergence can be considered the precision of the network in making this classification. In the familiar definition of the terms accuracy and precision, where precision can be obtained for an inaccurate response, convergence is necessary for generalization but does not assure generalization. Networks, capable of converging on input data may not necessarily extract the conceptual property to permit proper generalization. The primary factors affecting convergence and generalization are network size, amount of training data, and the quality of training data. While convergence and generalization are most often associated with back error propagation networks, the concepts of these terms are also applicable other networks. However, for the following discussion they will be discussed in reference to back error propagation networks.

In determining the size of a network, considerations must be made for the input data. For back error propagation networks, Kolomogrov’s [40] mapping theory provides a relationship between the size of the input vector and the size of the network. Kolomogrov’s theory states that to achieve any arbitrary mapping between input and output, the number of hidden nodes should equal 2n+1, where n is the size of the input vector. What is missing from the theory is the relationship between the size of the network and the number of input patterns. For large input vectors, application of this theory could lead to very large networks.
A problem with large networks is that they lose their ability to generalize unless the training set is also very large. In large networks with small training sets, there is a lack of interaction between weights. Certain weights come to represent or be associated with one particular pattern and no others. This association is essentially rote memorizing of a training pattern without extracting any information which will enable the network to generalize. Weights are merely "looking" for their assigned pattern. To overcome this problem, there needs to be a sufficient number of training patterns for the network size. According to Shelton [78] a good rule of thumb is to always have more training pairs than hidden nodes. This rule forces weight sharing for different patterns which ultimately ensures generalization.

In some instances, especially with large video input vectors, it is impractical to generate enough training pairs to meet the size of the network specified by the Kolomogrov's mapping theory and the rule of thumb provided above. In these instances it may be best to reduce the size of the network. Kolomogrov's theorem is for purely arbitrary mappings of input to output. For more structured input, it has been found that the number of hidden nodes can be significantly reduced. Further, by designing network connections in a manner that incorporates problem knowledge (discussed in Chapter Four), network size can also be reduced. Both of these approaches will result in a reduction in the size of training patterns which will assist in the network's effort to generalize.
In seeking generalization, an associated issue is the problem of convergence. Large networks with few training patterns are relatively easy to train. But, for the reasons mentioned above, these networks do a poor job of generalization. Smaller networks with larger training sets generalize better but they are harder to train. For these networks it is sometimes difficult or even impossible for the network to converge. Designers must therefore develop the proper mix of training pattern set with a network size that converges to an acceptable level of error.

2.4.2.2 Development of Training Patterns

Neural networks are pattern classifiers trained to discriminate between inputs. Applying networks to navigation requires that the problem be formulated in a pattern classification manner. Patterns selected for training must contain sufficient information to permit the extraction of the conceptual basis while at the same time being a representational sampling of the application’s input space. These criteria will ensure proper network generalization for the desired application.

Obtaining both convergence and generalization in a network can be facilitated by selecting patterns that exhibit the pivotal features of the problem. In selecting these patterns, the researcher is guiding the learning process along the desired lines of network performance. For example, if the training set consisted of different shaped objects of different color, it would be fruitless to input merely the object shapes when the concept of color is sought. Further, it would be wasteful to input both shapes and color when
only the concept of color is sought. Patterns should be selected that best describe the concept of the problem. Proper generalization and convergence will follow.

In selecting patterns, it is useful to apply the Gibsonian concept of invariants. Gibson [26] is the founder of the ecological approach to perception. This approach, also called direct perception has been receiving a great deal of interest in its application to vision, Albus [1]. The premise of the approach is that perception exists in the environment; the environment provides the clues for analyzing the scene. People extract relevant features from their environment which in essence comprise the perception of the scene. "One of the most important concepts in direct perception theory is the invariant. The emphasis on invariants may be Gibson's single most important contribution to psychology...," Gordon [29]. The basic idea of invariants is that there are some features of the scene that maintain an expected, unchanging relationship. Knowing invariant relationships permits the observer to filter through superfluous input data and extract the specific perception of the scene. In selecting input patterns for neural network training it is therefore appropriate to choose patterns based on an invariant feature. Generalization will then be based on an unchanging invariant feature of the input space.

2.5 Executive Summary and Conclusion

From an understanding of how current navigation systems address the problem, it can be seen that the neural approach is more in accordance with how an intelligent being approaches the problem. Neural networks, by being able to encode conceptual data,
are applicable to the problem of navigation. Network learning of navigational concepts is guided by training and structure. While biological anatomies provide some basis for network structure, dynamics of artificial neural networks must also be considered. It is within this realm that research experience and network understanding prove most useful. Designing networks is always a difficult problem because of the relationship between network size and performance. While the Kolomogrov's mapping theorem provides a starting point, the "rule of thumb" presented in this chapter provides an additional criteria to support proper network generalization.

There are a multitude of interrelated considerations that govern the design of networks. Researchers must find an acceptable mix of network parameters for the availability and size of input patterns. While attempts are made to select the best combination of network dimensions and patterns, obtaining the optimal blend is an elusive concept. In this regard, structures should be deemed adequate when exhibited performance is acceptable. This gauge was used for evaluating the network structures in this research.

The following summarize the issues presented in this chapter:
--As opposed to the intelligent system model, neural networks produce intelligent response more along the lines of how an intelligent being produces these responses.
--Motivation for pursuing the network approach came from the network's ability to encode conceptual data. By learning the concept of navigation, the system could be
applied to a myriad of environmental situations.

--Key issues in developing networks to exhibit a desired performance are those of network dimensions and training data. For back error propagation networks, Kolomogrov's mapping theory and a rule of thumb governing the relation between network size and the number of training patterns provide a starting point. Acceptance of a network structure should be based on issues of performance.

--Network response functions in a sense-association-response cycle. Processes are developed in this research to address these functions by dividing the problem into two groups. Chapter Three discusses the development of the sense-association process and Chapter Four discusses the response.
Chapter Three -- Sensing and Association

The previous chapter provided an overview of the objectives and principles guiding this research. From this overview, it was determined that a neural network navigation system consists of two sub-systems: the sense-association sub-system and the response sub-system. This chapter presents the development of the sense-association or sense-perception sub-system. Research efforts discussed in this chapter describe an artificial neural system that senses the environment, extracts obstacle locations, and then maps these locations for the response sub-system.

3.0 Extraction and Mapping of Environmental Information

For any system or organism to interact with its environment, it must be able to sense or extract environmental features relevant to the desired interaction. There is usually a proportional relationship between the degree of interaction and the sophistication of the sensing apparatus. For navigation, knowing where objects or obstacles are located will enable a system or organism to move through the scene. The pivotal environmental feature for navigation is then the determination of the spatial relationship of objects in the scene. A scanning laser range finder has been selected as the sensing device for this research because of its ability to readily extract this spatial relationship.

Once the environment has been sensed, association is the process of mapping the sensed data to some internal representation to permit a cognitive response. For the task
of low level navigation, the major sensory-perception or sensory-association task is to determine the traversable regions in the input space. Basic reasoning or response decisions are then predicated on answering the question "Can the robot move in a certain direction without running into an object?" Accepting this criteria as the basis for movement, a binary declaration as to the acceptability of a location in the environment is a sufficient representation. From the laser sensory data, a binary determination will be made as to the traversable conditions of the scene.

In deriving the sensory-association sub-system that maps laser sensory data to a binary representation of traversable regions, a study of biologically observed features proved most useful. Biological systems provided the basis and understanding for the development of the artificial system. From knowledge of biological systems it was possible to develop a neural network architecture to realize the desired mapping process. The following section discusses the biological principles germane to this implementation followed by the application of these principles in an artificial neural network.

3.1 Biological Considerations

The need to consider biological systems became apparent from the onset of this research effort. Solutions to two preliminary issues were derived from biological principles. Studies of the human visual system provided an insight into the problems of 1) the feasibility of applying laser data directly to the response sub-system and 2) how to handle the large amount of sensory data. From this initial review of biological
systems, a foundation for the rest of the system was developed.

It was originally hoped that laser data values could be directly applied as input to a neural network. Navigational responses would then be made directly from raw range values without any intermediate processing of range data. While this approach may seem plausible, research determined that a more biologically inspired system could be developed for increased performance and applicability. Applying range measurements directly would require a strict adherence to sensor configuration and sensing conditions. Networks trained to respond to a specific range measurement would be invalid if sensor or vehicle configurations were changed. The system would not be portable in the sense that new training would be required for every new application.

Biological inspiration also provided a means of dealing with the abundance of sensory input data. One of the many problems that has confronted the development of vision systems has been the profusion of data. It is a natural wonder how the human eye can integrate and decipher the extraordinary amount of visual input it is continuously receiving. Machine vision systems seek to overcome this dilemma through increased computing power and advanced algorithms. The biological understanding of the process may provide an insight into how the human visual system handles this profusion of data.

3.1.1 Human Visual System

Environmental visual data is sensed via photons striking retinal cells located within
the eye. The human retina is made up receptor cells called rods and cones depending on their shape. There are approximately 120 million rod shaped cells and 60 million cone shaped cells, Carlson [11]. These retinal cells respond to incoming light by producing a receptor potential that is transmitted down the cells’ axon to ganglion cells. Ganglion cells then transmit their output through the optic nerve to the dorsal lateral geniculate nucleus of the thalamus. From here, signals are then sent to the primary visual cortex located in the most posterior region of the cerebrum, the striate cortex. Of particular interest, and a fact that will be extensively exploited in this research, is that the visual system maintains the spatial organization of visual data completely through to the visual cortex. This is called a retinotopic representation on the cortex and can be considered a topographic mapping of visual data on the cortex, Carlson [11].

The connections between retinal cells and ganglion cells is not one for one. There are varying number of retinal cells that feed into a single ganglion cell. The collection of retinal cells assigned to a particular ganglion cell is called a receptive field. Kuffler [52] determined that these receptive fields are generally circular in shape. From his work and the collection of works presented by Dowling [20], the dynamics of the receptor fields are becoming better understood. A significant finding is that of the on-center off-surround activation characteristic of ganglion cells.

Kuffler [52] found that the output of a receptor field ganglion was highest when light stimulation was directed at the center of the field. When light is directed to the
surrounding retinal cells the activation significantly drops off. Further, output from the receptor field ganglion is highest when there is maximum contrast between a bright center and a darker surrounding area. This observation led to classifying the receptor fields as an on-center off-surround anatomy. There are also off-center on-surround cells which exhibit excitation to light stimulus in the surround cells and inhibition to light stimulus on the center cell. Carlson [11] indicates that these two types of cells are sensitive to different stimuli. The on-center off-surround cells, called X-cells, respond best to continuous stimuli whereas the other cells, Y-cells, respond best to rapidly moving stimuli. Presentation of environmental scenes via the laser scanner, can be considered as a continuous stimulus to the retinal cells. Consequently, this work will focus on the activations of X-cells. Furthermore, all X-Cells terminate in the same region of the visual cortex. More is understood of this particular part of the visual cortex than other regions. This understanding will be beneficial in the development of the neural network navigation system.

3.1.2 Visual Cortex

Through the work of Hubel and Wiesel [42] and von der Malsberg [84], a basic understanding of the organization and function of the mammalian visual cortex can be derived. Hubel and Wiesel determined that there are six layers (with additional sub-layers) in the primary visual cortex. All X-cells terminate at level IVc. At this level Hubel and Wiesel discovered the existence of orientation selective cells. When visual input indicates the existence of a particular orientation in the scene, a cell associated with
that orientation becomes active. These cells are called simple cells as they only respond to one type of stimuli. Von der Malsberg determined a relationship between the excitatory and inhibitory connections within cells of the visual cortex which leads to the formation of groups or clusters of cells. Within these clusters, several different orientation selective cells would compete with one another to represent the visual input received by the cluster. For each receptor field there is only one winning cell and therefore only one feature extracted. Hubel and Wiesel stress that the response of the cell in the visual cortex is purely a local process. Input to a cluster is provided only by its associated receptor field. Thus only a portion of the entire visual input is presented to a cluster.

While the preceding may account for the cell dynamics which occur in the visual cortex, it fails to explain the extraction of what was perceived or "seen" in the input. One explanation is that perception is the process of piecing together the winning cells of each cluster. Winners represent a feature extracted for each retinal patch. By maintaining the spatial relationship between patches and cluster output, subsequent cortical layers may piece together features in order to perceive the sensed image. Cells

![Figure 3.1 Perception by Feature Extraction of Retinal Patches](image-url)
of several subsequent cortical layers are called complex cells. These complex cells have a receptor field of simple cells feeding into them. By combining simple cell output with respect to their location, the sense of the scene is developed in the complex cell. For example, in Figure 3.1, the scene of a window is presented. Retinal patches provide input to their associated clusters. Each cluster consists of orientation selective cells for horizontal, vertical or corner orientations. These cells compete within the cluster to represent the retinal patch. Cluster outputs are then combined to provide a sense or perception of the scene. If the scene were changed to that of a door, this same network would still extract vertical, horizontal and corner features but their spatial relationship in the perceived image would be different. This difference would be the key in distinguishing between a door and a window.

It must be noted that the above example unduly simplifies the perception process. There is strong evidence that the neurons respond to spatial frequency instead of the example’s lines and edges. Spatial frequency is defined as the visual stimulus’ cycle per degree of visual angle. Many researchers (see Carlson [11]) believe that the visual cortex performs some Fourier analysis on the visual input’s spatial frequency. The process is extremely complicated and much remains a mystery. For the current application it is possible to accept the basic phenomenon presented by Hubel and Wiesel.

In closing this discussion of the human visual system it must be noted that the descriptions given above are anything but complete. There are numerous other
connections between retinal cells, ganglion cells, and other portions of the brain than those discussed. There are also three different types of ganglion which are sensitive to different types of inputs. There is an enormous store of medical and physiological research into the human visual system which is far too great and far too detailed to be presented here. What has been presented is merely the major components and primary processes of a vast and complex system. No attempt has been made to delete any particular aspect or overly emphasize a particular aspect in order to support the artificial anatomy that is to be developed.

3.1.2.1 Invariant Feature Extraction

In developing a sense or perception of the environment, the visual system must clearly extract distinguishable, unmistakable features that define the scene. Recalling the concept of invariant features, the objective of this process would then be to extract the invariant features of the image. A possible invariant feature, applicable to navigation, that exists in the environment, is the relative change in elevation between points. Almost assuredly gradient features guide humans in foot placement and route selection. Using the laser scanner and processes developed later, this invariant feature will provide the basis for perceiving the environment. Feature extractors of the visual cortex will be designed to detect this invariant feature.

3.1.3 Functional Characteristics of Visual System

Briefly diverting attention from the physiological aspects of the visual system, it
is useful to note some functional characteristics. An interesting phenomenon of the retinal cells is their ability to sense both minute and intense light stimulus. The response of the receptor cells to this wide range of inputs is nonlinear with increased sensitivity to lower stimulus. A consequence is that cells are more sensitive in detecting relative changes in intensity than in determine absolute levels of intensity. A common example of this phenomenon is that of brightness consistency.

Brightness consistency is a common human experience. A piece of white paper will appear brighter than a grey sheet of paper in either bright or dim lighting conditions. What is being detected is the relative disparity between the two sheets of paper. The converse to brightness consistency is brightness contrast. Brightness contrast is observed when one portion of the input becomes darker the other portion appears lighter. Again this phenomenon can be attributed to the relative changes in the input. These two traits will be discussed later during the discussion of networks that simulate retinal cell propagation.

3.2 Component Development

The foregoing brief description of the processes of the visual cortex provides the basis for the neural sensing and associating system developed here. Scanning laser range data provide the visual input of the environment. Retinal patches, or receptor fields, respond to the incoming visual data and feed forward their output to their respective cluster. Cells of the cluster become active when they see their associated feature. The
most active cell of the cluster, wins out over the other cells and its associated feature is ascribed to the retinal patch. This feature should be extracted based on an invariant in the input data to avoid any misinterpretation. In developing a binary representation of the environment for navigation, only the feature of open or blocked needs to be extracted for each receptor field. Winning cluster features are then combined by subsequent layers of neurons which form the basis of perception. Throughout this process the spatial relationship between visual data, retinal patches, clusters, and output are maintained. Figure 3.2 is a diagram of the biologically inspired structure for the navigation. The following sections describe the development and implementation of this process.

3.2.1 Laser Scanner

The sensing device can be thought of as the eye of the system. While there is no correlation between the scanning laser range finder and the biological eye, the laser range finder does provide the environmental information necessary for low level navigation.
A vision system may more accurately replicate the biological eye but it does a poor job in extracting range information. This shortcoming is one of the major disadvantages of vision systems. Efficient and practical methods have not yet been developed for the extraction of range information from 2-dimensional or even stereo vision images. In seeking a biological solution to this problem, there are some difficult and yet unresolved issues as to how a biological system accomplishes this task. For this reason it was determined that it was appropriate to employ a sensor that provides the information directly.

Scanning laser range finders are a relatively new technology with development first beginning in 1977, Nitzan [66]. While there are several different types of laser systems capable of producing range information of a scene, the scanning laser range finder is becoming dominant in the field of navigation. Within the field of scanning laser range finders, the most common type used for navigation measures distance through the time of flight phase shift distance measurement technique.¹

Figure 3.3 depicts the internal configuration of a typical scanning laser range finder. A transmitter produces a modulated laser light source which is split into two signals. One signal is projected onto the environment by a rotating scanning mirror. This signal is reflected back from the environment and sensed by the receiver. The second

¹For a discussion of other types of laser range finders, the reader is directed to a previous work by the author, Weiland [87].
signal acts as a reference beam and is provided directly to the receiver to be used for comparison with the returned beam. Phase difference between these two signals is proportional to the distance. The amount of returned light is also converted into a sinusoidal current at the modulation frequency. Amplitude of this current is proportional to the reflectance of the object. These laser systems produce a two-dimensional array of data for each sensed point of the environment. Each datum point is two bytes long. One byte represents a reflectance value based on the amount of laser light reflected from the object. The other byte is the range value.² This research is only concerned with the range data and the reflectance data is ignored.

²Vision systems commonly refer to a single datum point as a pixel for picture element. For range elements, a single point is occasionally called a range for range element. Often, a range is referred to as a pixel because of the similarity in the presentation of the data and the term pixel is more widely known.
There are two scanners of this type commercially available for research. One is made by the Environmental Research Institute of Michigan (ERIM) and the other by Odetics. The characteristics of each are summarized from Dunlay [21] and Odetics [70] as follows:

<table>
<thead>
<tr>
<th></th>
<th>Field of View</th>
<th>Output</th>
<th>Resolution</th>
<th>Ambiguity Interval</th>
<th>Frame Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERIM</td>
<td>30° *</td>
<td>80° *</td>
<td>64x256</td>
<td>3.0</td>
<td>64.0</td>
</tr>
<tr>
<td>ODETICS</td>
<td>60° *</td>
<td>60° *</td>
<td>128x128</td>
<td>1.44</td>
<td>32.74</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(.72 in 9 bit mode)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Scanning lasers are very appealing devices for quickly obtaining range information. Their capabilities in providing an eight bit distance measure of 16,384 points in less than a second is highly attractive. However, there are some problems in the quality of the data that must be considered. Problems of specular surface errors, longer frame capture time (compared to vision), noise, parallax, footprint averaging, and range ambiguity must be considered. For a more detailed discussion of these issues, the reader is directed to an earlier work by this author [87] and Jain [43]. While these issues must be considered, a larger issue that hampers actual implementation in this research is the prohibitive cost of the systems. These systems cost well over $100,000. For this reason, a simulation program was developed.

3.2.1.1 LASER IMAGING SIMULATION ALGORITHM (LISA)

Due to the prohibited cost of a scanning laser range finder, the system has been simulated in software. LISA, which stands for Laser Imaging Simulation Algorithm has
been developed at Rice University to emulate scanning laser range finding systems, Wu [90]. The simulation program was developed as an aid in developing laser image processing algorithms. It enables users to identify both environmental and internal laser system configurations. System parameters can be set to mimic the operation of the two actual sensors described above. For this research, emulation of an Odetics scanner was chosen because of potential interface with NASA/Johnson Space Center.³

Environmental conditions are considered in LISA through the development of an artificial robot world. The robot world for this research consists of a hexagonal track that is bounded by a one foot high wall along the exterior and a one foot deep ditch on the interior. Along the track there are several obstacles randomly placed. Figure 3.4 shows a typical wire frame image of the robot world. This image was obtained by configuring the sensor at a height of six feet with a downward tilt of 60 degrees. In this configuration range measurements begin at the robot's toe and encompass a 60° viewing area both horizontally and vertically. Several subsequent references will be made to the image of Figure 3.4.

³NASA/Johnson uses the Odetics scanner in the robotic Extravehicular Activity Retriever robot (EVA Retriever).
From this image, the simulation program calculates and writes the appropriate range measurements to a file which will be read by the sensing system.

Noise in laser images is associated with the internal configuration of the scanning laser sensor. LISA allows the user to include noise in the output image to test the robustness of routines in less than ideal conditions. This feature will be used in evaluating the performance of the navigation sensing system.

3.2.2 Retinal Patches

Retinal patches, or receptor fields, of the biological system extract a portion of the complete visual input. Local operations are then invoked to determine the principle feature of each patch. This concept can be applied to extract obstacle information for particular segments of the laser image. By sub-dividing the 128x128 input space into smaller retinal patches, obstacle states can be determined for different regions. Since the spatial organization of retinal patches is maintain throughout the vision process, there is a direct relationship between the associated mapping on the visual cortex and environmental conditions.

The size and number of retinal patches will dictate the resolution of the scene in the visual cortex. Although retinal patches will be classified as either traversable or not traversable, they may actually include regions of both classification. Any combination of classification within a patch will necessitate a declaration of non-traversable for the
entire patch. With large patches, this classification may overly restrict movement. To combat this inhibiting effect, patch size and arrangement should consider both the expected environmental conditions and robotic vehicle dimensions.

In developing the navigation system it was determined that a retinal patch size of 32x32 was adequate. For the sensor configuration presented above, this patch size equated to an environmental measurement of approximately one foot by one foot at the toe of the robot. Obviously at greater distances, a patch will represent a greater environmental area. This non-linear relationship is unavoidable but may be accounted for in the training of the network. Resolution of the mapping can be improved by shifting and overlapping the receptor fields. The desired effect of shifting is to locate all open areas of a size equal to one retinal patch. It may be possible to separate open from closed regions which jointly exist in a single patch by using this strategy. Again, the size of the patch will determine the extent of vehicular free space between obstacles. Figure 3.5 shows the effect of increasing the resolution of the image by overlapping the retinal patches.

If a 32x32 retinal patch is applied to the laser image with no overlap,
there will be four retinal patches per side for a total of sixteen regions in the mapping. If an overlap of 16 rangel is used, a resulting visual cortex mapping of 7x7 is achieved by adding a patch between each of initial four patches per side. There are limitless permutations and combinations of retinal patch size and degree of overlap. Optimal configuration is an elusive abstraction in this context and validating a particular structure size is not intended. The desired navigational performance can be obtained by training appropriate responses on this 7x7 representation. While the proceeding chapter discusses the training process, some training issues concerning this representation need to be presented here.

Robot dimensions are provided for in the retinal patch representation by selecting patch dimensions that will accommodate the size of the robot. In subsequent training, the robot will learn how to respond to the physical interpretation of this mapping. Training will relate open retinal patch elements with permissible regions of travel. Conversely, obstacle dimensions can be considered as the composite of the non-traversable patches. Improving the resolution will improve the selectivity of movements for a particular scene. While it may be impractical to obtain a high resolution for an entire image, there may be merit in obtaining a high resolution for foot placement operations.

3.2.2.1 Foot Placement-Robot Dimensions

When a human is walking to some destination he is moving with some goal in mind. Recalling the navigation hierarchy presented in Chapter 2, each step, while in
concert with the overall goal, must also avoid obstacles. During robotic movement, each iteration can be considered as the next foot placement process. With retinal patch size dictating resolution, the patch size should accommodate the robot. Thus open patches to the robot's immediate front would be considered acceptable foot placement locations. As the resolution increases, the possibilities of foot placements increase. The greatest resolution could be obtained when patch size corresponds to the robot dimensions and is shifted over by only one pixel across the input space. Choosing the next foot placement then becomes more of an analog choice rather then the selection of a discrete movement command to a retinal block. While this approach is completely feasible it was not incorporated in the final product. Foot placement will be confined to the center five retinal patches of the cortical image. These represent one foot square overlapping spaces spanning a three foot front centered on the robot. It is assumed that the system could be easily expandable to incorporate the improved resolution for foot placement. However, it is probably undesirable to apply this concept over the entire visual input due to the tremendous increase in the number of output nodes that will result.

3.3 Implementation of Sensing-Association System

Based on the preceding discussion of physiological evidence and artificial implementation, the specifics of the sensing system can now be established. The scanning laser range finder, simulated through the LISA program, provides the visual data of the scene. These range measurements will be extracted in 32x32 retinal patches and processed to determine their traversable condition. Retinal patches overlap in order to
increase the resolution of the image. Output from the sensing process will be the association of visual information onto the visual cortex representation. Spatial relationship between the original input and the visual cortex image is maintained. Figure 3.6 shows this basic structure.

In developing neural networks for the propagation of signals in the structure of Figure 3.6, several different approaches were considered. From previous research, Weiland [87], procedures were established that would permit the image of the visual cortex to be developed in a non-neural network manner. One approach would be to convert the laser data to a top down Cartesian map. Elevations from the Cartesian map which exceed a threshold could be declared as obstacle regions. Another approach would be to subtract from the input range values a second set of range values obtained from an obstacle free image. Values remaining which exceed a threshold indicate the presence of some object within a retinal area. These locations could then be mapped to the visual cortex map directly. While this first method has proven successful in other applications
(see Weiland [87]) there is an associated high computational cost. The second approach actually appears promising since it quickly determines regions of obstacles and can be mapped in retinal patch size elements. However, both of these conversion procedures require known and rigidly set sensor configurations. As with the direct application of sensory data discussed earlier, these approaches overly restrict the application of the system due to their reliance on a set configuration.

A neural network solution was sought in order to overcome the rigid requirements of these more traditional approaches. The network approach seeks to map the organizational relationship of obstacles onto the visual cortex without defining a specific scale for the mapping. While the sense of absolute distance measurements may be lost, spatial relations are retrievable in the interpretation of patches in the visual cortex. This approach seems feasible in that it is much easier for a human to describe the spatial relationship among objects than it is to determine absolute distances to objects. Studies by Kubie, Muller and Ranck [51] on the functioning of rat hippocampal cells indicate that distance is probably not used in biological systems for determining locations. Some other means is used to describe
spatial relationships besides distance. A possible approach is presented by both Zisper [91] and Gibson [27] that is based on the subtended arc of the retina caused by the projection of image obstacles. This angle is shown in Figure 3.7.

Sensing an object results in a retinal mapping of the object. The subtended angle of the retina image is proportional to the distance that the object is located from the individual. If "Object A" of Figure 3.7 moves closer to the viewer, this angle increases. As the angle increases, the size of the mapping onto the visual cortex, caused by "Object A" increases which provides the individual with a sense of distance to the object. An interesting dilemma occurs as shown in Figure 3.7 when two objects of different size result in the same retinal mapping. In this instance, see "Object B", the viewer cannot distinguish which is closer. While absolute distance measures of the scanning laser would be able to discern this difference, there is little biological support for the approach.

In order to solve the sensing problem with neural networks it was necessary to reflect on the processes of the biological system. Networks which propagate the input signal in the receptor field must exhibit the characteristics of brightness consistency and brightness contrast. They must be able to operate in a wide dynamic range. Biological evidence indicated that an on-center off-surround anatomy was desirable. Two such networks, based on this anatomy, were pursued in this research.
3.3.1 Shunting Equations

The basic on-center off-surround anatomy is shown in Figure 3.7a. Grossberg [34] has developed a network based on this anatomy for short term memory which has also been shown to obey membrane equations of biological cells. These networks have exhibited some interesting characteristics when analyzing processes of perception, conditioning and cognitive information processing. Applied to the sensing problem, they exhibit the many desirable characteristics such as conservation or normalization of total activity which permits an infinite dynamic range of input. Weber law modulation, which is related to contrast enhancement and brightness consistency, is also characteristic of these networks. Explanations for these features and how they are exhibited can be shown through an analysis of network dynamics. Propagation of cellular activities within the network are defined by the following equation:

\[ x_i = -Ax_i + (B-x_i)(w_if(x_i) + I) - (x_i + D)(\sum_j w_ig(x_j) + I). \]  

Eqn 3.1
where:
\[ x_i \] is the cellular activation level for \( i \)th cell
\[ A \] provides for short term memory decay
\[ B \] and \( D \) restrict activations between a finite range and will be discussed in greater detail later.
\[ I_i \] is the input to cell \( i \)
\[ I_j \] is the input to adjacent cells which will have an inhibitory affect on cell \( i \)
\( f(x_i) \) and \( g(x_i) \) provide excitatory and inhibitory feedback respectively, which is controlled by a weight matrix \( w_{ij} \)

Terms \( B \) and \( D \) are called the shunting terms. Their name is derived from their function in that they shunt or restrict the output of a cell to a workable range of between \( B \) and \( -D \). This permits an infinite dynamic range in the input space but maintains the output within a finite limit. Biological cells function in this manner by accepting a wide range of inputs but output is still constrained to a small working range. Figure 3.7b shows the same on-center off-surround anatomy with feedback or reverberation loops in the network. These feedback loops, identified by the function terms, \( f(x) \), \( g(x) \) in Eqn 3.1, provide for short term memory storage dynamics. For this discussion, feedback will not be included.

With constant input and no feedback, Eqn 3.1 can be rewritten as:

\[
x_t = -Ax_t + (B-x_t)I_t - (D+x_t)\sum_{j}f_j
\]

Eqn 3.2
Solving Eqn 3.2 for steady conditions gives:

\[ x_i(\infty) = (B+D) \frac{I}{(A+I)} \left( \theta_i - \frac{D}{(B+D)} \right) \quad \text{Eqn 3.3} \]

where \( \theta_i \) is the relative input to cell \( x_i \) calculated from \( I/I \).

Activation of cell \( x_i \) is proportional to the relative input to cell \( x_i \) regardless of the total input \( I \). If the total activity is calculated from Eqn 3.3, by summing activities \( x_n \), it can be seen that the total activity is also independent of the number of cells. There is a normalization of activity based on the total amount of input across the network. Thus, as one portion of the input becomes lighter, the other portion will become relatively darker which explains the characteristic of brightness consistency. "The brightness of an object depends not upon the intensity of light falling on the retinal image of that point but rather the relations among the intensities with the overall image," Cornsweet [16]. This normalization feature results in individual cellular activations being based on the relative input to the cell as opposed to the absolute input.

Weber's law explains the perceptual differences in intensities between two objects, see Cornsweet [16]. With small overall intensities, slight changes in intensity are recognizable whereas only large changes are recognized when intensities are high. In Eqn 3.3 the term \( I/(A+I) \) provides for this physically observed phenomenon. Further, for an object to appear twice as intense as it was before, the difference between the two
intensities must be twice as great. This rule can be summarized by stating that the change in intensities divided by the total intensity is equal to a constant. Thus, when the overall intensity is high, large changes are required in order to maintain the relative intensity difference between two objects.

In the expression \( \theta_i - D/(B+D) \) the second term is sometimes called the adaptability level. By adjusting the adaptability level, it is possible to threshold activations of cells based on their relative input. If this term is set to \( 1/n \), where \( n \) is the number of nodes in the retinal patch, then a uniform input will result in no cellular activation; the relative input to each cell, \( \theta_i \), will cancel the adaptation level term. This feature is extremely useful in vision applications as it permits the extraction of a figure from the background provided the figure is defined by darker pixel values than the background values.

In vision applications, shunting equation networks exhibit many of the qualities displayed in the visual system. How these networks will respond to laser data is a novel research topic pursued here. To the best knowledge of the author, there have been no other attempts at using shunting networks for the processing of range information. In analyzing how the network will respond to laser data, Figure 3.9 was used to illustrate a fictional sample of laser range data. Each section of Figure 3.9 can be considered as contiguous retinal patches. In the first and third patch, the scene is open and clear. Table
3.1 indicates the range measurement, the steady state cell activation from Eqn 3.3 (adaptation level of \(1/n\)), and the relative input of each distance measure for the patches of Figure 3.9. By comparing the first and third patch, it is noted that little information can be extracted by relying on the steady state activation levels of the cells. These two
patches both represent open space but final cell activity varies with the magnitude of the total input. A discrimination between patches can be made by permitting the cells to take on a binary representation with negative final activations being "0" or (-) and positive activations being "1" or (+). For clear open space, the transition zone between negative and positive activations will occur in a predictable manner as shown in the first and third patches. Obstacles within the field of view will disrupt this transition zone as shown in the second patch. This analysis was then applied to a complete retinal patch size element.

A shunting equation network was developed in software for the structure of the retinal patches discussed earlier. Direct laser range measurements were provided as input. Output from the network were converted to the binary (+,-) representation. Figure 3.10 shows the difference in retinal patch number 24 between an open scene (Figure 3.10 a.) and the simulated robot scene of Figure 3.4 (Figure 3.10 b.). The right side image shows the characteristic curvature in the transition caused by the radial measurement technique of the laser scanner. This transition is clearly disrupted by the presence of a box edge in the left image. It is evident that the earlier analysis of transition zones in laser image processing is expandable to the complete retinal patch. There is a clear and discernible difference in the two images of Figure 3.10.

Besides being able to distinguish clear space from obstacles, the shunting equation network exhibits the additional beneficial attribute of being relatively independent of sensor configuration. Changes in configurations do not radically affect the appearance
of the output. Sensor height and angular measures were adjusted in the LISA program with the output passed through the same shunting neural network. With minor variations, the transition zones in clear space patches were the same. Obstacles were in different locations because of the different spatial arrangement, but clear space could be readily identified by the characteristic transition.

After shunting equation processing, each retinal patch must be classified as to its traversable state. From Figure 3.10 it appears to be a simple task for humans to determine which patch is clear and which is not. Developing a network to perform this same classification did not prove to be so simple. Working with back propagation networks, limited success was achieved in classifying the patches. Numerous training schemes were attempted for a plethora of network structures. The best performing network only obtained an 80% correct classification rate for determining the proper
obstacle conditions of retinal patches. The main difficulty appeared to be a lack of sufficient training data for these networks. Open area training data was easily obtained. Likewise, completely blocked conditions or vertical obstacles could easily be obtained from the analysis of Figure 3.11a. However, there are undoubtedly other environmental conditions than horizontal or vertical surfaces. Attempts were made to develop additional training patterns for a greater array of obstacle conditions. While developing these training patterns a disturbing observation was made concerning the manner in which the shunting equation classifies regions.

![Figure 3.11 Shunting Equation Processing](image)

**Figure 3.11 Shunting Equation Processing** a) Vertical Surface b) Sloped Surface

**TABLE 3.2 Shunting Equation Processing for Figure 3.11**

<table>
<thead>
<tr>
<th></th>
<th>Figure 3.11 a)</th>
<th>Figure 3.11 b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>42 39 36 34 33</td>
<td>42 42 43 44 45</td>
</tr>
<tr>
<td>Theta</td>
<td>.22 .21 .19 .18 .17</td>
<td>.19 .19 .19 .20 .20</td>
</tr>
<tr>
<td>Binary</td>
<td>+ + - - - - - - + +</td>
<td></td>
</tr>
</tbody>
</table>
Figure 3.11b shows the output for a severely sloped surface. The binary output for this patch is identical to the open patch. The network cannot distinguish between these two conditions where one is a clear obstacle condition. Use of the transition zone approach is not as robust as initially thought. As long as the monotonic progression of range measures was maintained, the transition zones would occur predictably. What would ideally be needed is an adaptation level that would extract only the presence of obstacles from the scene and not merely transition zones. For a completely flat scene, the output should be comparable to the filtering of uniform patterns i.e. $\theta_i = D/(B+D)$. Obstacles or disturbances in the range image could then be detected since they would fluctuate from the uniform flat pattern and stand out from a uniform background. In essence, this approach would be detecting relative range changes between range measures. A network of this type could not be developed with shunting equations but was obtained in a cellular neural network.

3.3.2 Cellular Neural Networks

Cellular neural networks are a recent development introduced in 1988 by Leon O. Chua and Lin Yang [14]. Inspiring their evolution was the desire for real time neural network parallel processing for vision applications. Cells of the network are based on electric circuit components as opposed to biological specifics. Like neural networks, cellular networks rely on the interaction or connections between cells for their computing strength. Cells are arranged in a two dimensional fashion and are connected only to their nearest neighbor. As with the shunting equation networks, these networks are short term
memory networks as their connections do not have Hebbian type adjustable weight, see Appendix A. Connection strengths are governed by static weights assigned by a cloning template. There are both feedforward and feedback cloning templates. By adjusting the weights of the cloning templates and internal circuit parameters, different features of the input can be extracted.

Figure 3.12 shows the electric circuit layout of a typical cell in a cellular neural network. Each cell is composed of a nonlinear voltage-controlled current source $I$, a linear capacitor $C$, and a linear resistive circuit element $R$. A fully connected network is shown in Figure 3.13. Interaction between cells is conveniently described in terms of cloning templates. Cloning templates are positioned over individual cells and
control the interaction strengths of neighboring cells with the center cell. Cell activity is calculated by summing the output of all cells within the template area multiplied by the template strength. Propagation in the net is achieved by shifting the template across the entire network and calculating new values for the center cell. Typical feedback and feedforward templates are shown in Figure 3.14. Use of these specific templates will be discussed later.

From the electric circuit diagram of Figure 3.12 and Kirkoff's current and voltage laws, circuit equations for a cell can be developed. From these basic equations, Chua and Yang [13] and [14] have developed mathematical proofs for the dynamic range and stability of these networks. Extracted from their analysis are the following equations that are necessary for computer programming of the network. Terms of the equations are explained in the subsequent paragraph.

\[
\frac{dv_{xy}(t)}{dt} = -\frac{1}{RC}v_{xy}(t) + f_y(t) + g_y(u) + \frac{I}{C}
\]

Eqn 3.4
where

\[ f_y(t) = \frac{1}{C} \sum_{c(k,l) \in N_y} A(i,j;k,l) v_{yk}(t) \]  
Eqn 3.4(a)

\[ g_y(u) = \frac{1}{C} \sum_{c(k,l) \in N_y} B(i,j;k,l) v_{yk} \]  
Eqn 3.4(b)

\[ v_{yk}(t) = \frac{1}{2} (v_{yk}(t) + 1 - v_{yk}(t) - 1) \]  
Eqn 3.5

The indices \(ij\) refer to the cloning template where values are within the neighborhood \(N_y\). Indices \(k,l\) refer to the cell position \(c(k,l)\) within the network. Equations 3.4a and 3.4b are the mathematical descriptions for the effects of the feedback and feedforward templates respectively. \(A\) and \(B\) refer to values within the feedback and feedforward templates that control the amount of interaction. The feedback template obtains its input through adjacent cells, \(v_{yk}(t)\), where the feedforward operator obtains its input through the external input \(v_{yk}\). These feedforward, feedback features are similar to the those found in the shunting equation of Eqn 3.1. Individual cells will have an analog activation level, \(v_{xij}\), but Eqn 3.5 provides the network with binary output. This equation is also used to calculate the input of adjacent cells for the purposes of feedback in Eqn 3.4(b), i.e. \(v_{yk}(t)\).
The feedforward template of Figure 3.14 imparts the desirable characteristic of an on-center off-surround network structure. By applying the weights of this template to an input vector, the center cell's value will be normalized in relation to its neighbors. As with shunting equations, this on-center off-surround normalizing feature permits an infinite dynamic range of inputs. Further, Chua and Yang [13] have shown that the output will also be bounded within a finite working range. Thus, another similarity to both shunting equations and biological cells is observed. The feedback operator specifies cellular reaction to the activations of neighboring cells. In the earlier shunting equation analysis, feedback was not considered as it was germane only to short term memory dynamics. In cellular neural networks, the feedback operators have an edge detection and noise removal capability that are particularly beneficial for our application for navigation sensing.

Chua and Yang [13] [14] provide several examples for how feedforward and feedback templates can be applied to vision images for edge detection and noise removal. One of their examples uses the feedback operator of Figure 3.15 a). If this operator is applied to Equation 3.4 with no feedforward operator, and $C=10^3$, $R_x=10^3$, $I=0$, Equation 3.4 can be written out in long form as:
\[
\frac{dv_{xib}(t)}{dt} = -10^6[-v_{xib}(t) + v_{yib-1}(t) + 2v_{yib}(t) + v_{yib+1}(t)]
\]

Eqn 3.6

Applying the input of Figure 3.15 b), the feedforward operator of Figure 3.15 a) has the ability to detect the existence of horizontal edges in the input space. After network propagation, edges will be detected as indicated in Figure 3.15 c). If a vertical edge detector, defined by rotating the previous template 90 degrees, is applied to the same image, no vertical edges will be detected. However, the sensitivity of this edge detection feature may be adjusted through the selection of template parameters and the external current source that may extract some vertical features. Where the art of neural networks may reside in the selection of training patterns and network parameters, the art of cellular neural networks involves the selection of cloning templates and current source values. Selections of these parameters also affect the inherent noise removal capability of the network. Noise removal performance is most easily described through a visual inspection of the before and after pictures presented in Chua and Yang [14]. In addition to their examples, an example of this feature for use with laser range data will be presented later in this thesis.

Adjustment of template values and circuit parameters affect network output. Determining the parameters to suit a particular application is a difficult process. Chua and Yang [14] admit that "How to choose the filter parameters to achieve a desired image transformation is currently still an active research problem". They present several
different templates in their work and establish some general principles for their use. From their work and experimentation with several combinations of parameters and templates, the templates of Figure 3.14 were developed and employed in this research for the purpose of determining obstacle edges in range data. From knowledge of these edge positions, the location of obstacles in the scene can then be directly determined. These locations will then be mapped to a binary representation for the response system.

3.3.2.1 Edge Detection

The edge detection process discussed here is not to be confused with edge detection for object and shape recognition. As mentioned in the discussion of the navigation hierarchy, extracting object identifying features is a higher level navigation procedure. What is desirable for the current application is the detection of a change in range measurements that exceeds a robotic vehicle’s ability to traverse. Objects which are obstacles within a scene will cause a greater discontinuity in range measurements than free space. This discontinuity is often called a jump edge and is an invariant feature which has been exploited for determining obstacle locations, Weiland [87], Dunlay [21].

Jump edge detection is unlike typical vision edge detection. Vision systems routinely determine edges by calculating neighborhood gradients and then comparing the value to a threshold. In ranging systems, there is an expected greater change between adjacent range measurements at greater distances. Therefore, to determine if a discontinuity exists between two range points, the range difference of the points must be
divided by the distance to the candidate edge point. This ratio is then used in comparison to a threshold. Since the range measurements were normalized in the cellular neural network by the feedforward template, local processing by the feedback template would automatically include the dynamic ratioing aspect. By adjusting the current source $I$ of the network, the sensitivity of edge detection can be changed in much the same way as adjusting a threshold in traditional means.

### 3.3.3 Cellular Neural Network Performance

A computer program was written to implement a cellular neural network. Source code for this program is located in Appendix B. Network performance was first validated by replicating the examples of Chua and Yang that were discussed earlier. The network was used to replaced the shunting equation network in the structure of Figure 3.6. Again, to the best knowledge of this author, this is the only application of cellular neural networks for the processing of range data or use for navigation.

Simulated laser range data generated for the scene of Figure 3.4 was applied to the network. The left half of Figure 3.16

![Figure 3.16 Output of Cellular Neural Network](image)

(left) traditional edge detection, right) cellular neural network edge detection
extracts objects edges of the image using traditional methods developed by the author, [87]. A threshold of a 6% difference between adjacent range measures was used for declaring edge points. The right half of Figure 3.16 is the same image with edges extracted using the programmed cellular neural net with \( I=3.0 \), and the templates of Figure 3.14. In keeping with the navigation sensing model developed earlier, the cellular network processed information as separate receptor field inputs of 32x32. Retinal patch overlap was removed from Figure 3.16 to provide a means for comparing the two methods. It is clear from Figure 3.16 that the cellular neural network is equally capable of jump edge detection as compared to traditional approaches.

While there are many traditional methods of edge detection, see Fu [24], there are some significant advantages of the cellular neural network approach. First, cellular neural networks were designed for direct hardware implementation. For the processes described above, it would only take 5 msec to process an image. Further, because the network processes the image in parallel, processing time is also independent of image size.

Second, cellular neural networks also have an inherent noise removal property. If noise is added to the range values of Figure 3.4, the output for the same parameter settings used before will result in the image of Figure 3.17. From the figure it can be noted that this noise removal feature is more effective at closer ranges. While a noise removal process that cleans the entire image may be more desireable, these closer objects will have a more immediate affect on navigation decisions. Further, removing these
closer noisy points comes with no additional computational cost.

Third, detecting obstacles through jump edges eliminates a dependence upon rigid sensor configurations. Range discontinuities are an invariant feature that are not dependent on sensor configurations. Measurements are based solely on the physical relationship between objects in the scene. Therefore, the obstacles' edges will be extracted from any laser image regardless of sensor orientation.

Finally, while other jump edge detectors may provide similar output to that of the cellular network, an objective of this research is to explore the applicability and limits of the neural approach to navigation. Use of a cellular neural network is in accordance with this approach. Further, the cellular network exhibits some biologically observed characteristics. The network has an on-center off-surround structure through its feedforward template. This permits a wide dynamic range of output. Output is confined to a workable region as with biological cells. If liberties are taken in forming the biological analogy, output could be considered as the ganglion output which is transmitted
down the optic nerve to the visual cortex. From these outputs which indicate the existence or absence of obstacles, feature extractors can then determine whether the various retinal patches are traversable.

3.4 Classifying Regions

In comparing the outputs of the shunting equation and the cellular neural networks, one may think that the same problem of interpreting the output may arise. In the shunting equation no easy or workable method of classifying retinal patch regions was found because of the difficulty in training a classification network to meet all obstacle conditions. Now it appears that the same problem may arise in analyzing edge data of the cellular network. Fortunately, this is not the case as edge detection in range data is an invariant feature. Ergo, edge points indicate the existence of an object. Since the edge detector is adjustable, the declaration as to what constitutes an obstacle or edge point can be made in accordance with vehicular constraints.

After propagation in the cellular neural network, cells have an output of "1" for traversable background and "-1" for edge points. It must be noted that with this representation not all traversable areas are accessible. For instance, if the scene contained a desk, the transition between floor, desk and background would be detected as edge points which would outline the desk. However, the desk top and sides would be declared as a traversable areas. While this seems like an erroneous classification the solution lies in the interpretation. It must be remembered that these interior regions are surrounded
by obstacles; edge points. Thus the apparent free space of the desk top and sides are inaccessible as they are \textit{protected} by obstacles.

Processing of sensory data in the cellular neural network can be considered analogous to the response of retinal cells to incoming stimulus. For each retinal patch the feature of traversability must now be extracted in much the same way as cells within a cluster compete to represent the input. An additional process needs to be developed to classify retinal patches as to their traversability. This process can be considered that of the simple cells of the visual cortex.

3.4.1 Classifying Network

Classifying retinal patches is most easily accomplished through a simple thresholding network similar in design to the original McCulloch-Pitts model, see Appendix A. The minimal anatomy for classifying input into one of two categories, is a single cell with binary output. All cell activations in a patch are fed forward to the single classifying neuron. Instead of training the connections of this network, connections
can be set to "1". Thus, when the cells are summed at the classifying neuron, a threshold can be set to fire the cell when a certain number of edge points are detected. A singular edge point within a patch represents an object of approximately .25 inches in width. Under probable environmental conditions, this point is probably noise and does not represent a physical object. The threshold of the classifying neuron therefore provides for some additional noise filtering. Retinal patch regions of Figure 3.16 were mapped with the above network using a threshold of greater than three edge points being declared as a non traversable region. The output from this network is shown in Figure 3.18.

The resolution of retinal patch sized elements can be seen in Figure 3.18. At greater distances, these patches encompass a greater area which increases the likelihood that some obstacle will be located in the patch. Using a static threshold for classifying retinal patches regardless of their location in the image, results in several of the more distant patches being blocked. This is especially true in the scene of Figure 3.4 where a one foot high wall and a one foot deep ditch parallel the roadway. These obstacles cut across the majority of distal retinal patches. For more open environments this would not occur. It would be possible to modify the classification procedure by 1) changing the threshold to permit more obstacle points at greater distances or 2) changing retinal patch size to maintain the same physical space for each patch. However, the approach developed provides the foundation for these and other refinements. It is clearly capable of detecting obstacle locations in the image.
3.5 Summary of Sensing Issues

When attempting to mimic the process of a biological system, the biological model itself may provide many helpful insights into how a system should be configured. By studying mammalian visual systems, an appropriate structure was developed for a robotic sensing system. Properties and observations of biological systems led to a neural network structure that exhibits many similar characteristics. This structure consists of a sensing system to gather environmental information, and a mapping system. Spatial relationships between sensed objects of the environment are maintained through to the mapping on the visual cortex. These principles lead to the robotic navigation system depicted in Figure 3.6.

In the robotic system a scanning laser range finder was used to as the sensor to extract the spatial relationship of objects within the scene. While the system lacks a biological analogy, it is superior to vision systems in its ability to readily extract the spatial relationship of objects in a scene. Laser data range values are extracted in retinal patch size segments. Overlapping the retinal patches increases the resolution of the final image. Retinal patch range values are processed in a cellular neural network to determine the existence of obstacles. By adjusting network parameters the sensitivity of the edge detection process can be specified in accordance with vehicular characteristics. Network output is then analyzed by a classification network to determine the traversable state for the retinal patch. This binary declaration is then passed to the response system.
While the shunting equation network exhibits many of the observed characteristics of the biological vision system, it failed to provide adequate performance when presented with laser range data. Cellular neural networks, which exhibit some biological features, proved to be better suited for the processing of range data for this application. Further, cellular neural networks were developed based on a hardware implementation which could process a retinal patch in 5 msec independent of the size of the patch. Also, these networks have an exceptional noise removal capability inherent in their design.

Mapping sensory data to the visual cortex by the simple classifying network also provides addition noise removal. The sensitivity of this classification can be adjusted in accordance with expected environmental conditions and sensor noise. A single cell is sufficient for classifying whether a patch is traversable or not. This binary declaration is then mapped onto the visual cortex. Spatial relationships of objects in the scene are maintained throughout the complete sense-association process.

It must again be noted that no attempt is made at declaring the structure developed as the definitive sense-association process. There are undoubtedly many other configurations and structures that could obtain similar performance. When developing neural structures, the concept of an optimal configuration is elusive. What should be used as a guideline is the idea of sufficiency for the application. In this regard, the structure developed in this chapter is sufficient for making binary determinations as to the traversability of regions in laser sensory data.
3.5.1 Executive Summary of Specific and Significant Results

The following items highlight and summarize the results of the sensing process described in this chapter.

--The biological structure of mammalian vision systems provided the basis for developing a neurally inspired robotic sensing system.

--Application of the biological approach to sensing provided a method for handling the large amount of sensory data. It also permitted the application of the system over a wide range of environmental conditions that are independent of sensor configuration.

--A scanning laser range finder was selected as the sensing device because of its ability to rapidly extract the spatial relationship of a scene.

--Retinal patches extract a portion of the entire input scene. Each patch will be analyzed and rated as to its traversable state. The composite of all patches will show the traversable regions of the scene.

--A novel application of processing laser range data for navigation was presented for both shunting equation networks and cellular neural networks.

--Cellular neural networks were seen as vastly superior to other approaches for analyzing retinal patches. Cellular networks have the ability to normalize total input, determine the existence of edges, and remove a substantial amount of noise.

--Range discontinuities are an invariant feature which indicates the presence of some obstruction in the environment. This feature is invariant in terms of sensor orientation, distance, and ambient conditions. It is an acceptable criteria for determining
the traversable state of a region.

--A single cell classified each retinal patch as being traversable or non-traversable based on the number of edge points detected by the cellular neural network.

--Binary output from classifying cells are mapped to the visual cortex. The spatial relationship between input data, retinal patches, cluster output and visual cortex are maintained.

The visual cortex mapping terminates the sensing and association process of this chapter. Subsequent navigation responses will be based on this image. How the image is processed and interpreted is the subject of the next chapter. This chapter and the next are then combined in Chapter Five to demonstrate the workings of the complete system in the simulated robot world.
Chapter Four -- Navigational Behavior

Human behavior is commonly described in terms of actions or reactions to the application of stimuli. While these responses are often analyzed from an external, psychological perspective, they are ultimately controlled by internal neuronal processes. Somehow, the output of individual neurons are combined or arranged in manners that result in the exhibition of behavior. This chapter describes the development and organization of artificial neural components capable of exhibiting the behavior of navigation.

4.1 Artificial Neural Network Approach to Navigation

It was originally believed at the onset of this research that a network could be implemented to handle all facets of navigation in the proposed stimulus response manner by training a large, fully connected back prop network on a few pivotal input patterns. These training patterns would be selected based on their ability to define the concept or essence of navigation. It was hoped that the generalization capability of the network would then allow proper responses for all subsequent inputs. In attempting this implementation, an initial problem seemed to be an inability to select appropriate patterns. After extensive effort, it appeared that quantity of training patterns would win out over quality because of the difficulty in defining pivotal patterns. However, the true problem of this approach was in the basic premise that a single network was capable of exhibiting complex behavior.
There are limits as to how much a single neural network can learn. In the approach above, the network was basically being asked to learn or distinguish too much information from the input. In a way, this problem is similar to that discussed by Brooks [7] that led to the development of his hierarchical structure, see Figure 2.4. Brooks indicates that traditional sequential approaches to navigation are often hampered by extensive perception and reasoning processes that attempt to incorporate too many aspects of navigation. To alleviate the complexity, Brooks suggests sub-dividing the problem along behavioral lines, see Chapter 2. Performance of the single neural network approach is likewise hampered by trying to account for too many behavioral aspects of navigation. This problem can also be alleviated by subdividing the problem along behavioral lines.

The research discussed in this chapter is directed toward the development of a hierarchical structure of separate neural networks for exhibiting different levels of navigational behavior. Using the behaviors of Brook's hierarchy, networks were designed for the obstacle avoidance, wander, and goal seeking levels. An issue arises as to how neural networks can exhibit behavior. The answer to this exists in biological principles.

4.2 Biological Considerations

"It might seem that one could simply connect neurons together by means of synapses and make networks that mediate behavior, but this is not the way nature does it. A general principle of biology is that any given behavior of an organism depends on a hierarchy of levels of organization," Shepard [79]. The hierarchical organization
discussed here by Shepard differs from that of Brooks. Brooks' hierarchy can be considered a macro system view and discusses what behaviors need to be exhibited for navigation. Shepard's view is at a micro level which describe how neuronal process result in a behavior.

Shepard and Koch, [79] describe the relationship between neuronal processes and behavior through organizational levels within the nervous system, see Figure 4.1. Toward the bottom of this organization are the specific cellular components that comprise and define the functions of individual neurons. As these cells are connected into networks, denoted as local circuits in Figure 4.1, responses are produced that are germane to that network's particular location within the brain. These local responses are then connected to other regions, through interregional circuits, to achieve a synthesized behavioral response for the system. Thus, behavior is the result of neural systems and network organization and is not defined by individual cellular function. Grossberg [36] states that "An analysis of
individual cells is insufficient (to describe behavior) because key behavioral properties are often emergent properties due to interactions among cells. Different types of specialized neural circuits govern different combinations of emergent properties." In biological systems these different neural circuits are associated with different regions of the brain.

There are ten principal regions of the human brain.\textsuperscript{1} From the model above, local networks propagate stimuli within these separate regions which then collectively results in a behavioral response. There are several brain regions that are believed to contribute to a navigational response. The primary region is the cerebral cortex (commonly referred to as the neocortex). "The basic task of the cortex is the processing of sense data and the formulation of appropriate motor response," Douglas and Martin [19].

Within the cerebral cortex there are several sub-regions to include the primary somatosensory cortex, primary auditory cortex and primary visual cortex. These \textit{primary} cortices connect to their respective portion of the association cortex, i.e. primary visual cortex to the visual association cortex. It is within the association cortex that movement decisions are made. Output from the association cortex is directed to the primary motor cortex, also located within the cerebral cortex, which controls the movement of muscles. Recalling an earlier discussion from Chapter 1, it is evident that the organizational function of neural networks are identical to those exhibited in the brain for sensory motor

\textsuperscript{1} Much of the discussion concerning basic brain organization and function is derived from a basic textbook of physiology by Carlson [Carlson] except where noted.
Because regions of the brain are highly interactive, there are other regions that are also likely contributors to the movement decision making process. The hippocampus has been shown to be associated with spatial perception. O'Keefe [68] indicates that the hippocampus may allow an animal to determine its position in an environment and "calculate the behavior necessary to move from its current position to a desired location (e.g., one containing a reward)". Carlson [11] indicates that the hippocampus may also be associated with motivation which would provide the inspiration to seek the location containing the reward. Other brain regions affecting navigation include the cerebellum which is associated with motor coordination and the thalamus which relays and filters virtually all input to the cortex.

Undoubtedly there are many other regions that directly or indirectly affect movement decisions. What needs to be extracted from this discussion is the compartmentalization and combinational architecture of the brain. This research seeks to mimic this approach by developing separate local networks whose composite output defines the behavior of navigation. The previous chapter initiated this regional processing concept by describing the responses of retinal cells to visual stimuli. Output from this network resulted in an input or mapping of environmental data onto the primary visual cortex. Processes developed in this chapter can be considered analogous to those of the association cortex; determining an appropriate movement response for the environmental
In developing local networks for behavior, it is improper to overlook the functioning of individual cells that constitute the network. Cellular functions impart inherent characteristics that affect the computing performance of a network. Consequently, as in the previous chapter, artificial cellular processes should be based on the anatomies of the biological system. In mimicking the processes of cortical cells of the cerebral cortex, it is possible to use some of the existing and established artificial neural network paradigms. Development of these paradigms were based on principles of a generic biological neural cell, see Appendix A. While the structure of neuronal cells may vary with brain regions, the general function and organization of these cells provides the basis for several artificial neural network paradigms such as back error propagation. The self-organizing of neuronal cells exhibited in biological systems inspired the networks developed by Kohonen [47]. These artificial neural systems are not to be taken as biological equivalents but rather as useful tools that exhibit similar properties. While there may be several deficiencies in the artificial model, Douglas and Martin [19] note that "the potential usefulness of network models that are biologically based cannot be overestimated." Thus, this research seeks to exploit the capabilities of existing network paradigms in developing local networks for navigation.

4.2.1 Implementation of Biological and Behavioral Considerations

From the forgoing discussions it can be seen that behavior is a result of network
architecture. Neuronal cells are organized in levels of local networks that contribute to
the exhibition of behavior, see Figure 4.1. These behaviors can also be organized in
hierarchical levels to obtain more complex behavior, see Figure 2.4. By combining these
two principles of hierarchical organization, a viable neural network approach to the
problem of navigation can be developed. In this research, separate neural networks are
developed in a hierarchical fashion to achieve the behavior performances of the first three
levels of Brook's hierarchy of navigation. Individual cellular functions of each network,
which contribute to the display of a particular behavior, are based on an established
network paradigm. Networks for obstacle avoidance, wandering, and goal directed
navigation behavior were developed for implementation in a simulated environment.

4.3 Computer Simulation

A menu driven software simulation was developed for implementing and validating
the neural network approach to navigation as proposed above. This software package
consists of three major components: training pattern generation, network training, and
evaluation. A typical usage of the simulation package consists of the user generating
training patterns, training a network structure on these training patterns, and then
evaluating network performance. To facilitate the explanation of each of these sections,
features of the computer simulation will be discussed first.

4.3.1 Simulation Display and Representation

As in any simulation, some concessions must be made for practical
implementation. Efforts were made to limit the artificialness caused by these concessions as much as possible. Guiding this effort was the will to keep all aspects that were deemed critical to the neural navigation process from being simulated in ways that would preclude their actual implementation. This section discusses the format and issues pertaining to the simulation used in this research.

Simulated environmental information is mapped onto the visual cortex by the processes developed in the previous chapter. This mapping is represented as a 7x7 grid which is displayed in a perspective projection to offer the viewer a more logical sense of the scene, see Figure 4.2. Binary inputs are displayed as either open space (0) or, for obstacle locations, as boxes (1). While these obstacle locations could be obtained from the simulated laser data of the artificial robot world, they are generated randomly in the simulation program to reduce the time between screen updates. Density of obstacles can be controlled by varying a threshold in the random generation routine.

The circles under the perspective projection of Figure 4.2 indicate the candidate
robot moves. Initially, the robot is located at the bottom, center circle and can select one of the five movements commands that represent a movement into the simulated environment. Selecting a movement direction acts as the stimulus for a motor response which is simulated by scrolling the scene forward with allowances for any right or left movements. In this way, the robot remains stationary as the scene moves by the robot. New environmental information is added at the back row of the representation. In this way, the system simulates the movement through unknown, irregular terrain.

The simulation program was primarily designed for only forward robot movement in the direction of the upper row of circles. This may initially seem overly restrictive as in rare instances there may be a goal position that is unobtainable because of the necessity for right angle turns or reversing direction. However, it is believed that the processes developed here are sufficient for the movement commands selected and can be expanded to include a wider selection of moves. Furthermore, the orthogonality of the movement and updating scheme caused by the simulation may also be restrictive. It is probably more realistic that once a turning command is executed, the vehicle will remain in the direction of the completed motion and will not realign itself parallel to the direction of previous position as required by the simulation. If permitted to remain on course, it would be possible for the robot to achieve right angle turns or even to circle back within the space of a typical viewing segment.

The system does have provisions for a backward movement command but this
routine is only evoked when the robot encounters a position where movement in all directions is denied. This routine was not necessarily developed as a candidate movement direction but was included as a means for the robot to indicate failure. Upon its occurrence, subsequent processes can be called to take corrective action. Reversing directions results in the scene scrolling backward. An artificial aspect of this motion is that once obstacles are negotiated scrolling forward, their positional information is lost and not recalled for backward movement.

Another movement consideration pertains to the robot's ability to traverse between diagonally connected obstacle positions of the representation. This movement was deemed permissible after analyzing the affect of overlapping retinal patches. Overlapping results in detectable obstacles appearing in four contiguous retinal patches, see Figure 4.3 (a). If another obstacle point is added diagonally adjacent to the set of four retinal patches shown in Figure 4.3 (a), there will actually be one quarter of a retinal patch of free space between the two obstacle points, see Figure 4.3 (b). For the discussions that follow, it was assumed that this was sufficient space for the robot to traverse which permits the robot to cut corners, see Figure 4.3 (c). It will be evident from the discussion of training and network operation that this
assumption could be readily adjusted to permit any interpretation of representational data.

In addition to the graphical display of Figure 4.3, other screen information is provided by the simulation program. Above the grid representation of the environment is a matrix of simulated laser range values. These range measurements are measured from the robotic vehicle to the center of the corresponding grid position of the scene. Attempts were initially made at utilizing these range measures directly. However, training problems caused this approach to be abandoned in favor of the binary representation. These values continued to be displayed because of their potential application in future work which will be discussed in a later chapter. Also displayed in Figure 4.2 is a screen menu and other data pertinent to a particular use of the system. The following sections now describe the different components and use of the simulation package.

4.3.2 Generation of Training Patterns

Training patterns can either be generated manually or in the software package by invoking the program getpat.c. The source code for this program is given in Appendix C. A program menu within getpat.c initiates and terminates the program. At start up, the user is queried as to the number of training patterns to be generated and whether goal locations should be included. After entering the desired responses, pattern generation commences.

The program generates typical environmental scenes at random and displays the
scene in the format of Figure 4.2. Users enter an appropriate movement response for the scene by using the arrow keys of the keyboard and illuminating a movement direction circle. Movement responses are considered target vectors and are paired with the corresponding screen display of environmental information which serves as input vectors. This input-target vector training pair is then written to a file for subsequent training recall.

Users may exit or restart the pattern generation process at any time. Additionally, after generating the desired number of patterns, the user can either exit the program or restart the process. These features were added in case the user noticed any gross inconsistencies or errors in their responses. These improper responses will have a deleterious affect on the training process.

4.3.3 Training

Training methods for artificial neural networks depend on the network paradigm being used. For the current stimulus-response approach to navigation, a straight forward training approach would be to use supervised training and provide the system with appropriate responses or target vectors for environmental conditions. Two supervised learning networks, one being back error propagation (see Appendix A) and the other counter propagation network (CPN) developed by Hecht-Nielson [39], were investigated for use in this research.

\footnote{Chapter Five introduces concepts and proposes methods for the robotic system to determine appropriate movements autonomously.}
4.3.3.1 Counter Prop Networks

Counter prop networks combine the unsupervised learning Kohonen network, see Kohonen [46] [47] and Appendix A, with trainable Grossberg outstars neurons, [37]. The Kohonen layer of the network has the ability to self organize on the input training vectors. Self organization results in a spatial relationship of nodes within the network based on the similarity of input vectors. Kohonen layer output, or winner, is the network node that is most similar to the input vector. Nodes of the Kohonen layer are connected to trainable Grossberg outstar which are tuned to the target vector.

Initially, a solution to the navigation problem was sought through a CPN network. Motivation for using this network was the self organizing feature of the Kohonen layer. It was envisioned that as the traversable state went from clear to blocked, the winning Kohonen node would traverse across the Kohonen layer weight space. As this winning node moved, the values of the associated Grossberg outstar would correspondingly decrease. For subsequent scenes it was hoped that the network could determine when the traversable conditions permitted or precluded movement in a particular direction. A network of this type was developed using the Artificial Neural Simulation package produced by the Science Applications International Corporation, [3].

Using training data developed by the computer simulation of the preceding section, CPN networks were developed and trained using the ANSIM package. Unfortunately these networks were unable to display acceptable navigation behavior. Numerous network
sizes, parameters and training data sets were continually tried but to no avail. Their failure may stem from the fact that Kohonen networks function as statistical look-up tables, Hecht-Nielsen [39]. This feature could have precluded the extraction of conceptual navigational information. A complete analysis of CPN performance for this application is beyond the scope of this discussion. Especially when considering that success was achieved through the use of the back error propagation paradigm.

4.3.3.2 Back Prop

Back prop networks were found capable of exhibiting the behaviors necessary for navigation. In this research, the training of these networks was accomplished through the use of an existing software package called NETS. NETS is a product of the Software Technology Branch of the Johnson Space Center [5]. Training patterns generated by the program getpat.c are written in NETS format. These training patterns are then used to train the weights of a particular network structure using the NETS software. Once a network is trained, weights can be exported from NETS and imported into the computer simulation evaluation program. Particular structures and training procedures for the back prop networks will be discussed in subsequent sections.

4.3.4 Evaluation

The evaluation program, eval.c which is located in Appendix D, provides a means for determining the validity of network responses to randomly generated environmental scenes. Responses can be either analyzed individually or they can be compared to the
performance of user responses to similar data. Displays for this program are identical to those of the pattern generation routine of Figure 4.2.

*Eval.c* is a menu driven program that offers the user the ability to specify many different organizations and combinations of internally embedded back prop networks. Users may specify the type of network, whether goal locations are included, and the number of networks to be used in a hierarchical configuration. Two additional modes, one displaying network activation levels and the other providing a delay between moves, may also be specified. There are a total of 42 different operational combinations possible. The following table indicates the various modes which will be discussed in greater detail later:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Goal</th>
<th>Obst Avoid</th>
<th>Delay</th>
<th>Display Act</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Response</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Networks</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Obstacle Avoid</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Wander (2 Networks)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Upon specifying a particular mode with options, the necessary weight files are imported from NETS. The program then generates random scenes which serve as input to the neural networks. Network propagation results in a movement decision which updates the scene. Performance is judged by a scoring system which is based on one point for every obstacle position negotiated and fifty points for every goal position reached. Program duration is limited by an adjustable time segment currently set to thirty
seconds. Collisions with obstacles result in a three second penalty. The scoring feature permits the comparison of network performance to that of a user for a similar time segment.

This completes the discussion of the simulation program. While the source code for these programs are listed in the appendices, the actual weight matrices that were trained in this research are omitted. These weight matrices are merely a rather large set of floating point numbers. Examples of training patterns generated by the program getpat.c are offered in Appendix C. The following section discusses the development of these training patterns and weight files for each network of the navigational hierarchy. From this discussion and the source code provided, appropriate training patterns could be developed for network learning.

4.4 Obstacle Avoidance Network

According to Brooks' hierarchy, the foundation of a navigation system is constructed on the behavior of obstacle avoidance. Obstacle avoidance ensures the safety of both the robot and the contents of its environment. At its most primitive level, obstacle avoidance is the simple task of ensuring that the robot does not collide with an obstacle. If, for a moment, movement is considered as a foot placement, then the task is to insure that no obstacles are stepped upon or kicked. In numerous traditional approaches to navigation, the simplicity of obstacle avoidance was often lost by trying to combine the process with higher level reasoning schemes. The hierarchical format
permits developing the sophistication of each level only to the degree necessary.

Using the representation of Figure 4.2, the process of obstacle avoidance is to ensure that no obstacles exist in the direction of a proposed movement. Since the five movement directions of Figure 4.2 correspond to the five bottom center retinal patches, the traversable state of these patches will ultimately dictate whether movement into one of these retinal patch areas is allowed. If all positions are blocked, the network can output the aforementioned move backward command as way to indicate a blocked condition. It is noted here for a later discussion that the move backward command can only be evoked by the obstacle avoidance routine.

In the program eval.c, a small back prop network was developed for obstacle avoidance consisting of five input, nine hidden, and six output nodes. Binary values representing the traversable state of the five lower central retinal patches provide network input, see Figure 4.4. Output, also binary, indicates the acceptability of movement directions. The obstacle avoidance routine makes no decision as to what direction is preferable from its output. It is only concerned with which moves are allowable. From the set of allowable moves, some other process will determine which is the most desirable.
4.4.1 Obstacle Network Training

In training the back prop network it was possible to utilize the complete input data set for input training vectors. Training on the complete input set will guarantee proper network response. This is an extremely desirable property in this usage because of the importance of the obstacle avoidance process. Target vectors were developed that represent all collision free moves for each of the thirty two input states. Because of the overlap in retinal patches, it was assumed that the robot could not skirt pass an obstacle located to its immediate right or left to make the wider right or left turning movement, see Figure 4.4. Similar to the discussion concerning movements between diagonal obstacles, this assumption could be changed by merely modifying the target vectors. Using this training set, the back prop network described above was trained using the NETS software.

The network was trained for over 3000 cycles when the maximum error converged to .004 with an RMS error of .001. NETS normally uses a binary representation of .1 and .9 instead of 0 and 1. However, in this case a binary scheme of 0 and 1 was used in order to gate movement directions off or on, respectively, through multiplication. Subsequent sections will discuss the generation of movement activations levels but it is noted here that these values will be bounded between 0.0 and 1.0 by the sigmoid activation function of the back prop algorithm. The maximum value that could then be obtained in gating a direction off through multiplication would be .004; max error .004 * 1.0. If a threshold of .1 is used for determining the permissibility of directions after
multiplication, there is an absolute assurance that obstacles will be avoided. Likewise, in gating a signal on through multiplication of 1 ±.004, movement activation levels are not significantly affected. These network weights were imported into the program eval.c to evaluate the performance of the network.

4.4.2 Obstacle Network Performance

Obstacle avoidance network performance can be evaluated by two methods using the program eval.c. First, in the Obstacle Avoidance mode, the backprop network is used to determine collision free directions for randomly generated scenes. Network output specifies all acceptable moves of which one is selected. This selection then provokes a movement response which updates the scene and the process then reiterates. The second method of evaluating obstacle avoidance performance starts to build on the eventual hierarchical system structure by having the obstacle avoidance network verify the selection of a movement command of another network.

Selecting the Random network mode with the Obstacle Avoidance option, invokes two networks which are combined in a hierarchical fashion. One network, for obstacle avoidance, is identical to the one discussed above. The other is an untrained network that utilizes all 49 retinal patches as input and produces five random movement responses as output. By organizing these networks in a hierarchical fashion, the obstacle avoidance network eliminates unfavorable movement responses generated by the random network. A movement direction is then selected based on the highest activation for the remaining
cells, see Figure 4.5.

During evaluations in either mode, the obstacle avoidance network never failed to insure the selection of a collision free move. The obstacle avoidance routine is extremely robust and ensures success as it basically looks before the robot steps and guards against improper movement. If there are no acceptable moves, the robot selects the move backward routine. This routine could then be linked to some other process or heuristic to invoke some appropriate action. However, it would probably be even more desirable for the system not to enter a condition that requires a backward move.

The obstacle avoidance routine is purely a local process which can lead the system into undesirable states or local minima. Selection of movements that are locally obstacle free often resulted in all subsequent movement being blocked in several evaluations. These situations can be avoided by choosing more globally desirable obstacle free movements. With neural networks this could be accomplished by replacing the random network of Figure 4.5 with a network that can determine the desirability of movement direction based on the complete environmental

Figure 4.5 Hierarchy of Networks
scene. This new network would select moves that exhibit the behavior of wandering through the environment.

4.5 Wander Network

"Wander aimlessly around without hitting things" is how Brooks' [7] describes the second level of his hierarchy. The impetus for including wandering in the hierarchy is that it causes motion or interaction with the environment. If one is to analyze this process it can probably be deduced that wandering occurs in directions that appear the most promising for continued movement and away from dense obstacle areas. It is the aim of the wandering process to avoid blocked conditions or local minimums that would curtail movement. A wandering neural network was designed to provide for this type of motion by analyzing the complete scene and comparatively judging movement directions based on their potential for sustained movement. In a hierarchical structure, this Wander network replaces the Random network of Figure 4.5. Wander network output specifies the desirability of movements which are again validated by the Obstacle Avoidance network. In developing the wander network, two issues of pattern generation and network structure must be addressed.

4.5.1 Training Pattern Development

Since the wander network must derive a global response, all environmental information must be used as input. This results in a total of $2^9$ possible input patterns.
Since it is impractical to train a network on all of these input patterns, schemes were developed to extricate an appropriate representational sub-set of the input space. Initially, attempts were made to manually extract the key, pivotal patterns that would describe the concept of wandering. However, networks trained from these types of data sets performed poorly when evaluated on randomly generated data. It appeared that a lack of training patterns that represent something between blocked and open conditions resulted in abrupt and improper responses when these patterns were encountered during evaluation. Because of the difficulty in defining these appropriate transitional patterns, it was decided to train the network on a typical sampling of randomly generated patterns.

Random environmental patterns were generated and associated with target movement directions obtained from user responses. It was decided that if the ultimate goal is to achieve human performance, that it would be best to train the network on human responses. Using this approach the network would effectively *watch* and learn how a human responds to the environmental conditions.

The first attempt at having the network *watch* human responses was to run the evaluation program in the User mode and write environmental information and user responses to a training file. A problem with this approach was the inconsistency of human responses to the environmental data. There are numerous input states where several directions may be equally acceptable. If the user does not remain consistent in selecting movement responses, there is a potential for developing an erroneous training
set. Contradictions in the training set will prevent network convergence and ultimately affect performance. To alleviate this problem, \textit{getpat.c} was written in its current form to permit a user to select all acceptable moves for a given input pattern.

Although the user may now input all acceptable moves, there is still a chance of an inconsistency being introduced to the training set. This is especially true in large training sets. For example, if randomly generated input pattern \#3 is identical or extremely similar to \#88, the user may not recall their exact response to the earlier pattern. This is compounded by the binary declaration where an earlier choice may have been considered borderline. Again these inconsistencies will result in learning problems and adversely affect performance. An interesting dilemma then arises in determining the training set.

In general, increasing the size of the training set will refine the generalization capabilities of a neural network. However, in this application, increasing the size of the training set increases the chances of inconsistency which will adversely affect the generalization capability of the network. Consequently, the training set must be selected large enough for proper network generalization but not too large where inconsistencies may pollute the training set. Through research, a training set of 100 patterns was deemed adequate for the current application. This number was also selected in concert with network dimensions that are now discussed.
4.5.2 Network Structure and Training

Two network designs were developed in this research for the wander network. Both of these networks performed exceptionally well and therefore both were maintained in the final evaluation program. One network is a standard fully connected backprop network, *fullcon.net*, and the other is a pattern connected network, *patcon.net*. These networks can be respectively invoked as either Wander - fullcon or Wander - patcon in the evaluation program.

The network *fullcon.net* consists of 49 input nodes, 50 hidden nodes and 5 output nodes. Hidden layer size was selected as a compromise between the number recommended by the Kolomogrov mapping theory, a desire to keep the number of required training patterns low, limiting training time and, of course, providing sufficient weight interaction to attain convergence and generalization. Depending on the quality of the training set, network learning times where typically eight hours on a PC AT for a convergence of maximum error to less than .01. The definition of error in this application is more nebulous than in the obstacle avoidance network. Wander network output specifies a particular direction based on maximum output activation. There is seemingly no difference in choosing between movement direction that vary by as little as .01. In this regard, a network convergence of .01 level insures that the selected direction is proper for the set of inputs.

Network training time can be reduced, without adversely affecting convergence
and generalization, by embedding knowledge of the problem in the network structure. Pattern connected networks exploit the designer's knowledge of the problem through a perspicacious selection of connections. Instead of fully connected networks, connections are made to stress key relational aspects of the input and internal layers of the network. There are two approaches to developing these pattern type networks. One approach, called the morphological approach and championed by Steven S. Wilson [88], permits the network to determine what connections are necessary. The method starts with a fully connected network that eliminates connections if relationships do not exist. Another approach, and the one pursued here, is the use of shared weights for back prop networks, first presented by Rumelhart [75].

Shared weight back prop networks have an analogy to the feature detection process of retinal patches. A segment of the input space is extracted and processed through a set of weights that are successively passed over the complete input space. These weights are trained to determine the existence of some feature within the extracted segment of input space. Output from this set of weights serve as cellular activations for the hidden layers of the network. Hidden layers may likewise be connected through shared weights to the output layer. Both NETS and ANSIM software enable the user to specify various sizes and overlaps for the shared weights. Problem knowledge is imparted into network structure through selecting the appropriate patch dimensions and interconnects.

Applying the shared weights concepts to the wandering behavior, each movement
direction could be evaluated based on the environmental conditions directly affecting the movement direction. Decisions could then be made based on a comparative analysis of the movement directions. A network of this type was developed using NETS software.

The pattern connected network of Figure 4.6 uses two overlapping feature detectors to determine the acceptability for wandering through a particular segment of input space. The right side hidden layer of Figure 4.6 was designed to determine if movements were blocked horizontally through the patch segment. The left side layer seeks to determine the likelihood of vertical movement. These hidden layers are then fully connected to the five output layers as a means of promoting competition between candidate directions. This competition is desirable since selection of a movement direction is probably relative to the overall input.

Because of the reduced number of interconnects in the pattern network, the time for one complete pass through the training set is less than a fully connected network. However, the number of cycles increased for the same level of convergence which resulted in approximately the same eight hour training time. The fully connected network took 3868 cycles for convergence where the pattern connected network took 9875. It
must also be noted that it was found to be difficult to train the pattern connected network pass the convergence point of .01. Where the fully connected network was capable of additional convergence to a maximum error of .001, the pattern connected network could only converge to the .009 level. This difference did not appear to affect performance of the pattern connected network.

4.5.3 Wander Network Performance

The evaluation program was used to judge the performance of both networks. Random scenes are generated by the program which serve as input to the wander network. Network output specifies the desirability of a particular movement direction. The direction of highest response is selected as the best direction for the scene. Movement in the selected direction updates the scene and the process continues. Performance is judged through a subjective evaluation of system movement choices for the given environment.

Both networks were capable of selecting either one of the most desirable or, at a minimum, a completely acceptable movement direction for the environmental scenes. For example, in the fully connected network, the network would select one of the best directions in 94% of the time. Of this total, 68% can be attributed to the selection of the most desirable direction and 26% the selection of a completely acceptable direction. In the remaining 6%, 5% represents minimally acceptable responses and 1% represents questionable responses. The pattern connected network performed slightly better with an
81% selection rate for the most desirable, 14% acceptable, 3% adequate and 2% poor selection rate. Rating responses were based on a subjective judgement comparing system responses with human responses for identical situations. Based on a purely subjective evaluation of observed performance, it was concluded that the networks were capable of exhibiting the behavior of wandering. Further, in comparing the performance of both of these networks, it was also noted that there was little discernible difference between the two.

In addition to the subjective evaluation discussed above, other performance criteria were used in evaluating network performance. One criteria was to observe network responses with the obstacle avoidance routine turned off to see if the system would avoid obstacles. Since the network was trained on only collision free moves, a measure of network generalizing capability is to see if the collision free aspect of moves remain. Trained wander networks were continually capable of avoiding obstacles without the inclusion of the obstacle avoidance network. While this condition held in numerous tests, it cannot be assured for every input state. In actual applications, the obstacle avoidance network should always be employed to ensure collision free operation.

Another criteria used for evaluation was to insure that the network would not enter a position where it would need to back up. The whole purpose of the wander network is to promulgate movement through the environment by selecting globally desirable routes. If a wander network makes a move into a blocked condition, it means that the
network has failed to extract the desired essence of its purpose. In instances where this failure occurs, the network should be retrained with the scene that led to the failure included in the training set. Over time, the number of these pathological states included in the training set will increase the system’s versatility.

Since networks are trained on only a sampling of the input space, they cannot always guarantee that all blocked conditions will be avoided. There are undoubtedly numerous pathological conditions that could be developed in which the wandering aspect would fail. While they may be added to the training set for subsequent application, there should be some provision to account for these rare occurrences on line. A method will be discussed in the following section, designed for exploration, that may provide the means for the wandering robot to avoid these sporadic blocked conditions that are not immediately identified.

A final observation of wander network performance concerns the concatenation of moves. Each iteration contributes to the formation of a continuous route through the environment. This route may also be compared to one a human might choose. While each individual move was shown to exhibit human-like behavior, the composite route was usually unnatural. Human responses often form smooth, regular, logical paths through the environment. Consecutive network moves often oscillated in direction. This is due to the selection of movements being based solely on the highest activation for that particular scene even when there may be several acceptable choices that would have
contributed to the continuation of a smooth path. It may be possible to include an additional network that would favor the selection of these secondary acceptable routes. This network could maintain a brief history of moves and encourage the selection of moves that would produce a more human-like path through the environment. While this additional network is feasible, it was not incorporated into the evaluation program.

4.6 Goal Directed Movement Network

If a particular location, or goal position within an environment is sought, movement decision must be made in accordance with both the desired location and obstacle positions. Where wandering and obstacle avoidance movement could be characterized as the selection of a general move, obtaining goal locations requires the selection of specific moves. Brooks' defines [7] this level of navigational behavior as the ability to "Explore' the world by seeing places in the distance that look reachable and heading for them." In this application, this behavior is considered goal seeking. It is assumed that a goal can be seen and located by some higher level process of the robot. The navigation system must now provide the movement instructions that will enable the robot to reach this location.

Extensive investigation revealed that it is infeasible to obtain goal directed movement in a solely stimulus response manner as was possible for the lower levels of the hierarchy. While it appears that some elements of goal directed movement are governed by a stimulus response behavior, indications are that some planning scheme is
also invoked to account for complex, untrained, or unusual combinations of goal and environment. This duality is probably due to an inability to ascertain a conceptual basis for movement for all possible combinations of environmental conditions and goal locations. These observations were made while developing the robotic navigation system and where later found to be supported in biological evidence.

Nadel and O'keefe [69] have observed the responses of hippocampal cells of rats negotiating a maze and concluded that these cells play an important role in the rats’ ability to determine its location in the environment and direct movement towards a goal. Through their experiments, Nadel and O'keefe have developed a cognitive-map theory for the function of the hippocampal cells. Succinctly put, the theory postulates that memory processes develop a cognitive-map of the environment through exploration. Movements that lead to a goal for a particular input image are associated with the image in the mapping. In subsequent moves, current environmental input is compared to previous learned cognitive-maps. If the two match, the associated movement commands are invoked. Otherwise, the animal enters an exploration mode to determine the appropriate response (perhaps this explains the name Brooks uses for the third level of the hierarchy; "Explore"). From this biological evidence and the author’s observations, it seems feasible to subdivide goal oriented response into two conditional processes. One for a familiar environment that will function in a stimulus response manner and the other for uncertain conditions which will require exploration.
The cognitive-map theory of O'Keefe has been neurally implemented in a work by Zisper [91]. Zisper has concentrated on the matching and movement generation processes of the hippocampus for environments in which both objects and the directions from these objects to a goal are known. The matching process is based on the recognition and orientation of known objects within the environment. When a configuration is matched, the associated direction to the goal is recalled. The system developed here differs from this approach in that it is not based on object recognition. This research is predicated on the belief that objects need not be identified for avoidance. Matching, in this work, is based on a conceptual similarity between input and memory. If there is a match, the associated movement direction is recalled directly. When a goal location inspires movement that is contradictory to past learning, this creates a mismatch and evokes an exploration process. This approach permits the system to used in unknown or unexplored environments. Further, it provides a method for exploration that is based on past learning. Thus, where Zisper has based movement on the recognition of environmental landmarks, the approach here is for movement to be based on a recognition of environmental conditions.

In order to direct movement in the direction of a goal, the goal location must be known by the network. In specifying this location as network input, it is impractical to merely input the goal location with the environmental information. Using the binary representation of this research, goal information would be lost in the obstacle representation using this approach. Likewise, it is also impractical to double the input
space by separating goal input from obstacle input into two separate segments of an input vector. Not only would this increase the input space to $2^8$ but it is questionable whether the network will ascertain the importance of a single input in the goal vector component in comparison to the predominance of obstacle inputs.

A solution to the goal specification problem was derived from studies by Grossberg and Kuperstein [33] which has probably inspired the implementations for manipulator obstacle avoidance of Kuperstein [53], Graf [32], Mel [60]. In these approaches, goal positions are identified in head angle or in visual angle coordinate reference frames. This position is encoded in vector format and applied to subsequent networks where a correlation between target, environmental, somatosensory or other information is assimilated. For this application, the goal location is specified as a vector representing the horizontal and vertical location of the goal in the image, i.e. $(x,y)$ location. It is assumed that a vision system could locate the goal and encode the location in a similar format.

### 4.6.1 Goal Directed Network Training

A goal directed network must inspire the selection of movements that will lead to the goal. To train a network that provides this motivation, training patterns were developed to associate goal positions with suitable movements. Initially, attempts were made to have the system watch and learn from human responses to typical combinations of environment and goal locations. However, this approached failed to produce adequate
performance. As with the wander network, there may be several acceptable directions for
a given input. When the network was permitted multiple responses, it often would choose
a response that would move away from the goal. There was not an innate attraction
toward the goal. A solution was obtained by favoring movement directions that increase
the chances of reaching the goal.

Desirability of movements were
specified by weighting movement
directions with a Gaussian distribution
centered on the most appropriate direction,
see Figure 4.7. This distribution increases
the likelihood that a movement toward the
goal will be taken. In the event obstacle conditions preclude movement in the principal
direction, then an adjacent direction is taken. As the robot approaches the goal location,
the choices of directions diminish to insure that movement taken will enable the robot to
reach the goal.

Using the representation of this research, there are a total of forty-nine possible
goal locations to be associated with movements in the training set. While it is possible
to train on the complete input set, the generalization capability of the back prop network
permits the utilization of fewer training patterns. This will allow a reduction in network
size and an associated decrease in on-line processing time for sequential processors. In
selecting input patterns, it is necessary to insure proper network generalization for the omitted patterns. The gaussian weight function has a minimum step size of .1 between values, e.g. movement directions of the sixth row are weighted as follows [.1 .3 .6 .8 .9 .8 .6 .3 .1]. The generalization capability must, at a minimum, insure the monotonic ordering of this distribution. If this relationship is not maintained for each goal position, additional training pairs must be added to the input set.

Using the above criteria, thirty-two patterns were used to train a network consisting of fourteen inputs, fourteen hidden nodes and five outputs. Training patterns represented the complete set of first and second rows goal positions, and every other goal position from the remaining set. Network convergence to a maximum error to .010 and a RMS error of .003 was obtained after 10000 cycles. The complete data set was then applied to check the generalization capability of the network. While the monotonic order was preserved, activations varied as much as .14 between desired and output. This difference may affect the movement selection process. For this reason, an additional network was trained on all forty-nine patterns to evaluate the performance of the network trained on less than the complete data set.

The additional network consisted of fourteen inputs, twenty-five hidden nodes and five outputs. This network converged to a maximum error of .004 and an RMS error of .001 in 5000 cycles. Since this network was trained on all possible input conditions, the maximum variation form expected output will be .004. This relatively small value will
have a minimal affect on the subsequent movement selection process. The selection of movement directions is accomplished through the hierarchical network structure.

4.6.2 Goal Network Structure

Using the hierarchical navigation structure, a network for exhibiting goal seeking behavior was developed in the evaluation program as depicted in Figure 4.8. At the top of the hierarchy is the goal network. Input to this network is a binary vector representing the goal location. This network is trained as discussed above so that its output represents the movement direction most likely to lead to goal satisfaction. These movement directions are then verified by the obstacle avoidance network. Remaining directions are then compared with wander network output.

High correlation between goal and wander directions indicates the existence of a direct or open route toward the goal. This condition can be considered analogous to a match in the cognitive-mapping theory. In these instances, goal directed behavior operates in a stimulus response cycle with the appropriate output being the associated movement command. A mismatch in goal and wander network output indicates that
caution is needed because the goal is located in a dense obstacle area. In these instances some planning or exploration process may be necessary to reach the goal. The duality in goal seeking behavior is seemingly intuitive in everyday life. If one's goal is to reach an unobstructed door located to the individual's immediate front, response is direct and automatic. If the path to the door is not direct and compounded by the existence of obstacles such as chairs or desk, a person then calculates a potential route to the door before moving.

4.6.3 Development of Exploration Process

In instances of mismatch, the network is being encouraged to move in a direction conflicting with the desires of the wander level. An exploration of possible routes must be undertaken to ensure that the robot is not moving into a potentially dangerous or barricaded state. Processes of exploration may possibly be algorithmic in nature. This may enable the application of many existing algorithms for determining a suitable path through the environment. One particular approach that is well suited for the representation being used is a previous work of the author and Norwood [86] that uses potential fields (see Chapter Two). While this approach is neurally based, it is doubtful that biological systems analyze routes through the application of potential fields. A more biologically inspired approach was achieved through a review of neurally based systems developed for robotic manipulator obstacle avoidance.

Kuperstein [53], Graf [32] and Mel [60] have developed neural systems for
obstacle manipulator avoidance. Kuperstein has pioneered this effort which probably started from his work in sensory motor control with Grossberg [33]. In this earlier work, they developed biologically plausible neural anatomies to enable a system to autonomously correlate sensory input with a motor response. For the manipulator applications of the above researchers, the approach entails the random movement of the manipulator end effector and the correlation of the visual effect of the movement with the stimulus causing the movement. Points in the robot’s work space can then be related to a set of joint angles to reach the respective points. After these relationships are learned, goal locations in the work space can be obtained by following a trajectory of continuous work space points from start to goal. In the selection of these trajectories, procedures are used to insure a collision free route.

While there are some similarities, navigation within an unknown environment is unlike the development of manipulator trajectories discussed above. This is primarily due to a difference in work space. For navigation in unknown terrain, it is impractical to randomly move about the entire visual input space to calibrate potential goal positions with a trajectory. It must be noted that some robotic navigation systems have been developed that do wander through limited environments for the purpose of building maps, Moravec [63]. As with the manipulator approach, once these maps are built, subsequent trajectories can then be planned. However, the process of map making, which is the fourth level of Brooks’ hierarchy, is beyond the intended scope of this work.
What is being sought in this work are the procedures that can be used for the robotic vehicle to safely maneuver through unknown environments. As in the approach of Moravec, the robot must first safely wander through and explore an environment before a map is made. This clearly forms an association between the exploration and map making processes. To accomplish these movements, some process, not based on an existing map or known environment, must be used. This is the principal difference between the manipulator approaches and this navigation approach. It is the premise of this research that exploration movements are based on past experience for similar conditions through a stimulus response cycle. What is being extracted from the work of manipulator obstacle avoidance are procedures that may provide insight into the exploration process when there is not an immediate application of one of the moves from the existing set. The work of Mel in manipulator obstacle avoidance provides insightful behavioral information for implementing an exploration process.

Mel uses the concept of mental imagery for determining a collision free manipulator trajectory, Mel [60]. The idea of mental imagery, which is based in extensive behavioral research (see Mel [60]), is to compute a trajectory in the system's mind, off-line, before engaging an action. "In this way a (system)... can step through an imaginary task and predict the potential consequences", Albus [2]. As the system steps through potential actions, the environment is updated. The current state is then evaluated to determine the desirability of the action. This process persists until a continuous collision free route is found through the environment to the goal. Procedures were
developed to incorporate the concept of mental imagery into the evaluation program.

When exploration is deemed necessary, mental imagery is used in the evaluation program to plan potential moves. The program saves the current state and then makes off-line, mental determinations as to whether a goal position can be reached. Unlike Mel's approach where candidate moves are selected at random or through traditional AI search techniques, moves are selected based on network output. Not only does this maintain the complete neural nature of this implementation but it results in movement being based on learning. Network output directs the mental selection of possible movement directions as before. If a mentally developed state results in an additional mismatch, the mental imagery routine may be recursively called. On the detection of failure, defined by an inability to reach the goal through a certain direction, the system reverts to next best direction based on network output and the process continues. Once a route is determined to the goal, the original state is restored and the appropriate movements proceed. For user reference, the background color of the graphic display is changed to a light blue when the system enters its mental state.

4.6.4 Evaluating Goal Directed Behavior

The network of Figure 4.8, equipped with the aforementioned mental imaging capability, was incorporated into the evaluation program eval.c. Evaluating this network's performance was based on comparing network responses to human responses using two criteria. The first comparison was strictly based on the ability to reach goal locations
while the second criteria concerned the manner in which goals were reached. For the first criteria, the scoring feature of the evaluation program was used to directly compare user performance to network performance. On average, both the pattern connected and fully connected wander network based systems, outperformed a human operator. The results for ten trials for each network are as follows:

<table>
<thead>
<tr>
<th></th>
<th>Pattern</th>
<th>Fully</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>333</td>
<td>323</td>
<td>294</td>
</tr>
<tr>
<td>Standard Dev</td>
<td>118.6</td>
<td>16.1</td>
<td>57.3</td>
</tr>
</tbody>
</table>

The pattern connected network obtained higher scores compared to the fully connected network because its cycle time is less due to the reduction in connections. However, the pattern connected network showed a greater inconsistency in reaching all goal positions. It should be mentioned that goals were missed by both networks and users due to some randomly generated pathological obstacle conditions. It is also noted that the user missed more goal locations than either the pattern or fully connected network; four goals missed to three and one respectively. As expected, there was a reduction in network scores proportional to the number of calls to the mental imaging process due to the additional processing. The highest score was obtained for the user using an 8Mhz PC.

It is interesting to note that differences in user and network scores were also related to the processing speed of the computer. For 8Mhz PCs, the networks performed only slightly better on average than the user as described above. On 20 Mhz PCs, network performance significantly increased where user scores showed only a slight
increase. For processing on a Intel 486 processor, network scores showed almost a ten
fold increase. This is expected since network response time is machine dependent where
user response time is a constant.

A reliance on subjective evaluation is appropriate for comparing performance of
these systems. While the previous analysis has sought to quantify observations, there are
extenuating conditions that preclude a definitive objective analysis. First, a user will
improve in performance with practice. At what point is the user considered competent?
Network performance increases with computer processing speed. What processing speed
is appropriate? Further, there is a three second penalty for colliding with obstacles.
Since the obstacle avoidance network insures collision avoidance, this penalty can only
be incurred by the user. Despite these and other minor discrepancies, it can be concluded,
based on observed performance, the functioning of the goal network showed exceptional
performance in being able to reach goal positions. The network was capable of exhibiting
the goal seeking behavior. In the following chapter, an application will be presented in
support of this claim.

A second analysis of network performance concerns the individual movements
selected by the network. Moves were selected based on a combination of wander output,
goal output and obstacle avoidance network output. The goal network ensures that
movement is directed toward the goal. This direction can only be changed by activations
in the wander level and obstacle avoidance. As before in the wander level, there may be
several directions of high activations. When these values are combined with a goal network that also permits several directions, oscillation or unnatural movements occurred. It was found that these oscillations were more pronounced using the goal network trained on the reduced set of input patterns. While the number of these erratic moves could possibly be reduced by an additional network designed to keep movement directed in smooth paths, the goal network trained on all patterns performed more predictably.

Performance of the mental imaging system was deemed adequate for the planning of a route to difficult goal locations. Because of the additional processing, users were occasionally capable of exceeding network performance when the routine was habitually invoked. In most calls to the mental imaging system, the initial path selection process was almost always adequate. It is only in rare instances where problems arise. Through experimentation, it was determined that the potential for problems exist when the wander activation is below .50 for the direction chosen by the goal network. This value is used as a threshold for initiating mental imaging processes.

It is difficult for the mental imaging systems to efficiently determine where a particular move was improper. Because of the recursive nature of the procedure, failure will first be detected at the terminating node. Exploration of subsequent directions will likewise commence. If all paths are blocked from this position, the system reverts to the previous position and attempts a new direction. In some environmental conditions, this could lead to quite an exhaustive search.
In an actual application after the mental imaging process determines a route, it would be necessary for the system to verify the environmental state at each step with that expected during the mental imagery process. Unlike the manipulator work space of Kuperstein, Mel and LaGraf, all obstacle locations are not known in mobile robot environments. Obstacles may be occluded by other obstacles. As the robot moves through the environment unexpected obstacles may reveal themselves which may alter the route previously planned. Consequently what the environment looks like after an action should be compared to what was expected.

4.7 Summary of Navigation Network

This chapter has covered the development of a biologically inspired, neurally based navigation system. The behavioral components of navigation, described by Brooks, provide a framework for the development of this system. Networks were developed in a hierarchical fashion corresponding to the each behavioral component that collectively constitutes navigation. Composite output of these various networks enabled the system to exhibit the qualities of navigation. Performance of this system was deemed comparable to that of a human for similar situations in many regards.

Neural networks are an appropriate structure for exhibiting navigational behavior. Individual neurons contribute to network responses which then contribute to the formation of a behavioral response. Network responses can then be combined to form more complex behavioral responses. A network anatomy was develop in hierarchical
organizations as shown in Figure 4.8, for the purpose of navigation. Procedures were developed to train a specific behavior for each level of the hierarchy. These networks were incorporated into an evaluation program which permits a validation of both individual network and composite network performance.

As with earlier AI researchers who were confronted with the problem of determining whether intelligence was or was not achieved by a particular system, subjective evaluation is necessary for evaluating the performance of networks developed in this chapter. For the goal of human-like behavior, a proper evaluation criteria is a comparison of network performance with that of a human. Subjective evaluation of individual and composite structures, indicated a proper, human-like response for the majority of input scenes. The following chapter will present an application which will provide additional insight into network performance.

### 4.7.1 Executive Summary

The following highlights the significant developments of this chapter:

--Developed an neural implementation for the processes of exploration in an unknown terrain. Movement decisions were based on learned behavior for similar conditions and not chosen through random or other methods.

--Developed three separate networks, based on Brooks' hierarchy, to exhibit the behaviors of obstacle avoidance, wandering and goal seeking. Each of these levels were shown to exhibit the desired navigational behavior. The obstacle avoidance level insures collision
free movement. Wander network output selected an acceptable direction over 85% of the time. Goal network performance, on average, exceeded that of a human operator.

Several different considerations were used in the development of training patterns. For critical tasks such as obstacle avoidance, all input patterns were used. For wandering where there is an extremely large input space, it was deemed best to have the system watch user responses to similar conditions. Goal seeking training patterns were developed to attract the robot toward the goal locations.

Using an analogy to the cognitive mapping theory of hippocampal cells, procedures were developed for a bifurcation of goal directed movement responses. When there is clear direct path to a goal, a response is immediate in a stimulus response manner. Otherwise, a mismatch occurs and the system must plan a route.

A feasible planning method, based on behavioral evidence, was incorporated into the navigation system through the use of a mental imaging process. The process pre-plans a route internally before movements are engaged.

An extensive evaluation program enabled the networks developed to exhibit exceptional navigational performance.
Chapter Five -- Application

The preceding two chapters discuss the development of the two components that comprise the neural navigation system of this research. In each of the previous chapters, individual component performance has been evaluated locally. By combining the two components into a complete navigation system, this chapter seeks an evaluation of overall system performance. The composite navigation network structure is shown in Figure 5.1 which was applied to a simulated robot navigation problem. An analysis of system performance indicates the adeptness and applicability of the neural approach through its ability to traverse the simulated environment. Measures for increasing system performance and procedures for the conveyance of this system to use in a non-simulated environment are also discussed.

Figure 5.1 Complete Navigation Network
5.1 System Simulation

An artificial robot world has been modelled in the laser simulation program, LISA [90]. A top down view of this environment is shown in Figure 5.2. This robot world consists of a hexagonal shaped roadway that is bounded by a one foot high wall on the outside of the roadway and a one foot deep ditch on the inside. Along the roadway there are numerous one foot high obstacles of various shapes; spheres, boxes and pyramids. The objective of this simulation is to determine the efficacy of the robotic navigation system of Figure 5.1 in selecting proper movement directions as it navigates down the track.

The robot is initially positioned at the top center of the track indicated by the arrow in the figure. A view down the track was shown earlier in Figure 3.4 and is also reproduced here in Figure 5.3. Simulated laser data, corresponding to this image is generated by LISA. These data are then processed through the cellular neural network to extract obstacle positions. Obstacle position information is mapped to the visual cortex, which is the input to the hierarchical structure. The networks of the hierarchy then produce a motor response which is used to update the position of the robot in the simulated environment. This process continues as the robot moves through the environment.
For the purposes of this demonstration, there are several issues for consideration. First, the pattern connected network is being selected for use as the wander network. Second, a goal position will be established at the laser sensor visibility limit and centered on the roadway. This goal position will move with the robot in an analogous manner as the donkey being led by the carrot. In this way motion will be inspired down the center of the roadway. Finally, because of the cluttered environment, the majority of distal retinal patches will be blocked, see discussion in Chapter Three and Figure 3.18. This introduces two additional considerations. First, the wander networks of Chapter Four were trained for environments with fewer obstacles than will be encountered in the simulation. Response to the new, more obstacle-dense environment will indicate the adaptability and application of the methods developed. Second, the mental imaging procedure was designed to avoid the completely blocked conditions like those shown in Figure 3.18. However, by applying knowledge of the environment it is permissible to turn down the sensitivity of the mental imaging procedure for the purposes of this demonstration. It is known in the simulation that a path exists, even though one is not seen in the immediate image.

The following images walk the robot through the scene. Each iteration represents, approximately, a one foot move and are referred to sequentially by Step Number. In the left image for each step, the view of the laser scanner is seen. Mapping of obstacle positions from this image are indicated in the graphical perspective projection of the figure on the right for each step. Initially, the correlation of these positions between the
two images may not be apparent. It must be remembered that retinal patches overlap one another and the more distal patches represent a greater area which increases their propensity for being classified as blocked. In Figure 5.3 (Step 0), the box located to the immediate right foreground of the environmental scene (left side wire frame image) can be detected in the visual cortex representation (right side figure) as the box located directly above the right most movement direction. As the robot moves forward, pass the box, in Figure 5.4, this obstacle no longer exists. Progressive moves can likewise be traced for the remainder of the images, although some of the relationship between images are less intuitive than those of the initial steps.

Obstacle position information is processed through the hierarchical network structure. Network output, representing the desirability of movement responses, is listed next to the movement direction circle. The top value specifies wander network output and indicates the system’s rating for movement in that direction. The second value is that of the goal network. Goal network values represent the Gaussian distribution weighing scheme for directing movements in the direction of the goal. Output of these two networks are multiplied together to determine the best open route that will move the robot down the center of the roadway. Movement directions are also verified by the obstacle avoidance network which ensures the system does not move into an occupied position. The arrow in the right figure for each step indicates the winning direction. Robot position is then updated to reflect this movement decision and the process continues.
Figure 5.3  Step 0

Figure 5.4  Step 1

Figure 5.5  Step 2
Figure 5.6 Step 3

Figure 5.7 Step 4

Figure 5.8 Step 5
5.2 System Performance

If robot movement is followed through the images, a continuous path can be mapped out as shown in Figure 5.12. The robot successfully navigated through the cluttered environment by moving to the right of both the box and the ball. System output permitted proper responses even for an environment in which it was not specifically trained. While this type of performance was the specific goal of this research and indicates success, there are several observations that need to be made concerning this performance. First, there are some activation levels that appear inaccurate for the obstacle conditions. Specifically the values for the far right moves of the wander network in Steps 2 and 4 are extremely low in relation to their movement potential. This could possibly be caused by the application of a new environment for which the network was not trained. Other than these two discrepancies, the network appears to have adapted quite well to the new environment.

A different path would have been taken had it not been for the inclusion of the goal position, i.e. movement decisions based on wander network output only. In three of the nine steps taken, wander network output dictated a different response which would have taken the robot on a completely different course. Goal network strengths have a
significant impact on the selection of directions. Except for instances where the primary
goal direction was blocked by an obstacle, the primary goal direction was always selected
except once, Step 5. In this case, the desirability of the winning directions, measured by
wander network output, outweighed the pull of the goal network. Selecting a different
weighing scheme for the goal network can therefore result in a different path. Users can
most likely affect the movement characteristics of the system by adjusting the strengths
of the goal network output.

In two cases, the network appeared to enter a barricaded state, see Steps 3 and 5.
In these instances, there was a low correlation between movement direction and wander
network output. All other cases had a wander network output of .86 or higher where
these moves had wander network outputs of .78 and .21 respectively. If the robot had
made these forward moves in the simulation program of Chapter Four, these two scenes
would have scrolled forward and resulted in a blocked condition for all subsequent
movements. The low correlation between direction and wander would have made them
candidates for the mental imaging process. However, because the retinal patches of this
simulation represent different spatial areas than those of the evaluation program of
Chapter Four, open positions were revealed after making the moves.

Retinal patches representing the distal parts of the scene encompass a greater
environmental area. Because of this, their chances of being classified as blocked
increases. As the area represented by a retinal patch decreases, the resolution increases.
Thus, acceptable moves were made after conducting the moves of Steps 3 and 5. Because of this phenomenon, the application of the mental imaging process in a real application becomes suspect. All environmental conditions are not known from a single image. Plans made by the mental imaging process are subject to change after moves when new environmental data is presented. Applying the mental imaging subroutine, as developed early, to either Steps 3 or 5, will result in the declaration of no paths existing. Modifications to either the mental imaging process or the representation are needed. These will be discussed later.

The stimulus response approach to navigation has an additional benefit over some navigation systems because it only responds to the immediate environmental scene. Problems of slippage or other disturbances encountered during a move have less of an impact on performance than they would on absolute positioning navigation systems. A sense of where the robot is, is derived from its location in the environment and not where it thinks it is in some coordinate frame. Further, the speed of the environmental update permits an almost immediate response to environmental conditions and reduces blind travel time. In an actual system, a laser scanner can produce an image in 835 milliseconds (msec). Cellular network processing is 5 msec for a hardware implementation. Processing in the hierarchical networks of this examples took only 220 msec on an 8Mhz PC. With a faster serial processor, cycle time could easily be under one second. Amount of vehicular movement could then be adjusted in accordance with this cycle time.
From this simulation it can be seen that the system developed in this research was capable of navigation. Not only was individual component performance verified in previous chapters but the network as a whole was even able to navigate through a new and strange environment. Because of the network’s ability to navigate and its relatively fast cycle time, this approach to navigation has distinct potential for application in an actual system.

5.3 System Implementation

Since artificial aspects have been kept to a minimum during the development of this system, the system diagram of Figure 5.1 provides the blueprint for implementing the system in an real world application. Procedures used in the development of the simulated system are also applicable to the development of an actual system. Hardware components, such as the laser scanner, can be directly interchanged with the simulated processes of this research. If there is a lack of hardware capability in implementing the neural network structures, they may be developed in software as was done in this research. It must be noted that the gradient approach to jump edge detection operates significantly faster than cellular network structure in serial implementation. A typical system would then consist of a laser scanner with a direct memory access (DMA) to an on board computer. The computer would process laser values through software networks as in this research. Network output would then be directed to the vehicle’s drive system for movement.
The major concern in system implementation lies in the training of the hierarchical networks. Since the cellular network simply responds to data, this network is not as much as a concern as the hierarchical networks. Critical in cellular net development is the selection of network parameters that will extract range discontinuities in accordance with vehicular capabilities. For the hierarchical networks, three methods of developing training patterns were discussed in this research. Training an actual system should follow the same procedures.

For obstacle avoidance, all obstacle conditions that impede the next foot placement process should be included in the training set. It may be beneficial to include additional sensors, such as sonar or laser proximity sensors, in the obstacle avoidance process. By strategically locating sonar sensors along the base of the vehicle, obstacles having a direct and immediate impact on the next move can be sensed and subsequently avoided. While this process was accomplished through laser data in the simulation, it may be better to use a secondary sensor for this purpose. In this way, the scanning laser range finder could be freed from its downward attitude and thereby permitted to sense a larger portion of the environment.

In training the wander network it would probably be preferable to develop training patterns from the environment in which the system is expected to operate. As with the method used in this research where the network watched operator responses, wander network training patterns should be developed in a similar manner. A human operator
could *drive* the vehicle through a typical environment and record appropriate motor responses for each image. This also permits the user to evaluate subsequent system responses with how he would respond for similar situations. Gross inconsistencies indicate that the network did not properly learn; probably due to a poor training set.

The gaussian distribution developed in this research for training the goal network can be easily modified by the user to develop their own particular behavior. Distribution values assign a weight to a particular movement direction with respect to goal position. Changing the weighting scheme will affect movement behavior. As with all three networks, training patterns are developed with whatever assumptions the user desires to make. After training all networks off-line, learned weights are then imported into the software network structure and the system is ready for use.

Navigation system output produces a movement command. For actual system implementation, the specification of this movement command is vehicle dependent. The commands generated must be of a form that produces the desired movement response. Further, to avoid the oscillation problem discussed in Chapter Four, an additional smoothing network may need to be added to the system.

Because the simulation was developed for potential application in a non-simulated environment, it should be possible to directly transfer the system by following the guidelines presented above. There may be some slight nuances encountered when
conducting this transfer but a thorough understanding of the principles presented in this research should provide a foundation from which to seek a solution.

5.4 Extension of Current Work

While the network approach to navigation provided the requisite performance of determining a collision free path through the scene, there are some obvious logical extensions of the approach to improve on performance. These extensions of this work are being proposed as guidelines for the expansion of the neural approach developed here. Two primary areas of research are addressed. The first concerns system representation and the second addresses the learning of motor response.

The current method of declaring retinal patch traversability led to the dense obstacle scenes of the simulation presented above. This in turn impacted on both movement decisions and the mental imaging capability of the system. While the depth that the mental imaging procedure looks into the environment could be reduced in accordance with obstacle density, improving the resolution of the representation appears more promising. The resolution of the representation can be increased in two manners. First, the simply approach would be to permit an increase in the number of obstacle points per retinal patch with respect to the area that that patch represents. The more distal patches, representing an increased area, should be permitted more obstacle points per patch.
The second approach is to increase the number of retinal patches so that each one represents a similar amount of environmental area. This approach may make the representation overly complex in visual analysis. For this reason, it may be best for the human to avoid interpretation of this intermediate mapping and teach responses directly from the initial image. That is, proper motor responses are designated for the scene and not the internal representation. To carry this notion one step further, it may also be possible for the system to learn the proper response without a human supervisor. The previous works of Kuperstein [53], Graf [32], Mel [60] and Tolat [81] provide some insight to this process.

Kuperstein [53], Graf [32], and Mel [60] have developed neurally based robotic manipulator systems that learn a relationship between manipulator joint coordinates and visual stimulus autonomously. Tolat [81] has a developed a neural system that learns the dynamics of a system autonomously. In Tolat’s work, a broom is balanced on a cart which can be propelled in either direction horizontally by thrusters. Through properly designed learning, the network is inspired to determined the proper thrusting action to maintain the broom in an upright position. The key to all of these approaches resides in network training, and the key to the training is repetition and weight adjustment.

Mel cites the work of Piaget’s theory of childhood development as the basis for the learning method of his system, see Mel [60] and Ginsberg [28]. It is Piaget’s belief that an infant develops sensory motor control by learning the relationship between a
motor movement he generates and the associated outcome. After repeatedly flailing its arms about, at some point a relationship is developed so that if the child wishes to reach a goal, the proper motor response has been learned. The systems cited earlier are essentially based on this process. In these systems, sensory motor control is learned without any specific reference to system dynamics or kinematics. It is highly probably that a system could likewise be developed for navigation.

In developing systems of this type it is necessary to formulate the learning process so as to enhance the selection of proper movement responses. A current proposal of this author is to use the concept of operant conditioning as a basis for learning responses; improper responses are punished and proper responses are rewarded. Application of punishment and reward is through neural network weight adjustments where the likelihood of a particular response is decreased or increased respectively. At issue is how to determine whether punishment or reward is applied. In the development of biological systems there is clearly some internal process that evaluates actions. If one touches a hot stove, one is clearly punished for this action and learns not to associate the visual stimulus of a hot stove with the motor response of touch. In a robotic system, no such internal process exists. It is the aim of this proposal to genetically design an internal reward and punishment system that will enhance the learning of navigation behavior autonomously.

Genetic design consists of developing rules that can be used to evaluate network
response. Improper responses can be thought of as network error and are used for weight adjustments. Using simple rules such as 'it's best to move straight forward when possible or head in the direction that is most promising,' network output can be compared to what a human would think would be reasonable for the situation. The proposed system has been initially instituted to determine the feasibility of this approach. A simulated two wheel robot has been developed which uses seven evenly spaced sonar sensors for environmental sensing and network input. Range data are being used because of the potential use of using scanner laser range finder range values directly. These range values are processes in a back error propagation network that outputs two motor responses. Each wheel of the robot responds to its output and the robot moves in the environment. This response is evaluated and any error is used to update the weights of the network. Since the network is initially untrained, responses can be considered random at first. Slowly the network will learn the desired behavior. An example of this process is shown in Figure 5.13.
This approach is still in its infancy and is merely a concept at this time. It appears that some other network, such as Adaptive Resonance Theory, (ART) developed by Grossberg [35] may prove superior in this application because there will not necessarily be a repetition of training patterns as required by backprop. However, by incorporating the processes of this research, a system may be developed that could learn navigation autonomously.

5.5 Summary of System Performance

The neural navigation system developed in this research displays the requisite navigational behavior in the simulation problem. The network was capable of determining a collision free path through the environment autonomously. The system adapted adequately to an environment in which it was not previously trained. While there were some minor discrepancies in activation values, the system performed in a manner that could reasonably be considered as human-like. Applying this system to an actual implementation should be accomplishable by following the procedures discussed here.

This work provides a foundation for continuing the study of the neural approach to navigation. Several logical extensions to this work were presented in this chapter. In addition to these, it is also feasible to extend the navigation system upward one more level in Brooks’ hierarchy. The procedures developed in this research of basing the neural solution on biological and behavioral processes provides a method for developing these more advanced navigation systems.
Chapter Six -- Summary and Conclusions

This thesis has discussed the research and development of a connectionist approach to the problem of autonomous robotic navigation. Unlike most current, traditional approaches, this method seeks human-like navigational performance by basing the internal workings of the navigation system on the processes similar to those of human intelligence. Neural networks provide a means by which the concept of a problem can be learned. By training neural networks on the proper motor responses for a given environmental condition, the concept of navigation can be extracted and encoded in the internal representation of the network. Through the generalization capability of neural networks, a proper navigational response can then be obtained for a myriad of subsequent environmental input states. Network structures and training were developed in this research to implement this approach to navigation. The performance and applicability of the system was then verified in a simulated robotic navigation problem.

6.1 Summary of Development Processes

In developing the neural approach issues concerning network applicability, network design, and network training needed to be addressed. Neural networks were deemed applicable to the problem of navigation for several reasons. First, neural networks are relevant to problems which humans consider intuitive. Second, the stimulus-response processing manner of neural networks is typical of how human motor responses are generated. Additionally, concepts, which provide humans with a means of coping with
a multitude of conditions, can be learned by neural networks. Thus, to achieve humanlike navigational performance, it is appropriate to apply a solution technique, neural networks, which exhibits many of the characteristics of the intelligent being.

Biological and behavioral models provided the basis for the ontogeny of the navigation system which was developed in two parts; the sense-association process and the response process. For the sense-association process, the mammalian visual system guided the development of architectures capable of interpreting and mapping environmental sensory data. A laser scanner was selected as the system's eye because of its ability to readily extract the spatial relationships of objects within a scene. Incoming, simulated laser range data were applied to the system's retinal cells. These data were then extracted in overlapping retinal patch segments and processed in a cellular neural network. Attempts were made to process these data in shunting equations networks because they have a closer biological analogy to retinal cells than the cellular neural network. However, the shunting equation networks were not as well suited as the cellular networks for the processing of laser range data.

The cellular neural network was designed to detect an invariant feature, jump edges, within each retinal patch. Detection of range discontinuities within these retinal patches indicates the presence of an obstacle which will impede movement through the area represented by the retinal patch. A classifying network declares the patch segment as either open or blocked based on the number of edge points detected. This
classification is then mapped to the visual cortex for subsequent processing. A unique
feature of this process is that the spatial relationship of objects within the scene is
maintained throughout all processing steps to the visual cortex. Thus, laser range data
are converted to a relational, binary, internal mapping of the scene. From this mapping,
navigational responses are generated.

Navigation is a multifarious process. A navigation hierarchy developed by Brooks
[8] provides an insight into the many individual components that comprise intelligent
navigational behavior. Along the guidelines of this hierarchy, separate neural networks
for obstacle avoidance, wander and goal seeking behaviors were developed in this
research. It was determined that back error propagation networks provided a convenient
network paradigm for learning the requisite behavior for each level of the hierarchy.
Procedures were then developed to train the networks to exhibit their behavioral trait.

In developing training schemes for neural networks, there are numerous
considerations. The artistry of neural networks enters into the process when selecting
input vectors and training set size, defining the concepts of convergence and
generalization, and determining the proper relationship of training data sets with both
network size and parameters. In this research three separate methods of generating
training sets were presented. For pivotal tasks, such as obstacle avoidance, all possible
conditions should be trained. Wander network training data were obtained by watching
a human respond to typical environmental situations. The goal network was trained by
developing a weighing scheme that adjusts movement directions in relation to the goal position. Networks were then trained until they exhibited acceptable performance.

6.2 System Performance

Performance of the two sub-systems of the navigation process was evaluated separately at first and then jointly. Separate software programs implemented each sub-system. For the sense-association process, simulated laser range data were generated by a laser imaging simulation program, LISA. These data were then processed in retinal patch size elements as discussed above. By adjusting network parameters, the sensitivity of the edge detection process of the cellular neural network can be controlled. Consequently, the determination of what constitutes an object could be adjusted in accordance with vehicular constraints. Thus, the performance of this system is controlled by parameter adjustment.

An extensive evaluation program permits the evaluation of the response sub-system in either a separate network mode or a composite network mode. Scenes, represented in the format of the sense-association sub-system, are generated randomly. Network responses indicate a movement direction which can then be evaluated for the given scene. In evaluating the performance of the obstacle avoidance network, the benefits of training the network on all possible input states were clearly noted. The obstacle avoidance network was capable of insuring collision free movement for every input. Wander movements of the network were evaluated subjectively by comparing user
responses for similar conditions. Because of the network's ability to choose appropriate directions, its responses were considered to exhibited human-like wander movement behavior. A scoring system was used to judge goal network performance compared to user performance. In most all cases, the network was capable of reaching more goal locations than a human user. Furthermore, network performance increased with computer processing speed where user responses remained relatively constant.

The two components of the navigation system were then combined for system evaluation of movement through an artificial robot world. The nature of the problem supports a direct implementation of the neural approach to an actual system. The environment was sensed and data processed in the network structures of above. Network output was then used as a motor response to update the position of a robot in the simulated environment. The composite network structure was capable of exhibited human-like navigational qualities by finding a collision free path through the environment. Also, the wander network showed exceptional adaptability by producing adequate responses for an environment for which it was not trained.

6.3 Conclusions

Neural networks are an appropriate approach to the problem of autonomous robotic navigation. Their ability to encode conceptual information pertaining to navigation permits generation of proper motor responses for a myriad of environmental conditions. Biological structures and processes of intelligent systems provide useful insight into the
development of these artificial neural systems. An understanding of how intelligent systems produce intelligent behaviors facilitates the neural implementation. Through the learning of navigation concepts and the anatomies of network structures developed in this research, a neural navigation system was shown to display many human-like navigational capabilities.

One of the basic tenets of this research was that movement decisions function in a stimulus-response manner where the response has been learned for the stimulus. This approach permits the direct application of the neural approach. For the processes of obstacle avoidance and wander movement, it appears that this approach holds true. However, some aspects of goal directed movement appears to be more algorithmic. If there is not some clear, direct path to the goal, some search technique is necessary. A mental imaging process was developed in this research as a way of determining a path. While this approach works well in structured and set environments of Chapter Four, it was less effective in the environments of the simulated navigation problem. There is need for continued research in this area to develop more robust mental planning routines.

Neural networks do not produce exact solutions. Since output is a generalization to the training data, there will not usually be a unique solution for every input. Some of the movement decisions selected by the networks of this research might not have been the optimal choice. However, what should be considered in these cases is whether the selection was sufficient or adequate. After all, human navigational performance includes
stumbling ones toe on the coffee table (which would be avoided by the obstacle avoidance network). Further, there is no definitive, optimal, network structure for the networks of this type of problem. What should be considered acceptable are those networks that exhibit the desired performance.

6.4 Closing Remarks

This work should be regarded as one of the initial steps taken in achieving autonomous robotic behavior through the use of neural networks. It provides a foundation for continued research in this area and promotes the application of neural networks to problems of this type. In the preface of Mel’ book [60], Stephen M. Omohundro writes about an emerging experimental field that seeks to engineer biological and cognitive processes. He states that "It wouldn’t be entirely surprising if progress in the understanding of these ingredients [perception, action and learning] in some way recapitulates the steps that fledgling organisms must have taken as they pulled themselves out of the darkness." This work is seen as a step of the evolutionary process in the development of a connectionist autonomous robotic navigation system.
REFERENCES


APPENDIX A -- Neural Network Tutorial

Recent advances in neural network technology have revitalized both the study of neural networks and their application to many real-world problems. Neural networks display several characteristics that have proven advantageous to the solution of some problems that have been difficult to solve by existing traditional means. Specifically, these problems are typified by their input space being less than complete. "An incomplete application is one for which specific outputs need only be defined for a subset of the input permutations, or where large similar classes of inputs map into the same output space," Martinez [57]. In these cases, the solution generated must be a generalization or adaptation to the input. A prime example is the use of neural networks as pattern classifiers. Through the generalization capability of neural networks, similar or incomplete input patterns can be classified into an established category for identification. This ability has led to widespread research in the application of neural networks to many vision-related problems and is extensively used in this work for robotic navigation.

This tutorial is designed to provide a basic understanding of neural network dynamics and some of their more appealing characteristics. It is designed to give the reader a general, albeit layman's understanding of the neural network approach and is not intended to make the reader an expert. Rigorous proofs and exacting details of network paradigms and applications have been omitted. Emphasis has been placed on developing the paradigms used in this research. Specifically, the back error propagation network was
extensively used in this research and is therefore discussed in greater detail than other paradigms. For the more interested reader, references are cited in order to permit one to pursue a more detailed discussion of other paradigms.

A.1 In the Beginning...

The subject of artificial neural networks is a large interdisciplinary field with its foundation firmly established in the studies of biological neurons. From these studies, certain characteristics have come to be understood and modeled in ways that enable artificial implementation. While there are many, varied artificial implementations, the fundamental processes of neural networks remain based on the biological neuron.

A representation of a biological neuron is shown in Figure A.1. This neuron consists of a cell body with radiating axons that terminate in dendrites. Dendrites of a cell connect to dendrites of neighboring cells through a structure called a synapse. Neuronal signals, emanating from a cell body, travel down the axons and dendrites to other cells through this synapse. The effect of these signals is controlled
by the receiving cell's synapse which is capable of amplifying or decreasing the incoming signal.

Dendritic connections provide for a highly connective and interactive network structure. A Defense Advanced Research Project Agency (DARPA) study [18], estimates that the human brain consists of 100 billion neurons. Each neuron has about 1000 dendrites giving a total of about 100,000 billion synapses. It is this highly connective feature of the biological brain which many researchers believe gives the brain its computing strength. Likewise, it is within a similarly massively connected network in which artificial neural networks achieve their strength.

Origins of the current neural network implementations are generally attributed to the work of McCulloch and Pitts in the 1940's, McCulloch [59]. McCulloch and Pitts provided the first mathematical model to describe the workings of the biological neuron. Figure A.2 is a depiction of the McCulloch and Pitts neuron and the network equation that describes its operation. In the figure, there are two inputs to the neuron (often referred to as a cell or node) and a single output. Incoming signals are weighted by the synaptic value associated with an input line. These
weighted values are then summed by the cell body and the result is compared to an internal threshold. If the threshold is exceeded, the internal value is positive, otherwise it is negative. This resultant value is then processed through a non-linear activation function; the \( \text{sign} \) function, \( \text{sgn}(x) \). The \( \text{sign} \) function produces an output of \(+1\) for positive input and \(-1\) for negative input. As with biological cells, this \( \text{sign} \) function produces an output within a finite range independent of input magnitude. When output is high, the cell is said to have \textit{fired}.

McCulloch-Pitts used their neuronal model for studies of pattern matching and object recognition because it is capable of categorizing inputs into two categories (binary output). By making adjustments to the internal threshold and synaptic connections, the cell can be sensitized to a particular type of input. A single cell of this type has the ability to categorize any linearly separable problem. If the function \( g(x) \) is defined as the internal sum of the cell, then when \( g(x) > 0 \), the output is \(+1\), and if \( g(x) < 0 \), the output is \(-1\). When \( g(x) = 0 \), this function defines the decision boundary between the binary output. For the cell of Figure A.2,

\[
g(x) = w_0 x_0 + w_1 x_1 + T
\]

Eqn A.1

where \( w_0 \) and \( w_1 \) represent the synaptic weights of input lines 1 and 2 respectively, \( x_0 \) and \( x_1 \) their associated input and \( T \) the internal threshold. If \( g(x) = 0 \) for the decision boundary, then Eqn A.1 can be rewritten as
Eqn A.2 defines a line in the input space which will separate the input space into two distinct regions. By selecting appropriate parameters in Eqn A.2, input can then be classified into two categories. While these parameters can be calculated for the model, in a biological system these values must somehow be learned internally. The McCulloch-Pitts model accounts for many biologically observed phenomenon but it fails to explain the phenomenon of learning.

From the McCulloch-Pitts model, learning can be considered as the process of developing or determining the appropriate connection strengths and internal thresholds to achieve a desired discrimination of input data. D.O. Hebb has developed a theory for how a biological cell could learn these parameters internally, Hebb [38]. Hebb postulated that the synaptic strength between a source and destination neuron was increased if the source neuron led to the firing of the destination neuron. Furthermore, by strengthening connections where a relationship exists, there is an automatic relative decrease in strengths between neurons that do not enjoy a relationship. Neurons are then associated with one another for a particular input based on the strength of their synaptic connections. While there are few actual networks that utilize pure Hebbian learning, most existing learning algorithms have emerged from this concept espoused by Hebb.
Over time the McCulloch-Pitts model became known as the perceptron. If the McCulloch-Pitts model of Figure A.1 is expanded vertically by increasing the number of nodes, the resulting structure is considered a single layer perceptron. In 1962, Rosenblat developed a learning algorithm for a single layer perceptron, Wasserman [85]. While this marked a significant advancement in neural network technology it was short lived because of a work by Minsky and Papert in 1969 [62]. In their book entitled Perceptrons, Minsky and Papert describe the limitations of a single layer perceptron network. While the single layer perceptron can represent the logical and and or functions, it cannot represent the relatively basic exclusive or problem. While it was generally known at the time that multiple layer perceptrons could overcome this problem, there was little knowledge as to how to train these networks. As a result, neural networks research entered its dark ages and attention was directed toward achieving artificial intelligence through symbolic processing approaches.

Multi-layered perceptrons are capable of classifying more complex representations provided the cells use a nonlinear activation function. If the network is expanded horizontally by adding additional layers of nodes, with output of one layer providing input to the next, the result is a multi-layer perceptron. In this configuration, the interior layers of the network are called hidden layers. In Figure A.1, the summation of inputs multiplied by their weights can be written in vector form as $Net_i = XW_j$, where $Net$ and $X$ are row vectors and $W_j$ is a matrix. Without a non-linear activation function, this sum would then propagate directly through the next set of weights, $Net_2 = Net_1W_2$. Since
\[ Net_1 = XW_1 \] and matrix multiplication is associative, \[ Net_2 = X(W_1W_2) \]. Therefore, a weight matrix \( W_3 \) can be found to produce the same effect as the two weight matrices \( W_1 \) and \( W_2 \). By using a non-linear activation function between layers, each layer is able to impart its processing capability on the network. Instead of the \( \text{sign} \) function, \( \text{sign}(x) \), of the McCulloch-Pitts model, most current applications use the sigmoid function,

\[ Output = \frac{1}{1 + e^{-Net}}. \]

As the dimensionality of the perceptron network increases, the classification capabilities of the network increase. It was shown earlier how a single perceptron can classify linearly separable problems. In a two layer perceptron, shown in Figure A.3 (b), the classification regions can be described by two linear delineations. These two layer networks are capable of classifying convex regions. The complexity of the convex region in the two layer perceptron networks increases with the number of vertical cells added to the network, see Figure A.3 (c). Three layer networks are capable of classifying purely arbitrary regions. The sophistication of this capability is only limited by the number of vertical cells added to the network. In most

Figure A.3 Perceptron Classification Abilities, after DARPA [18]
applications, the two layer network is sufficient. It was not until 1986 that a training algorithm was developed for training these higher dimensional networks which is discussed in the following section.

A.2 In the Second Beginning...

In 1986, Rumelhart, Hinton and Williams [75] developed a training algorithm for multi-layered perceptrons. While this was not the single most significant development of the time, this work coincided with other developments that led to a resurgence of neural network research that has continued to this day. The significance of their work is that it provided a solution to the exclusive or problem exposed by Minsky, and also provided a means for classifying more complex representations. Their training algorithm is based on propagating an error signal back through the network to adjust connection strengths and is commonly referred to as back error propagation or back prop.

Back prop training is considered supervised training in that the network needs to be told what the classification or output should be for an input pattern. This is opposed to unsupervised learning in which this association is done internally. Back prop training data are formulated in training pairs consisting of input and target vectors. The composite of training pairs comprise the training set. Training consists of presenting training data, calculating an error, and then making any necessary weight changes. This procedure continually cycles through the training set until the error is reduced, or converges, to an acceptable level. Thus, the training procedure consists of a forward pass to calculate
output and a backward pass to modify connection strengths.

During the forward pass, calculation of cellular activity is identical to the McCulloch-Pitts model except for the activation function. The activation function used in back prop is the sigmoid function. This function is used because of its non-linearity for the forward pass and it has an easily calculated derivative for the backward pass. Differentiability is necessary in order to remove the effect of the activation function during the backward pass.

**Figure A.4** shows a typical two layer back prop network. This network is initialized with random weights that will be trained for a particular training set. The objective of the training process is to reduce the error between an output vector and the desired target vector. This training problem can be written as

\[ J = ||\text{target} - \text{out}||^2 \]  

Eqn A.3

where \( J \) is the objective function to be minimized. If the partial derivative of this function is taken with respect to weight changes, Eqn A.3 becomes
\[
\frac{\partial J}{\partial w_{jk}} = \Sigma_k (\text{target}_k - \text{out}_k)^2 - 2 \Sigma_k (\text{target}_k - \text{out}_k) \frac{\partial \text{out}_k}{\partial w_{jk}} \quad \text{Eqn A.4}
\]

Equation A.4 shows that the back prop learning algorithm is a gradient descent method since weight changes are made in the direction that minimizes the error. Since output, \(\text{out}_k\), is a function of the weights entering cell \(k\) from the hidden layer \(j\), and the activation of the associated cell in the hidden layer, the derivative of the output with respect to the weights can be calculated as the derivative of the activation function multiplied by the hidden layer activation.

\[
\frac{\partial \text{out}_k}{\partial \text{Net}_k} = \text{out}_k (1 - \text{out}_k) \cdot \frac{\partial \text{out}_k}{\partial w_{jk} z_j} \quad \text{Eqn A.5}
\]

Substituting Eqn A.5 into Eqn A.4 the minimization function can be rewritten as

\[
\frac{\partial J}{\partial w_{jk}} = -2(\text{target}_k - \text{out}_k) \text{out}_k (1 - \text{out}_k) z_j \quad \text{Eqn A.6}
\]

The term \((\text{target}_k - \text{out}_k)\) of Eqn A.6 indicate the direction of subsequent weight adjustments and are often referred to as the \textit{delta} value. Magnitude of change is controlled by a step size also referred to as a learning rate. These terms combine to specify the amount of weight adjustment that will reduce the error. Weight updating is then
\[ w_k(n+1) = w_k(n) + \eta \Delta z_j \]  

Eqn A.7

where

\[ \eta \] is the learning rate.

and

\[ \Delta = (\text{target}_k - \text{out}_k) \text{out}_k (1 - \text{out}_k) \]

Back prop learning basically determines the error and finds out which cells contributed to the error. These cells are identified by their activation level in the hidden layer, \( z_j \). Connection weights associated with these values are then adjusted in proportion to this activation strength. For the hidden layer, the delta term is different because there is no definitive value or target vector for determining an error. Hidden layer error is calculated by summing the product of the output layer delta values by their associated connection strength. In effect, this error value behaves like a step size in that if the cell contributed to a large error, there will be a large adjustment. Again, the activation function is removed by applying its derivative, which again controls weight changes based on its output level. Thus the delta value for the hidden layer becomes

\[ \Delta_j = \text{out}_j (1 - \text{out}_j) \sum_k \Delta_k w_{jk} \]  

Eqn A.8

and weight changes in the input layer are then

\[ w_j(n+1) = w_j(n) + \eta \Delta z_i \]  

Eqn A.9
where \( z_i \) is now the input vector applied to layer \( i \).

The back prop algorithm is probably one of the more widely used and analyzed network paradigms in the field of neural networks but it is not without its drawbacks. Since the method relies on gradient descent, local minimum problems may occur. An associated problem is the selection of an appropriate step size (learning rate). These factors may adversely affect network training and learning. Refinements and modifications have been made by different researchers to the basic algorithm to lessen the adverse impact of these problems (see Wasserman [85] for a more detailed discussion of both problems and solutions). However, the algorithm is rather robust and efficient if properly configured. In this research, several other network paradigms were pursued for various applications, but they failed to deliver the performance and stability of back prop.

Numerous researchers dislike the concept of supervised learning as used in back prop. They question the availability of target patterns for developing biological organisms. To this group, an unsupervised approach to learning seems more plausible. In 1984, Kohonen [46] developed an unsupervised learning algorithm for a one layer

![Figure A.5 Unsupervised Learning Algorithm](image)

Figure A.5 Unsupervised Learning Algorithm
network. Cells of the network consist of randomized weight vectors with a length equal to the length of the input vector. Input vectors are compared to each cell vector of the network to determine a closest match. Similarity is computed by taking the dot product of the input vector with the network's cell vectors. The closest vector is declared the winner and the components of the vector, considered its weights, are adjusted to be made more similar to the input vector. Through continual presentation of patterns, each cell of the network will be adjusted to correspond to a particular input or set of input vectors. This process is shown in Figure A.5.

In Figure A.5 weight vectors $w_1$, $w_2$, and $w_3$ initially have random values. On applying input $I_i$, the dot product of this input vector and each weight vector is calculated. Closest match is the greatest dot product which indicates the highest similarity in direction. In Figure A.5, the winning weight vector would be $w_j$. Only this vector is selected for updating. Weight updating is governed by the following expression

$$w_i(n+1)=w_i(n)+\eta(I_i-w_i)z_i \quad \text{Eqn A.10}$$

where

$\eta$ is the learning rate

and

$z_i = 1$ if $i$ is the winning cell, otherwise $z_i = 0$.

Each component of the winning weight vector is adjusted in the direction of the input. After training, weight vectors will be associated with a region or domain which they
represent. Any subsequent presentation of an input pattern that resides in this domain will be classified as belonging to the class represented by the weight vector. In this way the network has developed its own classifications without supervision.

There are numerous variations and applications of the unsupervised approach discussed above. Kohonen has improved on his original work by allowing the weight vectors of the network to self organize in a topological manner by updating the winning cells neighbors and not just a single winning cell Kohonen [47]. Grossberg has incorporated the processes of unsupervised learning into his Adaptive Resonance Theory (ART) [35]. Both of these networks are briefly mentioned in the text of this work.

While the foregoing discussion presents the basics of supervised and unsupervised learning it provides little insight into the mathematical basis for how the networks achieve some of their more remarkable performance capabilities. Specifically, networks are capable of generalizing an appropriate output from similar or incomplete input. From the discussion of network paradigms, this capability must reside in the process of weight adjustment. Network functions are based in weight space and network algorithms are developed to adjust these weights. Networks are also adaptable to changing input conditions. Again, this feature is achieved through weight adjustment, where weight adjustments are continually made to adapt the system to the current input. Thus the generalization and adaptability capabilities of neural network lie in the dynamics of weight adjustment.
In the adjustment of weights, network stability must be maintained. Network stability is pivotal for the proper implementation of a network paradigm and must be established before a network is accepted in the neural network research community. One of the significant contributions to the field of neural networks occurred in 1983 by Cohen and Grossberg who proved the stability of a large class of networks, [15]. Basically, they developed the Liapunov function for the set of networks governed by the dynamic equation

\[
\frac{dx_i}{dt} = a_i(x_i) \left[ (b_i(x_i) - \sum c_{ij} d_j(x_j)) \right]
\]

Eqn A.11

where \( a_i \) is called the amplification function, \( b_i \) is a self signal function, \( c_{ij} \) is a connection strength matrix, and \( d_j \) specifies other signal functions such as feedback from adjacent cells. In [34], Cohen and Grossberg show that the model of Eqn A.11 can be used to show the stability of several network models including the McCulloch-Pitts model. Stability analysis based on the Liapunov function provides the basis for the generalization capability of networks through the concept of content addressable memory.

The Liapunov function can be thought of as layers of energy planes within the system. Each memorized item, identified by its associated weight vector, results in a minimum in the energy plane. When a new input is first applied, it will be located somewhere in the energy field. For a properly trained network, this new point will exist in the basin of attraction of an existing energy minimum. Through network propagation,
this new input vector can be considered as being drawn down into the closest minimum and associated with the category corresponding to that minimum. In this way, items are retrieved in parallel by content instead of the more traditional computer approach of being sequentially retrieved by address. Further, less than complete or noisy input patterns can be classified correctly by being within the basin of attraction of the correct category. This content addressable memory feature of neural networks enables networks to generalize a proper response from less than complete input.

Most of this tutorial has concentrated on the dynamics of learning and memory through establishing a particular set of weights. There is another set of networks that propagate responses based on network parameters and not network weights which are used for short term memory processes. Short term memory defines the process in which an input pattern is stored only temporarily. Presentation of new patterns destroys any knowledge of previous patterns. A biological analogy is the human vision system in which visual data are only stored temporarily if they are not needed. Otherwise, the visual image decays over time and is forgotten; short term memory. While, long term memory is associated with Hebbian learning, short term memory can be characterized by the selection of parameters in an appropriate network dynamic equation. In the text of
this work, a short term memory network was implemented for the visual system of the navigation system.

While neural networks have received much recent attention, the real power of neural networks has yet to be realized. As with the biological system, artificial neural networks are designed for parallel implementation. Processing power of neural systems is most often calculated in connections per second as opposed to von Neuman machine classifications of instructions per second. Parallel processing can significantly increase the number of connections per second. Full parallel implementation has yet to be achieved but research is being conducted at several locations to achieve this capability. For more information concerning parallel implementation the reader is directed to the 1988 DARPA study, [18].

A.3 Of Bird and Brain

A schism has developed in the the field of neural networks as to the importance of retaining all or some of the biological aspects of neurons. In one sense it can be considered sacrosanct to neglect certain aspects of biological neurons or add obvious artificial augmentations to improve performance. On the other hand, explanations for some biological phenomenon can be overly complex or insufficient to be included in a design. In this vein it may be better to engineer certain functions into the neuron to achieve better performance.
Arguments in support of the engineering approach mention that airplanes do not "look" like birds, Freeman [23]. Whatever is sufficient or engineered that performs the task is acceptable. A point can be made that early attempts at flight, through mimicking the action of birds, failed miserably. Success was finally achieved when engineering solutions to the mechanical aspects of flight were discovered. While this may seem like an argument in favor of an engineering solution, the similarities between bird and plane are too numerous to rule out the importance of the biological system in the development of flight. Much was learned and adapted into the mechanical design of airplanes from the flight of birds. Assuredly even man's desire to attempt flight was first inspired by birds. In this same vain, man is now attempting to mimic the actions of humans through the development of intelligent robotic systems.

Most of the previous intelligent system work has been centered on purely engineered solutions with little regard for the internal biological processes that comprise intelligent action. The direction of this work is that it may be best to approach the solution from the biological perspective. The attempt is not so much to mimic the exact biological aspect in terms of the actual processes of neurons or wet ware, but more of the organizational and functional processes of the biological system. In one regard, the effort is to achieve the psychologically observed phenomenon by artificially designing the biological system.

It is difficult to delineate the point in which the artificial neural network becomes
too artificial. It is the direction of this research to maintain the sense of a biological network as much as possible. In many instances during this research it would have been much easier to abandon biological implementations for engineered neurons or established algorithmic solutions to a problem. However, by attempting to maintain a reasonable degree of feasibility, the domain and limitations of the approach can be determined. As with the airplane, success will probably, eventually reside in the combination of biology and engineering. It has yet to be shown whether artificial neural networks will provide the basis for the solution or whether the approach is like the early attempts at flight; merely a bunch of arms flapping.
/* cellnet.c*/

/* This program takes the laser data from the catch array of the 
original laser program and detects jump edges or range discontinuities. 
The occurrence of these features indicates the presence of an edge. The 
discontinuity between the adjacent values is proportional to the distance. 
These discontinuities are then calculated using a cellular neural network. 
The output from this operation is also displayed. Retinal patches are 
also classified and displayed in binary as to their traversable state. This 
information can then be written to a file for subsequent processing of 
the response sub-system. */

#include <stdio.h>
#include <graph.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <math.h>

#define ROW 128
#define COL 128
#define YSIZE 32
#define XSIZE 32
#define SHIFT 16
#define R 1.0
#define C 1.0
#define I 1.5 User prompted for value */
#define SENS 1
#define BUFSIZE (ROW*2)
#define ABS(u) u>0 ? u : -u
#define THRESH 1020
#define BBASE 280 /*Divisible by 8 for cursor psn.*/
#define YBASE 98 /*Divisible by 14 for cursor psn.*/
#define TRUE 1

/*COLORS*/
#define BLACK 0
#define BLUE 1
#define GREEN 2
#define CYAN 3
#define RED 4
#define MAGENTA 5
#define BROWN 6
#define WHITE 7
#define GREY 8
#define LT_BLUE 9
#define LT_GREEN 10
#define LT_CYAN 11
#define LT_RED 12
#define LT_MAGENTA 13
#define YELLOW 14
#define INT_WHITE 15

short scrn_width, scrn_height;
struct videoconfig video_info;
short videomode = _HRESBW;

unsigned char range[ROW][COL];
unsigned char buf[BUFSIZE];
float cell[YSIZE][XSIZE];
float input[YSIZE][XSIZE]; /* used for extracting patches */

main(argc,argv)
int argc;
char *argv[];
{
    int inhandle, clr, n=0, num_read, class;
    short i, j, k, m, edge_x, edge_y, cell_x, cell_y;
    float hthresh, vthresh, I;
    char buffer[80];

    if(argc != 2)
        { printf("Format: C> cellnet file.pic\n"); exit(-1); }
    if(( inhandle = open(argv[1], O_RDONLY | O_BINARY))<0)
        { printf("Can't open file %s.\n", argv[1]); exit(-1); }
    for(j=0; j<ROW; j++)
    {
        n=0;
        read(inhandle, buf, BUFSIZE);
        for(i=0; i<COL; i++)
            { /* *(*reflect+j)+i=buf[n++]; */
                n++;
                *(*range+j)+i=buf[n++];
        }
    }
}
}  
close(inhandle);

init_graphics();

while(hthresh!=0)
{
    _clearscreen(_GCLEARSCREEN);
    _settextposition(1,0);
    _outtext("Please enter Horizontal Threshold (float) or 0 to stop. ->");
    scanf("%f", &htresh);
    fflush(stdin);
    if(htresh!=0)
    {
        _settextposition(2,0);
        _outtext("Please enter Vertical Threshold (float) ->");
        scanf("%f", &vthresh);
        fflush(stdin);

        _settextposition(3,0);
        _outtext("Please enter value for current I ->");
        scanf("%f", &I);
        fflush(stdin);

        _clearscreen(_GCLEARSCREEN);
        set_up_screen(&edge_x, &edge_y, &cell_x, &cell_y, hthresh, vthresh, I);
        make_grid(400,300,20,7,RED);
        find_edges(htresh, vthresh, edge_x, edge_y);

        /* Extract a Patch */

        for(i=0; i<7/*ROW/YSIZE*/; i++)
            for(j=0; j<7/*COL/XSIZE*/; j++)
            {
                if(kbhit())
                {
                    fflush(stdin);
                    break;
                }
                _settextposition(16,70);
sprintf(buffer,"Patch #%d",(i*7+j));
_outtext(buffer);
for(k=0;k<YSIZE;k++)
for(m=0;m<XSIZE;m++)
  input[k][m]=(float)range[(i*SHIFT+k)][(j*SHIFT+m)];
cell_edge(i);
show_cell_edge( (i*SHIFT+cell_y), (j*SHIFT+cell_x) );
calc_output(&class);
if(class)
  set_grid(i,j,400,300,20,7,RED);
}
}/* end of if */
getch();
}/* end of while */
terminate_graphics();
}/* end of main */

set_up_screen(edge_x, edge_y, cell_x, cell_y, horz, vert, I)
short *edge_x, *edge_y, *cell_x, *cell_y;
float horz, vert, I;
{
  short h_offset, v_offset;
  char buffer[80];

  h_offset=(scrn_width-(2*COL))/3;

  *edge_x=h_offset;
  *cell_x=2*h_offset+COL;

  v_offset=(scrn_height-(2*ROW))/3;

  *edge_y=v_offset;
  *cell_y=v_offset;

  _setcolor(RED);
  _rectangle(_GBORDER,(edge_x)-1,(edge_y)-1,
            (edge_x)+COL, (edge_y)+ROW);
  _settextposition((v_offset/14-3),h_offset/8);
  _outtext("Range");
  _settextposition((v_offset/14-2),h_offset/8);
  _outtext("Discontinuities");
  _settextposition((v_offset/14+2),0);
  _settextcolor(YELLOW);
  sprintf(buffer, "Vert = %3.2fn", horz);
_outtext(buffer);
_settextcolor(BLUE);
sprintf(buffer, "Horz = %3.2f\n", vert);
_outtext(buffer);
_settextcolor(RED);
_outtext("Combined");

_rectangle(_GBORDER,(*cell_x)-1,(*cell_y)-1,
    (*cell_x)+COL, (*cell_y)+ROW);
_settextposition((v_offset/14-3),(*cell_x)/8);
_settextcolor(WHITE);
_outtext("Cellular");
_settextposition((v_offset/14-2),(*cell_x)/8);
_outtext("Network");
_settextposition((v_offset/14+1),((cell_x)+COL+16)/8);
sprintf(buffer,"I = %3.2f A", I);
_outtext(buffer);
}

/* this is the main cellular net sub routine. The cloning
templates are specified here as fbackt for feedback template
and ffwdt for feedforward template */

cell_edge(I)
float I;
{
    int i, j, t, u, v, w, tstart, tstop, ustart, ustop;
    int vstart, vstop, wstart, wstop, cnt;
    float fbackt_p[3][3], ffwdt_p[3][3];
    float fback, ffwd;
    double cntt;
    char buffer[40];

    fbackt_p[0][0]=0.0;
    fbackt_p[0][1]=1.0;
    fbackt_p[0][2]=0.0;
    fbackt_p[1][0]=1.0;
    fbackt_p[1][1]=2.0;
    fbackt_p[1][2]=1.0;
    fbackt_p[2][0]=0.0;
    fbackt_p[2][1]=1.0;
    fbackt_p[2][2]=0.0;

    ffwdt_p[0][0]=-.25;
    ffwdt_p[0][1]=-.25;
    ffwdt_p[0][2]=-.25;
ffwdt_p[1][0]=.25;
ffwdt_p[1][1]= 2.0;
ffwdt_p[1][2]=.25;
ffwdt_p[2][0]=.25;
ffwdt_p[2][1]=.25;
ffwdt_p[2][2]=.25;

for(cnt=0;cnt<5;cnt++)
{
    for(i=0; i<YSIZE; i++)
    {
        tstart=vstart=0;
        tstop=vstop=3;
        /* ffwdt_p[1][1]=2.0;/*
        if(i==0) { tstart=vstart=1;ffwdt_p[1][1]=1.25; }
        else if(i==(YSIZE-1))
            { tstop=vstop=2; ffwdt_p[1][1]=1.25; }
        for(j=0;j<XSIZE;j++)
        {
            ustart=wstart=0;
            ustop=wsstop=3;
            if(i==0 || i==(YSIZE-1)) ffwdt_p[1][1]=1.25;
            if(i==0) { ustart=wstart=1;ffwdt_p[1][1]=1.25;
                          if(i==0 || i==(YSIZE-1)) ffwdt_p[1][1]=.75; }
            else if(i==(XSIZE-1)) { ustop=wsstop=2; ffwdt_p[1][1]=1.25;
                          if(i==0 || i==(YSIZE-1)) ffwdt_p[1][1]=.75; }

            fback=0.0;
            if(cnt!=0)
            /* get initial conditions spread across entire array */
            {
                for(t=tstart;t<tstop;t++)
                    for(u=ustart;u<ustop;u++)
                    {
                        cntr=.5*(fabs((double)(cell[i-1+t][j-1+u]+1.0))
                        -fabs((double)(cell[i-1+t][j-1+u]-1.0)));
                        fback+=fbackt_p[t][u]*(float)cntr;
                    }
            }
    }
    ffwd=0.0;
    for(v=vstart;v<vstop;v++)
        for(w=wstart;w<wstop;w++)
        {
ffwd+=ffwdt_p[v][w]*input[i-1+v][j-1+w];
    }
}

_settextposition(15,70);
sprintf(buffer,"Iter %d", cnt+1);
_outtext(buffer);
}

show_cell_edge(y,x)
short y,x;
{
    short i,j;
    double cntr;

    for(i=0;i<YSIZE;i++)
        for(j=0;j<XSIZE;j++)
            {
                cntr=.5*( fabs( (double)(cell[i][j]+1.0) )
                        - fabs( (double)(cell[i][j]-1.0) ) );
                if(cntr<-.99)
                    {
                        _setcolor(YELLOW);
                        _setpixel(x+j,y+i);
                        cell[i][j]=1.0;
                    }
                else
                    {
                        _setcolor(GREEN);
                        _setpixel(x+j,y+i);
                        cell[i][j]=1.0;
                    }
            }
}

calc_output(val)
int *val;
{
    int i, j, sum=0;

    for(j=0;j<YSIZE;j++)
        for(i=0;i<YSIZE;i++)
sum+=1*(int)cell[j][i];

if(sum>THRESH)/*if its clear*/
    *val=0;/*already GREEN background*/
else
    *val=RED;
}

find_edges(hthresh, vthresh, edge_x, edge_y)
float hthresh, vthresh;
short edge_x, edge_y;
{
    short clr, j, i;
    float horz, vert;

    for(j=0;j<(ROW-1);j++)
        for(i=0;i<(COL-1);i++)
            {
                horz=(((float)(range[j][i])-(float)(range[j][i+1]))
                      /(float)(range[j][i]));
                _setcolor(GREEN);
                if(horz>hthreshll-horz>vthresh)
                    _setcolor(YELLOW);
                _setpixel(i+edge_x, j+edge_y);
            }

    for(j=0;j<(ROW-1);j++)
        for(i=0;i<(COL-1);i++)
            {
                vert=(((float)(range[j][i])-(float)(range[j+1][i]))
                      /(float)(range[j][i]));

                if(vert>vthreshll-vert>vthresh)
                    {
                        _setcolor(RED);
                        if(_getpixel(i+edge_x,j+edge_y)==YELLOW)
                            _setcolor(BLUE);
                        _setpixel(i+edge_x, j+edge_y);
                    }
            }

    init_graphics(void)
    {

_getvideoconfig(&video_info);
switch(video_info.adapter)
{
    case _MDPA:
        printf("This program needs a graphics adapter.\n");
        exit(0);
    case _CGA:
        videomode = _HRESBW;
        scrn_width=640;
        scrn_height= 200;
        break;
    case _EGA:
        videomode = _ERESCOLOR;
        scrn_width = 640;
        scrn_height= 350;
        break;
    case _VGA:
        videomode = _VRES16COLOR;
        scrn_width = 640;
        scrn_height = 480;
        break;
}
_setvideomode(videomode);
}

terminate_graphics(void)
{
    _setvideomode(_DEFAULTMODE);
}

make_grid(x,y,offset,num,clr)
short x,y,offset,num,clr;
{
    int i,j;

    _setcolor(RED);
    _moveto(x,y);
    _lineto(x,offset*num+y);
    _lineto(offset*num+x,offset*num+y);
    _lineto(offset*num+x,y);
    _lineto(x,y);

    for(j=1;j<num;j++)
    {
    
_moveto(x, offset*j+y);
_lineto(x+offset*num, offset*j+y);
}

for(j=1;j<num;j++)
{
    _moveto(offset*j+x,y);
    _lineto(x+offset*j, offset*num+y);
}

set_grid(rw, col, x, y, offset, num, clr)
int rw, col;
short x, y, offset, num, clr;
{
    x=x+offset*col;
    y=y+offset*rw;

    _setcolor(clr);
    _rectangle(_GFILLINTERIOR, x, y, x+offset, y+offset);
}
Appendix C -- Pattern Generation Routine Source Code

/*getpat.c*/

/* This program is used to generate training data for back prop neural networks used in this research. Use of this program is primarily for wander network training. Patterns are generated randomly and the user inputs a proper motor response. This input/output pair is then written to a file in the following form where the first 7x7 matrix represents the environmental input and the next row of 5 values correspond to the use input:

---Pattern # 1
(
.1 .1 .1 .1 .1 .1 .1
.1 .9 .9 .1 .1 .1 .1
.1 .9 .9 .9 .1 .1 .1
.1 .1 .1 .9 .9 .9 .1
.1 .1 .1 .1 .9 .9 .1
.1 .9 .9 .1 .1 .1 .1
.1 .9 .9 .1 .1 .1 .1
.1 .1 .1 .9 .9 .9 )
---Pattern # 2
(
.1 .1 .1 .1 .1 .1 .1
.1 .1 .1 .1 .1 .1 .1
.1 .9 .9 .1 .1 .1 .1
.1 .9 .9 .9 .1 .1 .1
.1 .1 .1 .9 .9 .9 .1
.1 .1 .1 .1 .9 .9 .1
.1 .9 .9 .1 .1 .1 .1
.1 .1 .1 .9 .9 .9 )

There are three additional sections of this appendix that list the source code for accompanying files required for the operation of the pattern generation routine. These files are also used in the compilation of the evaluation program of Appendix D. In section C.1, the file mcommon.h specifies structures types and defines constants. Section C.2 is the graphic routines used in this research. Section C.3 list the routines used for specifying screen updates. It is appropriately labelled moves.c. */

#include <stdio.h>
#include <graph.h>
#include <math.h>
#include "mcommon.h"
#include <stdlib.h>
#include <sys/types.h>

#include <sys/timeb.h>
#include <time.h>
#include <conio.h>
#include "common.h"
#include "weights.h"
#include "layer.h"
#include "net.h"
#include "netio.h"

extern Net *B_create_net();
extern Net *B_free_net();
extern int N_reset_wts();
extern void P_prop_input();
extern void PA_initialize();
extern void D_initialize();
extern Sint C_float_to_Sint();
extern float C_Sint_to_float();
extern void sys_init_rand();

extern struct seg_table segment[];
extern int num_segments;

/******************************************************************************
/* Global Variables */
/******************************************************************************
float fullcon_Inputs[INPUT_SIZE];
float fullcon_Outputs[OUTPUT_SIZE];
Net *fullcon_NetPtr;

int goal, goal_row, goal_col, obst[NUM_IN_ROW];

static char *game[] =
{ "Select Watch Mode",
  "No Goal",
  "Goal",
};

main()
{

    FILE *infile;
int curpos=1, game_type;
int input=49, target=6, num_div=7;
int code, total, test=0;
int num=3, i, go=1;  /* num= number of items in menu */

void fullcon_initialize();
void fullcon_propagate();
void fullcon_cleanup();

_clearscreen(_GCLEARSCREEN);
display_menu(game, num, curpos);
while(go)
{
    update_menu(game, num, curpos);
    code = getcode();
    switch(code)
    {
        case U_ARROW:
            if( curpos>0 ) --curpos;
            else curpos=num-1; break;
        case D_ARROW:
            if( curpos<num-1 ) ++curpos;
            else curpos = 1; break;
        case INSERT:
            go=0;
            if(curpos==1)
            {
                game_type=0;
                /*set_nrmfunc();*/
                goal=0;/*No goal*/
            }
            else if(curpos==2)
            {
                game_type=0;
                /*set_nrmfunc();*/
                goal=1;/*Goal active*/
                fullcon_initialize();
                goal_col=0;
            }
            break;
    }
    break;
}

initialize_graphics();
target = 6;
num_segments = input = 49;
segment[0].save = 1;
setsegvisibility((input + 1), 1, RED);
setsegvisibility(0, 1, GREEN);
curpos = 1;
while (TRUE) {
    display_game_menu(curpos);
    code = getcode();
    switch (code) {
        case U_ARROW:
            if (curpos > 1) --curpos;
            else curpos = num - 1; break;
        case D_ARROW:
            if (curpos < num - 1) ++curpos;
            else curpos = 1; break;
        case INSERT:
            action(curpos, num_div, input, target, game_type);
            break;
        default:
            break;
    }
}
/* end of main */
display_game_menu(pos)
int pos;
{
    static char *games[] =
        { "GAME MENU",
          "Start",
          "Exit",
        };
    char buffer[80];
    int j;
    _settextposition(2, 5);
    _settextcolor(WHITE);
    sprintf(buffer, "\n\n\n", *games);
    _outtext(buffer);
    _settextcolor(RED);
for(j=1; j<3; j++)
{
    if(j==pos)
        _settextcolor(CYAN);
    sprintf(buffer, "%v%s
", *(games+j));
    _outtext(buffer);
    _settextcolor(RED);
}

action(pos, num_div, input, target, game_type)
int pos, num_div, input, target, game_type;
{
    if(pos==2)
    {
        terminate();
        exit(-1);
    }
    else
        go_game(num_div, input, target, game_type);
}

go_game(num_div, input, target, game_type)
int num_div, input, target, game_type;
{
    int code, second_code, points, score=0, quit, first, write_it=1;
    int i, pat_num=1, move=0, no_move, go=1, curpos, first_code;
    time_t tstart, tstop;
    FILE *gfptr, *stdfptr;
    char buffer[80];
    int extra=2, total, in_loop;

    total=extra+input+target;
    
    if(!goal)
        if( (stdfptr = fopen("standpat.iop", "w")) == NULL)
            { printf("can't open standpat.iop"); exit(0); }

    if(goal)
        if( (gfptr = fopen("goalpat.iop", "w")) == NULL)
            { printf("can't open goalpat.iop"); exit(0); }

    clearall(input);
_settextposition(25,2);
_outtext("Enter the # of patterns desired ->");
cscanf("%d", &quit);
fflush(stdin);
sprintf(buffer, "%d", quit);
_settextposition(25,2);
_outtext(" ");
set_up_board(num_div, input);
sleep(2.0);
_settextposition(25,2);
_outtext("START");
beep(1);
_settextposition(25,2);
_outtext(" ");
getche();
while (pat_num <= quit && go)
{
  first=1;
cursos=input+1;
in_loop=1;

  resetsegvisibility();
  if (goal) turn_goal(ON, num_div);
  setsegvisibility(0, 1, GREEN);
  setsegvisibility((input+1), 1, RED);
  segment[input+1].save=0;
  for (i=(input+2); i<=(input+7); i++)
  {
    setsegvisibility(i, 0, BLUE);
    segment[i].save=0;
  }

  if (goal)
    calc_output_fullcon(input, target);

  while (in_loop && go)
  {
    display_extras(cursos, input, target, extra, pat_num);
    code = getcode();
    switch (code)
    {
      case U_ARROW:
        if (segment[cursos].save != 1)
          setsegvisibility(cursos, 0, RED);
if(curpos==total)
    curpos=input+1;
else curpos++;
setsegvisibility(curpos, 1, RED);
break;
case D_ARROW:
    if(segment[curpos].save != 1)
        setsegvisibility(curpos, 0, RED);
    if(curpos==input+1)
        curpos=total;
    else curpos--;
    if(curpos==total)
        curpos=input+1;
    setsegvisibility(curpos, 1, RED);
    break;
case R_ARROW:
    if(segment[curpos].save != 1)
        setsegvisibility(curpos, 0, RED);
    if(curpos==total) curpos=input+1;
    else curpos++;
    setsegvisibility(curpos, 1, RED);
    break;
case L_ARROW:
    if(segment[curpos].save != 1)
        setsegvisibility(curpos, 0, RED);
    if(curpos==input+1);
    else curpos--;
    setsegvisibility(curpos, 1, RED);
    break;
case INSERT:
    if(curpos<=(input+target)) /*if it's not the end */
    {
        if(segment[curpos].save==1) /*if deleting an item */
        {
            segment[curpos].save=0;
            setsegvisibility(curpos, 0, RED);
        }
        else
        {
            segment[curpos].save=1;
            if(first)
            {
                first=0;
                first_code=curpos-input;
} }
else if(curpos==(input+target+1)) /* action on "ENTER" */
in_loop=0;
else if(curpos==total) /* action on "EXIT" */
go=0;
break;
default:
break;
}
}

if(go)
{
/*write input pats to file do not save pat if it's a back */
/* or if there was a collision */
if(segment[input+1].save!=1 && write_it)
{
/* copy input patterns to training files */
if(!goal)
{
  fprintf(stderr,"--Pattern # %d \n", pat_num);
  fprintf(stderr, "("");
  for(i=1;i<=input;i++)
  {
    if((i-1)%7==0)
      fprintf(stderr, "\n");
    if(segment[i].save==0)
      fprintf(stderr, "1 ");
    else
      fprintf(stderr,".9 ");
  }
}
if(goal)
{
  fprintf(gfptr,"--Pattern # %d \n", pat_num);
  fprintf(gfptr, "(");
  for(i=1;i<=num_div;i++)
  {
    if( (i==goal_row) || (i==1 && goal_row<1) ||
        (i==num_div && goal_row>num_div) )
      fprintf(gfptr,".9 ");
    else

    fprintf(gfptr,".1 ");
}
}
}
setsegvisibility((input+1), 1, RED);
for(i=(input+2); i<(input+7); i++)
    setsegvisibility(i,0,BLUE);

code=first_code;
if(goal) turn_goal(OFF, num_div);
points=make_move(code, input, num_div, 0, 0);

if(points<-1)
{
    write_it=0;
    _settextposition(25,5);
    _outtext("CRASHED!!!-select another command");
    sleep(2.0);
    _settextposition(25,5);
    _outtext(" ");
}
else if(points==-1)
{
    if(goal) fprintf(gfptr, "Evasive action required");
    else fprintf(stdfptr, "Evasive action required");
    points=0;
    pat_num--;
}
else if(points>=0)
{
    write_it=1;
    update_pat(pat_num,quit);
if(!goal)
{
    fprintf(stdfptr, "\n");
    for(i=2;i<=target;i++)
    {
        if(segment[input+i].save==1)
            fprintf(stdfptr,".9 ");
        else
            fprintf(stdfptr,".1 ");
    }
    fprintf(stdfptr, " ) \n");
}
if(goal)
{
    fprintf(gfptr, "\n");
    for(i=2;i<=target;i++)
    {
        if(segment[input+i].save==1)
            fprintf(gfptr, ".9 ");
        else
            fprintf(gfptr, ".1 ");
    }
    fprintf(gfptr, " ) \n");
    pat_num++;
}
}
resetsegvisibility();
_settextposition(25,5);
_outtext("PATTERN GENERATION FINISHED");
sleep(2.0);
_settextposition(25,5);
_outtext(" ");
fcloseall();
remove_window();

}/* end of main */

update_pat(num_pat, max)
int num_pat, max;
{
    char buffer[80];
display_extras(pos, input, target, extra, pat_num)
int pos, input, target, extra, pat_num;
{
    static char *extras[] =
    { "Input New Patterns",
      "Enter Pattern",
      "Finish and Close", }
    char buffer[80];
    int j;
    _settextposition(2,5);
    _settextcolor(WHITE);
    sprintf(buffer, "%s\n\n", *extras);
    _outtext(buffer);
    _settextcolor(RED);

    for(j=1; j<(extra+1); j++)
    {
        if(j==(pos-input-target))
            _settextcolor(CYAN);
        sprintf(buffer, "%s\n", *(extras+j) );
        _outtext(buffer);
        _settextcolor(RED);
    }
    _settextposition(8,2);
    _settextcolor(WHITE);
    sprintf(buffer, "current pos= %d \n", pos);
    _outtext(buffer);
    _settextposition(9,2);
    sprintf(buffer, "Pattern # %d ", pat_num);
    _outtext(buffer);
remove_window()
{
    _settextposition(2,5);
    _outtext(" \\
    "
    _outtext(" \\
    "
    _outtext(" \\
    "
    _outtext(" \\
    _outtext(" \\
    _settextposition(9,2);
    _outtext(" \\
    "
    _outtext(" \\
    "
}

calc_output_fullcon(input, target)
int input, target;
{
    int i;
    short x, y, row, col;
    char buffer[80];

    for(i=1;i<=input;i++)
    {
        if(segment[i].save==0)
            fullcon_Inputs[i-1]=.1;
        else
            fullcon_Inputs[i-1]=.9;
    }
    fullcon_propagate();

    for(i=0;i<target-1;i++)
    {
        x=(short) segment[input+2+i].xsmax;
        y=(short) segment[input+2+i].ysmax;
        col=x/8;
        row=y/14;
        _settextposition(row,col+1);
        sprintf(buffer, "%3.2f ", fullcon_Outputs[i]);
        _outtext(buffer);
    }
}
void fullcon_initialize()
{
  int i;

  /* call initialization code */
  fullcon_NetPtr = NULL;
  sys_init_rand();
  PA_initialize();
  D_initialize();

  /* create network */
  fullcon_NetPtr = B_create_net(1, "fullcon.net");
  fullcon_NetPtr->use_biases = TRUE;
  fullcon_NetPtr->num_inputs = INPUT_SIZE;
  fullcon_NetPtr->num_outputs = OUTPUT_SIZE;

  /* reset weights and the input, output arrays */
  N_reset_wts(fullcon_NetPtr, "fullcon.pwt", PORTABLE_FORMAT);
  for (i = 0; i < INPUT_SIZE; i++)
    fullcon_Inputs[i] = 0.0;
  for (i = 0; i < OUTPUT_SIZE; i++)
    fullcon_Outputs[i] = 0.0;
}

void fullcon_propagate()
{
  int i;
  Layer *input, *output;
/*----------------------------* /
/* get pointers to network input and output */
/*----------------------------* /
input = fullcon_NetPtr->input_layer;
output = fullcon_NetPtr->output_layer;

/*----------------------------* /
/* load input values; propagate network */
/*----------------------------* /
for (i = 0; i < INPUT_SIZE; i++)
    input->node_outputs[i] = C_float_to_Sint(fullcon_Inputs[i]);
P_prop_input(fullcon_NetPtr);

/*------------------------*/
/* setup output values */
/*------------------------*/
for (i = 0; i < OUTPUT_SIZE; i++)
    fullcon_Outputs[i] = C_Sint_to_float(output->node_outputs[i]);

} /* fullcon_propagate */

/*----------------------------*/
/* call this routine once to free the space */
/* used by the network. */
/*----------------------------*/
void fullcon_cleanup()
{
    B_free_net(fullcon_NetPtr);
}

} /* fullcon_cleanup */
/* mcommon.h */

/* mcommon.h is the common header file for the programs eval.c and getpat.c. It is called "m"common to distinguish it from common.h of the NETS program. Common structures and definitions are provided in this file */

#include <stdio.h>
#include <graph.h>
#include <math.h>

#define U_ARROW 72
#define D_ARROW 80
#define R_ARROW 77
#define L_ARROW 75
#define INSERT 13

#define DBL_LFT 32 /* space bar */
#define LEFT 104 /* h */
#define FWD 106 /* j */
#define RIGHT 107 /* k */
#define DBL_RGT 108 /* l */
#define BACK 117 /* u */

#define INPUT_SIZE 49
#define OUTPUT_SIZE 5
#define OUTPUT 5
#define GOAL_INPUT 14
#define GOAL_HIDDEN 25
#define GOAL_OUTPUT 5
#define OBST_INPUT 5
#define OBST_HIDDEN 9
#define OBST_OUTPUT 6
#define THRESHOLD 8 /* random number threshold for declaring an obst */
#define NUM_IN_ROW 1 /* max # of obstacles in row */

#define TRUE 1
#define FALSE 0
#define ON 1
#define OFF 0
#define RAD_TO_DEG 57.29578
#define DEG_TO_RAD 0.0174533
#define SQR(u) u*u
/* COLORS
   for text and line drawings use short integers which are identified
   in the header file by name
   _setbcolor in text mode uses L_xxxxx(where xxx is the color name)
   _setbcolor in graphics mode uses predefined values _XXX
*/

#define BLACK 0
#define BLUE 1
#define GREEN 2
#define CYAN 3
#define RED 4
#define MAGENTA 5
#define BROWN 6
#define WHITE 7
#define GRAY 8
#define LT_BLUE 9
#define LT_GREEN 10
#define LT_CYAN 11
#define LT_RED 12
#define LT_MAGENTA 13
#define YELLOW 14
#define BR_WHITE 15

#define L_BLACK 0L
#define L_BLUE 1L
#define L_GREEN 2L
#define L_CYAN 3L
#define L_RED 4L
#define L_MAGENTA 5L
#define L_BROWN 6L
#define L_WHITE 7L
#define L_GRAY 8L
#define L_LT_BLUE 9L
#define L_LT_GREEN 10L
#define L_LT_CYAN 11L
#define L_LT_RED 12L
#define L_LT_MAGENTA 13L
#define L_YELLOW 14L
#define L_BR_WHITE 15L

enum commands {drw2, mv2, txt, circle, stop}; /* Drawing Commands */
struct display_cmds  /* Storage structure */
    {  /* for drawing commands, */
        enum commands cmds;  /* start and end points */
        int xstart, xend, ystart, yend;  /* pointers to text */
        char *ptrtext;  /* string and next struct */
        struct display_cmds *ptrnext;
    }

struct display_cmds *ptrthis, *ptrnew;  /* pointers to the above */

struct seg_table
    {  /* Segment Table */
        struct display_cmds *ptr;  /* with pointer to */
        int vis, used, save;  /* drawing commands */
        int xsmin, xsmax, ysmmin, ysmmax;  /* for that segment */
        float area;
        int las_on, las_off, las_data;
    }

struct vector
    {  /* to */
        float x,y,z,w;
    }
Appendix C.2 -- "mcommon.c"

/*mcommon.c*/

/* mcommon.c is the graphics package for the programs getpat.c and eval.c. The "m" in the name, "m"common.c is to distinguish this file from the the common.c file of the common.c file of the NETS package. Structure used in the software package are defined in mcommon.h. All screen objects are defined as graphic segments. All drawing commands are associated with the segment's number and stored in the global structure "segment table". All routines of the graphics package then have access to these structures. The screen displays were primarily set to an EGA configuration of 640x350 for both EGA and VGA to retain the spacing of the screen. The software package requires a file of viewing parameters to be present for operation; the file is called viewp. This file is read by the routine "initialize graphics()" which specifies the order and type of parameters required. */

#include <stdio.h>
#include <graph.h>
#include <math.h>
#include "mcommon.h"
#include <stdlib.h>
#include <sys/types.h>
#include <sys/timeb.h>
#include <time.h>
#include <malloc.h>

/************** GLOBAL VARIABLES ************/
float xwleft, xwright, ywbottom, ywtop; /* Window Globals */
float xvleft, xvright, yvbottom, yvtop; /* Viewport Globals */
float cur_x, cur_y, map_l, map_h;     /* Current Pen position conversion factors*/
int transform_flag, translate_flag, gi, open_seg; /*flags*/
int scrn_width, scrn_height;

struct seg_table segment[99]; /* Established for 99 */
/* Segments */

struct vector cur;
float N_par[4][4];
float t[4][4];
int num_segments;
getcode()
{
    int key;
    while(TRUE)
    {
        key=getch();
        if(key==13)
            return(INSERT);
        else if(key==0)
            return(getch());
    }
}

resetsegvisibility()
{
    int i;

    for(i=1; i<=num_segments; i++)
        if(segment[i].save==1)
            setsegvisibility(i,1,RED);
}

/****************************OBJECTS FOR DRAWING******************************/
grid(div)
int (div);
{
    int i;
    float length;
    length=div*10.0;

    for(i=0; i<=div; i++)
    { 
        move2( (float)i*10.0, 0.0, 0.0);
        draw2( (float)i*10.0, 0.0, length);
    }
    for(i=0; i<=div; i++)
    {
        move2( 0.0, 0.0, (float)i*10.0);
        draw2( length, 0.0, (float)i*10.0);
    }
}
box(i,j,num)
int i,j,num;
{
    float x,z,side=10.0;
    
    x=(float)i * side;
    z=(float)j * side;

    move2( x, 0.0, z);
    draw2( x, side, z);
    draw2( x+side, side, z);
    draw2( x+side, side, z+side);
    draw2( x, side, z+side);
    draw2( x, side, z);
    move2( x+side, 0.0, z);
    draw2( x+side, side, z);
    move2( x+side, 0.0, z+side);
    draw2( x+side, side, z+side);
    move2( x, 0.0, z+side);
    draw2( x, side, z+side);
    move2( x+side, 0.0, z);
    draw2( x, 0.0, z+side);
    move2( x, 0.0, z);
    draw2( x+side, 0.0, z+side);
}

initialize_graphics()
{
    int num_div, cnt;
    struct vector VUP;
    struct vector VRP;
    struct vector VPN;
    struct vector COP;
    struct vector VRP_prime;
    struct vector u;
    struct vector v;
    struct vector start;
    struct vector end;
    struct vector tmpvec;
    struct vector cross_product();
    struct vector subtract_vec();
    float dot_product();
    struct vector multiply_vec();
float u_min, u_max, v_min, v_max, F, B;
float x_cur, y_cur, scalar;
int i, j, object_end, go;
int xstart, xend, ystart, yend;
FILE *obj_ptr, *view_ptr;
cur.w=start.w=end.w=1.0;

if( (view_ptr= fopen("viewp","r"))==NULL)
{
    printf("Cannot open viewing angle file\n");
    exit(-1);
}

fscanf(view_ptr,"%f %f %f", &VRP.x, &VRP.y, &VRP.z);
fscanf(view_ptr,"%f %f %f", &VPN.x, &VPN.y, &VPN.z);
fscanf(view_ptr,"%f %f %f", &VUP.x, &VUP.y, &VUP.z);
fscanf(view_ptr,"%f %f %f", &COP.x, &COP.y, &COP.z);
fscanf(view_ptr,"%f %f %f", &u_min, &u_max, &v_min, &v_max);
fscanf(view_ptr,"%f %f", &F, &B);
fscanf(view_ptr,"%d", &num_div);

VRP.w=VPN.w=VUP.w=COP.w=1.0;

/*
Routines to calculate parameters from input data
*/

/****************************************************************************
 * Procedure to set initial rot matrix. These values
 * will first be used to calculate DOP prime. They are
 * being placed directly into the N_par matrix even though
 * the matrix does not contain the initial translation.
 * The initial translation matrix will be pre-multiplied
 * later. It will be easier to use the identity command
 * and set the values for the translation into the changing
 * global matrix. The matrix also accounts for a change
 * between right hand and left hand coordinate systems, i.e.
 * the matrix T_rl has been post multiplied into the rotation
 * matrix.
****************************************************************************/

N_par[0][2]=VPN.x;        /* multiplied by -1 for -z axis */
N_par[1][2]=VPN.y;
N_par[2][2]=VPN.z;

normalize(&VPN);
scalar = dot_product(VPN,VUP);
tmpvec.x = scalar * VPN.x;
tmpvec.y = scalar * VPN.y;
tmpvec.z = scalar * VPN.z;
tmpvec.w=1.0;

v = subtract_vec(VUP,tmpvec);
v.w=1.0;
u = cross_product(VPN,v);
u.w=1.0;

N_par[0][0]=u.x;
N_par[0][1]=v.x;
N_par[1][0]=u.y;
N_par[1][1]=v.y;
N_par[2][0]=u.z;
N_par[2][1]=v.z;
N_par[0][3]=N_par[1][3]=N_par[2][3]=0.0;
N_par[3][0]=N_par[3][1]=N_par[3][2]=0.0;
N_par[3][3]=1.0;

/****************************
* Set initial translation matrix. Translate VRP to origin.
****************************/
identity(t);
t[3][0]=-(VRP.x+COP.x);
t[3][1]=-(VRP.y+COP.y);
t[3][2]=-(VRP.z+COP.z);

/**** Combine these matrices into N_par by pre-multiplying only
the necessary terms****/

for(i=0;i<3;i++)
  for(j=0;j<3;j++)
    N_par[3][i]+=t[3][j]*N_par[j][i];

VRP_prime=multiply_vec(VRP,N_par); /* = VRP x T x ROT */

/**** Calculate shear_z matrix and post-multiply. Place results in
the N_par matrix****/

identity(t);
t[2][0]=-(VRP_prime.x+.5*(u_max+u_min))/VRP_prime.z;
t[2][1]=-(VRP_prime.y+.5*(v_max+v_min))/VRP_prime.z;
multiply_matrix(N_par,t);

/*** Calculate scale matrix and post multiply. Place results in the
 the N_par matrix/***

identity(t);

t[0][0]=2* (VRP_prime.z/(u_max-u_min)*VRP_prime.z + B));
t[1][1]=2* (VRP_prime.z/(v_max-v_min)*VRP_prime.z + B)));
t[2][2]=1.0/(VRP_prime.z+B);

multiply_matrix(N_par,t);

/* Set perspective and multiply into matrix */

identity(t);

t[2][3]=VRP_prime.z/(VRP_prime.z+B);

multiply_matrix(N_par,t);

/*** Viewing parameters now set, i.e. N_par. Shift to object ****/

set_laser_data(); /*determine laser measurements*/

cur.x=cur.y=cur.z=0.0;

window(0.0, 1.0, 0.0, 1.0);
viewport(0.0, 1.0, 0.0, 1.0);
initialize();
openseg(0);
grid(num_div);
closeseg();
setsegvisibility(0,1, GREEN);

cnt=1;
for(i=0; i<num_div; j++)
  for(i=0; i<num_div; i++)
    {
      openseg(cnt);
      box(i,j, num_div);
      closeseg();
      setsegvisibility(cnt++, 1, RED);
    }

num_segments = cnt-1;     /* number of squares*/

for(i=1; i<cnt; i++)
setsegvisibility(i,0,RED);

set_upcircles(i);

}

set_upcircles(i)
int i;
{
    int j;
    short x_cent, y_cent;
    short offset=77;
    short rad=12;

    x_cent=(short)scrn_width/2+offset-33;
    y_cent=(short)scrn_height*.9-20;

    _setcolor(BLUE); /* border for circle -- linked to filling routine*/
    _ellipse(_GFILLINTERIOR,x_cent-rad,y_cent-rad, x_cent+rad, y_cent+rad);
    openseg(i);
        ptrthis->cmds=circle;
        ptrthis->xstart=x_cent-rad;
        ptrthis->ystart=y_cent-rad;
        ptrthis->xend=x_cent+rad;
        ptrthis->yend=y_cent+rad;
        ptrnew= (struct display_cmds *)
            malloc(sizeof(struct display_cmds));
        ptrthis->ptrnext=ptrnew;
        ptrthis=ptrnew;
    closeseg();

    x_cent=(2*offset);
    y_cent=(offset-20);

    for(j=1;j<6;j++) /* for five circles */
    {
        _ellipse(_GFILLINTERIOR,x_cent-rad,y_cent-rad, x_cent+rad, y_cent+rad);
        openseg(i+j);
            ptrthis->cmds=circle;
            ptrthis->xend=x_cent-rad;
            ptrthis->yend=y_cent-rad;
            ptrthis->xstart=x_cent+rad;
            ptrthis->ystart=y_cent+rad;
            ptrnew= (struct display_cmds *)

malloc(sizeof(struct display_cmds));
    ptrthis->ptrnext=ptrnew;
    ptrthis=ptrnew;
closesegO;

    x_cent+=offset;
}
}
******************************************************************************
/* initialize()-- initialize turns the graphics package "ON". */
/* It primarily sets initial values and flags. It calls */
/* three routines in the process. */
******************************************************************************
initialize()
{
    struct videoconfig video_info;
    short videomode = _HRESBW;

    xwright=xwright=yvtop=ywtop=1.0;

    _getvideoconfig(&video_info);
    switch(video_info.adapter)
    {
        case _MDPA:
            printf("This program needs a graphics adapter.\n");
            exit(0);
        case _CGA:
            printf("This program needs an EGA adapter.\n");
            exit(0);
        case _EGA:
            videomode = _ERESCOLOR;
            scrn_width=640;
            scrn_height=350;
            break;
        case _VGA: /* only use ega mode to maintain proportions */
            videomode = _ERESCOLOR;
            scrn_width=640;
            scrn_height=350;
            break;
    }
    _setvideomode(videomode);
}
terminate()
{
    _setvideomode(_DEFAULTMODE);
}

move2(x,y,z) /* Move command updates current psp */
float x,y,z;
{
    struct vector dest;
    struct vector multiply_vec();

    cur.x=x;
    cur.y=y;
    cur.z=z;

    dest=multiply_vec(cur, N_par);
    dest.x/=dest.w;
    dest.y/=dest.w;
    dest.y=1.0-dest.y;

    convert_win_view_scrn(&dest.x, &dest.y);

    ptrthis->cmds=mv2;
    ptrthis->xend=dest.x;
    ptrthis->yend=dest.y;
    ptrnew= (struct display_cmds *)
        malloc(sizeof(struct display_cmds));
    ptrthis->ptrnext=ptrnew;
    ptrthis=ptrnew;
}

draw2(x,y,z) /* DRAW from current to x,y,z */
float x,y,z;
{
    struct vector dest;
    struct vector multiply_vec();

    cur.x=x;
    cur.y=y;
    cur.z=z;

    dest=multiply_vec(cur, N_par);
    dest.x/=dest.w;
    dest.y/=dest.w;
dest.y=1.0-dest.y;

convert_win_view_scrn(&dest.x, &dest.y);

ptrthis->cmds=drw2;
ptrthis->xend=dest.x;
ptrthis->yend=dest.y;
ptrnew= (struct display cmds *)
malloc(sizeof(struct display_cmds));
ptrthis->ptrnext=ptrnew;
ptrthis=ptrnew;
}

/******************************************************************************
 * openseg(i)-- opens a segment in the segment table for acceptance
 * of an image. Only one segment can be opened at a time and an
 * image cannot be placed into the system unless a segment is
 * opened. Openseg(i) sets the segment table flag used for that
 * particular value of i. The function allocates memory space
 * for the first drawing command to be placed in the structure.
 * It then points to this location with the global pointer
 * "ptrthis" indicating it is pointing to the current structure.
 * Openseg(i) also checks to insure that a stored segment is not
 * written over by checking the segment used flag. A global value
 * for i is set to be used by the closeseg() command.
 ******************************************************************************/

openseg(i)
int i;
{
    gi=i;
    if(open_seg==1)
        return(-3);
    if(gi<0||gi>99)
        return(-4);
    if(segment[i].used==1)
        return(-2);

    segment[i].used=1;
    open_seg=1;

    segment[i].xmin=segment[gi].xmin=scrn_width;
    segment[i].ptr=(struct display_cmds *)
malloc(sizeof(struct display_cmds));
ptrthis=segment[i].ptr;

return(0);
}

/*********************************************************
* closeseg()--closes the current open segment. It first places the
* end drawing command flag or stop command in the current open
* drawing command structure. It then cycles back through the linked
* list of drawing commands to find the extent of the segment. Each
* drawing command stores the limits or extent of the command.
* The maximum and minimum values are found for the segment and a
* segment area is calculated. The global flag gi is only used to
* to reset the pointer to drawing commands to conduct the extent
* search.
***********************************************************/
closeseg()
{

    if(open_seg!=1)
        return(-6);

    ptrthis->cmds=stop;    /* flag last entry */

    ptrthis=segment[gi].ptr;

    while(ptrthis->cmds!=stop)
    {
        if(ptrthis->xstart>ptrthis->xend)
        {
            if(ptrthis->xstart>segment[gi].xmax)
                segment[gi].xmax=ptrthis->xstart;
            if(ptrthis->xend<segment[gi].xmin)
                segment[gi].xmin=ptrthis->xend;
        }
        else
        {
            if(ptrthis->xstart<segment[gi].xmin)
                segment[gi].xmin=ptrthis->xstart;
            if(ptrthis->xend>segment[gi].xmax)
                segment[gi].xmax=ptrthis->xend;
        }
    }
}
if(ptrthis->ystart>ptrthis->yend)
  {
    if(ptrthis->ystart>segment[gi].ysmax)
      segment[gi].ysmax=ptrthis->ystart;
    if(ptrthis->yend<segment[gi].ysmin)
      segment[gi].ysmin=ptrthis->yend;
  }
else
  {
    if(ptrthis->ystart<segment[gi].ysmin)
      segment[gi].ysmin=ptrthis->ystart;
    if(ptrthis->yend>segment[gi].ysmax)
      segment[gi].ysmax=ptrthis->yend;
  }
ptrthis=ptrthis->pnxt;
}

segment[gi].area=(segment[gi].xsmax-segment[gi].xsm)
  *(segment[gi].ysmax-segment[gi].ysmin);
open_seg=0;
}

******************************************************************************
* setsegvisibility(i,x)--sets the segment specified in i to the visibility
* condition (ON,OFF) specified in x. It first checks for a valid
* segment number, if there are no open segments, and whether or not
* the segments specified already has the visibility desired. It
* then proceeds through the drawing commands specified by the segment
* number. "ptrthis->cmds" specifies the drawing action. There
* are only three drawing commands stored and an end of file flag.
* The command will specify which drawing routine will be called
* and the associated parameters are sent to the routine. "drw2"
* uses "dda", "pt2" uses "putpix", and "txt" uses "puttext".
* After reaching the stop flag, the procedure shows the new image
* and sets the visibility flag to match "x". Images and text are
* erased by sending the "OFF" flag to the drawing commands which
* overwrites the image with "white" pixels.
******************************************************************************
setsegvisibility(i,x,clr)
int i,x,clr;
{  
  struct xycoord far _moveto();
  short far _lineto();
if(open_seg==1)
    return(-1);

if(i<0||i>98)
    return(-4);

if(segment[i].used != 1)
    return(-5);

/*  if(segment[i].vis==x)
    return(-13);  */

ptrthis=segment[i].ptr;

if(ptrthis->cmds==stop)
    return(-7);
if(x==0)
    _setcolor(BLACK);
else
    _setcolor(clr);
update_las_data(i,x);
while(ptrthis->cmds!=stop)
{
    if(ptrthis->cmds==drw2)
        _lineto(ptrthis->xend,
                ptrthis->yend);

    else if(ptrthis->cmds==mv2)
        _moveto(ptrthis->xend,
                ptrthis->yend);

    else if(ptrthis->cmds==circle)
    {
        if(x==0) _setcolor(BLUE);
        _ellipse(_GFILLINTERIOR,(short)ptrthis->xstart,
                 (short)ptrthis->ystart, (short)ptrthis->xend,
                 (short)ptrthis->yend);
    }
    ptrthis=ptrthis->ptnext;
}

segment[i].vis=x;
return(0);
/**
 * convert_win_view_scrn(x,y)--converts x,y coordinates from the world
 * coordinate system through the viewport specifications to screen
 * coordinates. Conversion values for window to viewport mapping
 * are pre-calculated in the window and viewport commands and
 * set in map_l and map_h.
 **/ convert_win_view_scrn(x,y)
 float *x,*y;
 {
   *x=(map_l*(x-xwleft)+xleft)*scrn_width;
   *y=(map_h*(y-ywbottom)+ywbottom)*scrn_height;
 }

/**
 * window(x1,x2,y1,y2)--specifies the viewing window in the world
 * coordinate system. Conversion factors for window to viewport
 * mapping are calculated whenever this function is called to
 * keep the conversion factors current. Also sets global window
 * variables.
 **/ window(x1,x2,y1,y2)
 float x1,x2,y1,y2;
 {
   xwleft=x1;
   xwright=x2;
   ywbottom=y1;
   ywtop=y2;
   map_l=((xwright-xwleft)/(xwright-xwleft));
   map_h=((ywtop-ywbottom)/(ywtop-ywbottom));
 }

/**
 * viewport(x1,x2,y1,y2)--specifies the viewing viewport in the normalized
 * device coordinates. Conversion factors for window to viewport
 * mapping are calculated whenever this function is called to
 * keep the conversion factors current. Also sets global viewport
 * variables.
 **/ viewport(x1,x2,y1,y2)
 float x1,x2,y1,y2;
 {
xvleft=x1;
xright=x2;
yvbottom=y1;
yvtop=y2;

map_l=((xvright-xvleft)/(xright-xvleft));
map_h=((yvtop-yvbottom)/(yvtop-yvbottom));
}

*******************************************************************************/

*******************************************************************************/

float dot_product(v1, v2)
struct vector v1, v2;
{
  float answer;

  answer = v1.x*v2.x + v1.y*v2.y + v1.z*v2.z;
  return(answer);
}

*******************************************************************************/

*******************************************************************************/

struct vector cross_product(v1, v2)
struct vector v1, v2;
{
  struct vector answ;

  answ.x = v1.y*v2.z - v1.z*v2.y;
  answ.y = v1.z*v2.x - v1.x*v2.z;
  answ.z = v1.x*v2.y - v1.y*v2.x;
  answ.w = 1.0;

  return(answ);
}

*******************************************************************************/
* subtract_vec(v1,v2)-- subtracts one vector from the other.
* Returns a vector structure to the calling program.

struct vector subtract_vec(v1,v2)
struct vector v1, v2;
{
    struct vector answ;

    answ.x = v1.x - v2.x;
    answ.y = v1.y - v2.y;
    answ.z = v1.z - v2.z;

    return(answ);
}

/*********************************************/
* normalize(v1)--Normalizes a vector specified by the pointer v1.
* Result is returned to address specified by pointer.

normalize(v1)
struct vector *v1;
{
    float length;
    double d_length;

    d_length = (double) dot_product(*v1,*v1);
    length = sqrt(d_length);

    v1->x=(v1->x/length);
    v1->y=(v1->y/length);
    v1->z=(v1->z/length);
}

/*********************************************/
* multiply_vec(v1,m)-- Multiplies a matrix by a vector structure.
* The matrix is a 4x4 and the vector is a 1x4, thus the
* vector is pre-multiplied. The function returns a vector
* structure to the calling program.

struct vector multiply_vec(v1,m)
struct vector v1;
float m[4][4];
{
    int i,j;
float tmp=0.0;
struct vector answ;

for(i=0;i<4;i++)
{
    tmp = v1.x*m[0][i] + v1.y*m[1][i] + v1.z*m[2][i]
         + v1.w*m[3][i];

    if(i==0)
        answ.x=tmp;
    else if(i==1)
        answ.y=tmp;
    else if(i==2)
        answ.z=tmp;
    else if(i==3)
        answ.w=tmp;
}
return(answ);
}

="/***********************************************************************
* multiply_matrix(a,b)--a is given as an address and indicates
* where the results of this matrix multiplication will be stored
* after calling this function. This function is called
* with multiply_matrix(a,b);
*  
***********************************************************************/

multiply_matrix(a,b)
float a[4][4], b[4][4];
{
    float answ[4][4];
    int i,j,k;

    for(i=0;i<4;i++)
    {
        for(j=0;j<4;j++)
        {
            answ[i][j] = 0.0;
            for(k=0;k<4;k++)
                answ[i][j] += a[i][k]*b[k][j];
        }
    }
    for(i=0;i<4;i++)
for(j=0;j<4;j++)
    a[i][j]=answ[i][j];

/**************************************************************************
 * identity()-- used to reset a matrix specified by "t" back
 * to the identity matrix.
**************************************************************************/
identity(t)
float t[4][4];
{
    int i,j;

    for(j=0;j<4;j++)
        for(i=0;i<4;i++)
            {
                if(i==j)
                    t[j][i]=1.0;
                else
                    t[j][i]=0.0;
            }

    sleep(x)
float x;
{
    time_t tstart, tstop;
    time(&tstart);
    time(&tstop);
    while( difftime(tstop, tstart) < x)
        time(&tstop);
}

display_menu(titles, num, pos)
char *titles[];
int    num, pos;
{
    int j;
    char buffer[80];

    _settextwindow(10,25,20,65);/*shadow box*/
    _setbkcolor(L_BLACK);
    _clearscreen(_WINDOW);
_settextwindow(5,20,15,60); /* window and color for main menu */
_setbkcolor(14L);
_clearscreen(_GWINDOW);
_settextcolor(BLUE);
_settextposition(2,10);
sprintf(buffer, "\%s\n\n", *titles);
_outtext(buffer);
_settextcolor(BLACK);

for(j=1; j<num; j++)
{
    if(j==pos)
        _settextcolor(CYAN);
    sprintf(buffer, "\%s\n", *(titles+j));
    _outtext(buffer);
    _settextcolor(BLACK);
}

update_menu(titles, num, curpos)
char *titles[];
int num, curpos;
{
    char buffer[80];
    int j;
    _settextposition(3,10);
    sprintf(buffer,"\n");
    _outtext(buffer);
    for(j=1; j<num; j++)
    {
        if(j==curpos)
            _settextcolor(WHITE);
        sprintf(buffer, "\%s\n", *(titles+j));
        _outtext(buffer);
        _settextcolor(BLACK);
    }
}

get_val_from_menu(titles, num, skip)
char *titles[];
int num, skip;
{
    int curpos=1, code;
display_menu(titles, num, curpos);
while(TRUE)
{
    update_menu(titles, num, curpos);
    code = getcode();
    switch(code)
    {
        case U_ARROW:
            if (curpos>0) --curpos;
            else curpos=num-1; break;
        case D_ARROW:
            if (curpos<num-1) ++curpos;
            else curpos = 1; break;
        case INSERT:
            if(curpos==0 || curpos>num || curpos==skip)
                break;
            else
                return(curpos);
        default:
            break;
    }
}

set_laser_data()
{
    int num_div=7, i, j, cnt=1;
    double theta, phi, theta_incr, phi_incr, h_scan=60.0, v_scan=60.0;
    double on_dist, off_dist, conv_angle;
    float conv_on, conv_off;

    theta_incr=h_scan/(double) num_div;
    phi_incr=v_scan/(double) num_div;

    theta=h_scan-(theta_incr/2.0);
    for(i=0;i<num_div;i++)
    {
        phi=90.0-(v_scan/2.0)+(phi_incr/2.0);
        for(j=0;j<num_div;j++)
        {
            conv_angle=sin(phi*DEG_TO_RAD)*cos(theta*DEG_TO_RAD);
            on_dist=5.0/conv_angle;
            off_dist=6.0/conv_angle;
        }
    }
}
conv_on=(float)on_dist*256.0/16.0;
conv_off=(float)off_dist*256.0/16.0;

segment[cnt].las_on=(int)conv_on;
segment[cnt].las_off=(int)conv_off;

cnt++;
phi+=phi_incr;
}
theta+=theta_incr;

}

update_las_data(i,x)
int i, x;
{
    int input=49, num_div=7;
    short row=0, col=35, row_offset, col_offset;
    char buff[10];

    _settextcolor(WHITE);
    if(i<input || i<1)
        return(0);
    row_offset=i/num_div;
    col_offset=(i%num_div)*4-4;
    if(i%num_div==0)
    {
        row_offset-=1;
        col_offset=num_div*4-4;
    }
    _settextposition( (row+row_offset), (col+col_offset) );
    if(x==0)
        segment[i].las_data=segment[i].las_off;
    else
        segment[i].las_data=segment[i].las_on;
    sprintf(buff,"%3.3d",segment[i].las_data);
    _outtext(buff);
Appendix C.3 -- "moves.c"

/* moves.c specifies all of the common features of both
getpat.c and eval.s that are not graphics related. As its name indicates,
the subroutines that control the graphical moves of the simulation are
contained in this program. */

#include <stdio.h>
#include <graph.h>
#include <math.h>

#include "mcommon.h"
#include <stdlib.h>
#include <sys\types.h>
#include <sys\timeb.h>
#include <time.h>

#define INTERVAL 1.5
#define MV_DELAY .5

extern struct seg_table segment[];
extern int goal, goal_row, goal_col, obst[];

/************************ time_segment used for response from player ******/
time_segment()
{
    int key=FWD; /* key 2 (FWD) default for no response */
time_t tstart, tstop;

time(&tstart);
time(&tstop);
while( difftime(tstop, tstart) < INTERVAL )
{
    time(&tstop);
    while( kbhit() )
        return(getch());
}
return(key);
}

make_move(code, input, num_div, delay, test)
int code, input, num_div, delay, test;
{
if(code==4||code==FWD)
{
    setsegvisibility((input+1), 0, BLUE);
    setsegvisibility((input+4), 1, RED);
    if(delay) sleep(MV_DELAY);
    if(test) getch();
    return(move_fwd(num_div, input));
}
else if(code==5||code==RIGHT)
{
    setsegvisibility((input+1), 0, BLUE);
    setsegvisibility((input+5), 1, RED);
    if(delay) sleep(MV_DELAY);
    if(test) getch();
    return(move_right(num_div, input, 1));
}
else if(code==6||code==DBL_RGT)
{
    setsegvisibility((input+1), 0, BLUE);
    setsegvisibility((input+6), 1, RED);
    if(delay) sleep(MV_DELAY);
    if(test) getch();
    return(move_right(num_div, input, 2));
}
else if(code==3||code==LEFT)
{
    setsegvisibility((input+1), 0, BLUE);
    setsegvisibility((input+3), 1, RED);
    if(delay) sleep(MV_DELAY);
    if(test) getch();
    return(move_left(num_div, input, 1));
}
else if(code==2||code==DBL_LFT)
{
    setsegvisibility((input+1), 0, BLUE);
    setsegvisibility((input+2), 1, RED);
    if(delay) sleep(MV_DELAY);
    if(test) getch();
    return(move_left(num_div, input, 2));
}
else if(code==1||code==BACK)
{
    if(delay) sleep(MV_DELAY);
    if(test) getch();
return(move_bkwrds(num_div, input));
}
else if(code==0)
    return(0); /* stay put */
else
    return(0);
}

hold(start)
time_t *start;
{
time_t h_start, h_stop;
int code=1;

time(&h_start);
_settextposition(24,5);
_outtext("Strike any key to continue");
getch();
_settextposition(24,5);
_outtext(".");
time(&h_stop);

*start+=difftime(h_stop, h_start);
}

move_frwds(num_div, input)
int num_div, input;
{
    int i, points=0;

    i=(num_div/2);
    if(segment[input-i].save==1) /* check for crash with head-on box*/
        { 
crash(); 
        return(-2);
    }
    if(goal)
    {
    if(goal_col==num_div-1)
    {
        goal_col=0;
        if(i+1==goal_row)
        {
points+=50;
beep(1);
}
}
else
goal_col++;

for(i=input; i>input-num_div; i--)  
/*cnt front row for points*/
if(segment[i].save==1)
{
    setsegvisibility(i,0,RED);
    segment[i].save=0;

    points++;
}
for(i=input-num_div; i>0; i--)  
/* move each element up */
{
    if(segment[i].save==1)
    {
        setsegvisibility(i,0,RED);
        segment[i].save=0;
        segment[i+num_div].save=1;
    }
}  
new_row(num_div);
return(points);
}

move_bkwrdd(num_div, input)
int num_div, input;
{
    int i, points=0;

    for(i=1; i<=num_div; i++)  
/* remove back row */
    {
        if(segment[i].save==1)
        {
            setsegvisibility(i,0,RED);
            segment[i].save=0;
        }
    }
    if(goal)
goal_col--;
}
for(i=0;i<NUM_IN_ROW;i++)
    obst[i]=0;

for(i=num_div+1; i<=input; i++)
    { if(segment[i].save==1)
        { setsegvisibility(i,0,RED);
          segment[i].save=0;
          segment[i-num_div].save=1;
        }
    }
    return(0);
}

move_right(num_div, input, num)
int num_div, input, num;
{
    int i, points=0;

    i=(num_div/2-1);
    if(segment[input-i].save==1 || (segment[input-i+1].save==1 && num==2) )
        { crash();
          return(-2);
        }
    if(goal)
        {
            if(goal_col==num_div-1)
                { goal_col=0;
                  if(i+num+2==goal_row)
                      { points+=50;
                        beep(1);
                      }
                }
            else
                { goal_col++;
                  goal_row=num;
                }
        }
    for(i=0;i<NUM_IN_ROW;i++)
if(obst[i]!=0)
{
    obst[i]-=num;
    if(obst[i]<1)
        obst[i]+=num_div;
}
for(i=input; i>input-num_div; i--)
    if(segment[i].save==1)
    {
        setsegvisibility(i,0,RED);
        segment[i].save=0;
        points++;
    }
for(i=input-num_div; i>0; i--)
{
    if(segment[i].save==1)
    {
        setsegvisibility(i,0,RED);
        segment[i].save=0;
        if(i%num_div==1 || (i%num_div==2 && num==2) )
            segment[i+2*num_div-num].save=1;
        else
            segment[i+num_div-num].save=1;
    }
}
new_row(num_div);
return(points);
}
move_left(num_div, input, num)
int num_div, input, num;
{
    int i, points=0;

    i=(num_div/2)+1;
    if(segment[input-i].save==1 || (segment[input-i-1].save==1 && num==2) )
    {
        crash();
        return(-2);
    }
    if(goal)
    {
        if(goal_col==num_div-1)

{
  goal_col=0;
  if(i-num==goal_row)
  {
    points+=50;
    beep(1);
  }
  }
else
{
  goal_col++;
  goal_row+=num;
}
}
for(i=input; i<input-num_div; i--)
  if(segment[i].save==1)
  {
    setsegvisibility(i,0,RED);
    segment[i].save=0;
    points++;
  }
for(i=0;i<NUM_IN_ROW;i++)
  if(obst[i]!=0)
  {
    obst[i]+=num;
    if(obst[i]>num_div)
      obst[i]-=num_div;
  }
for(i=input-num_div; i>0; i--)
{
  if(segment[i].save==1)
  {
    setsegvisibility(i,0,RED);
    segment[i].save=0;
    if(i%num_div==0 || (i%num_div==num_div-1 && num==2 ))
      segment[i+num].save=1;
    else
      segment[i+num_div+num].save=1;
  }
}
new_row(num_div);
return(points);
new_row(num_div)
int num_div;
{
    unsigned seed;
    struct timeb time_buffer;
    int num, num1, i, new_cnt=0, cnt=0, skip=1, set=0;

    ftim(&time_buffer);
    seed=time_buffer.millitm;
    srand(seed);

    for(i=0;i<NUM_IN_ROW;i++)
        if(obst[i]!="0")
            {
                segment[obst[i]].save=1;
                if(obst[i]==num_div)
                    segment[1].save=1;
                else
                    segment[obst[i]+1].save=1;
                obst[i]=0;
                cnt++;
            }

    for(i=1; i<=num_div; i++)
    {
        num1=num=rand();
        num/=10;
        num*=10;
        num1-=num;
        if(goal & goal_col==0 & i==1)
            {
                if(num1>num_div) num1=(num1-num_div);
                else if(num1<1) num1=1;
                while( (num1<=num_div) & set!=1 )
                    {
                        if(segment[num1].save==0)
                            {
                                goal_row=num1;
                                set=1;
                            }
                        else
                            num1++;
                    }
            }
        if(num1>num_div) num1=num_div;
while(set!=1 && (num1>0))
{
    if(segment[num1].save==0)
    {
        goal_row=num1;
        set=1;
    }
    else
        num1--;
}

if(goal_row==i || goal_row==i+1)skip=0;
if(num1>THRESHOLD && new_cnt<NUM_IN_ROW && skip)
{
    segment[i].save=1; /*save obst pt*/
    obst[new_cnt]=i;    /*in global*/
    if(i<num_div)       /*in not end*/
        segment[i+1].save=1; /*fill adjacent*/
    else
        segment[1].save=1;  /*fill first*/
    i++; /*skip next block*/ /*adjacent filled*/
    new_cnt++; /*num of obst in row*/
    cnt+=new_cnt;/*old and new combined*/
}
    skip=1;
}

set_up_board(num_div, inputs)
int num_div, inputs;
{
    int i;

goal_col=1;
new_row(num_div);
resetsegvisibility();
for(i=1; i<num_div; i++)
{
    move_fwd(num_div, inputs);
    resetsegvisibility();
}
}
turn_goal(x, num_div, row, col)
int x, num_div, row, col;
{
    if(row>0 && row<=num_div)
    {
        setsegvisibility( (col*num_div+row),x,BLUE);
        _settextposition(21,20);
        _outtext(" ");
        _settextposition(21,60);
        _outtext(" ");
    }
    if(row<1)
    {
        _settextposition(21,20);
        _outtext("<-Goal Left");
    }
    if(row>num_div)
    {
        _settextposition(21,60);
        _outtext("Goal Right->");
    }
}
crash()
{
    int i;
    for(i=0; i<3; i++)
    {
        _remappalette(0, _RED);
        _remappalette(4, _BLACK);
        beep(1);
        _remappalette(4, _RED);
        _remappalette(0, _BLACK);
        beep(1);
    }
}
beep(x)
int x;
{
    int i;
    for(i=0; i<x; i++)
    printf("\a");
}
clearall(input)
int input;
{
    int i;
    for(i=1; i<=input; i++)
        if(segment[i].save==1)
            {
                setsegvisibility(i,0,RED);
                segment[i].save=0;
            }
    setsegvisibility(0,1,GREEN);
}
Appendix D -- Evaluation Program Source Code

/*eval.c*/

/*eval.c is the main operational program of this research. It is a menu driven program
that permits the user to specify up to 42 different operational modes. Like getpat.c, it
must be complied with moves.c, mcommon.c, mcommon.h and all of the NETS source
code. Because of this complicated compilation, a make file is also provided in section
D.1 of this appendix. Besides these companion files, additional network and weight files
are required for this program. Network dimensions must be in concert with the weight
files, the specifications of Appendix C.1 (mcommon.h) and the global values of this file.*/

#include <stdio.h>
#include <graph.h>
#include <math.h>

#include "common.h"
#include "weights.h"
#include "layer.h"
#include "net.h"
#include "netio.h"

#include "mcommon.h"
#include <stdlib.h>
#include <sys/types.h>
#include <sys/timeb.h>
#include <time.h>

#define INTERVAL .5 /* time between update in player mode */
#define GAME_OVER 30 /* time (secs) for game over */
#define MV_DELAY .5 /* delay to improve viewing */
#define GOAL 100
#define MISS 101
#define MISS_IT 102
#define MISS_ONCE 103

extern Net *B_create_net();
extern Net *B_free_net();
extern int N_reset_wts();
extern void P_prop_input();
extern void PA_initialize();
extern void D_initialize();
extern Sint C_float_to_Sint();
extern float C_Sint_to_float();
extern void sys_init_rand();

extern struct seg_table segment[];
extern int num_segments;

/*-----------------------------*/
/* Global Variables */
/*-----------------------------*/
float fullcon_Inputs[INPUT_SIZE];
float fullcon_Outputs[OUTPUT_SIZE];
Net *fullcon_NetPtr;
float patacon_Inputs[INPUT_SIZE];
float patacon_Outputs[OUTPUT_SIZE];
Net *patacon_NetPtr;
float random_Inputs[INPUT_SIZE];
float random_Outputs[OUTPUT_SIZE];
Net *random_NetPtr;

float goal_layer1[GOAL_INPUT][GOAL_HIDDEN];
float goal_layer2[GOAL_HIDDEN][GOAL_OUTPUT];
float goal_Inputs[GOAL_INPUT];
float goal_bias[GOAL_HIDDEN+GOAL_OUTPUT];
float goal_Outputs[GOAL_OUTPUT];
float obst_layer1[OBST_INPUT][OBST_HIDDEN];
float obst_layer2[OBST_HIDDEN][OBST_OUTPUT];
float obst_Inputs[OBST_INPUT];
float obst_bias[OBST_HIDDEN+OBST_OUTPUT];
float obst_Outputs[OBST_OUTPUT];

int goal, goal_row, goal_col, obst[NUM_IN_ROW], level=0;
int my_input=49;

int num_div=7;

static char *game[] =
    { "Select Operational Mode",
      "Player",
      "System",
      " Random",
      " Obstacle Avoid",
      " Wander -Full Connect",
      " Wander -Pat Connect "};

static char *obs[] =
{ "Include Obstacle Avoidance",
  "No Obstacle Avoidance",
  "Obstacle Avoidance Included" },

static char *goals[] =
{ "GOAL MENU",
  "No Goal",
  "Goal" },

static char *delays[] =
{ "Use Delay Between Moves",
  "No Delay",
  "Delay" },

static char *tests[] =
{ "Run in Step Mode",
  "No",
  "Yes" },

main()
{
   FILE *inphile;
   int curpos=1, game_type, delay;
   int target=6;
   int code, total, obst, go_goal, test;
   int num=7, i, go=1; /* num= number of items in menu */
   void fullcon_initialize();
   void fullcon_propagate();
   void fullcon_cleanup();
   void patcon_initialize();
   void patcon_propagate();
   void patcon_cleanup();
   void random_initialize();
   void random_propagate();
   void random_cleanup();

   _clearscreen(_GCLEARSCREEN);

   game_type=get_val_from_menu(game,7,2);
   goal=get_val_from_menu(goals,3,0)-1;
   if(game_type>3 & & game_type!=4)
      obst=get_val_from_menu(obsts,3,0)-1;
   delay=get_val_from_menu(delays,3,0)-1;
   test=get_val_from_menu(tests,3,0)-1;
if(game_type==3) randomInitialize();
if(game_type==5) fullcon_initialize();
if(game_type==6) patcon_initialize();
if(((game_type==3||game_type>4) && goal) goal_initialize();
if(obst || game_type==4) obst_initialize();
initialize_graphics();
target= 6;
um_segments=49;
segment[0].save=1;
setsegvisibility((my_input+1),1,RED);
setsegvisibility(0,1,RED);
curpos=1;
while(TRUE)
{
    display_game_menu(curpos);
code= getcode();
switch(code)
{
    case U_ARROW:
        if( curpos>1 ) --curpos;
        else curpos=num-1; break;
    case D_ARROW:
        if( curpos<num-1 ) ++curpos;
        else curpos = 1; break;
    case INSERT:
        action(curpos,target,game_type,obst,delay,test);
        break;
    default:
        break;
}
}

} /* end of main */
display_game_menu(pos)
int pos;
{
    static char *games[] =
    { "GAME MENU",
        "Start",
        "Exit", 0};
    char buffer[80];
    int j;
}
for(j=1; j<3; j++)
{
    if(j==pos)
        _settectcolor(CYAN);
    sprintf(buffer, "\t%3$s\n", *(games+j));
    _outtext(buffer);
    _settectcolor(RED);
}

action(pos, target, game_type, obst, delay, test)
int pos, target, game_type, obst, delay, test;
{
    if(pos==2)
    {
        terminate();
        exit(-1);
    }
    else
        go_game(target, game_type, obst, delay, test);
}

go_game(target, game_type, obst, delay, test)
int target, game_type, obst, delay, test;
{
    int code, points, score=0, test_cnt=0;
    int i, row, col;
    time_t tstart, tstop;

    level=0;
    clearall();
    set_up_board();
    update_score(score);
    beep(1);
    _settectposition(25,5);
    _outtext("Strike a key to begin.");
    getch();
    _settectposition(25,5);
_outtext(" ");
beep(1);

time(&tstart);
time(&tstop);

while( ( test && (test_cnt<GAME_OVER) )
    || ( !test && (difftime(tstop, tstart) < GAME_OVER) ) )
{
    time(&tstop);
test_cnt++; resetsegvisibility();
setsegvisibility(0,1,GREEN);
if(goal) turn_goal(ON, goal_row, goal_col);
setsegvisibility((my_input+1), 1, RED);
for(i=(my_input+2); i<=(my_input+6); i++)
    setsegvisibility(i,0,BLUE);
code=2;points=0;
fflush(stdin);

if(game_type==1) /* player */
    code=time_segment();
else if(game_type==3)
    code=random( obst);
else if(game_type==4)
    code=obstacle_avoid();
else if(game_type==5)
    code=wander_full_connect(goal, obst,test);
else if(game_type==6)
    code=wander_pattern(goal, obst, test);
row=goal_row; col=goal_col;
if(code==73) /* hold- page up key */
    hold(&tstart);
else
    points=make_move(code, delay,test);
fflush(stdin);
if(goal) turn_goal(OFF, row, col);

if(points<0)
{
    _settextposition(25,5);
    _outtext("CRASHED!!!-Select another command");
sleep(2.0);
    _settextposition(25,5);
_outtext(" ");
}
else
{
    score+=points;
    update_score(score);
}

_setbkcolor(_BLACK);
_settextposition(20,10);
_outtext(" ");
_settextposition(25,5);
_outtext("GAME OVER");
sleep(2.0);
}

update_score(score)
int score;
{
    char buffer[80];

    _settextposition(9,2);
    _settextcolor(WHITE);
    if(score==0)
        _outtext("SCORE=");
    else
    {
        sprintf(buffer, "SCORE= %d \n ", score);
        _outtext(buffer);
    }
}

wander_full_connect(goal, obst, test)
int goal, obst, test;
{
    int i, save_i, start,code;
    float max=0.0, wander[OUTPUT];
    short x,y,row,col;
    char buffer[20];

    for(i=1;i<=my_input;i++)
    {
        if(segment[i].save==0)
fullcon_Inputs[i-1]=.1;
else
    fullcon_Inputs[i-1]=.9;
}
fullcon_propagate();
if(test) print_output(fullcon_Outputs, 0);
for(i=0;i<OUTPUT;i++)
    wander[i]=fullcon_Outputs[i];
if(goal)
{
    for(i=0;i<num_div;i++)
    {
        if(goal_row==i+1)
            goal_Inputs[i]=.9;
        else
            goal_Inputs[i]=.1;
    }
    if(goal_row<1) goal_Inputs[0]=.9;
    if(goal_row>num_div) goal_Inputs[num_div-1]=.9;
    for(i=num_div;i<2*num_div;i++)
    {
        if(goal_col==i-num_div)
            goal_Inputs[i]=.9;
        else
            goal_Inputs[i]=.1;
    }
    goal_propagate();
    if(test) print_output(goal_Outputs, 1);
}
for(i=0;i<GOAL_OUTPUT;i++)
    fullcon_Outputs[i]=goal_Outputs[i];
if(obst)
{
    start=my_input-(num_div/2+2);
    for(i=0;i<5;i++)
        obst_Inputs[i]=segment[start+i].save;
}
obst_propagate();
if(obst_Outputs[0]> .8)
    return(BACK);
for(i=0;i<5;i++)
{
if(obst_Outputs[i+1]<.2)
obst_Outputs[i+1]=0.0000000;
fullcon_Outputs[i] *= obst_Outputs[i+1];
}
if(goal)
{
for(i=0;i<OUTPUT;i++)
    fullcon_Outputs[i] *= wander[i];
}

for(i=0;i<OUTPUT;i++)
    if(fullcon_Outputs[i] > max)
        max = fullcon_Outputs[i];
    save_i = i;
return_code(save_i);
if(wander[save_i]<.8)
{
code = dream_on(i, 0, obst, test, 1);
if(code==BACK && level==0)
{
    _settextposition(25,30);
    _outtext("Cannot Reach Goal");
    sleep(1.0);
    _outtext(" ");
    return(i);
}
else return(code);
}
else return(i);
}
dream_on(code, delay, obst, test, full)
int code, delay, obst, test, full;
{
    float sv_full_out[OUTPUT], max;
    int save_seg[50], save_i, points;
    int i, k, cnt=1, sv_goal_row, row, sv_goal_col, col;
    int cand_code, move_code, sv_move_code, miss=0;
    long bcolor;
    char buffer[10];
level++;  
_settextposition(20,10);  
sprintf(buffer,"Level %d",level);  
_outtext(buffer);  

if(full)  
{  
    for(i=0;i<OUTPUT;i++)  
        sv_full_out[i]=fullcon_Outputs[i];  
}  
else  
{  
    for(i=0;i<OUTPUT;i++)  
        sv_full_out[i]=patcon_Outputs[i];  
}  
sv_goal_row=goal_row; sv_goal_col=goal_col;  
for(i=1;i<=my_input;i++)  
    save_seg[i-1]=segment[i].save;  
bgcolor=_getbgcolor();  
_setbgcolor(_LIGHTBLUE);  

sv_move_code=move_code=cand_code=code;  

while(1)  
{  
    row=goal_row; col=goal_col;  
    points=make_move(cand_code, delay, test);  
    if(goal) turn_goal(OFF, row, col);  
    if(points>=50)  
    {  
        for(i=1;i<=my_input;i++)  
        {  
            setsegvisibility(i,0,RED);  
            segment[i].save=save_seg[i-1];  
        }  
        goal_row=sv_goal_row; goal_col=sv_goal_col;  
        reset_screen(goal,goal_row,goal_col);  
        _settextposition(20,10);  
        _setbgcolorbgcolor;  
        if(level==1)  
        {  
            level=0;  
            _settextposition(20,10);  
            _outtext(" ");  
        }  
    }  
}
return(move_code);
}
else
{
    level--;    
sprintf(buffer,"Level %d",level);
    _outtext(buffer);
    return(GOAL);
}
reset_screen(goal,goal_row,goal_col);
if(goal_col==0 || level>=7)
    cand_code=MISS_ONCE;
else if(full)
    cand_code=wander_full_connect(goal, obst, test);
else
    cand_code=wander_pattern(goal, obst, test);

if(cand_code==BACK || cand_code==MISS_ONCE )
    /*can't move from this position or missed the goal*/
{
    cnt++;

    _settextposition(21,10);
    sprintf(buffer, "Count = %d", cnt);
    _outtext(buffer);

    for(i=1;i<=my_input;i++)
    {
        setsegvisibility(i,0,RED);
        segment[i].save=save_seg[i-1];
    }
goal_row=sv_goal_row; goal_col=sv_goal_col;
reset_screen(goal,goal_row,goal_col);

    if(cnt>=5)
    {
        if(level==1)
            cand_code=MISS_IT;
        else
            return(BACK);
    }
}

for(k=0;k<cnt;k++)/*cnt starts @ 2 */
{
    if(k!=0)sv_full_out[save_i]=0.0;
    max=0.0;
    for(i=0;i<OUTPUT;i++)
    {
        if(sv_full_out[i]>max)
        {
            max=sv_full_out[i];
            save_i=i;
        }
    }
}
if(max<.2)
{
    if(level==1)
        cand_code=MISS_IT;
    else
        return(BACK);
}
if(cand_code!=MISS_IT)
    move_code=cand_code=return_code(save_i);
}
if(cand_code==GOAL || cand_code==MISS_IT)
{
    _settextposition(20,10);
    if(level==1)
    {
        for(i=1;i<=my_input;i++)
        {
            setsegvisibility(i,0,RED);
            segment[i].save=save_seg[i-1];
        }
    goal_row=sv_goal_row; goal_col=sv_goal_col;

    reset_screen(goal_row,goal_row,goal_col);
    _setbkcolor(bcolor);
    level=0;
    _outtext("   ");
    if(cand_code==MISS_IT)
    {
        _settextposition(25,30);
        _outtext("Cannot Reach Goal");
    }
}
sleep(1.0);
__outtext(" ");
return(sv_move_code);
}
return(move_code);

else
{
    level--;
sprintf(buffer,"Level %d", level);
__outtext(buffer);
return(GOAL);
}
}

reset_screen(goal, goal_row, goal_col)
int goal, goal_row, goal_col;
{
    int i;

    resetsegvisibility();
setsegvisibility(0, 1, GREEN);
if(goal) turn_goal(ON, goal_row, goal_col);
setsegvisibility((my_input+1), 1, RED);
for(i=(my_input+2); i<=(my_input+6); i++)
    setsegvisibility(i, 0, BLUE);
}

return_code(save_i)
int save_i;
{
    if(save_i==0)
        return(DBL_LFT);
    else if(save_i==1)
        return(LEFT);
    else if(save_i==2)
        return(FWD);
    else if(save_i==3)
        return(RIGHT);
else if(save_i==4)
    return(DBL_RGT);

print_output(array, clr)
float *array;
int clr;
{
    short x, y, row, col, i;
    char buffer[20];

    if(clr)
        _settextcolor(YELLOW);

    for(i=0;i<5;i++) /*5=# of movement directions */
    {
        x=(short) segment[my_input+2+i].xsmac;
        y=(short) segment[my_input+2+i].ysmac;
        col=x/8;
        row=y/14+clr;
        _settextposition(row,col+1);
        sprintf(buffer, "%3.2f", *(array+i));
        _outtext(buffer);
    }
    _settextcolor(WHITE);
}

wander_pattern(goal, obst, test)
int goal, obst, test;
{
    int i, save_i, start, code;
    float max=0.0, wander[OUTPUT];
    short x,y,row,col;
    char buffer[20];

    for(i=1;i<=my_input;i++)
    {
        if(segment[i].save==0)
            patcon_Inputs[i-1]=.1;
        else
            patcon_Inputs[i-1]=.9;
    }
patcon_propagate();
if(test) print_output(patcon_Outputs, 0);
for(i=0;i<OUTPUT;i++)
    wander[i]=patcon_Outputs[i];
if(goal)
{
    for(i=0;i<num_div;i++)
    {
        if(goal_row==i+1)
            goal_Outputs[i]=.9;
        else
            goal_Outputs[i]=.1;
    }
if(goal_row<1) goal_Outputs[0]=.9;
if(goal_row>num_div) goal_Outputs[num_div-1]=.9;
for(i=num_div;i<2*num_div;i++)
    {
        if(goal_col==i-num_div)
            goal_Outputs[i]=.9;
        else
            goal_Outputs[i]=.1;
    }
goal_propagate();
if(test) print_output(goal_Outputs,1);

for(i=0;i<GOAL_OUTPUT;i++)
    patcon_Outputs[i]=goal_Outputs[i];
}
if(obst)
{
    start=my_input-(num_div/2+2);
    for(i=0;i<5;i++)
        obst_Outputs[i]=segment[start+i].save;

    obst_propagate();
    if(obst_Outputs[0]>0.8)
        return(BACK);

    for(i=0;i<5;i++)
    {
        if(obst_Outputs[i+1]<.2)
            obst_Outputs[i+1]=0.000000;
            patcon_Outputs[i]*=obst_Outputs[i+1];
    }
if(goal)
{
    for(i=0;i<OUTPUT;i++)
        patcon_Outputs[i]*=wander[i];
}

}
for(i=0;i<OUTPUT;i++)
    if(patcon_Outputs[i]>max)
    {
        max=patcon_Outputs[i];
        save_i=i;
    }
    i=return_code(save_i);
if(wander[save_i]<.8)
    {
        code=dream_on(i, 0, obst, test, 0);
        if(code==BACK && level==0)
            {
                _settextposition(25,30);
                _outtext("Cannot Reach Goal");
                sleep(1.0);
                _outtext(" ");
                return(i);
            }
        else return(code);
    }
else return(i);
}

random(goal, obst)/*same as fullcon except untrained*/
int goal, obst;
{
    int i, save_i, start;
    float max=0.0, wander[OUTPUT];

    for(i=1;i<=my_input;i++)
    {
        if(segment[i].save==0)
            fullcon_Inputs[i-1]=.1;
        else
            fullcon_Inputs[i-1]=.9;
    }
fullcon_propagate();
for(i=0;i<OUTPUT;i++)
    wander[i]=fullcon_Outputs[i];
if(goal)
{
    for(i=0;i<num_div;i++)
    {
        if(goal_row==i+1)

            goal_Inputs[i]=.9;
        else

            goal_Inputs[i]=.1;
    }
    if(goal_row<1) goal_Inputs[0]=.9;
    if(goal_row>num_div) goal_Inputs[num_div-1]=.9;
    for(i=num_div;i<2*num_div;i++)
    {
        if(goal_col==i-num_div)

            goal_Inputs[i]=.9;
        else

            goal_Inputs[i]=.1;
    }
}

goal_propagate();
for(i=0;i<GOAL_OUTPUT;i++)
    fullcon_Outputs[i]=goal_Outputs[i];
}
if(obst)
{
    start=my_input-(num_div/2+2);
    for(i=0;i<5;i++)
        obst_Inputs[i]=segment[start+i].save;

    obst_propagate();
    if(obst_Outputs[0]>.8)
        return(BACK);

    for(i=0;i<5;i++)
    {
        if(obst_Outputs[i+1]<.2)
            obst_Outputs[i+1]=.00000000;

            fullcon_Outputs[i]*=obst_Outputs[i+1];
    }
}
for(i=0;i<OUTPUT;i++)
fullcon_Outputs[i]*=wander[i];

for(i=0;i<OUTPUT;i++)
if(fullcon_Outputs[i]>max)
{
    max=fullcon_Outputs[i];
    save_i=i;
}
if(save_i==0)
    return(DBL_LFT);
else if(save_i==1)
    return(LEFT);
else if(save_i==2)
    return(FWD);
else if(save_i==3)
    return(RIGHT);
else if(save_i==4)
    return(DBL_RGT);

obstacle_avoid()
{
    int i, start, save_i;
    float max=0.0;

    start=my_input-(num_div/2+2);
    for(i=0;i<5;i++)
        obst_Inputs[i]=segment[start+i].save;

obst_propagate();

if(obst_Outputs[0]>0.8)
    return(BACK);
else if(obst_Outputs[3]>0.8)
    return(FWD);
else if(obst_Outputs[2]>0.8)
    return(LEFT);
else if(obst_Outputs[4]>0.8)
    return(RIGHT);
else if(obst_Outputs[1]>0.8)
    return(DBL_RGT);
else if(obst_Outputs[5]>0.8)
    return(DBL_LFT);
else
    printf("ERROR WITH OBST");
}

/***************NETS FUNCTION***************/

/*----------------------------------------*/
/* call this routine once to setup the network */
/*----------------------------------------*/
void fullcon_initialize()
{
    int i;

    /*------------------------*/
    /* call initialization code */
    /*------------------------*/
    fullcon_NetPtr = NULL;
    sys_init_rand();
    PA_initialize();
    D_initialize();

    /*------------------------*/
    /* create network */
    /*------------------------*/
    fullcon_NetPtr = B_create_net(1, "fullcon.net");
    fullcon_NetPtr->use_biases = TRUE;
    fullcon_NetPtr->num_inputs = INPUT_SIZE;
    fullcon_NetPtr->num_outputs = OUTPUT_SIZE;

    /*------------------------*/
    /* reset weights and the input, output arrays */
    /*------------------------*/
    N_reset_wts(fullcon_NetPtr, "fullcon.pwt", PORTABLE_FORMAT);
    for (i = 0; i < INPUT_SIZE; i++)
        fullcon_Inputs[i] = 0.0;
    for (i = 0; i < OUTPUT_SIZE; i++)
        fullcon_Outputs[i] = 0.0;
}

/* fullcon_initialize */

/*----------------------------------------*/
/* call this routine every time you want to query */
/*----------------------------------------*/
/* the network. Note that it assumes the "Input" */
/* array is already loaded with input values */
/*--------------------------------------------*/
void fullcon_propagate()
{
  int i;
  Layer *input, *output;

  /*--------------------------------------------*/
  /* get pointers to network input and output */
  /*--------------------------------------------*/
  input = fullcon_NetPtr->input_layer;
  output = fullcon_NetPtr->output_layer;

  /*--------------------------------------------*/
  /* load input values; propagate network */
  /*--------------------------------------------*/
  for (i = 0; i < INPUT_SIZE; i++)
    input->node_outputs[i] = C_float_to_Sint(fullcon_Inputs[i]);
  P_prop_input(fullcon_NetPtr);

  /*--------------------------------------------*/
  /* setup output values */
  /*--------------------------------------------*/
  for (i = 0; i < OUTPUT_SIZE; i++)
    fullcon_Outputs[i] = C_Sint_to_float(output->node_outputs[i]);
}
/* fullcon_propagate */

void fullcon_cleanup()
{
  B_free_net(fullcon_NetPtr);
}
/* fullcon_cleanup */

/*--------------------------------------------*/
/* Pattern connect network */
/*--------------------------------------------*/
void patcon_initialize()
{
  int i;

  patcon_NetPtr = NULL;
}
sys_init_rand();
PA_initialize();
D_initialize();

patcon_NetPtr = B_create_net(1, "patcon.net");
patcon_NetPtr->use_biases = TRUE;
patcon_NetPtr->num_inputs = INPUT_SIZE;
patcon_NetPtr->num_outputs = OUTPUT_SIZE;

N_reset_wts(patcon_NetPtr, "patcon.pwt", PORTABLE_FORMAT);
for (i = 0; i < INPUT_SIZE; i++)
    patcon_Inputs[i] = 0.0;
for (i = 0; i < OUTPUT_SIZE; i++)
    patcon_Outputs[i] = 0.0;

} /* patcon_initialize */

void patcon_propagate()
{
    int i;
    Layer *input, *output;

    input = patcon_NetPtr->input_layer;
    output = patcon_NetPtr->output_layer;

    for (i = 0; i < INPUT_SIZE; i++)
        input->node_outputs[i] = C_float_to_Sint(patcon_Inputs[i]);
    P_prop_input(patcon_NetPtr);

    for (i = 0; i < OUTPUT_SIZE; i++)
        patcon_Outputs[i] = C_Sint_to_float(output->node_outputs[i]);

} /* patcon_propagate */

void patcon_cleanup()
{
    B_free_net(patcon_NetPtr);
}

} /* patcon_cleanup */

void random_initialize()/*same as fullcon except untrained weights */
{
    int i;
fullcon_NetPtr = NULL;
sys_init_rand();
PA_initialize();
D_initialize();

fullcon_NetPtr = B_create_net(1, "fullcon.net");
fullcon_NetPtr->use_biases = TRUE;
fullcon_NetPtr->num_inputs = INPUT_SIZE;
fullcon_NetPtr->num_outputs = OUTPUT_SIZE;

N_reset_wts(fullcon_NetPtr, "random.pwt", PORTABLE_FORMAT);
for (i = 0; i < INPUT_SIZE; i++)
    fullcon_Inputs[i] = 0.0;
for (i = 0; i < OUTPUT_SIZE; i++)
    fullcon_Outputs[i] = 0.0;

} /* fullcon_initialize */

/***** other network functions ***********/

goal_initialize()
{
    int i, j;
    FILE *nets_weights;
    float temp;

    if( (nets_weights=fopen("goal.pwt", "r"))==NULL)
    {
        printf("cannot open weight file");
        sleep(1.5);
        exit(-1);
    }
    for(i=0; i<GOAL_HIDDEN; i++)
        for(j=0; j<GOAL_INPUT; j++)
        {
            fscanf(nets_weights,"%f", &temp);
            goal_layer1[j][i]= temp;
        }
    for(i=0; i<GOAL_OUTPUT; i++)
        for(j=0; j<GOAL_HIDDEN; j++)
        {
            fscanf(nets_weights,"%f", &temp);
            goal_layer2[j][i]= temp;
        }
for(i=0; i<GOAL_HIDDEN+GOAL_OUTPUT; i++)
{
    fscanf(nets_weights, "%f", &temp);
    goal_bias[i]=temp;
}
return(0);

obst_initialize()
{
    int i, j;
    FILE *nets_weights;
    float temp;

    if( (nets_weights=fopen("obst.pwt", "r"))==NULL)
    {
        printf("cannot open weight file");
        sleep(1.5);
        exit(-1);
    }
    for(i=0; i<OBST_HIDDEN; i++)
        for(j=0; j<OBST_INPUT; j++)
        {
            fscanf(nets_weights, "%f", &temp);
            obst_layer1[i][j]=temp;
        }
    for(i=0; i<OBST_OUTPUT; i++)
        for(j=0; j<OBST_HIDDEN; j++)
        {
            fscanf(nets_weights, "%f", &temp);
            obst_layer2[i][j]=temp;
        }
    for(i=0; i<OBST_HIDDEN+OBST_OUTPUT; i++)
    {
        fscanf(nets_weights, "%f", &temp);
        obst_bias[i]=temp;
    }
    return(0);
}

goal_propagate()
{
    float net, sigmoid(), hidden_act[GOAL_HIDDEN];
    int i,j,k;

for(j=0;j<GOAL_HIDDEN;j++)
{
    net=goal_bias[j];
    for(i=0;i<GOAL_INPUT;i++)
        net+=goal_Inputs[i]*goal_layer1[i][j];
    hidden_act[j]=sigmoid(net);
}
for(k=0;k<GOAL_OUTPUT;k++)
{
    net=goal_bias[k+GOAL_HIDDEN];
    for(j=0;j<GOAL_HIDDEN;j++)
        net+=hidden_act[j]*goal_layer2[j][k];
    goal_Outputs[k]=sigmoid(net);
}
return(1);
} /* end of propagate() */

obst_propagate()
{
    float net, sigmoid(), hidden_act[OBST_HIDDEN];
    int i,j,k;

    for(j=0;j<OBST_HIDDEN;j++)
    {
        net=obst_bias[j];
        for(i=0;i<OBST_INPUT;i++)
            net+=obst_Inputs[i]*obst_layer1[i][j];
        hidden_act[j]=sigmoid(net);
    }
    for(k=0;k<OBST_OUTPUT;k++)
    {
        net=obst_bias[k+OBST_HIDDEN];
        for(j=0;j<OBST_HIDDEN;j++)
            net+=hidden_act[j]*obst_layer2[j][k];
        obst_Outputs[k]=sigmoid(net);
    }
    return(1);
} /* end of propagate() */

float sigmoid(x)
float x;
{
    double y, xd;

xd=(double)x;
y=1.0/(1.0+exp(-xd));
x=(float)y;
return(x);
}
Appendix D.1 -- Makefile for "eval.c"

This a make file for compiling the source code provided in this thesis with the NETS source code. Compilation with the NETS source code is only necessary when the use of NETS capabilities is desired.

```
compile=cl /c /AL /Od $(flgs)
link=cl /AL /Od /Os $(flgs)
linkpend=link /SE:512 /ST:14000 /NOE /F /E /PACK
exec=/Fe

game.exe : activate.obj lnrate.obj compile.obj net.obj layer.obj \
          dribble.obj weights.obj convert.obj game.obj buildnet.obj \
          teach.obj prop.obj pairs.obj show.obj netio.obj parser.obj \
          sysdep.obj mcommon.obj moves.obj
$(link) /stack:20000 $(exec)game *.obj $(linkpend)

game.obj : game.c buildnet.c net.c prop.c pairs.c dribble.c convert.c \
          sysdep.c common.h weights.h layer.h net.h netio.h mcommon.c \
          moves.c mcommon.h
$(compile) game.c

activate.obj : activate.c common.h
$(compile) activate.c

buildnet.obj : buildnet.c convert.c layer.c lnrate.c netio.c parser.c \
             weights.c common.h layer.h weights.h net.h netio.h
$(compile) buildnet.c

compile.obj : compile.c buildnet.c net.c netio.c sysdep.c common.h layer.h \
             weights.h net.h netio.h
$(compile) compile.c

convert.obj : convert.c common.h
$(compile) convert.c

dribble.obj : dribble.c convert.c net.c netio.c common.h layer.h weights.h \ 
              net.h netio.h
$(compile) dribble.c

layer.obj : layer.c convert.c lnrate.c weights.c common.h layer.h weights.h \ 
           net.h netio.c
```
$(compile) layer.c

lnrate.obj : lnrate.c convert.c common.h
  $(compile) lnrate.c

mcommon.obj : mcommon.c mcommon.h
  $(compile) mcommon.c

moves.obj : moves.c mcommon.h
  $(compile) moves.c

net.obj : net.c convert.c netio.c pairs.c parser.c prop.c common.h layer.h \ 
  weights.h net.h netio.h
  $(compile) net.c

netio.obj : netio.c common.h netio.h
  $(compile) netio.c

pairs.obj : pairs.c convert.c netio.c parser.c sysdep.c common.h layer.h \ 
  weights.h net.h netio.h
  $(compile) pairs.c

parser.obj : parser.c convert.c netio.c pairs.c common.h netio.h
  $(compile) parser.c

prop.obj : prop.c activate.c common.h layer.h weights.h net.h
  $(compile) prop.c

show.obj : show.c layer.c net.c netio.c weights.c common.h layer.h weights.h \ 
  net.h
  $(compile) show.c

sysdep.obj : sysdep.c common.h
  $(compile) sysdep.c

teach.obj : teach.c convert.c dribble.c layer.c netio.c pairs.c prop.c \ 
  sysdep.c common.h layer.h weights.h net.h
  $(compile) teach.c

weights.obj : weights.c convert.c netio.c common.h layer.h weights.h netio.h
  $(compile) weights.c