Detection of Highway Warning Signs in Natural Video Images Using Color Image Processing and Neural Network Techniques on a PC

A Thesis Presented to the Faculty of the College of Engineering and Technology Ohio University

In Partial Fulfillment of the Requirements of the Degree Master of Science

By
David Kellmeyer
June 1992
This Thesis Has Been Approved by the Department of Industrial and Systems Engineering and the College of Engineering and Technology

Russ Professor of Industrial and Systems Engineering

Dean of the College of Engineering and Technology
ACKNOWLEDGMENT

I wish to express my deepest appreciation to Dr. Helmut Zwahlen for guidance throughout my undergraduate and graduate programs at Ohio University. Without his vast knowledge and funding none of this research would have ever been possible. I would like to thank Dr. Luis Rabelo for expressing his great enthusiasm and excitement about this work when I was wondering if I would ever finish and Dr. Costas Vassiliadis for having the insight to first introduce me to the field of artificial neural networks.

I would be amiss without thanking my Uncle John Grimes who as an expert in this field was invaluable to my completion of this thesis. Moreso however than his knowledge, I would like to thank him for his patience over the phone in dealing with my initial ignorance of the topic and of course for the free consulting! Good luck on your SigGraf conference this summer!!!

Finally, I say Thanks to my Dad and family, fellow graduate students Murali and Sati and my all time favorite Fer-Ball French!!!
# Table of Contents

Summary .......................... 1

Chapter One  Introduction ........ 5

Chapter Two  Literature Review ... 12

  2.1 Advantages of Color ........ 12
  2.2 Disadvantages of Color ...... 14
  2.3 What is Color? ............... 15
  2.4 Human Perception of Visual Input 16
  2.5 Image Enhancement .......... 20
  2.6 Color Segmentation .......... 23
  2.7 Neural Networks ............ 30
      2.7.1 Inherent Advantages of Neural Networks 31
      2.7.2 Neural Network Basics .......... 32
      2.7.3 Neural Network Development Suggestions 38

Chapter Three  Objectives and Contributions 42

  3.1 Objectives .................. 42
  3.2 Contributions to Engineering 43
      3.2.1 Data Pre-Processing for Neural Networks 46
      3.2.2 Artificial Vision .......... 46
      3.2.3 Learning and Adaptivity .... 47

Chapter Four  Requirements and Alternatives 49

  4.1 Hardware Requirements for Implementation 49
      4.1.1 Computer Processor .... 49
      4.1.2 Video Input Devices .... 49
      4.1.3 Image Acquisition Hardware 49
      4.1.4 Monitors ............... 50
      4.1.5 Input and Output Cables 50
4.2 System Design Requirements
4.3 Technical Requirements
4.4 Alternatives
  4.4.1 Digitized Images
  4.4.2 Scene Segmentation
  4.4.3 Sign Recognition

Chapter Five Design Approach

  5.1 General Approach
  5.2 Use of Color
  5.3 Image Acquisition
  5.4 Resolution Reduction
  5.5 Smoothing and Filtering
  5.6 Saturation Enhancement
  5.7 RGB to HSI Conversion
  5.8 Color Segmentation
    5.8.1 Inputs
    5.8.2 Outputs
    5.8.3 Learning
    5.8.4 Post-Processing
  5.9 Scanning Routine
  5.10 Region Outlining
  5.11 Pre-Processing for Sign Detection Network
  5.12 Sign Recognition Neural Network
    5.12.1 Inputs
    5.12.2 Outputs
    5.12.3 Post-Processing
  5.13 Recorded Output for Each Detected Sign

Chapter Six Discussion of the Software

  6.1 Image Acquisition
  6.2 Resolution Reduction
  6.3 Saturation Enhancement
  6.4 Color Segmentation
List of Charts

1. Chart 1  Saturation Enhancement Using a Natural Log Function

2. Chart 2  Size of both the visual field and a 3 ft. warning sign in pixels for a 640 x 480 image as a function of distance
## List of Figures

1. Figure 1  Single Neural Network Processing Element, Node  33
2. Figure 2  Multiple Layer Neural Network  35
3. Figure 3  Flowchart of General Approach  57
4. Figure 4  Image Acquisition System Used in the Human Factors and Ergonomics Laboratory at Ohio University  60
5. Figure 5  Resolution Reduction Procedure  63
6. Figure 6  Conversion of RGB Color Space to HSI Color Space  68
7. Figure 7  Color Segmentation Back-Propagation Neural Network with 18 inputs, 2 Hidden Layers and 8 Output Colors  72
8. Figure 8  Scanning Procedure Flowchart  85
9. Figure 9  Scanning Procedure Rules and Sample  87
10. Figure 10  Standard and Modified Outlining Procedures  88
11. Figure 11  Sample Outcomes of Standard and Modified Outlining Procedures  90
12. Figure 12  Definition of an Adjacent Path  91
13. Figure 13  Pre-Processing and the Neural Network Used for Sign Recognition  94
14. Figure 14  Flowchart of the Main Modules in the Program  100
15. Figure 15  Color Segmentation Module Flowchart  104
16. Figure 16  Cascading Networks for Color Segmentation  115
SUMMARY

This thesis deals with the possible use of color and neural networks in the further development of the image processing field. Its specific problem is concerned with the detection and location of highway warning signs in natural roadway video images. The goal is to show that through the use of color and neural networks, a robust target detection procedure capable of detecting highway warning signs under varying conditions can be developed. Previous studies have shown color to be a valuable asset in the development of a robust imaging system that must perform under variable lighting conditions. Also, since highway signs rely heavily on color to provide drivers with information, they present an application where color provides an extremely valuable attribute.

The imaging system can operate on any IBM AT Style PC or 100% compatible equipped with a Targa+ 64 image acquisition board and some method of inputting images to the system. The basic approach is to digitize a roadway image and segment this image into basic color regions using a set of colors that are both important in highway sign recognition and that provide an even span of the color spectrum. The next step is to search this segmented image for color regions that could possibly represent a highway warning sign. These possible regions are then further analyzed to determine if their shape corresponds to highway signs of that region's color.

The system is capable of digitizing either S-Video, Composite Video or RGB Video input into a 640 x 480 color image with 16 bit pixel depth. This image's resolution is then reduced through software to simulate an image digitized at a resolution of 160 x 120 still with 16 bit pixel depth. Since the image is to be segmented into regions of basic colors, the colorfulness of each pixel is now
increased using a natural log function so that saturated pixels become more colorful and unsaturated pixels are basically left alone. This saturated image is now ready to be segmented into eight major colors using an artificial neural network. A supervised learning back-propagation network with two hidden layers is used to segment the image into eight basic colors. The neural network learned these colors from examples selected from standard highway images. The inputs to the network are provided by the color value of the pixel being segmented along with the color values of each pixel in a surrounding 3 x 3 neighborhood. These color values are represented using two inputs for each pixel that are derived from color differences that simulate the manner in which the human eye may use to encode color information. For each pixel, the color segmentation network must decide from the neighborhood of inputs which of eight colors (red, orange, yellow, green, blue, purple, brown or achromatic) most closely represents the pixel of interest's color. The result is a 158 x 118 (edge pixels are excluded) image represented using eight colors.

A scanning routine then searches the segmented image for regions of interest based on the color of the sign to be detected. Since the goal is to detect highway warning signs, the routine searches for first yellow and then orange regions. Upon finding a yellow or orange region, the system outlines the region and submits this region to be checked for parametric shape features to determine if the region could possibly be a highway warning sign. If the system feels the region is a possible sign, it then bounds the region for further analysis by a sign recognition neural network.

Before entering the sign recognition network the possible sign region is converted into a 10 x 10 boundary square of binary values. Binary values are used because the sign recognition network no longer uses color but only the shape of the region to determine if it is a warning sign. This sign recognition
network is another supervised learning, back-propagation network but this time with 100 inputs. The network's 100 inputs are binary values provided by each of the 100 pixels in the 10 x 10 boundary square and are used to produce two outputs that judge the region as either a sign or a non-sign. A post-processing algorithm analyzes these two outputs to give the final determination of whether or not the region represents a highway warning sign.

For each detected warning sign, a set of attributes is now stored that consists of the coordinates of the sign's bounding rectangle in the original 640 x 480 image, the intensity value of each pixel within this boundary rectangle and the sign's basic color. These attributes are stored for further evaluation of the region under higher resolution to confirm the presumption that the region contains a warning sign and if so to determine what type of a sign it is. This final recognition procedure is the topic of a another study which is presently in progress.

The results of the this study prove that neural networks can be used as an alternative to complex image processing algorithms like color image segmentation. The ability to substitute a neural network for a complex image processing algorithm has two main advantages. First, image processing is no longer a highly intense mathematical process only for the Albert Einstein's of the world, but rather, understandable to anyone who has a practical application for machine vision. Second and most importantly, the complexity of traditional image processing algorithms usually stems from trying to develop a more robust algorithm and most still end up requiring some sort of threshold that fails under changing circumstances. This problem can be alleviated by substituting a neural network that is trained to operate under varying conditions for the image processing algorithm. This statement is supported by the results of the color segmentation procedure that is consistently able to depict a variety of
natural images in only eight colors (ten, including gray levels) using a neural network.

The main observation found in using neural networks is that the paradigm, the number of hidden layers, the learning rate nor any other obscure factor is of greatest importance when developing a neural network. Rather, the way the information is presented to the network and the complexity of the network's task are the major factors in determining if a network will provide desirable results.

The use of color proved very effective in locating highway signs but its use is not recommend for all machine vision applications as the complexity and processing time for the algorithms are each increased. The developer should first look at his application to determine if color is a prominent attribute (as in detecting highway signs) and if color may be needed because of poor or uncontrollable lighting. If one of these conditions is present, then the use of color may be justified instead of using luminance values.

The system seems to provide very positive results even when a sign is set against a similar background. Of 35 testing images the system located 86% of the warning signs within these images. The processing time requires about 4 minutes per image on 486, 25 MHz processor. About half of this processing time is spent reducing the resolution of the image, which would not be needed in a dedicated system. The saturation routine takes about nine seconds and the color segmentation using the neural network takes about 1.5 minutes. The time required to scan for signs varies for each image. For real-time applications the use of designated hardware or parallel processing would be essential, especially if each frame is to be analyzed using a live video at 30 frames per second.
Chapter One -- Introduction

Over the years, machines have been called upon to replace people in an ever increasing number of areas. This is because machines have inherent advantages over people in areas such as repeatability and vigilance. However, people still prevail in many types of tasks. Real-world vision tasks are specific examples of tasks where people can often outperform machines. Vision tasks require vast amounts of visual data to be considered simultaneously in producing some sort of judgment as to the status of the scene. The human visual system is capable of processing large amounts of data by coding this data and sending its relevant information to succeeding levels of processing. Each succeeding level provides for further analysis of the data so that generalizations can be made about the scene being viewed.

Given this fantastic visual ability, many systems, such as traffic signs, have been developed to interact with people using visual information. These systems have been designed to take advantage of a person's visual abilities and are also designed within the framework of a human's visual limitations. This means making the best use of features such as color, size, shape and positioning to pass along information to the human user. These visual attributes can be used orthogonally to express the most information possible or redundantly to maximize the probability of transferring important information to the user. Most traffic information systems make use of redundancy to maximize the likelihood of transferring important information to the driver. In many traffic control devices, the desired information is coded using both the color and the shape of the traffic device. Thus, any machine vision system used in a traffic environment should take full advantage of these attributes and the fact that they are often used in a redundant fashion. By doing so, a
machine vision system may be able to approximately simulate human vision for a desired vision task and also possesses the advantages of machines. This is extremely important for monotonous tasks where vigilance is a problem.

This thesis concentrates on the detection of highway warning signs to design a general philosophy for developing a machine vision procedure that is to operate in environments designed for people. By designing the system using neural networks that key on the same visual inputs that people use, a more robust vision system will hopefully be obtained.

There presently exists many situations where systems display visual information to their users and these types of systems will undoubtedly be in place for many years to come. When substituting machines into systems originally developed to interact with people, it is desirable for the machines to operate using the same inputs, "senses", that people use. Furthermore, machines operating under real-world conditions must also be general enough to handle the wide variety of conditions that may occur. Thus, the need for a vision system that keys on the same visual attributes as humans and is robust enough to operate under varying conditions is established.

In the majority of past machine vision applications, achromatic (black & white) imaging has been used to analyze images. For many situations, using an achromatic imaging is preferred as these gray scale images contain the required information and do not flood the machine vision system with irrelevant color information. Until recently, achromatic images where not only desirable but often times a requirement given the cost and complexity of color imaging. Along with only having the capability to analyze achromatic images, the objects to be analyzed had to always be the same objects, of the same size, in the same position and viewed under carefully controlled lighting conditions. While
many of these limitations have been overcome, using a vision system under natural lighting conditions still causes problems for many imaging algorithms.

With a decrease in the cost of color cameras and color imaging hardware, more work has recently been done in the area of color image processing. This work has revealed that color may be helpful in solving the problem of adverse lighting conditions. This is because an object's color (hue) does not change as much relative to its luminance in situations where lighting can not be controlled. This consistency of color under adverse conditions is a necessity for any real-world image analysis system. As a further advantage, in environments designed for people, color is very beneficial as it represents a key attribute that helps people to acquire and detect important visual information quickly and reliably. In some instances, color is the only factor used to pass along information to the user.

However, the use of color is not enough to allow machines to operate in the same environments as humans. They must also be able to generalize inputs into rules so that they can respond correctly to inputs they have not seen before. This development of rules requires the ability for a system to learn or for the rules to be pre-specified to the system. In dealing with real-world applications, it is very difficult to specify a set of rules that work under varying conditions. This is the major problem that expert systems and decision trees have faced when used to accomplish real-world vision tasks. Furthermore, even if a set of perfect rules existed, it is often the case as in human vision tasks that no one actually knows the rules used to accomplish a visual task.

For these reasons, the ability of neural networks to learn from examples may prove helpful in increasing the robustness of image processing routines. By using examples to show the network "human" visual qualities, this replaces the need for attempting to quantitatively explain these qualities. Neural networks
have proven to be very effective in applications that humans do naturally but that are difficult for traditional artificial intelligence systems. This is because in these situations a designer can easily show a neural network examples and provide it with the proper answers but he or she finds it hard to develop a series of rules for how he determined the proper answer for each example. The learning and generalization qualities of neural networks are exploited in this application to perform both image segmentation and target recognition. The task of image segmentation is a difficult image processing task that will test the capabilities of neural networks to mimic human performance under varying conditions. As stated by [1], "Automatic segmentation represents the single most difficult step in modern image processing." By proving that neural networks can be trained to learn some of our basic visual abilities, more machine vision systems may incorporate neural networks as a tool to solve any vision application where humans excel. This study concentrates on the usefulness of neural networks and color for highway warning sign detection but they could also prove useful in many image processing applications.

The development of robust target detection systems is being sought by the military for obvious reasons. With the increasing speed and complexity of their fighting machines, there is constantly a need for an artificial vision system that can both locate and recognize targets under adverse conditions. However, in military applications, vision systems using stimuli outside the human visual spectrum are more common because the enemy often uses camouflage to make targets less conspicuous to humans. For the exact opposite reason, highway signs are made of highly visible colors and materials that are conspicuous to humans and thus lend themselves to a machine vision system that operates on the same visual stimuli as a person.
This type of road sign, detection system could also provide numerous uses in future automobiles as there are mainly two tasks involved in driving. One involves maneuvering the vehicle and the other requires watching out for expected and unexpected objects. These tasks must be done simultaneously and thus require the use of divided attention. Once the complexity of either of the task reaches a certain level, it is approximately true that the more effort spent on one task the less that can be devoted to the other and thus the opportunity for an accident increases. Currently, 95% of all automobile accidents have some component of human error.[2] Most preventative highway safety efforts have been devoted to making objects more apparent, so that less effort is required for the object detection task. From this standpoint, it would be very helpful for drivers to have an object detection system that serves as a back-up in case they missed an important piece of information.

Two European research projects, Drive (Dedicated Safety in Europe) and Prometheus (PROgraM for European Traffic with High Efficiency and Unprecedented Safety) are aiming to implement new technology in order to meet the growing public demand for increased highway safety.[2] According to these studies, the greatest potentials for increased highway safety are road transport informatics (RTI) systems. These systems are designed to provide the driver with helpful information concerning the driving environment through some type of in-car interface system. Phillips Research Laboratories has developed a system for detecting potential hazards in the roadway.[2] Their system uses radar to find objects and to calculate the distance, size and location of these objects. If the object is determined to be in the roadway, the system would inform the driver or possibly avoid the object using an anti-collision system.
A highway sign detection system would also be vital in developing a machine guided vehicle that would operate in the same environments as traditional vehicles. General Motors, Carnegie Mellon, the US Army and NASA have already begun research on such vehicles. The GM (LaneLok[3]) and Carnegie Mellon (NavLab) projects concentrate on how a vehicle could guide itself by following pavement markings or the road-side. The next logically step would be to develop an object detection and recognition system. The US Army's and NASA's applications have dealt with obstacle detection and the development of vehicles that can maneuver themselves around these obstacles in unknown off-road environments. If a robust vehicle maneuvering and object detection system could be developed, then a crude approximation of the human driver would be obtained.

The most practical use of a sign detection system would be for its applications in keeping an automated inventory of highway road signs for the Department of Transportation of different States. Many of these departments use a photo or video log system to view the State's different roadways and inventory their highway road signs. Presently, a person must analyze each roadway scene to determine if a sign is present and then record the desired information for each sign, if an accurate sign inventory is desired. By having an automated sign detection system, this type of monotonous work could be replaced by a machine vision system that would automatically inventory each sign.

These examples, show some practical and futuristic applications for image processing systems that can operate in environments designed for humans, specifically the driving environment. Going a step further, a single vision system capable of any human vision task has been the ultimate goal for a computer interface. It is the opinion of this author, that color image processing
and the use of neural networks are two aspects that will eventually help in achieving this type of general vision system.
2.1 Advantages of Color

For the past 25 years the vast majority of image processing applications and research have dealt with monochromatic images.[4] This is mainly because until recently the practicality of using color in artificial vision systems was limited by the cost of color image processing systems. These costs were imposed on both budgets and on computation time. However, many researchers agree that color imaging should continue to grow because of the decreasing cost of color acquisition and display equipment and the increasing performance of personal computing systems.[5]

One of the primary advantages of color is that color is not as susceptible to changes in illumination and geometry of an object, as are luminance values. [4,6,7,8] This means that the shading of an object will not distort an objects color as much as its luminance values. Another key advantage to color is that color analysis tends to produce a set of edges that are usually more "meaningful" to the goals of the system. [4,6,8] This is directly related to the first advantage of color. Since color is not as susceptible to changes in luminance values, the edges found tend to be the outlines of objects of interest and not spurious interior edges. This is in contrast to luminance edge finders which tend to find all edges (outlines and interior edges) and must then rely on other algorithms to extract the meaningful edges from the image. Mevattia [8] and Malowany [4] confirm this observation by concluding that color analysis shows fewer but more desirable edges. According to Malowany [4], Novak and Schafer agree by described color edges as consistently better than intensity edges.
Some factors that produce changing luminance values within an object are shadows, changing light sources, orientation angles, the distance of the object from the viewer and/or changes in an object's texture. All of these factors pose a problem when dealing with natural scenes. These factors among others are why most successful imaging systems have required many restrictions on the viewing environment to be maintained. Restrictions that are practically impossible to maintain in natural images. The first generation of machine vision systems required that the object of interest to be of the same size, in the same location and of the same orientation at all times along with the absence of any foreign objects. Also, the lighting had to be strictly controlled to distinctly outline the object against the background. The technology of today's imaging systems has increased dramatically and does not require many of these restrictions. The one restriction that still poses many problems is the constant control of lighting. The reason for this is that lighting can change an objects luminance values as well as make an object appear to be a different color. This makes it extremely difficult to set thresholds to discern desirable objects under all different lighting conditions. The control of lighting is something that can not be overcome in natural images as sunlight is always different and shadows occur outside of the designer's control. Because of its greater consistency under adverse lighting, color is seen by many researchers as a requirement for general vision systems that are to be used in analyzing natural images.

The majority of color vs. luminance papers deal with comparing color and achromatic values by their abilities to perform edge finding algorithms. However, Malowany [4] remarks that the attribute of color may be better suited to other imaging techniques such as region segmentation. Although
this suggestion lacks much scientific evidence, a few researchers have suggested that color is more of a global attribute while intensity is more a local attribute and since edge-finding is a local process this may not be where color would excel the most.[1] On the other hand, region segmentation is more of a global operation and thus color may far outdistance intensity in this type of operation. As a final argument, it appears that the above suggestion agrees with what is found in the human visual system. This is that the human visual system is less susceptible the degradations in color than in degradation to luminance values and that the human visual system may use color as a more global attribute.[5]

2.2 Disadvantages of Color

Although becoming ever less popular, the cost of color image processing is still a valid argument against using color. More commonly in real-time systems, the cost of concern is not money but that of processing time. Since color images use three separate values (Red, Green and Blue for example) for each pixel instead of a single luminance value, the computational time for color images is generally about three times that of monochromatic images.[8] However, this is only the case if the same monochromatic algorithms are applied to all three color components. Until recently, this is exactly how most color image processing applications made use of color and is probably why the advantages of color have not been exploited to a greater degree.[4] Today, development is being conducted on algorithms that exploit the additional information provided by color images.[4]

Another argument against color is that monochromatic edge detectors provide essentially the same edges as color edge detectors. Proponents of
the use of color do not dispute the argument except in the case of low-contrast images, but instead argue that monochrome edge detectors find too many irrelevant edges that must be sifted through to find the important edges. While Novak and Schafer agree that over 90% of their color edges were about the same as their luminance edges, they believe these other 10% are invaluable to a robust machine vision system in low-contrast and noisy images.[4] A counter argument to the monochromatic edge detectors providing too many irrelevant edges is that through edge linking these spurious edges can be eliminated and edges similar to those found using color are then obtained. Nevattia [8] has found that depending on the linking algorithm used, results similar to color results can be obtained. However, this means another algorithm for edge linking must be used and more than likely this algorithm will use one or more thresholds that may have problems dealing with varying conditions.

It appears the use of color has merits in image processing situations where conditions can not be adequately controlled and/or when a large number of spurious internal edges are not desired. Furthermore, with the development of new algorithms designed especially to exploit color's added information the advantage of color may become even greater.

2.3 What is Color?

In the physical world color is nothing more than different wavelengths in the visible light spectrum. Newton discovered the basis of color in 1666 when he separated sunlight using a prism into different wavelengths of violet and indigo (380-450 nm), blue (450-490 nm), green (490-560 nm), yellow (560-590 nm), orange (590-630 nm) and red (630-780 nm) light. There does not seem to exist an obvious relation between the way this
physical aspect of color is sensed by a computer system and the way color is perceived by humans. The study of color spans across many fields of study. Physics is concerned with the light spectrum of color, Physiology studies the impulses the eye and brain transmit to portray a color, Psychology studies the perception or color sensation of different colors and Psychophysics tries to relate the perception of a color with the physical (spectral) color. This attempt to relate the physics of color with our perception is where the study of color generally breaks down.\[10\] The perception of a given color is made up of three basic components that include the light source, the object illuminated and the observer. A change in any or all of these factors can lead to the perception of a different color. A particular color name assigned by the radiation of a given spectral composition is probably good enough for many people and industrial purposes but this has nothing to do with the human observer and is not sufficient for a general vision system.\[10\] Thus, any research done to advance the art of machine vision closer to that of human vision should take the time to study the basics of the human visual system.

2.4 Human Perception of Visual Input

The capabilities of the human eye and visual system are astonishing and the study of them is a prerequisite when trying to use neural networks to simulate some of the eyes many capabilities. Visual input is obtained by each eye's approximately 106 million receptors.\[11\] These receptors can be compared to a machine vision system's receptors within a camera. However, the eye's receptors each measure only one aspect of light; red, green, blue or luminance information where as each machine vision receptor may contain red, green and blue information. Another difference is that
machine vision receptors are usually evenly spaced over the entire scene, while the eye's receptors tend to be concentrated towards the center or fovea of the eye.

The eye's receptors come in two main classes, the rods for obtaining luminance information and the cones for obtaining color information. The rods vastly outnumber the cones approximately 100 million to 6 million. However, out to about 1 degree, making up a region called the foveola, there are only cones. An area out to about 1 1/2 degrees is called the fovea and this is where people view all their detailed information. Our visual perception goes out to about 40 degrees and from the foveola outwards the ratio of rods to cones increases until there are only rods. Thus, while the rods may vastly outnumber the cones, under ample lighting conditions the cones are responsible for all color information and make up the majority of receptors used to focus on an object. The important factor being the amount of light available. With luminance values given in $\log_{10} \text{cd/m}^2$, in situations less than -3 scotopic vision occurs using only the rods, from about -3 to -1 mesopic vision takes place using a combination of rods and cones and from -1 to 4.2 photopic vision occurs using only the cones.[10] Since this thesis deals with color imaging, the remainder of the discussion of the eye will deal with its use of the cones and color vision.

There are three groups of cones that consist of red, green and blue cones. By combining the input of each in different ratios, humans can distinguish many different colors. According to Hunt [11], in 1975 Judd and Wyszicki estimated that humans could distinguish about 10 million colors. Although this is not to say that people can give an object's color precisely to one of 10 million different colors. In fact, the number of colors that can truly be distinguished is relatively few. What the 10 million colors
refers to the our ability to notice different hues and shades when compared with another color. The different cones are not present in the same numbers and it is estimated that the ratio of Red to Green to Blue cones is 40 to 20 to 1. One key in designing a neural network is how information received at the cones is coded to be passed along through the visual system. First, of the 106 million receptors there are only about 1 million nerves used to carry this information, which means the input of many receptors must be combined and sent along a single nerve. This is important in trying to reduce the size of the neural network required to process a given section of pixels. This ratio though is not constant throughout the eye. In the foveola, the ratio of nerves to cones or rods (although rods are not present here) is approximately 1 to 1 and increases in the periphery to about 1 nerve to several hundred receptors. This means the eye is using its limited number of nerves primarily to transmit information the eye is focused on and the periphery is only given enough nerves to detect possibly important information. Secondly, it would help to know how the eye then transmits color information to the brain once these inputs are combined. According to Hunt [11], it is believed that RGB and luminance information is not what is passed out of the eye but rather a series of color differences. He suggests that the first input to the brain is a combination of all cones and rods \(2R + G + B/20 + \text{Rods}\) that provides a measure of luminance, the second input is a red minus green color difference \((R - G)\) and the third input is a blue minus yellow \((R + G)\) color difference \((2B - (R + G))\). In Parkkinen and Jaaskelainen's findings [12], they suggest that there could be more that three types of color receptors, which each focusing on a different color differences. Each of these receptors increases or decreases its firing rate when exposed to different wavelengths
of light. Furthermore, they attempt to support this by finding six major cell types in the parvocellular layers of the LGN (where the optic nerve terminates).

The human visual system could be separated into two primary stages; early vision processing in the retina and then a more complex analysis in the visual cortex.[12] More is known about the early processing of signals at or near the eye than in the cortex of the brain.[13] The early process called the primal sketch is constructed of intensity changes, edge-segments, bars, blobs and terminations.[9] From here the visual cortex develops more abstract features, like highway signs, from these primal features. [9]

In the detection of highway road signs, the problem is not to distinguish all the different colors that can perceived by the eye, but rather, to take all the possible yellows and call them a single yellow, all the possible oranges and call them a single orange and etc. for each of the primary colors that make up highway road signs. This mainly entails mimicking two of the eyes most fascinating capabilities: color constancy and scene segmentation. It may prove desirable to use the previous information on how the eye receives and transmits color information to achieve a machine vision version of color constancy and scene segmentation.

Color Constancy is important when using an object's color to recognize it under different lighting conditions and/or when the object is not exactly the same color each time.[11] The human visual system is very good at perceiving an object as the same color under different lighting conditions. It uses both chemical changes in its receptors and apriori knowledge about the object and the scene to maintain color consistency. The perceived color of an object is a product of the human mind and no definite equation can be drawn to determine an object's perceived color from its physical
properties.[13] This is a major reason for choosing a neural network to try to learn from example what determines an object's perceived color. In referring to hue, color constancy is the ability to distinguish approximately equal amounts of R, G and B values as achromatic regardless of the illumination source's level and color. Perception of brightness and color for figure-ground separation or scene segmentation are early vision processes that are needed to identify regions that are objects of interest. This perception of brightness or color is strongly influenced by the values of its surrounding (neighborhood) parts.[13] For this reason, when determining a pixel's perceived color not only its color but the color of its neighboring pixels must be considered. The size of this neighborhood is highly dependent on the resolution of the image.

The above section has dealt with how the human visual system preprocesses visual information and uses this information to perceive images. The pre-processing of visual input is extremely important to the human visual system and similarly the pre-processing of input is also highly important to any artificial neural network. By pre-processing the data, the work that must be done by the neural network to divide the training sets into appropriate classes can be greatly reduced. Neural networks cannot provide a solution for any problem by just shoving the data into a series of input nodes. They follow the general rule of garbage in, garbage out.

2.5 Image Enhancement

Image enhancement deals with modifying an image to bring out important information or to reduce the amount of noise in the image so that important features become more apparent. In its true sense, enhancement refers to improving the detectability of features that a machine uses to
detect objects rather than enhancing the humanly perceived quality of the image. Filtering processes can be used to reduce the amount of noise in an image or to accentuate edges.

In Low-pass filtering, high-frequency noise is eliminated by adjusting a pixel according to the values of its neighboring pixels. Thus, random noise should be eliminated when each pixel is averaged into its neighboring pixels. However, this operation often results in the blurring of edges. On the other hand, High-pass filtering is used to enhance the edges in an image but this tends to accentuate the high frequency noise within the image. Thus, a single smoothing or filtering operation that will be optimal at all times is hard to develop. Edge-preserving smoothing algorithms can be developed by using the standard deviation of a neighborhood of pixels, k-nearest neighbor smoothing or sigma-filtering. Each of these methods uses local statistics on neighboring pixels to better determine which pixels should be used in altering a pixels value.

Histogram enhancement techniques can also be used to enhance an image. Remapping is a histogram technique where the dynamic range of the image is stretched to its maximum range to increase the difference between regions of interest. This is done by determining the high and low values of an image's pixels and then stretching this to the maximum range. A variation is to use the middle 90% of the pixels so to avoid a single outlying pixel from disrupting the entire spread. While histogram stretching is a linear operation, this variation leads to a piecewise-linear operation. These remapping methods differ from high-pass filtering in that high-pass filtering only uses neighborhood pixels to change a pixel's value and histograming uses all pixels to spread the dynamic range to the greatest possible values.[5] Histogram equalization allocates more levels to where
the majority of pixels lie and less levels to less common pixels. This helps to increase the contrast in the most heavily populated areas and often reveals details that were previously hidden. The downfall of any histogram equalization routine is that by readjusting the histogram for each image the same color always has a different representation depending on the image it is in. The major difficulty with using either filtering or histograming techniques in natural images is that changes occur over a wide range of values making it difficult to use a given procedure for all situations.

Once the image enhancement routines have been chosen, the components to be altered in these algorithms must be chosen. For instance, the image could be altered using only luminance values or all three color values could be altered using the same algorithms. The key being to choose the component(s) that will provide the most enhancement with the least computation time. As stated before, the majority of color imaging routines have used algorithms previously designed for luminance values and simply applied these to all three color components thus tripling the processing time. Mitra et. al. [5], discuss the use of YIQ values to reduce the amount of processing required on color images. By changing RGB values to YIQ values, the Y value represents luminance and the I and Q components represent the color. In Mitra et. al [5], they experiment with three processing methods: 1) using RGB components and altering each one 2) Altering only the Y component and leaving the I and Q components alone 3) Altering the Y component and then adjusting the I and Q components back to their previous ratios with the Y component. For linear operations such as high-pass and low-pass filtering, processing only the Y component produces comparable results to processing each color attribute with the advantage of decreased computational complexity. For nonlinear operations such as
histogram equalization, Y component processing could decrease the false color effect of RGB processing. This effect is furthered if the Y to I and Y to Q ratios are preserved. They conclude that in terms of quality and complexity of processing, the YIQ method is more efficient than RGB processing in most applications.

2.6 Color Segmentation

Color segmentation is the segmentation of an image by the grouping of like colors into regions that presumable make up the major areas of interest. Segmentation involves a spatial analysis of the image and is intimately concerned with the spatial relationships between like pixels. It is based on being able to group pixels into discrete regions, either because they have some attributes in common or because they are bounded by a discontinuity such as an edge. Scene segmentation is another visual capability that humans perform extremely well. According to [1], it is unfortunate that no current, computer based segmentation technique can provide the discrimination required to distinguish many natural objects from their backgrounds. Furthermore, [1] states that "Automatic segmentation represents the single most difficult step in modern image processing." [1] further states that general segmentation is very difficult and emphasize that when considering a new imaging application, image segmentation should receive the most careful consideration. In spite of and maybe because of the lack of a general image segmentor, many different techniques have been developed. The first and according to [1] the only commonly used segmentation technique in practical recognition systems is some form of thresholding.
In thresholding, a histogram is usually used and an algorithm developed to split the histogram(s) into similar groups of pixels. These algorithms usually try to define the major mountains and separate adjacent mountains at their common valley. Problems arise when trying to determine which peaks constitute the major mountains and which are just part of a larger mountain. Thresholding is further complicated when viewing conditions can not be controlled, specifically, variations in lighting can be very problematic. As more varying conditions arise, the mountain finding algorithm must become increasingly complex. Even if a perfect thresholding algorithm is developed, there still exists a fundamental problem that can not be overcome through simple thresholding alone. The problem is that when a small area of an important color exists this color may still be overlooked because it does not constitute a large enough mountain. When using this type of histogram thresholding to reduce the color resolution, the above produces a custom palette. One way to overcome this is to combine colors which are too close to each other according to a chosen color metric threshold. This can still result in failure and the underlying problem is that the error is distributed to only infrequently occurring colors. Another problem of this method is that different colors are chosen for each image and thus when they are passed to the next stage of the imaging processing application different values are always passed along. This could be extremely troubling to a neural network that was trying to use this input to detect a sign in the image.

To ensure that a certain set of colors is used to depict the image, a universal palette can be used. In this method each pixel in assign to its closest palette entry. With an ample number of color palette entries, this method is guaranteed to render any color with a reasonable fidelity. This is
because unlike the custom palette the universal palette's error is distributed to all colors rather than a few. The use of custom or universal pallets are also known as color resolution reduction.

An alternative type of segmentation is region growing. Region growing tries to form clusters of similar pixels base on some attribute say intensity or hue. Initially, groups of adjacent pixels with similar values are grouped together. Depending on the threshold used this technique tends to create a large number of small similar regions. These small clusters are then enlarged by merging neighboring regions based on their similarities and spatial relationships and then stops when neighboring clusters fail to meet some measure of similarity. Notice again how previously chosen thresholds determine when adjacent pixels are similar enough to be grouped together and also when similar regions are to be merged together.

Region splitting is just the opposite of region growing. Here the entire image is considered a single region, then smaller regions are created by separating similar pixels in groups. Again, this procedures is troubled by the need to set prescribed values to determine region separation.

A combination of these two techniques known as split-and-merge techniques can be used to try to reduce the requirement of threshold values. A specific type of split-and-merge technique is known as a quadtree. In this technique you switch from merging to splitting and visa-versa until it is found that no more regions can be split or merged. By being able to constantly go back and forth from merging to splitting, this method is more tolerant than simple region-growing procedures to the effects of the initial region formations. [13]
Another method of segmentation is statistical classification. These methods tend to ignore the spatial relationships between pixels and concentrate on grouping pixels only by their statistical measures.

In some cases, relaxation can be used to group pixels into similar groups. The method was developed by Rosenfeld and Kak in 1982. This method makes explicit use of spatial information and takes into account the local neighborhood of each pixel during classification. This idea of using a pixel's neighborhood to determine its class seems very logical and is used in the approach of this thesis. The problem with this and also with probabilistic networks is that you need some initial estimate of the probabilities of the number of pixels in each class. For this reason, this method is normally used after some other classification has been run and the pixels are split into known regions.

Another method of segmentation is the use of neural networks and the like for the separation of pixels into separate regions. This can be accomplished by unsupervised learning as promoted by Duda and Hart in 1973 or could be accomplished by the less commonly used method of supervised learning. Although supervised learning seems more common in the majority of neural network applications, for image segmentation purposes, unsupervised learning seems to be more common. Coleman and Andrews[14], give many arguments for their choice of using unsupervised learning. The advantages of their system to all the above segmentation routines is that they use no human interaction or adjustable thresholds. Other advantages include that the construction of training sets is less time consuming and tedious than developing a supervised training set. They also seem to feel that with changing conditions such as natural lighting, that supervised learning is incapable of satisfactory performance. This
observation is not clear depending on their meaning of satisfactory and very disputable. If a supervised network is trained under varying circumstances then the network should learn how to operate in varying conditions. A further advantage of unsupervised networks is their ability to continue to learn without the use of human intervention. They feel that being able to continually learn is a key to developing a general image segmentor. A final advantage is that the unsupervised approach may also reveal characteristics in the image that were unnoticed by human observers. Whereas supervised networks will tend to mask these characteristics because they where taught to ignore them.

There are also apparent disadvantages that the authors admit to. One being that so little is known about the human-perceptual system that the resulting segmentations will usually not be as satisfying as segmentations made by a human being or those performed by a carefully trained segmentor operating in a supervised mode. Also, the unsupervised segmentor does not know what type of segmentation is desired except by that provided implicitly through the features selected as the inputs to the network.

For many of their presumed advantages, there are some questions to be answered before these advantages should be assumed. First, regardless of what unsupervised algorithm is used some sort of threshold is mandatory at least within the operation of the network. All unsupervised networks require either a factor to determine the number of segmentation regions to be created or a threshold to determine when the network should build a new region and when to put a pixel in an existing region. Secondly, it is true that an unsupervised training set can be easily developed but whether this will promote a properly trained network is another question. If the network uses a random set of color inputs, the input data space will be difficult to
separate since the data space should be fairly uniform, as most color systems are designed to be as uniform as possible. Since all a neural network does is try to find clusters in the data space, a uniform data space may cause many problems. Plus, if continual learning is used then the training set should become more uniform as more training sets are applied. If a selective data set is used, this should not be a problem but then this data set will be just as hard to develop at a supervised training data set. Finally, the ability to detect patterns unrecognized by human may prove highly desirable in many instances. However, in many vision applications the human is usually well adapted to pick out the regions of interest in a visual image.

A final and probably the most widely used type of segmentation procedures are edge detection techniques. Here, edges in an image are assumed to separate objects of interest and therefore can be used for segmentation. First, the meaning of an edge needs to be defined. An edge can be defined as a discontinuity in some image attribute, usually luminance. As has been stated earlier, unless careful control of lighting is maintained these edges do not usually coincide with objects of interest. The use of color may help but first a color edge must be defined. A true color edge would be a discontinuity in a 3-color space. To avoid this complexity, some alternate definitions have been developed but they do not take full advantage of the added information of color. One alternative is to define a color metric distance in the color space and use discontinuities in distance to find edges. Another method is to compute a set of edges in all three colors separately and then merge these edges to find a final edge depiction by some algorithm. Finally, only a certain attribute like the hue value, could be used to find edges. While each of these methods usually
provides more desirable results than luminance values, none take full advantage of the color attribute. Even if edges are detected using a complex 3-color space edge detector, the advantage of color may still not be fully exploited simply by the possibility that color is better suited towards region segmentation techniques than to edge detection techniques.

Other than color being better suited to region segmentation, there are many other factors that must be considered in the decision to use edge detection or some sort of region segmentation approach. Just by the nature of the operation, region segments always yield closed boundaries while after edge segmentation a line following algorithm is required to transform edges into regions. This use of another algorithm is another chance for errors and also entails the use of yet another threshold. Secondly, edge detection is a local process and thus often produces many irrelevant edges and may have problems with irregular boundaries. Alternatively, region segments are global processes and do not show some of these problems. However, by being global processes they are more likely to miss small objects where the area of the object determines its size. This could produce major problems in the detection of a small highway sign. As a general finding, the additional information provided by color seems to help region segments more than edge detectors. This has to do with edge detectors being local operators and region segments being global operators as color is believed to be used by humans as more of a global property. Finally, edge detectors are insensitive to the position of edges and will find all edges in the image equally well independent of their position. In region segmentation, the position of an object with respect to other regions may cause it to be incorrectly merged depending on which region it is adjacent to and the threshold used.
So as with most problems without an optimal answer, there seems to be advantages and disadvantages to each method depending on the application. Which ever segmentation process is chosen, it often helps to use some prior knowledge of a scene to help in the segmentation process. In the vast majority of successful machine vision applications, prior knowledge of the scene is used to its fullest. This is the reason that a single segmentor is not used for all imaging problems. It is also a widely known fact the human visual system relies heavily on apriori knowledge when viewing images.

After the segmentation has been completed, boundaries must be placed around the segmented regions of interest. One simple method of outlining is to simply turn left if inside a desired region or turn right if outside that region. The process is completed when you arrive back to the pixel where you began.[13] The only problem found with this algorithm is that pixels cady-corner to each other are considered as belonging to the same region. Although this is undetectable in images of high resolution it becomes apparent in lower resolution images and may cause problems. This might be desirable depending on the application but it also increases the chance of "bridging" two major regions together via noisy pixels.

2.7 Neural Networks

The goal of this thesis is not in the advancement of neural network technology but rather to use neural networks as a tool in image process. Thus, this paper concentrates on what helps in developing a successful neural network and not on developing a new paradigm or some new learning procedure. Even when a neural paradigm is chosen the developer is still faced with a large number of design alternatives such as the number of
hidden layers, the number of nodes in each network, learning rates, transfer functions, etc.. This field lacks any exact rules that can be used to determine these factors, which is why many consider the design of neural networks more of an art than a science. Any advancement towards establishing standard neural network design methods would be highly beneficial to bringing added credibility to neural networks but is not the goal of this thesis. However, this thesis deals more with the practical side of neural network application techniques and less with the conceptual theory of neural networks.

2.7.1 Inherent Advantages of Neural Networks

As eluded to earlier, neural networks are different processors than conventional Von Neumann Computers and thus possess different characteristics. In many cases, such as pattern recognition, these characteristics make neural networks much better suited to solve problems where humans excel and conventional computers tend to struggle. The reason for this is that humans use natural neural networks to acquire knowledge and then to apply it to everyday situations. The main difference from classical, single processor computers and neural networks is that neural networks use many simple processing units, nodes, operating in parallel instead of a step-by-step procedure. This is why classical computers are so efficient in solving problems where a given path is always taken to find the answer. An advantage of the parallel structure of neural networks is that they can be extremely fast if implemented in hardware. Also, since many nodes are used to solve the problem, neural networks tend to be highly fault tolerant as their nodes often work in a redundant fashion and can pick up for damaged nodes. This is extremely important in
developing a real-world system as the parallel structure is inherent to the network and does not have to be designed in with additional circuits.

The major advantage of neural networks surface in situations such developing a machine vision procedure where a set of exact rules can not be established. This is because neural networks learn from being shown examples and can adapt to solve most classification problems provided proper examples are used. Because of this, neural networks can learn from a designer's knowledge instead of having to be told explicated rules. More importantly is that the network not only learns the examples but actually learns the general rules used to classify the examples. It can then use these general rules to produce answers for situations it has not yet seen before. Even if the network can not come up with the exact answer, it will at least provide a reasonable answer whereas expert systems often produce bizarre answers when they encounter unfamiliar data.

2.7.2 Neural Network Basics

The power of neural nets do not come from their complexity, but rather, from their number of simple processing units. These simple processing units (see Figure 1) called nodes are each composed of a set of weighted input values, a summation function, a transfer functions and then and output. The input values are provided to a node via of set of weights that can either be fixed or altered if learning is to occur at this node. The summation function sums these weighted inputs and sends this value to a non-linear transfer function to determine the final output of the node. The reason this transfer function is always non-linear is that if a linear function is used, then any layering of the network is redundant as it could be expressed using a single layer.
Figure 1 -- Single neural network processing element, Node.
Some of the most common transfer functions consist of hard limiters, threshold logic, sigmoids and tangents. Hard limiters have a threshold where the weighted sum is either converted to a one or a zero. Threshold logic functions define an upper and lower bound where the weighted summation is either one or zero and between these limits the function is linear. While there is a linear region, the entire function is piecewise linear. Sigmoids and Tangents are basically the same as each provides a consistently non-linear function but the Sigmoid’s outputs range from zero to one and the Tangent’s range from negative one to one. Regardless of the transfer function used, the output of each function is connected to the next level of processors again using a set of weights and this process is repeated until the final level of nodes are reached.

These different levels of nodes are call layers (see Figure 2) and each layer consists of one or a group of nodes. Usually, as the network moves forward the number of nodes in each layer decreases as more general and important data is extracted from the data in the previous layer.

Often times, neural networks are discussed as if how they operate is a great mystery. While the exact method of how to create an optimum network is mysterious, they all tend to achieve their goal in the same fashion. Neural networks attempt to segment a given data space into a set of distinct regions.

The first networks, called perceptrons, were simple, two layer networks consisting of input nodes and output nodes. Because they only had one layer of adjustable weights (learning weights), Minsky proved that these networks could only segment linearly separable data spaces. This meant they could not solve any but the most simple problems and the study of
Figure 2 -- Multiple layer neural network
neural networks was largely abandoned for a long period of time. The reason more layers were not added is because no one could figure out how to distribute the error to more than one layer. However, a few researchers kept with their studies of neural networks and eventually solved this problem. Some of these older names associated with neural networks are McClulloch, Pitts, Hebb, Rosenblatt and Widrow.

This break through learning algorithm is know as back-propagation and represents by far the most widely and successfully used training method to date. With the use of back-propagation, the error in the output is sent back through the weights so as to correct those weights that provided erroneous information and reward those weights that provided correct information. With this came a new serge in the study of neural networks that included names like Hopfield, Rumelhart, McClelland, Sejnowski, Feldman and Grossberg among others.

In an introduction to neural computing[15], the author looks at six of the most popular neural networks and provides decision tree for deciding the proper network for a given situation. Different styles of networks are known as paradigms and each has their own advantages and disadvantages.

The first paradigm spoken of is the Hopfield Net. The inputs to this network are normally binary values that may represent black and white pixels. This paradigm is usually used to solve associative memory problems or optimization problems. If used in associative memory problems the inputs are usually degradations of the possible outputs and the network tries to find the closest matching output for each input. The problem with Hopfield Nets are they are limited in the number of outputs they can classify by the number of neurons in the network. The number of possible outputs it can discriminate between can only be about 15% of the number of inputs.
The Hamming Network also used binary inputs but it operates by looking for the strongest output node and having all other possible output nodes go to zero in a winner take all fashion. This network is advantageous over the Hopfield network in that its number of required connections grows linearly with the number of inputs while the Hopfield Net grows as the square of the number of inputs.

The Carpenter/Grossberg Classifier also know as Adaptive Resonance Theory (ART) is also a binary input paradigm but can be altered using the ART2 paradigm to a variable input network. It uses unsupervised learning to define groups of prominent clusters within the data space. As the input feeds forward, each node in the next layer produce a score based on how well it matches the input data. Lateral inhibition is used between these matching nodes to produce only a single winner. The winner's weights are then adjusted to make it even more similar to the input and thus more likely to win the next similar example. If not any nodes are seen as matching the input, then a new node is created for this pattern. Problems arise in determining the vigilance factor as to what constitutes a match. If it is set too low, the network will not learn all the patterns in the training set, while if it is too high, noisy data will create duplicate patterns for a single region in the data space. Many researchers suggest using a high vigilance during training and lowering this vigilance when running the network. However, this assumes that your training set is fairly well segmented.

The next paradigm is the classic perceptron. The perceptron can handle analog inputs but as already stated can only solve linearly separable problems. This leads to the Multi-layered Perceptron also known as the Back-propagation Network. This paradigm can also can handle analog inputs. Their are no guaranties that the training will converge or that the
training will not get stuck in a local minimum causing it to fool itself into thinking it had converged. The number of nodes in each layer is determined by the separation required of the input. If too few nodes are used the network may not converge while too many nodes causes "Grandmothering". This is when the network memorizes each input set instead of learning the general rules governing all data sets. The number of layers is basically determined by the input space. To solve a non-linearly separable problem, at least one hidden layer (a layer other than input and output layers) is required. If two hidden layers are used, the network will be able to simulate any desired region given the proper number of nodes and a proper training set.

Finally, the Kohonen Net is a self-organizing feature map the accepts analog input and uses unsupervised learning. Unlike the ART network, this network can work well in noisy conditions because the number of outputs is fixed and no further learning is done after training.

2.7.3 Neural Network Development Suggestions

The development of a neural network consists of many factors such as choosing the paradigm, the number of layers, the number of inputs, the connectivity, the learning rule, the transfer function, etc.. The first step is to choose a paradigm that shows promise in solving your application. This study concentrates on the back-propagation paradigm since supervised learning was preferred and this network has been very effective in the past.[18] Once a paradigm is chosen, the required number of hidden layers must be determined unless the data is linearly separable. The hidden layers act as features extractors that separate similar groups of data. When a series of hidden layers are used they tend to solve the problem in stages.
More hidden layers tend to increase the power of the network[17], however, two hidden layers is usually sufficient for any data space[18]. The drawback to multiple hidden layers is that more data sets are required to properly train the network and the training time is increased[19]. For this reason, most designers start with a single hidden layer and add hidden layers if the network cannot converge. However, if the net does not converge with two hidden layers, the designer should look at his data and pre-processing approach to see if anything can be done to make learning easier for the neural network.

After the number of layers has been determined, the number of nodes in each layer needs to be established. However, it should be noted that the number of layers and the number of nodes tend to interact. If you have multiple hidden layers you probably do not need as many nodes in each layer. A general procedure for determining the number of nodes is to start with the input layer. Here the number of input factors in the data depict how many input nodes will be required. Next, the number of output nodes is a function of the number of classes the network is supposed to divide the data space into. The two main approaches are to use binary coding or one-to-one coding. If binary coding is used, the number of output nodes is reduced but the learning tends to be harder and the data is more difficult to analyze during post-processing. Hence, one output node per data class is usually preferable if post-processing is required. The number of hidden layer nodes can then be approximately determined by adding the number of inputs and outputs and dividing by two. This usually provides a good place to start and nodes can be added or subtracted if needed.

The conductivity of the network, which is the fashion in which the layers of nodes are connected to each other via the weights, needs to now be
addressed. The two main choices are fully connected or randomly connected. Fully connected systems require the each node in succeeding layers has a set of weights attaching it to all nodes in the previous layer and this is generally the standard procedure. Randomly connected weights are more like the human neural system and require that the designer specify the percentage of conductivity from layer to layer.

The type of transfer function at each neuron can be selected from a wide range of functions, but the common element among all of them is non-linearity. The only general rule is that the range of the function should match the range of the inputs. Thus, if the inputs are binary a sigmoid function could be used that ranges from zero to one. However, if the inputs range from negative one to one, then the network should use a tanH which is similar the sigmoid function but ranges from negative one to one.

A final selection that needs to be made is what type of learning rule should be used. Once this is selected the network is ready to learn. Although many different variations of learning rules can be used, they all perform basically the same task: to propagate the error back though the network to the nodes causing the error via the weights.

A massive amount of research effort has been spent studying the best way to apply each of these network variables but still they are largely a mystery and basically come down to trial and error. Often when a network does not converge, designers start to speculate on how to alter any of the above to produce a better learning and can spend much time testing new ideas. These tend to be the people interested in the conceptual aspect of neural network development and thus are always trying to produce better networks by altering any of the above network variables.
On the other hand, almost anyone that is involved with the practical side of implementing neural networks will first question the pre-processing of the data. This is the bread and butter step for developing neural networks for practical applications. While the design of the network may not be as elaborate or have as fast of learning speeds, the results speak for themselves when proper pre-processing is used. In many cases learning times are even reduced because learning is not as difficult. Again, there are no scientific rules on how to pre-process data. However, since the network is trying to find natural clusters in the data space, anything that further segments the data space or that enhances the attribute the network receives, will probably help the learning process. Another goal of pre-processing may be to reduce the amount of noise in the data so that the basic segments become more apparent. If proper pre-processing is done then the network should most likely learn to segment the data space as long as the basic rules for developing a neural network are followed and a proper training set is used.
3.1 Objectives

The general objective of this thesis is to advance the art of machine (artificial) vision in target detection applications by incorporating the use of color and neural networks. Neural networks are used along with present artificial vision techniques to hopefully provide for a more robust approach than presently seen and one that takes advantage of the same visual stimuli that people find advantageous in target detection. Such a system, should also perform better than traditional vision systems in situations were it has not specifically been introduced to that exact visual image.

The specific objective of this thesis is to develop a machine vision software package that can observe static images of video roadway scenes and determine if a highway warning sign (yellow diamond) is present. If so, the system must then specify the location of that sign using a bounding rectangle to enclose the sign. The system should accomplish this under normal daylight conditions and using the available shape and color information provided by the sign. This system is being designed to show the feasibility of its basic approach in detecting warning signs and not its actual practicality in a real-time system. Because of this, the program should allow the user to interact with the software to see the effects of using different configurations of the system. Furthermore, the manner in which with operator can interact with the program should be logical and require little or no previous instruction. In a dedicated system, these variables would be fixed to achieve a desired detection goal and the system would be totally automated.

The thesis also addresses issues that are dealt with in almost all machine vision applications which include target acquisition, target detection and target
location. Target acquisition involves obtaining an image for further study, in this instance one that can be used by a computer. Target detection is then the determination of when an object of interest appears within an image. Once the desired target has been detected, target location attempts determine its position within the image. Target location is probably the more difficult of the tasks for a neural network to accomplish and is expected to be better left to present machine vision procedures.

Research on the human visual system is used to help in the development of a neural network that attempts to perceive colors similarly to the human visual system. The reason for learning how humans handle visual input is because humans also use neural networks when processing visual information and probably have evolved a way encoding visual stimuli in the most useful fashion possible to aid in neural learning. The goal is to obtain available information on what stimuli and processing procedures a person uses to process visual information. It is known that the human visual system works in separate stages that each feed different types of information along to the next visual level but exactly which information and in what form it is past from stage to stage is not know for sure. Some of the theories on what happens as information is passed through the visual system will hopefully prove useful in developing a pre-processing system that supplies a neural network with the most useful information possible.

3.2 Contributions to Engineering

The study of artificial neural networks is an exciting new field that has many applications in areas where present artificial intelligence techniques have had a formidable task in dealing with the available information and making decisions based on this information. This is readily apparent in machine vision
applications that deal with target detection in changing environments. It is extremely difficult to develop a given set of rules that govern how people are able to detect targets in a visual image. The main reason we can not develop these rules is that it is unknown exactly how we convert the visual stimuli in an image into perceived information. The easiest way we can teach others to detect targets is to show them a series of images and tell the person when a desirable target appears, as opposed to trying to give an elaborate set of features that distinguish a target under all circumstances. Thus, the goal is to try to use this simple method of showing examples to a machine vision system to teach it to learn image processing tasks.

People have often desired to give computers two of our most precious and highly adaptive capabilities, sight and learning. These two abilities are powerful tools that are very difficult to pass along to a computer but that are required if a machine vision system is to be a successful substitute or partner for people in an environment designed for people. Systems for use by humans are designed to use our five major senses so that humans can perceive information provided by the system. A person without senses in these environments would be like a computer without a keyboard, mouse or any other input device. Without senses it would be impossible to operate in these types of systems. Furthermore, vision accounts for a vast majority of what we perceive in a normal day. Since vision is such an efficient means of transferring information to humans, many of our everyday systems rely on vision to pass information on to their users. Thus, for us to lose our vision would be like trying to use a keyboard with only a few its keys. Also, these machines need to not only "see" but "see" as humans since the environment has usually been designed to take advantage of the attributes of the human visual system. However, it could be argued that many people are blind and they learn to manage quit well
in our environment. This is a valid point and many machines have been
developed with this in mind but often the environment they operate in is alerted
to meet their sensing capabilities. Since most machines do not have a visual
system, they must perceive man's environment in some other fashion than
vision. Because of this, we have devised many elaborate ways for machines to
perceive our world. Some use feelers as sensors, while others are equipped
with different types of senses than ours such as radar. In any case, it is often
harder for these machines to accomplish perception in a vision oriented world,
just as the blind have a harder time to adapt in our society.

The second and most important point is that people who are blind learn to
adapt. Learning and adaptation are the other traits we have tried to convey to
computers. Probably the most fascinating aspect of humans is their ability to
learn. This ability to take in vast amounts of perceived data, develop
generalizations that apply to these situations and then to be able to relate these
generalizations to similar situations would greatly expand the number of areas
where computers could be applied. Also, the ability to modify these
generalizations as more data is brought into the system is a major factor of our
survival.

Providing artificial vision systems with similar traits represents an objective
of this thesis. In this way we could show the vision system examples like we
do to other humans and let it decide how to detect and recognize objects. By
attempting to incorporate sight and learning into a machine, it will most
certainly increase the types of situations where machines could be used and
also increase the effectiveness of machines in their present applications.
3.2.1 Data Pre-processing for Neural Networks

The human visual system does not pass the exact data received by the receptors in the eyes back to the brain where it is perceived as an image. Instead, the neurons in the vision system are able to code information, translate it, and filter out useful information into a form that is easier for the brain to handle. The brain then does even further processing of the information before you "see" an image. Similarly, teachers do not just read straight out of the textbook. Rather, they alter the way the information is presented so as to make it easier for students to understand.

This same idea is used to alter the incoming video images so as to present the artificial neural network with a better chance of learning the important features of the image. Another reason for pre-processing the data is to shrink the amount of data that the network has to filter through. This amount of information can be extremely vast (4,915,200 bits) when dealing with a 640 x 480 image with 16 bit per pixel color resolution. In all cases, it is easier to learn a smaller amount of more meaningful information than to have to sift through vast amounts of data to find important information. This is a major concern when trying to develop useful pre-processing procedures. Thus, one section of the thesis will deal with the procedures used for pre-processing the data to be presented to the neural network.

3.2.2 Artificial Vision

All visual tasks consist of basically a combination of one or many of the following: target acquisition, target detection, target location and target recognition. Each not being possible until the prior task is accomplished. Hence, we may look at a visual task as a series of steps, some requiring prior steps to be completed while others can be processed simultaneously. In
general, to view an object the object must first be detected. This detection process is the focus of this thesis. Once accomplished, the viewer can focus on that specific object and determine any of a number of details such as its location, shape, color, size, texture, depth, motion, etc. These details of the object can then be used as characteristics by themselves or as clues to the proper identification (recognition) of the object. Since the goal of most visual systems is target detection and then some sort of recognition, any study producing desirable results in these areas has great promise in being successful in other similar applications.

3.2.3 Learning and Adaptivity

What is meant when someone or something is said to have learned? People go to school for as much as a third of there life to learn and are often judged on the amount of knowledge they have learned throughout their life. Therefore, their must be something about learning that makes its so desirable and a standard of someone's prowess. An objective of this thesis is to bring some of the advantages of learning to machine vision applications.

Learning can be accomplished by two major techniques: by being taught and/or by observing examples and relating these examples by one's self. These two methods are referred to as supervised and unsupervised learning. In most cases, we use a combination of both methods. Someone first teaches us or we read some material to gain a knowledge base, then we learn from experience as we actual take part in or observe the problem at hand. By being taught information, we can learn from someone else's experiences and do not have to redevelop the wheel. It would be highly desirable to pass along peoples' experiences to a computer as is attempted in expert systems. The problem is that for many tasks such as vision we do rely understand how we
use visual stimuli to develop perceived images. Thus, we can not provide the system with a series of rules that can be used to "see" images. However, we can tell the computer what we see in each image. Thus, the objective is to teach a computer specific visual tasks through a series of examples. Through a series of examples a computer vision system will hopefully be able to achieve the specific goal of detecting highway warning signs. If this is achieved then any vision task that humans perform well could probably be accomplished though the use of examples instead of detailed rules or algorithms.
4.1 Hardware Requirements for Implementation

4.1.1 Computer Processor

The system runs on an IBM AT Style computer or fully compatible computer with a minimum 640K of RAM. A minimum hard drive memory of 500K is required to store the basic program and to allow the program to save images. An additional 250K is required for the programs that develop a color segmentation training file and that develop a sign recognition training file. A 386 processor with a math co-processor or 486 processor is highly recommended in order to reduce imaging times. The addition of a math co-processor increases the speed of the neural networks by about 30 times.

4.1.2 Video Input Devices

Either a CCD color video camera, video tape recorder (VTR), video disc or digital image can provide the system with color images. The camera, VTR or video disc should be able to produce analog images in either Composite Video, S-Video and/or RGB Video formats. Digital images are required to be stored in the Targa format (*.TGA) and should have a resolution of 640 x 480.

4.1.3 Image Acquisition Hardware

The analog input images require a Targa+ 64 image acquisition board to digitize the analog images into a digital format. Before running the software, the board should be set to digitize images at 640 x 480 resolution and with 16 bit pixel depth in the interlaced mode.
4.1.4 Monitors

Any conventional computer monitor will suffice for viewing message and prompts from the program. However, for viewing the video images an analog multi-scan rate monitor with horizontal scan rates between 15 and 35 KHz, and vertical scan rates between 25 and 100 Hz is recommended. This type of multi-scan monitor can be used in a dual monitor or single monitor setup. The dual monitor setup is recommended so that messages from the program can be viewed on one screen while the analog images can be viewed on the analog monitor. If a multi-scan rate monitor is not selected, images are outputted using standard NTSC format and thus require a monitor with a 15.734 KHz horizontal scan rate and a 59.9401 Hz vertical scan rate.

4.1.5 Input and Output Cables

Two Truevision CA-6 interface cables are recommended. These allow for RGB, S-Video or Composite Video images to be feed into the Targa+ 64 board via a 9-Pin "D" type connector. This same cable will also allow for images to be sent out of the Targa+ 64 using either RGB, S-Video or Composite Video.

4.2 System Design Requirements

1. Run on a PC under DOS versions 3.1 or later.
2. Accept images from any source as long as they are Composite Video, S-Video or RGB Video format.
3. Digitize images as full color RGB images.
4. Detect and frame highway warning signs (yellow diamonds) within natural roadway images.
5. Provide the coordinates and luminance values for each pixel and the basic sign color for each sign.
6. Operate with any input image taken under normal daylight conditions and free of excessive blur from the motion of the camera.

7. No time constraint on the system as its purpose is to show a basic approach to target detection.

8. Be user friendly enough so that anyone with minimum computer skills can use the program.

9. Allow the user to adjust variables during the program to see their effects on the outcome.

10. Easily converted to a totally automated system by fixing these variables.

11. Store images on file for each major step of the sign detection process. This should be done in manner so that stored images do not keep taking up more and more disk space.

12. Create an output file for all detected signs that consists of the boundary rectangle that encompasses that sign, the luminance value of each pixel with this rectangle and the basic color of the sign.

Using the Ohio Department of Transportation's (ODOT) Photolog or some other similar roadway representation, the imaging software should conduct a scene analysis for each picture to determine if a highway warning sign is present within that picture. If so, the system should determine where the sign is located within the picture. This will be conducted in a static situation (looking at individual pictures not a moving image) and thus there is no constraint put on the system as to how long it can analyze each picture. One of the findings will be how fast such a system can step through individual photos with a given success rate for target detection and location.

The software should run under any version of DOS 3.1 or later. Its basic design should be to run in a totally automated situation but since this is a
conceptual approach to the problem the software should allow the user to experiment with adjusting variables within the program. The software should be easily used by anyone with minimum computer skills.

4.3 Technical Requirements

1. Provide a color image segmentation routine that gives a meaningful rendition of any highway image under normal daylight conditions with only eight basic colors.

2. Detect signs for a distance of at least 100 feet.

3. Provide a detection rate of at least 90%.

4. Provide a maximum of 10% false alarms, type II errors.

4.4 Alternatives

4.4.1 Digitized Images

Images can be digitized in any number of formats. The major choices are whether to use gray-scale or color, the resolution and the pixel depth. Since color appears to be more robust under variable lighting conditions than luminance values and because color is a major attribute of highway signs, the images where digitized in color. A resolution of 640 x 480 was chosen to provide ample resolution for both the detection and recognition of highway warning signs and to provide the system with square pixels. Finally, a pixels depth of 16 bits was chosen to represent the color values of each pixel. This value was not highly important because the system was to reduce the color resolution to only eight colors in the segmentation routine.
4.4.2 Scene Segmentation

The most classic form of scene segmentation is by edge-finding techniques. One major advantage of edge detection techniques is that they are well adapted to finding small regions within an image. However, these techniques tend to find spurious edges within objects and do not create naturally segmented regions. The next most common method of scene segmentation would be through the analysis of histograms. Here, the system attempts to find distinct mountains within the histogram that probably represent the basic regions of the image. This method is very susceptible to thresholds that must be used to define "distinct" regions. Also, in histograming, small regions may be excluded because they do not have enough pixels to produce a distinct mountain even if they are highly meaningful.

The last basic set of segmentation routines are region splitting and merging techniques. In region merging or region growing, similar pixels in the same proximity to each other are merged to make small regions. These small regions are then merged into each other if they are similar to each other and in the same proximity. Region splitting techniques work in the exact opposite manner of region growing techniques. The entire image starts as one region and then pixels that are dissimilar enough from each other are split into their own regions. Each of these techniques requires that some sort of threshold be establish as to when pixels or regions are similar enough or dissimilar enough. Also, errors may occur based on the spatial relationship of pixels and/or regions.

Split-and-Merge techniques were developed to elevate the need for strict thresholds in the above techniques. This is done by allowing the system to alternate back and forth between splitting and merging so that pixels can move in and out of regions until both the splitting and the merging algorithms are
satisfied with the segmentation. Quadtrees are a common application of split-and-merge techniques.

The alternative chosen to segment the scene into its basic colors was the use of neural networks. There were two major reasons for electing to use a neural network. First, the above segmentation systems all require heavily on thresholds to be set properly and this is very difficult under varying real-world conditions. Thus, these systems often fail when the environment can not be controlled. Secondly, natural scene segmentation is viewed by many as the most difficult application in artificial vision. In contrasts, scene segmentation comes very naturally to people and we perform this in a very efficient manner. However, we can not list the specific rules we use when performing scene segmentation so that they could be transferred to an expert system. Thus, this is a perfect example of the usefulness and versatility of neural networks. While we can not explain rules for scene segmentation we can easily provide the network with examples and let it learn some of the basic rules used to segment an image.

4.4.3 Sign Recognition

Once the image was segmented, color was use to find possible regions of interest. Color was chosen to locate the general areas of possible signs because humans seem to use color as a means to locate information quickly. Once a possible region was located, there where many techniques that could have been used to recognize warning signs. As color was used to detect a possible object, shape was chosen to recognize this object. Shape was elected to recognize warning signs because shape offers the most direct correspondence between an image and the objects of interest.[1]
One method to recognize shapes is template matching. This is accomplished by comparing the region of interest against pre-select shapes or templates. This method is effective under controlled conditions where objects are generally of the same size and orientation. If this is not the case, then the system must repeat the matching for all possible sizes and orientations which quickly becomes computationally explosive.

Parametric shape features can also be used to recognize an object's shape. This method studies a region for any of a number of features such as length, width, area, perimeter, center of mass, central moments, rectangularity, etc. Most of these features are clearly dependent on the distance from the object. However, they are basically rotationally invariant as the features are independent of an objects orientation. This method of feature analysis was seen as better for eliminating regions instead of recognizing regions and thus was implemented into the thesis in this fashion.

A neural network was again chosen to perform the sign recognition but not because present methods had so many shortfalls as compared to humans as in scene segmentation. Rather, a neural networks was selected for its ability to generalize and handle noisy data. After the drastic reduction in data, the warning signs in the images where no longer perfect diamonds and on occasion shading or a yellow background could produce noise in the image around the sign. Thus, in template matching and parametric shape analysis a threshold would have to set to determine when a region was close enough to a sign. Instead, a neural network was used to develop a generalized system that could deal with instances it had not seen before and be able to extract signs out of noisy data.
Chapter Five -- Design Approach

5.1 General Approach

The general approach used to detect highway warning signs in this thesis contains the same basic steps as any other image processing system. An overview of the software procedure is pictured in Figure 3. First, an image is digitized (grabbed) and then enhanced to bring out the desirable features of highway warning signs. This enhanced image is then segmented into its basic colors so that it can then be searched for warning signs among these segmented regions. The differences between this machine vision application and most applications lie in the approach taken to accomplish each fundamental task.

The procedure begins by grabbing the image in color which although is more common place today, is definitely not the norm. The next step is the enhancement routine which does not try to accentuate edges as in many cases but instead increases the colorfulness of the image. By increasing the colorfulness of the image, the job of the following segmentation routine is usually made much easier. This segmentation task focuses on color segmentation and is accomplished using a neural network, which is highly uncommon compared to the more common approach of some sort of histogram thresholding or edge segmentation. Finally, the sign is detected by first using color to find possible sign regions and then using a second neural network that concentrates on the shape of these possible sign regions to determine which are probable signs.

The segmentation of a natural color image and then the detection of a relatively small target, a highway warning sign, is by far not an easy image processing application. This alone would lead one to believe a highly complex
Digitize Color Roadway Image at 640x480

Reduce Resolution to 160x120

Conduct Saturation Enhancement

Segment Image using Neural Network -- SegNet

Scan for Colors of Interest

if scanning done

no

if proper color

yes

outline region

if proper size

yes

Scale Region for Neural Network -- SinNet

Analyze Region with Neural Network -- SinNet

Analyze output w/ post-processing

if output = sign

yes

Save boundary rectangle for detected sign

Convert region for 640x480 image

For each sign store boundary coordinates, luminance values and sign's basic color

Figure 3 -- General Approach Flowchart
set of algorithms would be required to achieve a robust system for this purpose. Coupled with the fact that image processing is a mathematically intense field, this would almost certainly involve a complex procedure. However, this approach is very straightforward and the most complex equation is the use of a natural logarithm, if the workings inside the neural networks are ignored.

It is the opinion of this author, that the highly complex imaging algorithms with all of their inherent thresholds are exactly why more general vision systems have not yet been achieved. The reason being, that each time a threshold is used this further limits the bounds that the system can operate within. The use of neural networks appears to be a viable alternative to complex image processing techniques and thus could lead to more general machine vision systems in the future. From the results of this machine vision task, it is suggested that the general philosophy of this approach with its use of color and neural networks may prove beneficial towards the development of machine vision systems that can operate under adverse conditions without highly complex algorithms.

5.2 Use of Color

As seen in the literature, the use of color can increase the effectiveness of algorithms designed for use with intensity values. The primary advantage of color is usually its ability to increase the effectiveness of these algorithms in environments with less than desirable conditions. The effectiveness of color under poor lighting conditions is a result of the relative hue of an object not being as drastically effected as its luminance values under shading conditions. This not only allows for edges to be detected under poor lighting but means that fewer spurious edges will be detected. These spurious edges generally
result when the luminance values within an object varying because of the shading or texture of the object. Probably an even a greater reason for the use of color in the detection of highway signs is that these signs use color as a vital means to passing on information to the driver. Not only can the edges of a sign be detected using its color but the color of the sign provides information as to what type of highway sign is present. Thus, a given set of highway signs can be located by scanning an image for their designated color. Since highway signs are coded using orthogonal features (color and shape), the knowledge of a sign's color leads to knowledge of the sign's possible shapes. This presumed shape can then be used to ensure that the system has located an actual sign.

5.3 Image Acquisition

A graphical depiction of the image acquisition system is shown in Figure 4. This system has been developed to handle a number of different video and digital input formats. Some of the video input systems that may interface with the system are video tape recorders (VTRs), video disks or live color video. Each have the option of providing the system with RGB Video, S-Video or Composite Video formats, all of which can be connected through the Truevision CA-6 cable to the image acquisition board. Another option is to directly load a digital image stored as a Targa file (*.TGA). It should be noted that it is a misconception that video disks provide digital output as most video disks still output analog images that must be digitized using the image acquisition board. Presently a Targa+64 is used in this system is set to digitize (grab) these images at a resolution of 640 x 480 pixels. This provides ample resolution for the detection and then later the recognition of different highway signs. It is a fair approximation that a resolution of 640 x 480
Figure 4 -- Image acquisition system used in the human factors engineering and ergonomics laboratory at Ohio University
provides an image composed of square pixels (technically an image of 648 x 486 provides square pixels). Having square pixels is important in avoiding the complications of using an image's aspect ratio when incorporating parametric shape features to distinguish objects or when using square neighborhoods.

5.4 Resolution Reduction

The first step of this procedure is to reduce the resolution of the image to a more usable level. This is desirable because while the 640 x 480 resolution may be required for recognizing the symbols on highway warning signs it is not required in detecting the presence of these signs and thus provides the system with an excessive amount of information. A color image digitized at 640 x 480 produces an image with 307,200 pixels that each contain three color values. By reducing the resolution to a minimum, the time required to accomplish each imaging tasks can be greatly reduced. Imaging time is a key issue in the development of a real-time system but it is also important in reducing the development time of the system. In developing an imaging system, many trials have to be run to improve and debug each routine resulting in many hours spent watching the results of each change. If a general approach can be developed using the smallest possible resolution, this same approach can be applied to images of higher resolutions if the program is written, as in this thesis, to handle different resolutions.

The images are not digitized at the desired resolution because the image acquisition board is not previously set to grab images in any resolution smaller than 640 x 480 and still obtain square pixels. Thus, a routine was developed to simulate an image that would result from digitizing the image at smaller resolutions. This routine is capable of reducing a 640 x 480 image to resolutions of 320 x 240 (76,800 pixels), 160 x 120 (19,200) or 80 x 60
(4800 pixels). A reduction to $320 \times 240$ only requires $1/4$ of the original information, the $160 \times 120$ image only requires $1/16$ of the original information and the $80 \times 60$ image requires only $1/64$ of the original information. From viewing images at these different resolutions, the $160 \times 120$ images seem to provide adequate information to detect highway warning signs while a further reduction to $80 \times 60$ resulted in warning signs occasionally being lost in the background. The resolution reduction algorithm (see Figure 5) achieves these reductions simply by converting a square of pixels in the original image into a single pixel in the reduced image. For the reduction to $160 \times 120$, a square of $4 \times 4$ pixels in the original image is converted to a single pixel in the reduced image. The red, green and blue components of each pixel in this $4 \times 4$ square are each averaged separately and these averaged values then represent the red, green and blue values of a single pixel in the reduced image. In a dedicated system, this resolution reduction step would not be required as the image could be grabbed originally at $160 \times 120$ pixels.

5.5 Smoothing and Filtering

For this application no explicit smoothing or filtering procedures were conducted. Some reasons for this are that the resolution reduction is a similar to a smoothing operation, plus both smoothing and filtering procedures require some sort of trade off. High pass filters enhances the edges of an image but tend to increase the amount of noise in the image, while low pass filters smooth away noise but tend to blur the edges in the image. Although there are algorithms designed to reduce these trade offs, they still fall victim to set thresholds and it is often hard to choose one filter or smoothing operation that is optimal for under varying conditions. Finally, the color segmentor used later
Figure 5 -- Resolution Reduction Procedure

Sum pixels within each large square
Sum Red
Sum Green
Sum Blue

Average Sums to obtain RGB of new reduced pixel
Average Red
Average Green
Average Blue
in the procedure uses a neighborhood of pixels to determine each pixel’s color and thus is similar to some filters.

5.6 Saturation Enhancement

Before a neural network can be used to analyze the image, a series of pre-processing steps need to be conducted to provide the network with the most useful information possible. The first true pre-processing step is the saturation enhancement procedure. As the name implies, the goal of this procedure is to increase the colorfulness of the image without effecting the hue and intensity values of the image. The purpose of increasing the colorfulness of each pixel is that later in the system a neural network is asked to segment the image into eight basic colors. By saturating each pixel, its apparent color is increased and thus made more obvious to the neural network. Also, after the saturation process a larger percentage of pixels are fully saturated, thus reducing the number of differently saturated pixels the neural network will have to discriminate against. The problem this saturation algorithm has to overcome is that if a pixel’s saturation level is low it should stay low so as to be seen by the network as achromatic (gray scale). By the same reasoning, the higher the saturation of the pixel the more its saturation should be increased until its maximum saturation level is reached. An algorithm was thus developed to convert each pixel’s original saturation to a new saturation based on these principles.

The algorithm implemented to achieve this requires that a pixel’s color be transformed from RGB values into Hue, Saturation and Intensity (HSI) color values. This allows the saturation component to be adjusted without effecting a pixel’s hue or intensity components. Once this is accomplished, each pixel’s percentage of saturation in the original image can be determined with 0%
representing an achromatic value and 100% representing the maximum
colorfulness for a given hue (i.e. no white mixed with the hue). If this
saturation is less than or equal to 1% a new saturation of 0% is assigned to
that pixel. However, if the saturation is greater than 1% then the natural log of
this percentage times 100 is taken to determine the factor by which the
saturation will be increased.

\[
\begin{align*}
\text{sat} \leq 1\% & \quad \text{new sat} = 0 \\
\text{sat} > 1\% & \quad \text{new sat} = \text{sat} \times \ln(\text{sat} \times 100) \quad [1]
\end{align*}
\]

sat = original saturation %;
new sat = final saturation %;

The reason for using the natural log is that it fulfills the requirement of
keeping the saturation levels low for unsaturated pixels and increases the
saturation of other pixels by a greater factor as their saturation increases (see
Chart 1). After a pixel's saturation is enhanced, the HSI values are converted
back to RGB values which are used to display the pixel's newly saturated color.
This process is conducted for each pixel in the image.
Saturation Enhancement Using a Natural Log Function

Chart 1 -- Saturation Enhancement using a Natural Log Function
5.7 RGB to HSI Conversion

A simple conversion from RGB to HSI color values can be achieved by a straightforward process using color percentages of the red, green and blue components of a color. First the percent of red, green and blue in a pixel's color is determined:

\[
%R = \frac{R}{R + G + B} \quad [2]
\]

\[
%G = \frac{G}{R + G + B} \quad [3]
\]

\[
%B = \frac{B}{R + G + B} \quad [4]
\]

A pixel's color is then mapped into a 2-D space with the %Red on the abscissa and the %Green on the ordinate as shown in Figure 6. The relation of this color location with respect to "white" determines a color's hue and saturation. Normally, "white" is defined as an equal amount of red, green and blue and thus is positioned at (1/3, 1/3). The angle (\(\alpha\)) from the horizontal of white represents a color's hue and the distance (\(S\)) from white gives a color's saturation level. If a color is positioned on "white" its hue is not zero but rather the null value which means this point is achromatic and does not have a hue. Furthermore, this does not mean the point is white but instead could represent any shade of gray. The percentage of saturation is determined by finding the maximum saturation distance for a given hue and then dividing \(S\) by this maximum value.
Figure 6 -- Conversion of RGB color space to HSI color space
5.8 Color Segmentation

After the color of the image has been enhanced, the next procedure is the segmentation of the image into a basic set of colors using the image's enhanced color attributes. The goal of this procedure is to reduce the color resolution of the image from 32,768 colors to a standard set of eight colors. The eight possible colors that each pixel must be represented by are red, orange, yellow, green, blue, purple, brown and achromatic (grays). These eight colors where chosen to be the most useful colors in detecting highway signs and to provide an approximately even span of the color space. The segmentation routine uses a universal palette of these eight colors in an attempt to simulate the way a person would divide the image given the same standard set of eight colors. Not only should this method of color segmentation be similar to human color segmentation but it also should try to incorporate the advantages of other segmenting procedures such as edge detectors and region segmentors while avoiding many of their down falls.

The disadvantages of edge detectors are that they tend to find many spurious edges and that these edges do not naturally form bounded regions. Thus, these edges must further be analyzed to determine which edges are of interest and a method must be developed to link edges that make up a region of interest. The disadvantages of region growing techniques result in small meaningful regions sometimes being lost in larger similar regions. This problem is highly susceptible to the spatial arrangement of the image as region segmentors are generally global operators. Finally, in histogram techniques small meaningful regions may be ignored because they do not produce a distinct enough mountain in the histogram. The underlying reason for each of these failures is that thresholds must be set to control each of these functions
and thus these methods are more susceptible to failure under adverse conditions.

The desirable advantage of edge detectors is that they perform as local operators which make them well suited in detecting small or thin objects. Region segmentors are advantageous as they always produce bounded regions and probably make better use of color since they are global operators. The advantage in histograming is that the designer can specify the colors the image should be segmented into if a universal color palette is desired and this provides for at least an approximate representation of each pixel in the image. If the system is left to determine a custom palette of colors for each image, infrequently occurring pixels often are represented by unsimilar colors even if these pixels represent an important region in the image.

Since the segmentation of an image into similar colors is a process that humans perform quite well, it was presumed that this might be an application where a neural network could prove beneficial. Some inherent advantages of a neural networks are that they can learn by example the most appropriate way of segmenting a given input space. This is extremely beneficial when an mathematical formula proves insufficient or can not be developed for a given segmentation problem. In the color segmentation of an image, mathematical formulas fail as the physical color values that represent a given perceived color can change under different lighting conditions. Thus, it becomes hard to set thresholds that perform sufficiently under all conditions. Neural networks do not require the setting of any thresholds to discern when two inputs should be considered as different and therefore were used to provide a more robust system that is capable of operating with inputs from a wide range of viewing conditions. Finally, the robustness of the system benefits from the ability of neural networks to generalize for inputs they have not seen before. This
capability results from the network learning the general rules used in classifying all inputs and can then apply these rules to any input case. This is not true for traditional expert system applications where an answer for each possible situation must be provided or the system often gives extremely poor answers when it comes across an unfamiliar input. When a system is used to view natural images or which the designer has no control, the ability to generalize is an important attribute. Furthermore, since a neural network is almost assured to give at least a reasonable answer, incorrect answers usually only occur for inputs that lie on a boundary defined by the network to separate two possible outputs. Using this observation, a series of cascading networks can be developed to reduce the effects of these types of errors. An example of the how cascading networks could be used to reduce the errors in color segmentation is explained in the results section.

The above are inherent advantages of neural networks but when designing a specific network the designer can incorporate other desired features into the network. The color segmentation network's inputs, outputs and learning procedure were designed in such a way as to incorporate some of the advantages of the standard segmentation routines presented above.

5.8.1 Inputs

The inputs to the network were chosen to be not just the color value of the pixel of interest, but rather, its color value and the color values of its neighboring pixels (see Figure 7). The major reason for using a neighborhood was that when a person describes a pixel's color they are really not referring to the specific color of that pixel, but rather, to the color it appears to be given its color and the color of surrounding pixels. In cases of high resolution, using a neighborhood is also required because a person can not distinguish a single
Figure 7 -- Color Segmentation back-propagation neural network with 18 inputs, 2 hidden layers and 8 output colors
pixel and is really referring to the average color of a small group of pixels. Using a neighborhood also makes this procedure similar to a local smoothing operation and thus blends irrelevant pixels into larger groups of colors. This aspect proved beneficial in eliminating the symbol from the sign and producing a solid yellow diamond to represent a warning sign. For the 160 x 120 resolution, a 3 x 3 neighborhood was chosen to input to the network for each pixel. The reason for this is that a 5 x 5 neighborhood provided for too much smoothing which eliminated important information in some instances. For higher resolution images this neighborhood should probably be increased to achieve the optimal results. However, the present color segmentor works well for all resolutions upto 640 x 480 and usually produces even more desirable images.

The color of each pixel in the neighborhood was depicted to the neural network using a set of two inputs. Only two inputs are required for each pixel because the network is concerned only with a pixel's color and not its intensity. If a pixel's hue was the only attribute of interest then one input for each pixel could be used. However, the color segmentation network also picks out achromatic pixels and thus requires the use of saturation information. Furthermore, the hue-saturation interaction could be important for the network in developing boundaries between adjacent colors in the color space.

Two color difference values similar to those presumably used by the human eye to encode color information where chosen to denote each pixel's color. Three color difference formulas for the RGB values can be established as in the following equations:

\[ C_1 = \%R - \%G; \]  \[ C_2 = \%R - \%B; \]  \[ C_3 = \%B - \%G; \]
These three equations can then be reduced to two equations since,

$$\%R + \%G + \%B = 1;$$ \hspace{1cm} [8]

The first input to the neural network is then just equation [5], the difference between the percentage of red and the percentage of green in a pixel.

$$\text{input1} = \%R - \%G;$$ \hspace{1cm} [9]

Where positive values represent reddishness and negative values represent greenishness. The second input is produced from subtracting equation [6] from equation [7] and results in the color difference between the percentage of blue and the percentage of yellow in a pixel. Although yellow is not an attribute in the RGB color system, it is represented by red plus green.

$$\begin{align*}
\text{input2} &= \%B - \%G - (\%R - \%B) \\
&= 2*\%B - (\%R + \%G) \\
&= 2*\%B - \%Y;
\end{align*}$$ \hspace{1cm} [10]

Here positive values denote blueishness and negative values denote yellowishness. It was observed that a network trained using the hue and saturation values did not perform as desirably in the same amount of training time as a network that used these two color difference input values. This would lead to the assumption that the color difference values provided information that was more meaningful to the segmentation network in learning basic colors.

5.8.2 Outputs

A universal color palette was chosen over a custom palette so that each image would be segmented into the same set of colors. This means a standard set of colors is chosen to represent the entire range of possible colors for any original image. By using a universal instead of a custom palette, the chance of ignoring small, meaningful regions of color is almost eliminated if the universal
A set of colors is chosen to evenly span the entire color space. A custom palette often misses small regions of color, like warning signs, because if the scene is dominated by say leaves and sky, it may choose many distinct greens and blues to represent the image and not have any color slots left for yellows or oranges. For this same case, a universal palette may not be able to classify the greens and blues as accurately as a custom palette, but it will always have a color reserved for each color the designer specifies. The reason for this is that a custom palette tends to place the reduction error on infrequently occurring pixels, thus, sometimes representing these pixels by an entirely different color. A universal palette distributes the error to all pixels and for this reason each pixel is always represented by at least a similar color but not as exactly for some pixels as a custom palette. Since a universal palette segments each image into the same colors, it holds another advantage over custom palettes when the output of the segmentation is to be analyzed by a neural network.

For this application, a routine was developed that attempted to use features of both universal and custom pallets to produce an optimum segmentation. In this routine, a histogram of all the pixels' hues was divided into six basic colors as a universal palette would be. However, within each of these standard color regions the routine acted as a custom palette by choosing the hue that most closely represented all the hues within that region. The result was a red, orange, yellow, green, blue and violet hue that each represented the pixels of their hue in the best way possible. However, selecting the proper boundaries between the regions proved difficult under varying lighting conditions and the continually varying set of output colors was presumed to be troublesome for the sign detection network to deal with.

After deciding on a universal palette, the output of the color segmentation network then had to represent each of the colors in the universal palette in
some fashion. A separate output node for each of the eight colors possible colors in the universal palette was chosen as the representation scheme (see Figure 7). While this method of encoding the possible output colors requires more output nodes than would binary coding (which would only require three output nodes to represent the eight colors), there are two distinct advantages to allowing each color to be represented by its own output node. First, the network usually has a much easier time learning this type of training set and thus learning time is reduced. In some instances, the network may not be able to learn the binary coded data and thus a more elaborate network will have to be used which negates the size saved by using fewer outputs. However, the major advantage is the ease in which the output can be used by post-processing algorithms to analyze the networks results. Even if a neural network is properly trained, the outputs will usually not consist of ones and zeros for input sets it has not seen before. Customarily these output values will range from zero to one and can be used to determine the "confidence" the network has in its answer. By using one output node per color, the confidence of each color being the proper output is readily observed. This allows for post-processing algorithms to be easily developed that optimize each set of output data given by the network depending on one's application.

5.8.3 Learning

The back-propagation paradigm was chosen since it has proved successful for a wide variety of applications. Back-propagation uses supervised learning which is desirable in this instance so that the network can be taught some of the general rules that the human visual system uses in segmenting colors into basic classes. If an unsupervised learning approach were to be used, the network may segment the inputs into classes other than those that represent
the different sign colors or may even find some underlying attribute other than color within the data to use as a segmentation rule.

In back propagation, the network produces an output for a given set of inputs and compares this to the correct output as specified by the designer. This error is then sent back through the network and the weights are altered according to this error to learn a more desirable answer.

The term learning has been used in place of training for a very important reason. When teaching humans, the term training implies that one is shown certain circumstances and the correct procedure to follow for each circumstance. During training different circumstances are shown repeatedly to a person until there is no question as to what to do under each defined set of circumstances. However, no emphasis is placed on developing the person's ability to generalize the knowledge within each circumstance to learn how to respond under different situations. This ability to generalize knowledge in order to produce tailor made answers is what moves learning beyond training. Training can be achieved using neural networks if the network has enough nodes to "memorize" each input set and this is often what happens if the network has too few training examples or is over-trained. In the case of too few training examples, the network adjusts its weights to memorize all possible inputs but these weights do not reflect general rules to be used for all situations. In an over-trained network, the network at one point "learned" the general rules for a task but then it tried to "train" itself to memorize specific characteristics in each training set. Thus, the network trains itself not only to the general rules of the data but also to any idiosyncrasies within the specific data set that can not be applied to all possible of inputs. While this may help in classifying inputs within the training set, it hurts in the overall classification of unseen inputs.
The network was taught using a training set of 2400 color examples. Approximately 3% red, 10% orange, 21% yellow, 18% green, 10% blue, 3% purple, 5% brown and 30% achromatic. The different percentages of each color reflect the importance of that color in detecting warning signs in natural images or the abundance of that color in highway scenes. These exact values were not decided upon before building the training set but rather this general philosophy was used when building the training set. The training set was built by selecting pixels within representative highway images that appeared to be one of the basic colors. An attempt was made to select a representative range of pixels for each color but no scientific method was used to determine which pixels represented which colors. In fact, any formulas or numeric values were undesirable as they have proven ineffective in the past in segmenting natural images. Instead, only the human perception of each pixel was desired in developing the training set. The development of this type of training set can be an extremely tedious task which called for the development of a computer program to quicken the process as much as possible and to reduce the amount of errors in the training set. This program is described in the discussion of the software.

The network learned for about 200,000 trails of 2600 training sets and still proved unable to converge. However, the segmentation network seems to have learned the overall segmentation rules and maybe it is better that the network did not completely converge. One reason being, that too much training may make a network begin to memorize inputs instead of learning underlying rules in the data. Also, these inputs were developed by using one person's judgment of color and obviously this judgment of what made a yellow different from an orange or a color pixel different from an achromatic pixel was not entirely consistent for each training set. The main reason among many for
this inconsistency is that humans are not very good at absolute judgment and instead tend to perceive colors relative to the entire image. If the data set is thus assumed to contain errors and one believes that neural networks tend to learn the basic rules first and then start picking out the idiosyncrasies in the data, it would better for the network to learn only to a point as further learning could lead to the network learning mistakes within the data set. Another problem is that while the network is able to view a pixel relative to its neighbors, it lacks the information needed to perceive a pixel’s color relative to the entire image. In future studies it would be interesting to find some sort of color metric for the entire image that could be inputted to the network along with neighborhood color values. These hypothesis receives support by viewing the positive results of the network even though it did not converge.

Taking the idea of sequential learning a step further, it would be logical that the network would try to learn the most basic rules first and then branch out to learn sub-rules under these most fundamental rules. As an example of this presumption, the color segmentation network seemed to first learn the difference between colors and achromatic pixels and then attempted to discern different colors. It is very interesting that when most people are given a set of colors to segment they too will start by separating these colors into major hues, then minor hues and finally segment them by saturation levels. Further study into how humans learn better will most probably increase the ability of neural network learning if applied properly. By designing a series of networks that start with one network that learns fundamental rules in the data and then uses a separate networks to analyze each fundamental rule, the chore of each network can be reduced. A more elaborate discussion and example of this type of cascading approach for color segmentation is given in the discussion section.
5.8.4 Post Processing

Each pixel in the image can only be depicted by one color and thus only one output in the segmentation network can be declared the winner (see Figure 7). For this reason, the maximum output of the network is used to represent the color of the pixel in question. The only further type of post-processing done is in determining how to separate the achromatic values into white, gray and black values. One method could be to build a Gray Network that would be called upon to analyze the neighborhood of pixels each time an achromatic output was produced by the primary network. This Gray Net would most probably receive the luminance values of the pixels as inputs. However, this system was not concerned with achromatic values at this time, requiring only the use of a simple threshold to separate the achromatic values into white, gray or black values in order to give a more appealing depiction of the image to human viewers.

\[
\begin{align*}
% \text{luminance} & < 25\% & \text{black;} \\
25\% & \leq % \text{luminance} & \leq 75\% & \text{gray;} \\
% \text{luminance} & > 75\% & \text{white;} \\
\end{align*}
\]  

(11)

Once the achromatic values have been determined the color segmentation process is complete and the image consists of only ten possible colors: red, orange, yellow, green, blue, purple, brown, white, gray and black.

In all image processing techniques that use a neighborhood to determine a pixel's value, edge pixels in the image will not have full neighborhoods. A decision must therefore be made as to how to handle this problem. The three primary options are to ignore the edge pixels, pretend the image wraps around itself and use the opposing edge to make up a pixel's neighborhood or to expand the image in all directions by copying the existing edge pixels. Any method of creating a neighborhood will not provide an accurate neighborhood
for these edge pixels but sometimes an approximation is better than not being able to process these edge pixels. One of these times is when other image processing algorithms are to follow which make it desirable to keep the image in a standard resolution. However, in this instance a standard resolution is no longer required allowing the edge pixels to be simply ignored since they are not extremely relevant to the image. Thus, the final image after color segmentation is a 158 x 118 image composed of ten colors. From an original image requiring 14,745,600 bits of information (640 x 480 pixels @ 16 bits per pixel), this segmented image requires only 167,796 bits of information (158 x 118 pixels @ 3 bits per pixel if a single achromatic is used instead of white, gray and black). This represents only 1.01% of the original information and in many case the location of a sign is even more distinct than in the original image. If the 160 x 120 image is considered the starting point then the color segmentation routine produces an image with only about 18.21% of the information contained in the 160 x 120 image.

5.9 Scanning Routine

Once the image has been segmented, the system now attempts to find warning signs within this image. A scanning procedure uses the colors produced by the color segmentor to locate areas where possible signs may be located. A sign's color is used as the primary attribute to locate these possible areas and then parametric shape features are used to determine if this area could possibly be a sign.

The way the procedure works is that the designer must tell the system which color(s) of the ten basic colors it should look for and then it scans the image from bottom to top in search of this color. Once it finds a pixel of the desired color, it then proceeds to outline the region made up of this pixel and
its adjacent pixels of the same color. While outlining this area, the procedure keeps track of the minimum and maximum horizontal and vertical pixels in the region. The procedure then uses these minimum and maximum values to determine the horizontal and vertical sizes of a bounding rectangle that would enclose the entire region. If the horizontal and vertical sizes of the bounding rectangle are within the expected size boundaries of a given sign, this region is bounded and marked as a possible sign location. The horizontal and vertical sizes are the only parametric features used in checking to see if a region is possible a sign.

There are many other parametric features that could be used to evaluate these regions such as the regions area, perimeter, center of mass, central moments, rectangularity, circularity, etc.. Of these, the regions rectangularity would probably prove most helpful in detecting warning signs. However, the goal in using parametric features was not to find signs but rather to exclude regions that where definitely not signs. Using this philosophy, size seemed to be the safest measure for determining possible sign regions. The allowable size ranges where exaggerated ensure that size did not exclude any sign regions. A minimum horizontal and vertical size of seven pixels in a 160 x 120 image (28 pixels in a 640 x 480 image) represents the smallest group of pixels that could represent a valid shape to the sign recognition network. This region of seven by seven pixels represent an area that covers only 0.25% of the image. A maximum horizontal and vertical size of 30 pixels in a 160 x 120 image (120 pixels in a 640 x 480 image) was chosen as the largest area that could possible represent a sign. This maximum region would represent an area that covered 8.33% of the image. To depict how large this region would be, a 30 by 30 pixel region would cover just under 1/5 of image in the horizontal direction and 1/4 or the image in the vertical direction. By examining the solid line in Chart
2, it can be seen that a 3 ft. x 3 ft. warning sign can be detected by the system between the distances of 50 and 200 feet using the above size restrictions. The reason for wanting to reject only obvious non-sign regions is that a neural network is used to determine which regions are signs instead of an elaborate definition of what constitutes a sign region.

This scanning procedure was developed to find possible locations for all types of signs or any other object, provided the possible colors and range of sizes is known for the object. It is possible to run the procedure without a size range but usually some sort of parametric shape information is know of the object to help exclude areas that obviously could not be the desired object. For the detection of highway warning signs, the system is told to scan for yellow regions first and then to scan for orange regions.

As mentioned above, an upward scanning procedure is used. However, a further algorithm was needed to control the scanning so as not to miss and possible regions and to prevent it re-scanning previously outlined regions for each line the region contains (see Figure 8). If each line of the image was scanned in succession, then each region would have to be outlined once for each pixel of the object contained in the given line. For this reason, after a region is outlined the scanning is continued on the line where that region was encountered but scanning is begun outside of the bounding rectangle for that region. However, when the algorithm increments to the next line it will again encounter the same region it just bounded and will continue bounding this region until the scanning routine is above the region. The first attempt in overcoming this problem was to move the scanning procedure to the upper right corner of the bounding rectangle after a region had been outlined. However, this allowed for some regions to be overlooked if the where to the right of a previously bounded region. To solve this problem, the scanning
Length of the Visual Field and Sign Size in Pixels for a 640x480 image as a Function of Distance

Chart 2 -- Size of both the visual field and a 3 ft. warning sign in pixels for a 640 x 480 image as a function of distance
Figure 8 -- Scanning Procedure Flowchart
routine was altered to operate as follows. First it scans upward line by line until it finds a desired color. It then proceeds to outline this region and place a bounding rectangle around the region (see Figure 9(a)). Upon completion of bounding the region, scanning is continued on the line it found the region but skips over to the right edge of the bounding rectangle (see Figure 9(b)). When the scanning routines comes to the end of a line it then increments to the next line and continues to scan again for a desired color. However, for each pixel it now determines if this pixel is already part of any bounded regions. If so, it skips over to the end of this bounded region and continues to scan the line (see Figure 9(b)). This process is continued until each line has been fully scanned (see Figure 9(c)).

5.10 Region Outlining

The region bounding routine used is a common outlining procedure described by Navattia [13]. In this procedure, if you are located on a pixel within the desired region, in this case a pixel of desired color, you turn left to move to the next pixel. However, if you are located on a pixel of some other color you turn right to move to the next pixel (Figure 10(a)). The procedure ends when you arrive again at the beginning pixel. The result is that the minimum and maximum pixels touched will be those just outside the region of interest. One problem may arise if the starting pixel is the minimum vertical pixel in the object of interest. This is often the case since the procedure scans from bottom to top and causes the bounding rectangle to exclude this lower pixel. As a simple solution, when the algorithm returns to the starting pixel it does not end but rather makes one last move to the pixel beneath it. This ensures that the bounding rectangle will encompass all of the pixels in the region of interest.
Figure 9 -- Figures (a) and (b) are the general scanning procedure rules. Figure (c) shows a sample scanning routine.
Figure 10 -- Outlining procedure for (a) standard outlining procedure, (b) modified outlining procedure.
While this procedure does produce an outlined region, its definition of the spatial relationship of pixels belonging to the same region is not what is desired for this application. In the above algorithm, two pixels located cady-corner to each other are defined as components of the same region (see "bridging pixels" in Figure 11(a) ). In situations where the background of the image contains noisy yellow areas, these cady-corner pixels tend to lead small groups of pixels into other small groups and sometimes will provide a bridge into a sign region (Figure 11(a) ). This is much less common if truly adjacent pixels are used to define a region (Figure 11(b) ). In this definition, two pixels must lie directly to the side, below or above each other to be part of the same region. For every image analyzed, all pixels within a sign met this stricter definition of a region, allowing it to be used in reducing the possibility of noisy pixels bridging into sign regions.

A small modification was made to the standard algorithm to enforce the stricter definition of a region. This modification consists of checking each pixel of desired color to see if there is a path of truly adjacent pixels leading to the most recently included pixel in the region (Figure 12). If a path is available, this pixel is included in the region thus becoming the most recently included pixel and the standard left turn is made to the next pixel. However, if the pixel does not meet this requirement, it is treated as being outside the region even though it is of the same color as the region and a right turn is made to the next pixel (Figure 10(b) ).

5.11 Pre-Processing for Sign Detection Network

The requirement of a proficient pre-processing routine for proper neural learning was truly realized when trying to detect the presence of a sign in the image. The original method of sign detection consisted of a 20 x 20 pixel
Figure 11 -- Sample outcomes for (a) standard outlining procedure, (b) modified outlining procedure
Adjacent Path Available

$X = \text{present pixel location}$

$P = \text{previous pixel location}$

Adjacent Path Unavailable

Figure 12 -- Definition of an adjacent path
"eye" that was moved throughout the entire image in 10 pixel increments looking for warning signs. Once a sign was detected, the procedure used the outlining routine to bound the region which was presumed to be a sign. The neural network was able to "memorize" the examples given but performed poorly on images it had not seen before. After this failure, a few hypotheses were drawn for why the network failed. For one reason the network usually did not get a "look" at the entire sign but rather a piece of the sign since the sign was usually not located entirely within the "eye". Secondly, each input to the network could be any of ten colors. While inputs do not have to be binary values, neural networks learn much easier if binary values can be used. This use of analog inputs required the network to consider not only shape but also color. Finally, since the 20 x 20 "eye" contained 400 pixels, every pixel could not be inputted to the network and still maintain a moderately sized network. Instead, a sampling pattern requiring only 1/4 of the pixels had to be incorporated to handle the vast amount of input pixels. Coupling this with the problem of the "eye" not encompassing the entire sign, made for a wide variety of input shapes that would represent both sign and non-sign regions.

Although other problems also existed, these problems were selected as those to overcome in designing the next sign detection network. To obtain regions of interest that were centered within a window, the method of the previous approach needed only to be reversed. The scanning and outlining routines were now used first to find possible sign regions instead of using a neural network to find sign regions. The neural network could then be sent a window that contained a centered region that possibly represented a sign. Secondly, it was realized that when looking for a certain type of sign there was a single color or at most two colors of interest. Thus, pixels being the color of interest could be represented by a one to the network and those of other colors
could be represented by a zero to the network. This would then provide a window of binary values that would give the shape of the region of interest and the network could decide if this constituted a sign.

However, a new problem of having varying sized windows and the old problem of how to reduce the number of inputs for large windows were still major problems. The problem of varying boundary rectangle sizes was solved by increasing the horizontal and vertical sizes of the rectangle to the next highest tens value (see Figure 13). Then the smaller side, either the horizontal or vertical, was increased to the other so that a bounding square was always obtained (see Figure 13). Thus, either a 10 x 10, 20 x 20 or a 30 x 30 bounding square is created from windows ranging from 7 to 30 pixels. The region was converted to a square because the final input to the neural network is a square and rectangular regions would have to be distorted to convert them into a square. When increasing the size of the bounding rectangle, it was done in a fashion that kept the region of interest in the center of the bounding square. Furthermore, when a region is expanded it should not then pick up noisy pixels and present these to the network. The procedure avoids this by assuming that each pixel added to the window is not the color of interest. To maintain a network of moderate size, it was decided that a network with no more than 100 inputs should be used. For regions of size 10 x 10, each pixel could then be inputted to the network in one-to-one fashion. However, an algorithm to represent the 20 x 20 and 30 x 30 regions with only 100 inputs was still required.

A simple sampling of certain pixels was not desired as this was one the presumed problems in the original network. Instead, the algorithm was designed to incorporate the way the human eye uses a single neuron to represent many receptors. For a 20 x 20 region a single input neuron would
Figure 13 -- Pre-processing and the neural network used for sign recognition
have to represent 4 pixels (2 x 2 square) and for a 30 x 30 region each input neuron would have to represent 9 pixels (3 x 3 square) (see Figure 13). A simply averaging routine was not chosen since a binary input was required. The final algorithm that was decided upon was that for 2 x 2 squares if at least 2 pixels where of the desired color an input of one would be used to represent the entire 2 x 2 square, otherwise, a zero input would be used to represent the 2 x 2 square. For 3 x 3 squares, if at least 3 pixels where of the desired color an input of one would be used to represent the entire 3 x 3 square, otherwise a zero input would be used. Once this is accomplished, the same 100 input network could be used to analyze all possible sizes of bounding squares.

The result being that the neural network is presented the entire region of interest in the center of the bounding square. Also, each pixel in the bounding square is used to pass on shape information to the neural network using binary values. After these changes, the network learned extremely quickly and works for images it has seen as well as images it has not seen. In fact the pre-processing routine proved so effective than any number of methods could have been used to determine if the shape of the region of interest coincided with the shape of a warning sign. However, a neural network was still chosen to determine if the region was a sign for reasons given in the following section.

5.12 Sign Recognition Network

The goal of the sign detection neural network is to determine whether or not possible areas of interest obtained by the scanning routine are highway warning signs. A neural network was used to accomplish this task mainly because their ability to generalize and handle noisy conditions. These are probably two of the main advantages of neural networks and are extremely helpful in developing a robust system that must operate under real-world
conditions. While the basic shape of all warning sign regions is a diamond, each region appears a little different for many reasons. In the reduction of the image and the following saturation enhancement, the edges of the sign sometimes become blurred. This may alter the shape of the sign if the segmentation routine sees these blurred pixels as yellow. In some cases, sign degradation occurs when the background of the sign contains some yellow pixels or the segmentation network mistakenly makes some background pixels yellow. Finally, degradation can occur when the larger boundary squares are reduced to fit the 100 input network. These types of problems always occur under natural conditions and when data reduction is used. They represent the reason that a neural network was chosen to determine probable sign regions since neural networks can handle noisy inputs extremely well. The network uses the inputs from the pre-processing routine to observe the shape of the region and from this information determines which areas represent highway warning signs. The network chosen for this task is again a standard back-propagation network with one hidden layers as can be seen in Figure 13.

5.12.1 Inputs

The 100 inputs are received from the pre-processing routine and are always binary values. Since binary inputs are used, a sigmoid transfer function is used throughout the network. These 100 pixels represent the 10 x 10 boundary square that holds the region of interest and provides the network with the shape of this region (see Figure 13).

5.12.2 Outputs

The network uses two outputs to represent the two possible outcomes that consist of the region being in the shape of a sign or not in the shape of a
sign (see Figure 13). Again, more efficient post-processing routines can be developed by using a separate outputs for each possible outcome.

5.12.3 Post-Processing

The post-processing goal is to use the outputs in fashion that maximizes the effectiveness of the neural network. Thus, depending on the reasons for wanting to detect a highway sign this process should be altered to fit a specific application's requirements. This is basically a choice of how many type I and type II errors are tolerable. The purpose of this system is to frame a highway warning sign for later analysis to determine exactly what type of highway warning sign it is. The algorithm used for the recognition of these signs should also be able to exclude invalid signs. Thus, the consequences of sending a non-sign to this process are not extreme. However, this system has been developed to send as few non-signs to the recognition routine as possible while still detecting a high percentage of warning signs. Another fact taken into consideration is that the system does not only get one look at a highway sign. Depending on the number of images used per second, the system should get a minimum of two or three good "looks" at any sign and if real-time imaging could be achieved the system would get between 40 and 80 good "looks" at a sign. Thus, if a sign is not highly detectable in one scene because of background or some other reason the system may get a better "look" at the sign in the next image. Utilizing these considerations, the two neural network outputs, sign or non-sign, where used in the fashion shown in equation [12]. If the sign output is very high the system will call the region a sign regardless of the non-sign output. If the sign output is high and the non-sign output is marginally low the system will again call the region a sign. If the sign output is moderately high and the non-sign output is moderately low the system will
again call the region a sign. Finally, if the sign output is marginally high and the non-sign output is low the region will be called a sign. In all other cases region is determined to be a non-sign.

\[
\begin{align*}
\text{sign} & \geq 0.95; \\
\text{sign} & \geq 0.90 \land \text{nonsign} \leq 0.50; \\
\text{sign} & \geq 0.75 \land \text{nonsign} \leq 0.25; \\
\text{sign} & \geq 0.50 \land \text{nonsign} \leq 0.10; \\
\text{else}; & \quad \text{non-sign}
\end{align*}
\]

5.13 Recorded Output for Each Detected Sign

The overall goal of this study is to eventually develop an automated system that can not only detect but recognize all types of highway traffic signs. To achieve this, the sign detection routine needs to provide the recognition routine with certain information. Thus, for each detected sign the coordinates of its bounding rectangle in the 640 x 480 image, the luminance values for each pixel in this rectangle and the sign’s basic color is written to a file called \textit{Sign.txt}. Once this has been accomplished, the sign detection phase is completed.
Chapter Six -- Discussion of the Software

The software was developed using the Turbo C and C++ languages. It consists of a main application that uses a series of modules developed for each image processing task. The most basic imaging tasks such as loading images, saving images, getting and setting pixel color values, etc. were accomplished using library of functions called the Targa+ Tool Kit developed by Truevision Inc.. The neural networks were developed and trained using a neural network development package called Professional II/Plus from NeuralWare. This package then provided "C" source code for the networks that were used as a module in the main program.

The modular approach to programming allows for easy to follow flow code within the main program and allows the developed imaging processing routines to be used in any program. Figure 14 shows a flowchart of how each of the modules is incorporated into the main program and after this the purpose and a brief explanation is given for each module.

The program automatically saves sequential images as it goes through its routine. These images are saved as ORIGINAL.TGA, the original image; SAT.TGA, the image after resolution reduction and saturation enhancement; SEG.TGA, the image after color segmentation and FINAL.TGA, the final image with warning signs framed. These images will overwrite themselves each time a new image is processed so they must be saved under different names if future reference is desired. A file called SIGN.TXT is also provided that gives the position of each sign in the image as a boundary rectangle, the luminance values for each pixel in the boundary rectangle and the basic color this sign.
Figure 14 -- Flowchart of the software modules in the main program
6.1 Image Acquisition

The function named GrabImage is used to obtain a digital image from Composite Video, S-Video or RGB Video. It also has the capability to load an image directly from a file. Within this routine the user is prompted to enter the number corresponding to the manner in which an image is to be obtained:

0 = Composite Video
1 = S-Video
2 = RGB Video
3 = Load Image from a File

When grabbing an image from video, the video will appear on the analog monitor once a video format is selected with a message stating what type of video was selected. The user is then prompted to press "one" and then press the enter key at the exact moment the desired image appears in video. If an image is to be selected from a file then the user is prompted to enter the file name with extension (usually .TGA) and the program will load this image or tell the user an error was encountered in loading the specified file. If an error occurs the user should be sure the correct file spelling was used with extension and that the file is in standard Targa format as all files must be of this type. Once an image is grabbed or loaded the routine provides a message as such and informs the user of the resolution of this image. For proper execution of the rest of the program, the image should be in a 640 x 480 resolution and a warning is issued to this is not the case.

6.2 Resolution Reduction

A program call Mosaic is used to reduce the resolution of any size image by factors of two, four or eight. Upon entering the program a message is provided as to the original resolution and the resolution the image is being reduced to.
The purpose of this program is to simulate how an image would appear at different resolutions if actually digitized by the frame grabber at that resolution. As this program progress, the reduced image is drawn in the lower left hand corner. Upon completion of the reduction, the reduced image is zoomed by the factor of the reduction to fill the entire screen and a message is provided that the reduction procedure is completed. For the method used to accomplish this see the Resolution Reduction section in the Approach chapter.

6.3 Saturation Enhancement

The function SatEnhance is used to increase the colorfulness of any sized image. It accomplishes this by altering the saturation values of each pixel without effecting the hue or intensity of the pixels. When the function begins, a message is provided that saturation enhancement using a natural log function is being executed. A function called RGBtoHSI is used within this program to convert each pixels RGB value to an HSI value so that the saturation can be obtained. Once a pixels saturation is obtained, its percentage saturation is determined for its hue and this percentage is then increased by a factor obtained using a natural log function. The natural log is used so that pixels of weak saturation remain weak and higher saturated pixels become more saturated. This enhancement is done for each pixel starting in the bottom left corner and ending in the upper right corner and can be seen progressing on the analog monitor. At the end of the routine another message is provided that the saturation enhancement has been completed.

6.4 Color Segmentation

The color segmentation is conducted within a routine call ColorSeg that enacts the color segmentation neural network call ColNet. The basic procedure
is shown in Figure 15. When the routine begins the user is prompted with a message stating that color segmentation is taking place. This message is followed by a count down from the number of vertical lines that must be segmented in the image. While this is occurring the routine is writing each pixel’s segmented color to a file call Color.CMP. When the count down reaches zero, the segmentation is complete and each pixel’s color is set according to its value in the segmented file. The segmented pixel colors are show on the analog monitor as they are read from the file. Upon completion of this, a message is provided the color segmentation has been completed.

6.5 Sign Detection

The sign detection is accomplished using a routine call SegOut4. This routine involve finding possible regions, pre-processing these regions and analyzing them using a neural network. Before the procedure is enacted the user is prompted for the color of objects which he wants to scan for. Once in the routine, another prompt is given to provide the routine with the minimum and maximum sizes of these regions. The user can alter these values to see their effect or can not use this size restriction by entering zero for the minimum size and any number greater than the horizontal resolution for the maximum size. When pixels of the desired color are found, their regions are bounded and checked against the size restrictions. If the area is too small it is ignored, if it is too large a message is provided saying Wrong Size and giving the coordinates of this region.

If the region is of proper size, a message is given as such with its coordinates. This region is then scaled using a pre-processing routine called SignDetection and the results of the scaling is sent for analysis by the neural network. This sign recognition routine is called SinNet and basically is a
Obtain input values for each pixel in the neighborhood

Send these inputs to ColNet

Determine Max Output and which color it represents

Store this color as a file for the pixel of interest

If all pixels have been segmented

Read each pixel's color from the file and set them to this color on the screen

Figure 15 -- Color Segmentation module flowchart
diamond detection network. The output of this network then undergoes post-
processing and the result is given as message under the possible region. Upon
completion of the scanning, the user is prompted to scan for another color or
to quit the sign detection routine.

6.6 Framing

Before this routine is enacted the original 640 x 480 image is recalled to the
screen. The framing routine called FinalSignFrame then bounds each detected
sign with a blue rectangle.

6.7 ColorNet

The ColorNet program was written to improve the efficiency and reduce the
errors in developing the training file to be used in teaching the color
segmentation network. A mouse driven format is used to reduce the time
required to develop the training file and make the program more user friendly.
A program of this type is a requirement for anyone wanting to train a
supervised neural network for a specific color segmentation application. For
this purpose the source code for this program is provide along with the actual
sign detection program. This program begins by the user selecting a
representative image. This image is then reduced and the saturation enhanced
to produce the same type of image that would be presented to the
segmentation network. A color palette is then drawn along the bottom of the
image to represent the eight possible primary colors. Secondary colors are also
provided to build a training sets for cascading networks. The user then selects
a color from this color palette using the right mouse button and then places the
cross-hair on a pixel that appears to be this color and clicks the left mouse
button. The program then obtains the RGB values of this pixel and its
neighboring pixels (the user selects the size of the neighborhood in the beginning), converts these values to the two color difference inputs and writes these values along with the color selected from the color palette to a training file. It also records a file of HSI values for each pixel if this is desired to train the network with. During the program the computer keeps track of the number of training sets developed for each color. The program is ended when the right mouse button is clicked above the color palate.

6.8 SignNet

A SignNet program is also included to develop a training set for the sign recognition network. This program loads an already segmented image from a file and searches these images for possible sign locations. Upon finding a possible sign location, the program shows this region as it would appear as a 10 x 10 input matrix and prompts the user to tell the program if this region represents an actual sign. Normally, the region is not framed as the frame may interfere with possible sign regions but markers are provided on each edge of the image to show where the sign is located. However, if the user is still unsure of the exact region of interest he or she can have the program to frame the region of interest in the image. Once an answer is provided, the program writes the 10 x 10 binary matrix to a file along with the correct answer for that region. This allows for training sets to be developed very rapidly provided that a set of segmented images is available.
Chapter Seven -- Results and Discussion

7.1 Results of Sign Recognition Network

A total of 55 images containing warning signs were used to test the system. The program was able to detect 56.36% of these signs. However, of the 55 images only 40 represented separate warning signs, the remainder where images of a previous sign at a different angle. Of the 40 possible warning signs the program was able to detect 75% of them. This demonstrates how being able to view the sign at different angles as the car approaches greatly increases the reliability of the system. Furthermore, only 35 of the 40 signs were large enough in the image to be detected by the program. Thus, the program should not be blamed for these errors but, rather, the zoom factor of the camera is to blame and should be increased to alleviate this problem. If only the properly sized signs are considered then the reliability of the system increases to 86%. Within the 55 total images, the system only produced three false alarms which translates into a 9% false alarm rate.

A series of templates in Appendix A depicts how the original image is altered as it progresses through the program. Template 1 (p. A-1) shows an original image with a resolution of 640 x 480. However, the blue box around the sign would not be present at this time. Template 2 shows this same image after having the resolution reduced to 160 x 120 and the saturation increased using the natural log function. Template 3 then shows this same image after the ColNet neural network has chosen one of the eight colors for each pixel. Template 3 also shows the blue bounding rectangles that represent possible warning signs found by the scanning routine. After the SignNet analyzes these regions, Template 1 shows the original image with a blue box around the only region it designated as a warning sign. Templates 4-6 show a second image.
7.2 Processing Times

486 25MHz

Total Processing Time: 2 minutes 30 seconds
Digitize Image: real time*
Resolution Reduction: 48 seconds
Saturation Enhancement: 10 seconds
Color Segmentation: 1 minute 27 seconds
Search for Possible Signs: 2 - 5 seconds
Sign Recognition: real time*

386 25MHz no math co-processor

Total Processing Time: 52 minutes
Digitize Image: real time*
Resolution Reduction: 1 minute
Saturation Enhancement: 6 minutes
Color Segmentation: 45 minutes
Search for Possible Signs: 2 - 8 seconds
Sign Recognition: real time*

7.3 General Results

As an overall evaluation, the system perform extremely well in varying conditions, which is contributed to the use of neural networks and proper preprocessing of the data. Presently, this is a dedicated system to detect warning signs but it could be easily changed to detect any type of object given its color and shape. The only changes that would have to be made is to tell the system what color to look for and to build a network to recognize the shape of the object.
There are to date only a few circumstances that have led to failure in detecting a sign. The first is when a sign is so heavily shaded that it appears black or gray to the system. One method to solve this problem would be to retrain the color segmentation network using examples where a sign is highly shaded. The other problem occurs when a sign is set against a consistently yellow background. While this may be a problem for a single scene, it is not often the case where a sign is set against a yellow background for each of the frames as the camera approaches the sign. The number of frames, or chances, the system has to "see" a sign is determined by the number of frames taken per second, the size of the sign, the visual angle of the camera and the speed of the car.

Referring back to the outlining procedure, the system will accept an area between 7 and 30 pixels in a 160 x 120 image as being a possible sign which translates into an area between 28 and 120 pixels in the original 640 x 480 image. This range can be altered by the designer to meet specific design standards but reducing the minimum detectable sign will always increase the number of type II errors (false alarms). Chart 2 (p. 83) assumes a standard 36" x 36" warning sign and a visual angle of 38 degrees to show the width of the sign in pixels for various distances. Since this graph is for 640 x 480, the sign size can range from 28 to 120 pixels which translates into possible sign detection distances of 50 to 200 feet. Traveling at 55 mph this translates into 1.86 seconds of viewing time. If the system was capable of operating in real-time (30 frames/sec), it would then have at least 55 frames in which to detect a warning sign. These duplicate frames could be used in many ways to increase the reliability and the number of uses for the system. This could consist of analyzing duplicate frames to increase the number of signs detected and to reduce the number of false alarms. These sequential images would also
provide information about the size and placement of the sign which would be very important in using this system for inventorying signs. If the present Ohio Department of Transportation's photo log is used as the input images, then the images would be presented to the system at the rate of one frame per 52.8 feet. This would allow for the system to have a minimum of two frames to detect the presence of a 36" x 36" warning sign and most probably three frames would be present.

7.4 Discussion

The main factor in determining the success of the system is the image developed by the color segmentation network. If the warning sign in this image is for the most part segmented from the background and remotely resembles a diamond shape, then the rest of the system will have no problem in locating this region. For this region, the majority of the discussion deals with the color segmentation network.

7.4.1 Color Segmentation Network

While the arguments given for terminating the training of the color segmentation network before it converged may seem just a defense for a poorly trained network, the results of the network in segmenting actual roadway scenes speak for themselves and corroborate these arguments. Although it is not desirable to use the training file as a testing file this seems to provide some indications of the where the network had trouble learning. When the same training set is used as a testing set, it results in about 75% of the sets being identified as the correct color. As it was assumed that many errors would stem from the network confusing adjacent colors, 85% of the answers either gave the correct color or an adjacent color. In instances where adjacent
colors are confused, a second stage of networks could be used to rectify this problem and this approach is elaborated on later in this discussion. This means that adjacent colors constituted about 40% of the total error. The error in determining if a pixel is a color or an achromatic value is a similar problem where pixels lie adjacent to each other in the color subspace and would be expected to thus contain at least another 40% of the error as many achromatic training sets were used. If this is true, this problem of adjacent subspaces would account for at least 80% of the total error and as stated above cascading networks may prove beneficial in solving this problem.

For reasons discussed earlier, some of the other 20% of the error may have consisted of errors within the training set and thus would not be desirable for the network to learn. Further analyzing the errors occurring at adjacent locations in the color space, these types of errors were assumed to be caused by the inconsistencies in the training set as specified. Although the data may have seemed inconsistent to the network in its definition of color boundaries, this may not be the case if the overall lighting of the image is taken into consideration and means of feeding this information to the segmentation network could have been developed. This problem stems from the effect of the same object appearing as different colors under different lighting conditions. The human system deals with this problem by using color adaptation techniques which are still only vaguely understood. However, the result is that as lighting conditions change, the perceived value of "white" is shifted in our perceptual color space so that objects that may actually be red but that may now appear purple are still perceived as red.

Although numerical answers are impossible to obtain for analyzing the color segmentation routine as each of over 19,000 pixels in each image would have to analyzed, the network always provides a fair representation of the image
under varying conditions. At first it was thought that the segmentor might have problems segmenting pixels located on color edges. This is because the neighborhood of pixels would contain two separate colors. Fortunately the network learned to overcome this situation.

Another situation of concern was when a yellow sign was depicted against a similar yellow-green background. The network proved extremely good at correctly segmenting pixels within a warning sign as yellow while segmenting yellow-green background pixels as green. This property of the network is extremely important in this application for obvious reasons. The only noticeable problems arise in confusing adjacent colors and this is mainly true for blue and green pixels. Also, the network sometimes confuses a color for achromatic or visa-versa in cases of excessive shading.

For the most part the results were extremely positive, so much so that the idea of using cascading networks to improve results was not required. This is because in all cases tested the segmented image provided a clear depiction of the warning sign and an accurate depiction of the entire image. One note is that the network may require more training with shaded images for a more robust segmentor. Another concern arose when video of winter time roadway scenes was analyzed. In these images, a sign is sometimes set against a field of dead grass which has a brownish-yellow color. The segmentation network often calls the dead grass yellow and thus hides the warning sign in the background. The solution would be to retrain the segmentation network with images for these situations. In fact, the present segmentor was trained with images from the summer and fall months and thus should also include winter and spring images for a more robust segmentation system.

In future training, the segmentation network should be tested under varying possible conditions at periodic intervals in the learning process to determine the
optimum place to stop the learning process so that the network does not come over-trained.

7.4.2 Sign Recognition Network

The sign recognition network was very useful in proving the point that neural networks are garbage in, garbage out operators. Designing a neural network does not just consist of finding a way to represent the data to input and nodes and choosing the desired output nodes. The final sign recognition network is basically a diamond recognition network. This negates the need to explain to a computer what a diamond looks like and the hardest part of setting a threshold as to how far the shape can deviate from a diamond and still be called a diamond. The network learns this required information by transmitting what a person already knows is a proper diamond. In a system that would recognize all sign types, it is recommend to use many individual networks for the reasons already discussed. Thus, one network for each shape (diamond, rectangle, octagon, circle, etc.) should be used. As some examples, if a red region was detected the octagon network would be used, if a blue or green region was detected the rectangle network would be used.

7.4.3 Neural Networks

The characteristic that distinguishes this machine vision application from most others is its use of neural networks. In this application neural networks are used for scene segmentation and object recognition. The scene segmentation network accomplishes color resolution reduction in a manner unlike many histograming techniques that fail under varying conditions. The network used proved effective in represent many different natural scenes using only eight colors. This application may have benefited from the use of more
colors and a more diversified training set. Also, with the addition of more colors this method of color segmentation could be used in others applications. To achieve this, a series of cascading networks is suggested over trying to develop a single network to classify each color. This idea of cascading networks offers many advantages. The first is an increased number of colors only limited by the number of cascading networks chosen. Plus, the ability to only use more colors where needed. Thus if one wanted to detect only a variety of yellows, the cascading networks could be used only on the yellow regions. Cascading networks could also reduce the number of errors in segmenting colors fall on boundaries in the color spaces. Finally, cascading networks can be trained and maintained separately making for easy development and maintenance.

The first advantage of providing added colors works by introducing a second layer of networks. The segmentation network in this thesis segments what is referred to a primary set of colors (red, orange, yellow, etc.). A set of secondary colors could be obtained by training a second layer network for each output color (see Figure 16). This results in the primary network being followed by a red network, orange network, yellow network, etc.. These secondary networks are feed the same inputs as the primary network but only have three outputs each. These three outputs consist of moving a step down the color scale, staying on the same color, or moving a step up the color scale. As an example, if the primary net outputted orange then the orange net would be used and could output orange-red, orange or orange-yellow. This type of progression would continue until the desired color resolution is achieved. Notice how cascading networks only have to be included for colors that need more resolution. The way this reduces the number errors at boundaries in the color space is as such. If a pixel was yellow-orange the primary net would
Figure 16 -- Cascading Networks for Color Segmentation
have to pick either yellow or orange as the output. Suppose it choose yellow by a small margin, then the secondary yellow net would most likely classify the pixel as yellow-orange. However if the primary net chose orange, the orange net would also have the opportunity to output yellow-orange.

With a more extensive training file, better results could be achieved for adverse conditions such as a shaded sign. Although training sets are monotonous to develop, once the training is done it can be used for any application. This idea of a set of cascading networks that can be implemented in an any or all fashion could allow a designer to develop a custom natural scene segmentor for any application.

As a more general finding, the use of cascading networks could help in any neural network application. This idea uses a type of expert system approach to designing a series of networks. By using the existing knowledge of the situation, a designer can break the problem into learning steps to make the learning process easier for the neural network. In the same fashion, you would not expect a person to learn a whole problem at once but rather to use a series of fundamental steps to reach a conclusion. However, it is important to understand that these are fundamental steps that represent an entire procedure and not specific steps that could be programmed into a conventional computer.

7.4.4 Saturation Enhancement

The saturation enhancement routine is the only stage of the system where a set threshold is used to alter the image. In this case the natural log function. Although is performs well in most cases the use of a specific function in natural images will not prove optimal under all circumstances. For images of poor illumination, this procedure is very beneficial. However, on the other extreme when there is an abundance of color in the picture, the saturation enhancement
tends to increase the color of some pixels too much. This shows the problems with trying to develop a general vision system using formulas and thus why neural networks are so beneficial in this area.

7.4.5 Use of Color

The main advantage of color proved to be in segmenting the image into objects of interest and then being able to locate possible regions of interest quickly. It is interesting that humans also use color to quickly access information. Color did prove helpful when a sign was obscured with shading as long as the sign did not appear gray to the segmentation network.

7.4.6 Data Reduction

The major hurdle in using neural networks in image processing is the large amount of information provided in the image. This was not a problem when conducting operations on a single pixel or small neighborhoods but for larger neighborhoods or object recognition, this is a serious problem. The problem is that a small area of only 20 x 20 pixels would require 400 inputs if only a sign's luminance value for each pixel was used. This constitutes a rather large network when using a personal computer to simulate neural networks and takes an extremely long time to learn and recall information. The strategy used in this thesis was to use the information the humans found most useful and teach this to the neural networks while discarding other information. It is interesting that the images with less pixel resolution and less color resolution actually made sign more obvious to the human eye as well as the computer. This is an indication that final images only used the basic information most pertinent to the sign detection problem and eliminated other information that cluttered up the image.
7.4.7 Real-Time System

The present system takes about 3 minutes to observe an image and 1 minute 45 seconds not including the resolution reduction routine. To view video images at real time, it would have to analyze 30 images a second. This means this present system would have to operate at 3150 times its present rate. The first step in achieving this would be to use all integer math in the program and make better use of LUT's (look-up tables). This however would not come close to a real-time system. The problem lies in that to segment the image each of the 18,644 pixels in the 160 x 120 image must be analyzed by the network. Thus, the use of parallel processing or neural hardware would be required.

The use of parallel computing will probably be the standard within the next decade. In 1991 Thinking Machines Corps. introduced a massively parallel-processing supercomputer called the CM-5. This computer is reported to have a peak performance of one trillion instructions per second. Comparing this value to the super fast Cray Computer, the CM-5 could do more in a day than the Cray could do in a year. Intel has introduced the XP/X supercomputer for the UNIX system. This computer can operate from 5 to 30 billion operations per second and they are working with this system to develop a trillion floating point operations per second machine. The technology for these types of computers is here today and in the future will be able to make short work of the largest neural networks and most complicated color imaging routines.

A more practical solution for a dedicated system would be to put the trained network in hardware which is not only available now from many producers but is also affordable. The final observation is that in the near future (and today for the right price) systems will be available to run large neural
networks and these might be the key to developing a general vision system that has been desired for so long.
Chapter Eight -- Conclusions

The primary fields of study within this thesis have investigated the use of color in image processing and the possibility of using neural networks in place of more traditional image processing techniques. The thesis has attempted to determine the feasibility of using these techniques to detect highway warning signs under real-world conditions.

This study does not attempt to provide any color versus achromatic conclusions. However, it is concluded that color can be used as an effective attribute in detecting targets that use color as a means of identification. Since the human visual system finds color so appealing, these types of applications are abundant and the use of color is highly recommended for these situations.

The general results from both of the neural networks used in this thesis show dividends for incorporating neural networks into image processing. These networks do not require elaborate algorithms but only for the designer to pass on examples to the network. By no means is it recommended that all image processing steps be accomplished in this manner or that any image processing task can be achieved by a neural network.

What is recommended to produce successful networks are two main observations; pre-processing and cascading or multiply networks. Not enough could be said about the need for pre-processing and this is especially true in color image processing where so much information is provided and the network needs to be given the greatest percentage of meaningful information possible. When training a network on examples of people's perception of an image it makes sense to pre-process information by enhancing the features that make objects easier to see for people. This should not be mistaken for suggesting that neural networks learn from the same features that humans do or that
images seen as enhanced by humans will be seen as enhanced by a neural network. What is suggested is that the features appreciated by humans, such as color, are a good place to start when teaching a network by human example.

The second recommendation of using cascading or multiple neural networks is recommended to reduce the task of each network and thus increase its chances of learning. The use of a single network to solve an entire problem is highly discouraged. Even with our vast parallel processing capabilities, human still solve problems sequentially and rely heavily on apriori knowledge.

An example of using apriori knowledge is that stop signs are red octagons, warning signs are yellow diamonds, construction signs are orange diamonds, etc. This apriori knowledge allows for specific networks to be devoted to recognizing signs based on possible shapes given their color or visa-versa. A single network to recognize signs would have to simultaneously consider color and shape to determine if the inputs represent one of numerous types of signs or a non-sign. In general the recommendation is to include apriori knowledge in an expert system type architecture that accesses specific neural networks given the situation at hand.

The recommendation for using cascading networks can prove extremely effective in certain types of recognition problems. A good example of this type problem is color segmentation. If given a set of colored objects and told to separate them into different colors, most people will start by separating the objects into major hues, minor hues and then by their saturation. This procedure makes use of breaking the problem into stages where the findings of previous stages can help to reduce the task of the next phase. Also, this shows that humans absolute judgment is not very good but by segmenting objects in steps we can take advantage of our extremely well adapted relative
judgment. It is suggested here that neural networks may also not be as adapted to absolute judgment as they are to relative judgment. Thus, a better approach to color segmentation might be to train a network to find major hues, as in this thesis. Then if needed segment major colors into minor colors. This thesis had full intentions of incorporating this procedure but it turned out that the major hue segmentor provided the colors needed for sign detection.

By using neural networks in the manner suggested, neural networks could be substituted for most complex image processing algorithms and probably prove more robust and human-like. With the increasing processing power available and by using neural hardware this could lead to a long desired general vision system in the future.

As for the feasibility of this method to detect warning signs, all results show that a fairly robust sign detection system has been achieved. The robustness of this method is contributed to the use of color and the use of neural networks to replace image processing applications that tend to fail because of thresholds. With processing times of around 1 minute 45 seconds, its potential for real-time operation is well within technology if dedicated hardware is used and may be within the reach of parallel processing alone without dedicated hardware. This justifies a final conclusion that the methodology of this system could be implemented for automatic inventorying of highway signs or for use as an in-car information system to assist drivers in obtaining roadway information.
BIBLIOGRAPHY


Appendix A -- Roadway Images
Template 6  Color Segmented Image with Possible Sign Regions
Appendix B -- Warning Sign Detection Program
Main Program COLOR4S -- Main program used to call the functions required to achieve sign detection

Written by David Kellmeyer, Feb. 10, 1991

--- Specification of Include Files ---
#include <stdio.h>
#include <targraf.h>

--- Declaration of Internal Functions ---
void Mosaic(int x, int y, int side); /* provides resolution reduction */
void SatEnhance(int c, int d, double satlevel); /* increases the colorfulness of the image */
int GrabImage(); /* digitizes an image of given input source */
void SegOut4(int hSize, int vSize, int minsize, int maxsize, RGBColor color, int *SignNum, Rect SignRect[30]); /* outlines possible sign regions */
void SignFrame(int RectNum, Rect PossibleRect[30]); /* draws boundary rectangles for declared regions */
void Seg2(int hSize, int vSize); /* segments the image into eight colors and creates a file "color.cmp" to store the results in */
void FinalSignFrame(int side, int RectNum, Rect PossibleRect[30]);

--- Declaration of Variables ---
unsigned int hSize, vSize; /* the horizontal and vertical resolution of the image */
int side; /* the length of the averaging square used in Mosaic, the resolution will be reduced by this value */
int original;
Rect YellowSign[30], OrangeSign[30], RedSign[30], GreenSign[30], BlueSign[30], outrect;
int YellowNum, OrangeNum, RedNum, GreenNum, BlueNum;
RGBColor color;
RGBColor black = {0,0,0};
RGBColor yellow = {0xffff,0xffff,0};
RGBColor orange = {0xffff,38911.0,0};
RGBColor red = {0xffff,0,0};
RGBColor green = {0,0xffff,0};
RGBColor blue = {0,0,0xffff};
Rect imagerect;
Rect redrect;
Point upperleft;
int result;
int minsize, maxsize;
int searchcolor;
double satlevel;

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

/**
 ** checks to see if a proper Targa board and driver are available
 ** and if so initializes the graphics capabilities
 */

if(InitGraphics() < 0)
{
    puts("TARGA Driver not available");
    exit(-1);
}

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

flushall();        /* flushes all buffers */
SetZoom(FloatToFixed(1.0),FloatToFixed(1.0));     /* set Zoom back to 1 */
SetPanPos(0,0);   /* pan to lower left corner */

result = -1;

while(result == -1)
    result = GrabImage();             /* Digitizes an Image */

GetDisplaySize(&hSize, &vSize);             /* gets the horizontal and vertical resolution of the image */
printf("\n\nImage resolution is set to %d by %d\n", hSize, vSize);

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

/**
 ** Save the Orignal Image
 */
if(result != 3) {
    printf("Enter name of Original Image (*.TGA): ");
    scanf("%s", imagename); */
    SetRect(&imagerect,0,0,hSize,vSize);

    printf("Saving the Original Image...");
    if ( PutPic("original.tga", imagerect) == 0 )
        printf("Original Image Saved as -- Original.TGA\n");
    else
    { printf("There was a problem in saving the original image!!!!!!!!!!!!\n"); exit(-1);
    }
    /*
    ******************************************************************
    *********** Conducts Resolution Reduction  ***********
    ******************************************************************
    */
    printf("Enter Desired Resolution(1 = %dx%d, 2 = %dx%d, 4 = %dx%d or 8 = %dx%d): ", hSize, vSize, hSize/2, vSize/2, hSize/4, vSize/4, hSize/8, vSize/8);
    scanf("%d", &side); /* obtains a resolution reduction factor */
    if(side == 2 || side == 4 || side == 8) /* if resolution reduction selected */
        Mosaic(hSize, vSize, side); /* conducts resolution reduction */
    else
        printf("No Resolution Reduction -- Image Resolution is still %d by %d\n", hSize, vSize);
    /*
    ******************************************************************
    */
    printf("Enter value: ");
    scanf("%f", &satlevel);
    if(satlevel >= 0)
        SatEnhance((hSize/side), (vSize/side), satlevel); /* Increases the colorfulness of the image */

    SetRect(&redirect,0,0,hSize/side,vSize/side);

    printf("Saving the Saturated Image...");
if ( PutPic("sat.tga", redrect) == 0 )
    printf("Saturated Image Saved as -- Sat.TGA\n");
else
{
    printf("\nThere was a problem in saving the saturated image!!!!!!!!!!!!\n");
    exit(-1);
}

Seg2((hSize/side), (vSize/side)); /* Segments the image into eight colors */

/*
**   Colors Edge Pixels Black since they are not
**    effected by the previous color segmentation
**
**   Segments the image into eight colors
**
*/
SetRect(&outrect,0,0,(hSize/side),(vSize/side));
SetRGBForeColor(black);
FrameRect(outrect);

printf("\nSaving the Segmented Image...");
if ( PutPic("seg.tga", redrect) == 0 )
    printf("Segmented Image Saved as -- Seg.TGA\n");
else
{
    printf("\nThere was a problem in saving the segmented image!!!!!!!!!\n");
    exit(-1);
}

/*
**   Scan the image for desired color regions and then
**    outline these regions and then sends these regions
**    for analysis by the sign detection network. Those
**    regions being signs are returned for framing.
**
*/

printf("\nEnter the minimum size of a sign: ");
scanf("%d", &minsize);
printf("Enter the maximum size of a sign: ");
scanf("%d", &maxsize);

RedNum = 0;
OrangeNum = 0;
YellowNum = 0;
GreenNum = 0;
BlueNum = 0;

searchcolor = 100;
while(searchcolor != 0)
{
    printf("\n\nColor that can be searched for:\n 1 = red\n 2 = orange\n 3 = yellow\n 4 = green\n 5 = blue\n 0 = No More Searches\nEnter Color Value: ");
    scanf("%d", &searchcolor);

    if(searchcolor >= 0 && searchcolor <= 5)
    {
        if(searchcolor == 1)
            SegOut4(hSide, vSize, minsize, maxsize, red, &RedNum, RedSign); /* scan for yellow regions */
        else if(searchcolor == 2)
            SegOut4(hSide, vSize, minsize, maxsize, orange, &OrangeNum, OrangeSign); /* scan for yellow regions */
        else if(searchcolor == 3)
            SegOut4(hSide, vSize, minsize, maxsize, yellow, &YellowNum, YellowSign); /* scan for yellow regions */
        else if(searchcolor == 4)
            SegOut4(hSide, vSize, minsize, maxsize, green, &GreenNum, GreenSign); /* scan for yellow regions */
        else if(searchcolor == 5)
            SegOut4(hSide, vSize, minsize, maxsize, blue, &BlueNum, BlueSign); /* scan for yellow regions */
        else
            searchcolor = 0;
    }
    else
        printf("\nInvalid Search Color Value Chosen!!!!!!!!!!!\n");
}

if(RedNum > 0)
{
    printf("\n\nFraming Red Signs...\n");
    SignFrame(RedNum, RedSign); /* draws the boundary rectangle for yellow regions */
}
if(YellowNum > 0)
{
    printf("\n\nFraming Yellow Signs...\n");
    SignFrame(YellowNum, YellowSign); /* draws the boundary rectangle for yellow regions */
}
if(OrangeNum > 0)
{
    printf("\n\nFraming Orange Signs...\n");
    SignFrame(OrangeNum, OrangeSign); /* draws the boundary rectangle for orange regions */
if(GreenNum > 0)
{
    printf("\n\nFraming Green Signs...
");
    SignFrame(GreenNum, GreenSign);  /* draws the boundary rectangle for yellow regions */
}
if(BlueNum > 0)
{
    printf("\n\nFraming Blue Signs...
");
    SignFrame(BlueNum, BlueSign);     /* draws the boundary rectangle for orange regions */
}

/******************************************************************
** Draws the saturated 160x120 image
*******************************************************************/

original = 0;
printf("\n\nEnter 1 to View Original Picture: ");
scanf(" %d", &original);

if(original == 1)
{
    SetZoom(FloatToFixed(1.0), FloatToFixed(1.0));  /* resets the zoom back to 1 */
    SetPanPos(0,0);

    printf("\nLoading -- Original.TGA...");

    upperleft.x = 0;
    upperleft.y = vSize/side;

    if ( GetPic("original.tga", 0, upperleft) >= 0 )
        printf("Original.TGA successfully loaded\n");
    else
        {
            printf("\nThere was an ERROR in loading the Original image!!!!!!!\n");
            exit(-1);
        }

    FinalSignFrame(side,YellowNum,YellowSign);  /* Frames detected yellow signs */
    FinalSignFrame(side,OrangeNum,OrangeSign);   /* Frames detected orange signs */

    printf("\nSaving the Final Image...");
    if ( PutPic("final.tga", imagerect) == 0 )
        printf("Final Image Saved as -- Final.TGA\n");
    else
        {
            printf("\nThere was a problem in saving the final image!!!!!!!\n");
            exit(-1);
        }
}
SetZoom(FloatToFixed(1.0),FloatToFixed(1.0));  /* resets the zoom back to 1 */
SetPanPos(0,0);
EndGraphics(); /* Ends the Targa Board Graphics */
}
Function Mosaic
---
it combines a square group of pixels into a single pixel for all pixels in an image---
IT REDUCES THE IMAGE RESOLUTION

Written by David Kellmeyer, September 17, 1991

Specification of Include Files

---

/*
Begin
*/

void Mosaic(int x, int y, int side)

Declaration of Arguments

---

*x: the present horizontal resolution of the image
*y: the present vertical resolution of the image
*side: the length of a side in pixels of the square that will be averaged into a single pixel value

Declaration of Variables

---

/*
Begin
*/

int ij; /* counters */
long sumred, sumgreen, sumblue; /* sum of the pixels to be averaged */
RGBColor mosaicpixel; /* the averaged value of the original pixels */
float fside; /* floating point value of side */
RGBColor color; /* pixel color of the original image */
int u,v; /* counters */
printf("\nOriginal Resolution was %dx%d\n", x, y);
printf("\nResolution being reduced to %dx%d\n", x/side, y/side);

for(j=0; j<y; j=j+side)           /* Moves vertically up the screen in increments equal to the length of
   the averaging square */
   {
      for(i=0; i<x; i=i+side)     /* Moves horizontally across (left to right) the screen in increments
          equal to the length of the averaging square */

          {

              sumred = 0;
              sumgreen = 0;        /* set the sum values to 0 */
              sumblue = 0;


              for(v=j; v<j+side; v++)     /* Moves vertically within each averaging square one line at
                  a time */
                  {
                      for(u=i; u<i+side; u++)   /* Moves horizontally (left to right) within each averaging
                                          square one pixel at a time */
                          {
                          GetRGBPixel(u,v,&color);   /* Gets the RGB values of the specified pixel */

                          sumred = sumred + color.red;
                          sumgreen = sumgreen + color.green;       /* add this RGB value to the RGB sum of the rest of
                          the pixels in that square */
                          sumblue = sumblue + color.blue;

                          }
                  }

          }

          mosaicpixel.red = sumred/(side*side);
          mosaicpixel.green = sumgreen/(side*side);      /* divide the sum by the total number of pixels in
          each averaging square to get the average RGB value */
          mosaicpixel.blue = sumblue/(side*side);

          SetRGBPixel((i/side),(j/side),mosaicpixel);      /* set a single pixel equal to the average RGB value */
          }
          }
}

fside = side;
SetZoom(FloatToFixed(fside),FloatToFixed(fside)); /* Zoom the image by the factor that the image was reduced by */
SetPanPos(0,0); /* pan down to the lower left corner of the image */

printf("...Resolution Reduction has been completed\n");
}
Function -- SatEnhance is a function that increases the saturation of each pixel using a natural log function.

Written by David Kelmeyer; October 24, 1991

#include <targraf.h>
#include <rnath.h>
#include <erm0.h>

void RGBtoHSI(RGBColor pixelcolor, double *Hue, double *Sat, double *Inten);
void HSItoRGB(double Hue, double Sat, double Inten, RGBColor *color);

void SatEnhance(int c, int d, double satlevel)

#include <targgraf.h>
#include <rnath.h>
#include <erm0.h>

void RGBtoHSI(RGBColor pixelcolor, double *Hue, double *Sat, double *Inten);
void HSItoRGB(double Hue, double Sat, double Inten, RGBColor *color);

void SatEnhance(int c, int d, double satlevel)
/**
 * Declaration of Variables
 */

double pi, angle1, angle2, angle3, k1, k2, s1, s2, s3, x, y, sumsq1, sumsq2; /* constants used in defining the HSI color space */
double theta, alpha, beta; /* angles made by a given hue in the HSI color space */
double maxsat; /* maximum distance from white for a given hue */
double persat; /* % saturation of the original color */
double newsat; /* % saturation after saturation enhancement */
double Hue, Sat, Inten; /* hue, saturation and intensity values used in describing a color in HSI color space */

register int i, j; /* counters */

RGBColor pixelcolor, color; /* color of the original pixel, color of the saturated pixel */

/**
 * Constant Specifications and Angle Equations
 */

pi = 3.141592654;
x = 2.0/3.0;
y = 1.0/3.0;

angle1 = atan2(y, x);
angle1 = angle1*180.0/pi;
angle2 = 45.0; /* see report meaning of variables */
angle3 = angle2 - angle1;

s3 = 360.0 - angle1;
s1 = 90.0 + angle1;
s2 = 225.0;

sumsq1 = (1.0/3.0)*(1.0/3.0) + (2.0/3.0)*(2.0/3.0);
sumsq2 = (1.0/3.0)*(1.0/3.0)*2.0;
k1 = sqrt(sumsq1);
k2 = sqrt(sumsq2);

*/

printf("\nExecuting SatEnhance using natural log...");

/**
 * Constant Specifications and Angle Equations
 */
for(j = 0; j < d; j++) /* move vertical up the image line by line */
{
for(i = 0; i < c; i++) /* moves horizontally across the image (left to right) */
{

GetRGBPixel(i,j,&pixelcolor); /* Gets RGB values for the pixel of interest */
RGBtoHSI(pixelcolor, &Hue, &Sat, &Inten); /* Convert RGB values to HSI values */

/*****************************/
** Determines the Maximum Saturation distance **
** for the this pixels Hue **
(See report for explanation of equations)
/*****************************/

if(Hue == 1000.0)
{
    maxsat = 1.0/10000.0;
}
else if(Hue > s3 && Hue < 360.0)
{
    theta = Hue - s3;
    alpha = angle3;
    beta = 180.0 - theta - alpha;
    maxsat = k1*sin((alpha*pi/180.0))/sin((beta*pi/180.0));
}
else if(Hue >= 0.0 && Hue <= s1)
{
    theta = Hue + angle1;
    alpha = angle3;
    beta = 180.0 - theta - alpha;
    maxsat = k1*sin((alpha*pi/180.0))/sin((beta*pi/180.0));
}
else if(Hue > s1 && Hue <= s2)
{
    theta = Hue - s1;
    alpha = angle1;
    beta = 180.0 - theta - alpha;
    maxsat = k1*sin((alpha*pi/180.0))/sin((beta*pi/180.0));
}
else if(Hue > s2 && Hue <= s3)
{
    theta = Hue - s2;
    alpha = angle2;
    beta = 180.0 - theta - alpha;
    maxsat = k2*sin((alpha*pi/180.0))/sin((beta*pi/180.0));
}
else
{
    printf("ERROR -- No Hue Value Found in SatEnhance\n");
}
persat = Sat/maxsat; /* determines the percentage of saturation for the pixel of interest */

/*****************************
** Increase this % of saturation using **
** the natural log function **
*****************************/

persat = persat-satlevel;
if(persat*100.0 >= 1.0) /* if % Sat >= 1 % */
    newsat = persat*log(persat*100.0); /* increases saturation % */
else
    newsat = 0;
if(newsat > 1.0) /* allows for a maximum Sat % of 100 % */
    newsat = 1.0;

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

Sat = maxsat*newsat; /* sets Saturation to its new level */

HSItoRGB(Hue, Sat, Inten, &color); /* converts HSI back to RGB */

SetRGBPixel(i,j,color); /* set pixel to saturated RGB value */
}
}

printf("\n...Saturated Enhancement Completed\n");
/** Function GrabImage -- Sets the Live Display Mode and Grabs a Frame **
****************************************************************************/

/* Written by David Kellmeyer, Jan. 30, 1991 */

/******************************************************************************
** Specifies the Include Files **
****************************************************************************/

******************************************************************************
**
#include <stdio.h>
#include <targraf.h>

/******************************************************************************
**
int GrabImage()

/******************************************************************************
** Begin */
{

/******************************************************************************
** Declaration of Variables **
****************************************************************************/

******************************************************************************
**
unsigned int dispMode; /* the display mode, the state of the buttons on the mouse */
int inputsource; /* use to define the type of input source desired by user */
int dummy; /* dummy variable */
Point upperleft = {0,0};
int result;
char imagename[20];

/******************************************************************************
**
printf("\nValid Video Input Sources\n 0=Composite\n 1=S-Video\n 2=RGB\n 3=Image on File\nSelect Source : ");
scanf("%d", &inputsource); /* Recieve Video input source */

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

/******************************************************************************
** Set the Targa board to recieve the chosen Video **
**
******************************************************************************

/* Written by David Kellmeyer, Jan. 30, 1991 */

if(inputsource == 0) {
    SelectLiveSource(vidInputComposite);
    printf("nComposite Video Selected\n");
    result = 0;
}
else if(inputsource == 1) {
    SelectLiveSource(vidInputSVideo);  // set Targa board to accept composit video */
    printf("nS-Video Selected\n");
    result = 1;
}
else if(inputsource == 2) {
    SelectLiveSource(vidInputRGB);
    printf("nRGB Video Selected\n");
    result = 2;
}
else if(inputsource == 3) {
    printf("nEnter name of image File: ");
    scanf(" %s", imagename);
    printf("nLoading -- %s...", imagename);
    if ( GetPic(imagename, 0, upperleft) >= 0 )
        printf(" %s successfully loaded\n", imagename);
    else
        {
        printf("nThere was an ERROR in loading %s!!!!!!!!!!!\n", imagename);
        exit(-1);
        }
    result = 3;
}
else {
    printf("nInvalid video input source selected!!!!!!!!!!!!\n");
    result = -1;
}

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

if( result != 3 & result >= 0 ) {
    GetDisplayMode(&dispMode);  // stores the present display mode */
    EnableGenlock();  // set the Targa board to GenLock */
    SetDisplayMode(dispModeLive);  // set the display mode to receive live video */
printf("Enter 1 to Grab Frame...: ");
scanf("%d", &dummy);

GrabFrame();                      /* Grabs the live video frame when any value is ENTERED */

printf("...Frame has been Grabbed\n");

SetDisplayMode(dispMode);         /* set the board back to it original display mode */

return(result);


/* Function SegOutline -- Outline areas given by the color variable and then frames them */

/* Written by David Kellmeyer, September 17, 1991 */

#include <targraf.h>

int SignDetection(RGBColor color, Rect PossibleRect);

void SegOut4(int hSize, int vSize, int msize, int maxsize, RGBColor color, int *SignNum, Rect SignRect[30])

/* Begin */
{
  int j;
  register int c,d;
  int b,z,dir,a,w,turn;
  int xmin,xmax,ymin,ymax;
  RGBColor pixelcolor, corner1, corner2;
  Point startpoint, prev;
  Rect ColorRect[50];
  Rect WrongRect[50];
  Rect PossibleRect[30];

  b = 0;
  z = 0;
  w = 0;

  /********** OUTLINES THE COLOR AREAS**********/

  for(j=1; j<vSize-1; j++)
  {
    dir = 2;
    d = j;

    xmin = hSize;
    xmax = 0;
    ymin = vSize;
    ymax = 0;

    for(c=0; c<hSize-1; c++)
    {
      GetRGBPixel(c,d,&pixelcolor);

      /**********Check to see if this pixel is already within a bounded rectangle**********/

      for(a=1; a<=b; a++)
      {
        if(c<ColorRect[a].x2 && c = ColorRect[a].x1 && d < ColorRect[a].y2 &&
          d = ColorRect[a].y1)
        {
          if(ColorRect[a].x2 > c)
            c = ColorRect[a].x2;
if(pixelcolor.red == color.red && pixelcolor.green == color.green && pixelcolor.blue == color.blue) {
    startpoint.x = c;
    startpoint.y = d;

    if(c < xmin) 
        xmin = c;
    if(c > xmax) 
        xmax = c;
    if(d < ymin) 
        ymin = d;
    if(d > ymax) 
        ymax = d;

    prev.x = c;
    prev.y = d;

    d++; 
    dir = 1;

    while(c != startpoint.x || d != startpoint.y) 
    {
        if(c < xmin)
            xmin = c;
        if(c > xmax)
            xmax = c;
        if(d < ymin)
            ymin = d;
        if(d > ymax)
            ymax = d;

        GetRGBPixel(c, d, &pixelcolor);

        if(pixelcolor.red == color.red && pixelcolor.green == color.green && pixelcolor.blue == color.blue) 
        {
            if(c == prev.x || d == prev.y)
            {

turn = 1;
prev.x = c;
prev.y = d;
}
else {
GetRGBPixel(c,prev.y,&corner1);
GetRGBPixel(prev.x,d,&corner2);
if (corner1.red==color.red &amp; corner1.green==color.green &amp; corner1.blue==color.blue) || (corner2.red==color.red &amp; corner2.green==color.green &amp; corner2.blue==color.blue) )
{
turn = 1;
prev.x = c;
prev.y = d;
}
else
turn = 0;
}
else
turn = 0;

if(turn == 1) /* Left Turn */
{
if(dir == 1)
{
--;
dir = 4;
}
else if(dir == 2)
{
++;
dir = 1;
}
else if(dir == 3)
{
++;
dir = 2;
}
else
{
--;
dir = 3;
}
}
else /* Right Turn */
{
if(dir == 1)
{
c++;
dir = 2;
}
else if (dir == 2) {
    d--;  
    dir = 3;
} else if (dir == 3) {
    c--;  
    dir = 4;
} else {
    d++;  
    dir = 1;
}

//**************Upon return to the startpoint***************/

if(c == startpoint.x && d == startpoint.y) {
    d--;  
    if(d < ymin)  
        ymin = d;
}

//**************For regions of proper size***************/

if ( (xmax-xmin > minsize) && (ymax-ymin > minsize) && (xmax-xmin <= maxsize) && (ymax-ymin <= maxsize) )
{
    b++;  
    SetRect(&ColorRect[b],xmin,ymin,xmax+1,ymax+1);  
    PossibleRect[b] = ColorRect[b];  
    printf("%dPossible Sign (%d, %d) Min = (%d, %d) Max = (%d, %d)\n", b, xmin, ymin, xmax, ymax);

    if( (SignDetection(color, PossibleRect[b]) == 1) )
    {
        z++;  
        SignRect[z] = PossibleRect[b];
    }

    d = j;  
    c = xmax + 1;  
    xmin = hSize;  
    xmax = 0;  
    ymin = vSize;  
    ymax = 0;
}

//**************For regions too large***************/
else if( (xmax-xmin > maxsize) || (ymax-ymin > maxsize) )
{
    w += 1;
    SetRect(&WrongRect[w],xmin,ymin,xmax+1,ymax+1);
    printf("Wrong Size (%d) -- Min = (%d, %d) Max = (%d, %d)\n", w,xmin, ymin, xmax, ymax);
    d = j;
    c = xmax + 1;
    xmin = hSize;
    xmax = 0;
    ymin = vSize;
    ymax = 0;
}

/**********For regions too small***********/

else
{
    d = j;
    c = xmax + 1;
    xmin = hSize;
    xmax = 0;
    ymin = vSize;
    ymax = 0;
}

*SignNum = z;
Function SignFrame -- Frames the possible sign locations */

Written by David Kellmeyer, September 17, 1991 */

#include <targraf.h>

void SignFrame(int RectNum, Rect PossibleRect[30])

/* Begin */
{
    int a;
    RGBColor crazycolor = {10000,5000,50000.0};

    SetRGBForeColor(crazycolor);

    for(a=1; a<=RectNum; a++)
    {
        FrameRect(PossibleRect[a]);
        printf("Probable Sign Location (%d, %d) (%d, %d)\n", PossibleRect[a].x1, PossibleRect[a].y1, PossibleRect[a].x2, PossibleRect[a].y2);
    }

    .

```c
#include <stdio.h>
#include <targraf.h>
#include <math.h>

int ColNet(void *NetPrt, float Yin[18], float Yout[8]);
void NNMax(double Inten, float input[8], int *color);
void CompactPixel(int i, int j, int input);
void RGBtoNAT(RGBColor pixelcolor, float *input1, float *input2, float *input3);
void RGBtoHSI(RGBColor pixelcolor, double *Hue, double *Sat, double *Inten);

Seg2(int hSize, int vSize)
{
    register int c, d;
    int i, j, k, p;
    int color, dummy;
    RGBColor pixelcolor, ijcolor; /* the color of the designated pixel */
    float input[18], output[8];
    float input1, input2, input3; /* the natural color inputs computed from the RGB values */
    double Hue, Sat, Inten;
    FILE *ColorFile;

    printf("Executing Color Segmentation...
");
    ColorFile = fopen("color.cmp", "w");

    for(j = 1; j < vSize-1; j++)
    {
        printf("%d ", vSize-1-j);
        for(i = 1; i < hSize-1; i++)
        {
            GetRGBPixel(i, j, &pixelcolor);
            RGBtoHSI(ijcolor, &Hue, &Sat, &Inten);

            k = 0;
            for(d = j - 1; d <= j + 1; d++)
            {
                for(c = i - 1; c <= i + 1; c++)
                {
                    GetRGBPixel(c, d, &pixelcolor); /* gets the RGB values of a pixel */
                    RGBtoNAT(pixelcolor, &input1, &input2, &input3); /* changes the RGB to natural color values */
                    if(input2 > 0)
                        input2 = input2/2;
                    if(input1 < -1.0 || input1 > 1.0 || input2 < -1.0 || input2 > 1.0)
                        printf("\n\n********ERROR IN INPUT VALUES***********");
                    exit(-1);
                }
            }
        }
    }
}
```
ColNet(&dummy, input, output);
NNMax(Inten, output, &color);
fprintf(ColorFile, "%d ", color);
}
}
fclose(ColorFile);
ColorFile = fopen("color.cmp", "r");
for(j = 1; j < vSize-1; j +++)
{
    for(i = 1; i < hSize-1; i +++)
    {
        fscanf(ColorFile, "%d", &color);
        CompactPixel(i, j, color);
    }
}
fclose(ColorFile);
printf("\n...Color Segmentation Completed\n");
/* Function FinalSignFrame -- Frames the possible sign locations */

/* Written by David Kellmeyer, September 17, 1991 */

#include <graf.h>

void FinalSignFrame(int side, int RectNum, Rect PossibleRect[30])

/* Begin */
{
  int a;
  RGBColor crazycolor = {10000,5000,50000.0};
  Rect BigRect;

  SetRGBForeColor(crazycolor);

  for(a = 1; a <= RectNum; a++)
  {
    BigRect.x1 = PossibleRect[a].x1*(side);
    BigRect.x2 = PossibleRect[a].x2*(side);
    BigRect.y1 = PossibleRect[a].y1*(side);
    BigRect.y2 = PossibleRect[a].y2*(side);
    FrameRect(BigRect);
    printf("Possible Sign Located at (%d, %d) (%d, %d)\n", BigRect.x1, BigRect.y1, BigRect.x2, BigRect.y2);
  }
}


/* Function HSItoRGB -- it converts HSI values to RGB values */

/* Written by David Kellmeyer, September 17, 1991 */

#include <math.h>
#include <errno.h>
#include <targraf.h>

void HSItoRGB(double Hue, double Sat, double Inten, RGBColor *color)
/*
** x:The X value of the pixel to be converted to HSI
** y:The Y value of the pixel to be converted to HSI
** HSI: a pointer to the variable holding the HSI value of the pixel
*/
/* Begin */
{
    double max, ratio;
    double colorsum;
    double cR, cG, cB; /* NTSC constants */
    double tR, tG, tB; /* the % of red and green in a pixel */
    double v, h; /* vertical and horizontal distance from point white */
    double Wx, Wy; /* constant coordinates for the point white */
    double pi, Sat2, Tan2, TanHue, tred, tgreen, tblue;
    RGBColor tempcolor;

    cR = 0.299000;
    cG = 0.587000;
    cB = 0.114000;
    Wx = 0.333333;
    Wy = Wx;
    pi = 3.141592654;

    /* printf("Executing HSI to RGB Conversion...\n"); */

    /* printf("cR = %lf, cG = %lf, cB = %lf
", cR, cG, cB); */
    /* printf("Wx = %lf, Wy = %lf
", Wx, Wy); */

    Sat2 = Sat*Sat;

    if(Hue == 1000.0)
    {
        h = 0.0;
        v = 0.0;
    }
    else if(Hue == 90.0)
    {
        h = 0.0;
        v = Sat;
    }
    else if(Hue == 270.0)
    {
        h = 0.0;
        v = (-1)*Sat;
    }
```c
} else if(Hue == 0.0)
{
    h = Sat;
    v = 0.0;
}
else if(Hue == 180.0)
{
    h = (-1)*Sat;
    v = 0.0;
}
else
{
    TanHue = tan((Hue*pi/180.0));
    Tan2 = TanHue*TanHue;
    h = sqrt( Sat/(1.0+Tan2) );
    if(Hue>90.0 && Hue<270.0)
        h = h*(-1.0);
    v = h*TanHue;
}
/* printf("h = %lf, v = %lf\n", h, v); */

tR = h + Wx;
 tG = v + Wy;
 tB = 1.0 - tR - tG;
/* printf("tR = %lf, tG = %lf, tB = %lf\n", tR, tG, tB); */

colorsum = Inten/(cR*tR + cG*tG + cB*tB);
/* printf("colorsum = %lf\n", colorsum); */

tred = tR*colorsum;
tgreen = tG*colorsum;
tblue = tB*colorsum;
/* printf("tred = %lf, tgreen = %lf, tblue = %lf\n", tred, tgreen, tblue); */

max = 65535.0;
if(tred > max || tgreen > max || tblue > max)
{
    if(tred > max)
    {
        max = tred;
        ratio = 65535.0/max;
    }
    if(tgreen > max)
    {
        max = tgreen;
        ratio = 65535.0/max;
    }
    if(tblue > max)
    {
        max = tblue;
        ratio = 65535.0/max;
    }
    tred = tred*ratio;
```
tgreen = tgreen*ratio;
tblue = tblue*ratio;
}

tempcolor.red = (unsigned int)tred;
tempcolor.green = (unsigned int)tgreen;
tempcolor.blue = (unsigned int)tblue;
/* printf("tempred = %u, tempgreen = %u, tempblue = %u\n", tempcolor.red, tempcolor.green, tempcolor.blue); */

*color = tempcolor;

/* printf("...HSI to RGB Conversion Completed.\n"); */
}
/ Function RGBtoHSI -- it converts RGB values to HSI values */

/* Written by David Kellmeyer, September 17, 1991 */

#include <math.h>
#include <errno.h>
#include <targraf.h>

void RGBtoHSI(RGBColor pixelcolor, double *Hue, double *Sat, double *Inten)
{ /* x:The X value of the pixel to be converted to HSI
    ** y:The Y value of the pixel to be converted to HSI
    ** HSI: a pointer to the variable holding the HSI value of the pixel */

    /* Begin */
    {
    long colorsum;
    double cR, cG, cB; /* NTSC constants */
    double tR, tG, tB; /* the % of red and green in a pixel */
    double v, h; /* vertical and horizontal distance from point white */
    double Wx, Wy; /* constant coordinates for the point white */
    double pi;
    
    cR = 0.299000;
    cG = 0.587000;
    cB = 0.114000;
    Wx = 1.0/3.0;
    Wy = Wx;
    pi = 3.141592654;
    
    /* printf("Executing RGB to HSI Conversion...\n"); */

    /* printf("cR = %lf, cG = %lf, cB = %lf\n", cR, cG, cB); */
    /* printf("Wx = %lf, Wy = %lf\n", Wx, Wy); */
    
    colorsum = (long)pixelcolor.red + (long)pixelcolor.green + (long)pixelcolor.blue;
    /* printf("colorsum = %lf\n", colorsum); */

    if(colorsum == 0)
    {
        tR = 1.0/3.0;
        tG = tR;
        tB = 1.0 - tR - tG;
    }
    else
    {
        tR = (double)pixelcolor.red/(double)colorsum;
        tG = (double)pixelcolor.green/(double)colorsum;
        tB = 1.0 - tR - tG;
        /* printf("perRed = %lf, perGreen = %lf, perBlue = %lf\n", tR, tG, tB); */
    }
}
*Inten = cR*(double)pixelcolor.red + cG*(double)pixelcolor.green + cB*(double)pixelcolor.blue; /* conversion for intensity */

    h = tR - Wx;
    v = tG - Wy;
    /* printf("h = %lg, v = %lg\n", h, v); */

    *Sat = sqrt((h*h+v*v)); /* conversion for Saturation */

    if(h == 0.0 || v == 0.0)
    {
        if(h==0.0)
        {
            if(v==0.0)
                *Hue = 1000.0;
            else if(v>0.0)
                *Hue = 90.0;
            else
                *Hue = 270.0;
        }
        else
        {
            if(h>0)
                *Hue = 0.0;
            else
                *Hue = 180.0;
        }
    }
    else
    {
        *Hue = atan2(v,h)*180.0/pi; /* conversion for Hue */
        if(*Hue < 0.0)
            *Hue = 360.0 + *Hue;
    }

    /* printf("...RGB to HSI Conversion Completed\n"); */
/* Function SegOutline -- Outline areas given by the color variable and then frames them */

/* Written by David Kellmeyer, Feb. 11, 1991 */

#include <stdio.h>
#include <targraf.h>

int simet(void *NetPtr, float Yin[100], float Yout[2]);

int SignDetection(RGBColor color, Rect PossibleRect)

/* Begin */
{
    int c,d,i,j,k;
    int Xindex, Yindex, Maxindex;
    int SignSizeX, SignSizeY;
    int Xdif, Ydif;
    int colorcount, nullcount;
    Rect temprect;
    RGBColor pixelcolor;
    float input[100];
    float output[2];
    int dummy;

    /***************************GET INPUT FOR NEURAL NET FOR YELLOW SIGNS***********/

    SignSizeX = (PossibleRect.x2-1) - (PossibleRect.x1+1);
    SignSizeY = (PossibleRect.y2-1) - (PossibleRect.y1+1);

    if(SignSizeX <= 10)
        Xindex = 1;
    else if(SignSizeX <= 20)
        Xindex = 2;
    else if(SignSizeX <= 30)
        Xindex = 3;
    else if(SignSizeX <= 40)
        Xindex = 4;
    else if(SignSizeX <= 50)
        Xindex = 5;
    else if(SignSizeX <= 60)
        Xindex = 6;
    else if(SignSizeX <= 70)
        Xindex = 7;
    else if(SignSizeX <= 80)
        Xindex = 8;
    else if(SignSizeX <= 90)
        Xindex = 9;
    else if(SignSizeX <= 100)
        Xindex = 10;
    else if(SignSizeX <= 110)
        Xindex = 11;
else if(SignSizeX <= 120)
    Xindex = 12;
else
{
    printf("There is an X index problem!!!!!!!!!!!");
    exit(-1);
}

if(SignSizeY <= 10)
    Yindex = 1;
else if(SignSizeY <= 20)
    Yindex = 2;
else if(SignSizeY <= 30)
    Yindex = 3;
else if(SignSizeY <= 40)
    Yindex = 4;
else if(SignSizeY <= 50)
    Yindex = 5;
else if(SignSizeY <= 60)
    Yindex = 6;
else if(SignSizeY <= 70)
    Yindex = 7;
else if(SignSizeY <= 80)
    Yindex = 8;
else if(SignSizeY <= 90)
    Yindex = 9;
else if(SignSizeY <= 100)
    Yindex = 10;
else if(SignSizeY <= 110)
    Yindex = 11;
else if(SignSizeY <= 120)
    Yindex = 12;
else
{
    printf("There is an Y index problem!!!!!!!!!!!");
    exit(-1);
}

if(Yindex > Xindex)
    Maxindex = Yindex;
else
    Maxindex = Xindex;

Xdif = 10*Maxindex - SignSizeX;
Ydif = 10*Maxindex - SignSizeY;

temprect.x1 = PossibleRect.x1 + 1;
temprect.y1 = PossibleRect.y1 + 1;
temprect.x2 = PossibleRect.x2 - 1;
temprect.y2 = PossibleRect.y2 - 1;

temprect.x1 = temprect.x1 - (int)(Xdif/2);
temprect.x2 = temprect.x2 + (int)((Xdif+1)/2);
temprect.y1 = temprect.y1 - (int)(Ydif/2);
temprect.y2 = temprect.y2 + (int)((Ydif+1)/2);

k = 0;
for(j=temprect.y1; j<temprect.y2; j=j + Maxindex)
{
    for(i=temprect.x1; i<temprect.x2; i=i + Maxindex)
    {
        colorcount = 0;
        nullcount = 0;

        for(d=j; d<j+Maxindex; d++)
        {
            for(c=i; c<i+Maxindex; c++)
            {
                GetRGBPixel(c,d,&pixelcolor);
                if(c<temprect.x1 || d<temprect.y1 || c>=temprect.x2 || d>=temprect.y2)
                    nullcount++;
                else if(pixelcolor.red == color.red && pixelcolor.green == color.green &&
                        pixelcolor.blue == color.blue)
                    colorcount++;
                else
                    nullcount++;
            }
        }

        if(colorcount >= Maxindex)
        {
            input[k] = 1;
            k++;
        }
        else
        {
            input[k] = 0;
            k++;
        }
    }
}

sinnet(&dummy, input, output);

printf("Sign out = %f, Non-Sign out = %f\n", output[0], output[1]);
if(output[0] > = 0.5 && output[1] < = 0.5)
    return(1);
else
    return(0);
/* Function RGBtoNAT -- it converts RGB values to natural visual input values /*

/* Written by David Kellmeyer, September 17, 1991 */

#include <targraf.h>
#include <stdio.h>

void RGBtoNAT(RGBColor pixelcolor, float *input1, float *input2, float *input3)

  /** pixelcolor - the incoming RGB value of a single pixel */
  /** input1 - a pointer to the location the first natural color value will be stored */
  /** input2 - a pointer to the location the second natural color value will be stored */

  /* Begin */
  {
    float perR, perG, perB; /* the % of red and green in a pixel */
    long colorsum; /* the sum of the red, green and blue values */

    printf("\nExecuting RGB to NAT Conversion...
"); */

    /* printf("\nExecuting RGB to NAT Conversion...
"); */

    /* computes the sum of the R,G and B values */
    colorsum = (long)pixelcolor.red + (long)pixelcolor.green + (long)pixelcolor.blue;

    *input3 = colorsum/98302.5 - 1;

    if(colorsum == 0) /* if equal to zero make the sum equal one so division by zero does not occur */
      colorsum = 1;

    /* compute the percentage of R, G and B in the pixel color */
    perR = (float)pixelcolor.red / (float)colorsum;
    perG = (float)pixelcolor.green / (float)colorsum;
    perB = (float)pixelcolor.blue / (float)colorsum;

    *input1 = (float)perR - (float)perG; /* compute first natural value, (+) -> red, (-) -> green */
    *input2 = 2*(float)perB - (float)perR - (float)perG; /* compute second natural value, (+) -> blue, (-) -> yellow */

    /* printf("...RGB to NAT Conversion Completed.\n"); */
  }

  .
/* Function CompactPixel -- determines a single pixel color */

/* Written by David Kellmeyer, September 17, 1991 */

#include <targraf.h>

void CompactPixel(int i, int j, int input) {

    RGBColor red = {Oxffff,0,0};  /* the color of the designated pixel */
    RGBColor orange = {0xffff,38911.0,0}; /* the color of the designated pixel */
    RGBColor yellow = {0xffff,0xffff,0}; /* the color of the designated pixel */
    RGBColor green = {0,0xffff,0}; /* the color of the designated pixel */
    RGBColor blue = {0,0,0xffff}; /* the color of the designated pixel */
    RGBColor violet = {28672,0,40959.0}; /* the color of the designated pixel */
    RGBColor gray = {32500,32500,32500}; /* the color of the designated pixel */
    RGBColor brown = {25000,15000,0}; /* the color of the designated pixel */
    RGBColor black = {0,0,0};
    RGBColor white = {0xffff,0xffff,0xffff};

    if(input == 1)
        SetRGBPixel(i,j,red);
    else if(input == 2)
        SetRGBPixel(i,j,orange);
    else if(input == 3)
        SetRGBPixel(i,j,yellow);
    else if(input == 4)
        SetRGBPixel(i,j,green);
    else if(input == 5)
        SetRGBPixel(i,j,blue);
    else if(input == 6)
        SetRGBPixel(i,j,violet);
    else if(input == 7)
        SetRGBPixel(i,j,black);
    else if(input == 8)
        SetRGBPixel(i,j,gray);
    else if(input == 9)
        SetRGBPixel(i,j,white);
    else
        SetRGBPixel(i,j,brown);
}

/* 
** NEURALNAT -- Takes the output of the neural network file and determines 
** which colors its has determined for each pixel 
*/

#include <targraf.h>

void NNMax(double Inten, float input[8], int *color)
{
    float max;
    int tempcolor, k;

    max = 0;
    for(k=1; k <= 8; k++)
    {
        if(input[k-1] > max)
        {
            max = input[k-1];
            tempcolor = k;
        }
    }
    if(tempcolor == 7)
    {
        if(Inten < 16384)
            *color = 7;
        else if(Inten <= 49151.0)
            *color = 8;
        else
            *color = 9;
    }
    else if(tempcolor == 8)
    {
        *color = 10;
    }
    else
    {
        *color = tempcolor;
    }
}.
Recall-Only Run-time for <segnet>

Control Strategy is: <backprop>

---

extern double exp ARGS((double));

*** MAKE SURE TO LINK IN YOUR COMPILER'S MATH LIBRARIES ***

double RRand(double lv, double hv);

---

double ColNet( void *NetPtr, float Yin[18], float Yout[8] )

{ float Xout[56]; /* work arrays */
  long ICmpT; /* temp for comparisons */
  /*
  */
  /* *** WARNING: Code generated assuming Recall = 0 *** */

  for(p=0; p<18; p++)
  {
    if(Yin[p] < -1.0 || Yin[p] > 1.0)
      printf("\n**************ERROR IN PASSING THE INPUTS TO COLOR NET**************\n")
  } /*

  /* Read and scale input into network */
  Xout[2] = Yin[0] * (1.5428384) + (-0.54283845);
  Xout[3] = Yin[1] * (0.8974276) + (-0.1025724);
  Xout[4] = Yin[2] * (1.8081662) + (-0.39724056);
  Xout[5] = Yin[3] * (0.8124919) + (-0.1875081);
  Xout[6] = Yin[4] * (1.8082104) + (-0.39727467);
  Xout[7] = Yin[5] * (0.83332117) + (-0.16667883);
  Xout[8] = Yin[6] * (1.6247297) + (-0.41510543);
  Xout[9] = Yin[7] * (0.85712279) + (-0.14287721);
  Xout[10] = Yin[8] * (1.6899128) + (-0.47187858);
  Xout[11] = Yin[9] * (0.89854162) + (-0.10145838);
  Xout[12] = Yin[10] * (1.5384438) + (-0.53844378);
  Xout[13] = Yin[11] * (0.79165844) + (-0.20834156);
  Xout[14] = Yin[12] * (1.6153627) + (-0.61536274);
  Xout[15] = Yin[13] * (0.77776888) + (-0.22223112);
  Xout[16] = Yin[14] * (1.7363251) + (-0.4791146);
Xout[17] = Yin[15] * (0.77776888 + (-0.22223112));
Xout[18] = Yin[16] * (1.7165584) + (-0.49508637);
Xout[19] = Yin[17] * (0.79165844) + (-0.20834156);

LAB110:

/* Generating code for PE 0 in layer 3 */
if ((1) > RRand(0.0, 1.0)) {
        (float)(-1.7742026) * Xout[3] + (float)(-0.072222) * Xout[4] +
        (float)(-0.647964) * Xout[5] + (float)(-2.5363438) * Xout[6] +
        (float)(-0.81920868) * Xout[15] + (float)(-0.47291142) * Xout[16] +
        (float)(-3.1045301) * Xout[17] + (float)(-0.14839515) * Xout[18] +
        (float)(-2.243197) * Xout[19];
    Xout[20] = 1.0 / (1.0 + exp(-Xout[20]));
}

/* Generating code for PE 1 in layer 3 */
if ((1) > RRand(0.0, 1.0)) {
        (float)(-2.5345175) * Xout[17] + (float)(1.6959513) * Xout[18] +
        (float)(-2.2573876) * Xout[19];
    Xout[21] = 1.0 / (1.0 + exp(-Xout[21]));
}

/* Generating code for PE 2 in layer 3 */
if ((1) > RRand(0.0, 1.0)) {
        (float)(-2.5345175) * Xout[17] + (float)(1.6959513) * Xout[18] +
        (float)(-2.2573876) * Xout[19];
    Xout[22] = 1.0 / (1.0 + exp(-Xout[22]));
}

/* Generating code for PE 3 in layer 3 */
if ((1) > RRand(0.0, 1.0)) {
        (float)(-0.81920868) * Xout[15] + (float)(-0.47291142) * Xout[16] +
        (float)(-3.1045301) * Xout[17] + (float)(-0.14839515) * Xout[18] +
        (float)(-2.243197) * Xout[19];
    Xout[23] = 1.0 / (1.0 + exp(-Xout[23]));
}
Generating code for PE 4 in layer 3

if ( (1) > RRand( 0.0, 1.0 ) ) {
               (float)(-0.3014886) * Xout[13] + (float)(0.23910847) * Xout[14] +
               (float)(2.1506777) * Xout[15] + (float)(0.027256031) * Xout[16] +
               (float)(-1.8690614) * Xout[17] + (float)(0.77813071) * Xout[18] +
               (float)(-1.9805211) * Xout[19];
    Xout[23] = 1.0 / (1.0 + exp(-Xout[23]));
}

/* Generating code for PE 5 in layer 3 */
if ( (1) > RRand( 0.0, 1.0 ) ) {
               (float)(-0.3014886) * Xout[13] + (float)(0.23910847) * Xout[14] +
               (float)(2.1506777) * Xout[15] + (float)(0.027256031) * Xout[16] +
               (float)(-1.8690614) * Xout[17] + (float)(0.77813071) * Xout[18] +
               (float)(-1.9805211) * Xout[19];
    Xout[24] = 1.0 / (1.0 + exp(-Xout[24]));
}

/* Generating code for PE 6 in layer 3 */
if ( (1) > RRand( 0.0, 1.0 ) ) {
    Xout[26] = (float)(0.46971807) + (float)(4.0227432) * Xout[2] +
               (float)(0.10236996) * Xout[17] + (float)(2.8670878) * Xout[18] +
               (float)(-2.4480059) * Xout[19];
    Xout[26] = 1.0 / (1.0 + exp(-Xout[26]));
}
Generating code for PE 7 in layer 3
if ((1) > RRand(0.0, 1.0)) {
    Xout[27] = (float)(-2.2325943) + (float)(2.1561933) * Xout[2] +
               (float)(-0.0098416302) * Xout[9] + (float)(1.9049731) * Xout[10] +
               (float)(-0.99914974) * Xout[19];
    Xout[27] = 1.0 / (1.0 + exp(-Xout[27]));
}

Generating code for PE 8 in layer 3
if ((1) > RRand(0.0, 1.0)) {
               (float)(-2.7846603) * Xout[13] + (float)(0.26240945) * Xout[14] +
               (float)(-2.5776019) * Xout[19];
    Xout[28] = 1.0 / (1.0 + exp(-Xout[28]));
}

Generating code for PE 9 in layer 3
if ((1) > RRand(0.0, 1.0)) {
               (float)(-2.4093823) * Xout[13] + (float)(0.48089254) * Xout[14] +
               (float)(-2.5251324) * Xout[19];
    Xout[29] = 1.0 / (1.0 + exp(-Xout[29]));
}

Generating code for PE 10 in layer 3
if ((1) > RRand(0.0, 1.0)) {
               (float)(0.5155099) * Xout[9] + (float)(6.8786674) * Xout[10] +
(float)(-0.49378499) * Xout[13] + (float)(-1.063915) * Xout[14] +
(float)(2.5675201) * Xout[17] + (float)(2.602556) * Xout[18] +
(float)(1.0605212) * Xout[19];
Xout[30] = 1.0 / (1.0 + exp(-Xout[30]));
}

} /* Generating code for PE 11 in layer 3 */
if ( (1) > RRand( 0.0, 1.0 ) ) {
(float)(-2.801364) * Xout[9] + (float)(0.94719243) * Xout[10] +
(float)(0.50880307) * Xout[17] + (float)(3.9267495) * Xout[18] +
(float)(-0.7465437) * Xout[19];
  Xout[31] = 1.0 / (1.0 + exp(-Xout[31]));
}

} /* Generating code for PE 12 in layer 3 */
if ( (1) > RRand( 0.0, 1.0 ) ) {
(float)(-0.51656181) * Xout[11] + (float)(0.7751559) * Xout[12] +
(float)(-0.69649833) * Xout[19];
  Xout[32] = 1.0 / (1.0 + exp(-Xout[32]));
}

} /* Generating code for PE 13 in layer 3 */
if ( (1) > RRand( 0.0, 1.0 ) ) {
(float)(-3.9030939) * Xout[17] + (float)(2.9812748) * Xout[18] +
(float)(-3.8884451) * Xout[19];
  Xout[33] = 1.0 / (1.0 + exp(-Xout[33]));
}

} /* Generating code for PE 14 in layer 3 */
if ( (1) > RRand( 0.0, 1.0 ) ) {
    Xout[34] = (float)(6.8635616) + (float)(1.9290524) * Xout[2] +
               (float)(-3.6240778) * Xout[17] + (float)(2.0454423) * Xout[18] +
               (float)(-3.2891936) * Xout[19];
    Xout[34] = 1.0 / (1.0 + exp( -Xout[34] ));
}

/* Generating code for PE 16 in layer 3 */
if ( (1) > RRand( 0.0, 1.0 ) ) {
               (float)(-2.5831137) * Xout[13] + (float)(-0.15154296) * Xout[14] +
               (float)(-0.1118091) * Xout[17] + (float)(2.726731) * Xout[18] +
               (float)(-3.059429) * Xout[19];
    Xout[35] = 1.0 / (1.0 + exp( -Xout[35] ));
}

/* Generating code for PE 17 in layer 3 */
if ( (1) > RRand( 0.0, 1.0 ) ) {
    Xout[36] = (float)(-16.134388) + (float)(0.74680454) * Xout[2] +
               (float)(-0.546414361) * Xout[17] + (float)(5.7726731) * Xout[18] +
               (float)(-2.8342457) * Xout[19];
    Xout[36] = 1.0 / (1.0 + exp( -Xout[36] ));
}
if ( (1) > RRand(0.0, 1.0) ) {
    (float)(0.64720243) * Xout[3] + (float)(1.0 + exp(-Xout[37]));
}

/* Generating code for PE 18 in layer 3 */
if ( (1) > RRand(0.0, 1.0) ) {
    (float)(-1.041573) * Xout[17] + (float)(-0.82231277) * Xout[18] +
    (float)(-0.86807096) * Xout[19];
    Xout[37] = 1.0 / (1.0 + exp(-Xout[37]));
}

/* Generating code for PE 18 in layer 3 */
if ( (1) > RRand(0.0, 1.0) ) {
    (float)(-0.651398) * Xout[13] + (float)(-1.584844) * Xout[14] +
    (float)(-3.4528327) * Xout[15] + (float)(-2.3660283) * Xout[16] +
    (float)(-2.3583987) * Xout[17] + (float)(-1.847643) * Xout[18] +
    (float)(0.40037084) * Xout[19];
    Xout[38] = 1.0 / (1.0 + exp(-Xout[38]));
}

/* Generating code for PE 0 in layer 4 */
if ( (1) > RRand(0.0, 1.0) ) {
    Xout[40] = (float)(15.170350367) + (float)(-3.6494558) * Xout[2] +
    (float)(-0.651398) * Xout[13] + (float)(-1.584844) * Xout[14] +
    (float)(-3.4528327) * Xout[15] + (float)(-2.3660283) * Xout[16] +
    (float)(-2.3583987) * Xout[17] + (float)(-1.847643) * Xout[18] +
    (float)(-2.0011094) * Xout[19];
    Xout[38] = 1.0 / (1.0 + exp(-Xout[38]));
}

/* Generating code for PE 1 in layer 4 */
if ( (1) > RRand(0.0, 1.0) ) {
    Xout[41] = (float)(15.170350367) + (float)(-3.6494558) * Xout[20] +
    (float)(-4.0172029) * Xout[21] + (float)(-0.22612832) * Xout[22] +
    (float)(-2.9409354) * Xout[23] + (float)(-3.2920206) * Xout[24] +
    (float)(-2.2641747) * Xout[25] + (float)(-2.73054) * Xout[26] +
    (float)(-3.3555164) * Xout[27] + (float)(-3.466701) * Xout[28] +
    (float)(-2.9303498) * Xout[29] + (float)(-0.50372899) * Xout[30] +
    (float)(-1.288024) * Xout[31] + (float)(-2.1734722) * Xout[32] +
    (float)(-2.2617415) * Xout[33] + (float)(-3.8837829) * Xout[34] +
    (float)(-3.641372) * Xout[35] + (float)(-1.2173759) * Xout[36] +
    (float)(-2.2316325) * Xout[37] + (float)(-0.34898686) * Xout[38] +
    (float)(-1.0237931) * Xout[39];
    Xout[40] = 1.0 / (1.0 + exp(-Xout[40]));
}
Generating code for PE 2 in layer 4

if ( (1) > RRand(0.0, 1.0) ) {
    Xout[42] = (float)(-0.44099468) * Xout[21] + (float)(2.8389966) * Xout[22] +
              (float)(-4.7977605) * Xout[23] + (float)(-2.7840567) * Xout[24] +
              (float)(-2.9244261) * Xout[27] + (float)(-2.7169788) * Xout[28] +
              (float)(-1.590512) * Xout[31] + (float)(0.068275221) * Xout[32] +
              (float)(1.4673207) * Xout[33] + (float)(0.65567964) * Xout[34] +
              (float)(-3.4292729) * Xout[35] + (float)(-8.108078) * Xout[36] +
              (float)(-4.0663815) * Xout[37] + (float)(1.6851736) * Xout[38] +
              (float)(3.3135936) * Xout[39];
    Xout[41] = 1.0 / (1.0 + exp(-Xout[41]));
}

/* Generating code for PE 3 in layer 4 */
if ( (1) > RRand(0.0, 1.0) ) {
    Xout[42] = (float)(0.91919553) + (float)(5.3256903) * Xout[20] +
              (float)(-3.3754776) * Xout[21] + (float)(-5.690063) * Xout[22] +
              (float)(-2.973981) * Xout[27] + (float)(-4.4546256) * Xout[28] +
              (float)(-6.855989) * Xout[31] + (float)(-4.1038733) * Xout[32] +
              (float)(-4.3446116) * Xout[33] + (float)(-3.2830739) * Xout[34] +
              (float)(-4.4055243) * Xout[35] + (float)(-10.892209) * Xout[36] +
              (float)(7.0642729) * Xout[37] + (float)(-6.2361326) * Xout[38] +
              (float)(7.4219456) * Xout[39];
    Xout[42] = 1.0 / (1.0 + exp(-Xout[42]));
}

/* Generating code for PE 4 in layer 4 */
if ( (1) > RRand(0.0, 1.0) ) {
    Xout[42] = (float)(0.059878841) + (float)(-1.1604657) * Xout[20] +
              (float)(-3.521539) * Xout[21] + (float)(-1.0659382) * Xout[22] +
              (float)(-2.7729677) * Xout[23] + (float)(-3.1605082) * Xout[24] +
              (float)(-2.2555475) * Xout[25] + (float)(-2.4770536) * Xout[26] +
              (float)(-3.0069206) * Xout[27] + (float)(-3.2068317) * Xout[28] +
              (float)(-3.793614) * Xout[29] + (float)(-1.1785052) * Xout[30] +
              (float)(-2.6133866) * Xout[31] + (float)(-2.0037165) * Xout[32] +
              (float)(-3.353138) * Xout[33] + (float)(-3.3655438) * Xout[34] +
              (float)(-3.2050788) * Xout[35] + (float)(-1.2935634) * Xout[36] +
              (float)(-2.2045648) * Xout[37] + (float)(-1.0305663) * Xout[38] +
              (float)(-0.99845153) * Xout[39];
    Xout[42] = 1.0 / (1.0 + exp(-Xout[42]));
}

/* Generating code for PE 4 in layer 4 */
if ( (1) > RRand(0.0, 1.0) ) {
    Xout[44] = (float)(-8.2699919) + (float)(-5.7764201) * Xout[20] +
              (float)(0.8039237) * Xout[21] + (float)(4.1147308) * Xout[22] +
              (float)(-2.9784491) * Xout[23] + (float)(2.286792) * Xout[24] +
              (float)(-7.6966573) * Xout[25] + (float)(0.65617132) * Xout[26] +
              (float)(-1.238855) * Xout[27] + (float)(2.4924499) * Xout[28] +
              (float)(0.53189057) * Xout[31] + (float)(4.0457978) * Xout[32] +
              (float)(-3.3754776) * Xout[33] + (float)(-5.690063) * Xout[34] +
              (float)(3.7525468) * Xout[37] + (float)(-5.077518) * Xout[38] +
              (float)(-2.973981) * Xout[39];
    Xout[44] = 1.0 / (1.0 + exp(-Xout[44]));
}
Generating code for PE 5 in layer 4

```c
if (RRand(0.0, 1.0) ) {
    Xout[45] = (float)(-0.12488957) * Xout[33] + (float)(0.057023659) * Xout[34] + (float)(1.7923009) * Xout[35] + (float)(0.33163247) * Xout[36] + (float)(-10.786066) * Xout[37] + (float)(3.3825743) * Xout[38] + (float)(-6.2619824) * Xout[39];
    Xout[44] = 1.0 / (1.0 + exp(-Xout[44]));
}

/* Generating code for PE 6 in layer 4 */
if ( (1) > RRand(0.0, 1.0) ) {
    Xout[46] = 1.0 / (1.0 + exp(-Xout[45]));
}

/* Generating code for PE 7 in layer 4 */
if ( (1) > RRand(0.0, 1.0) ) {
    Xout[47] = 1.0 / (1.0 + exp(-Xout[47]));
}
```
/* Generating code for PE 0 in layer 5 */
if ( (1) > RRand( 0.0, 1.0 ) ) {
    Xout[48] = (float)(-2.9332964) + (float)(-0.92407799) * Xout[40] +
               (float)(-6.7505493) * Xout[41] + (float)(-1.3528953) * Xout[42] +
               (float)(-0.80142742) * Xout[43] + (float)(-0.87213277) * Xout[47];
    Xout[48] = 1.0 / (1.0 + exp( -Xout[48] ));
}

/* Generating code for PE 1 in layer 5 */
if ( (1) > RRand( 0.0, 1.0 ) ) {
    Xout[49] = (float)(-0.36441499) + (float)(-1.0964102) * Xout[40] +
               (float)(-6.9870901) * Xout[41] + (float)(-3.3162463) * Xout[42] +
               (float)(-1.1177913) * Xout[43] + (float)(-2.9839637) * Xout[44] +
               (float)(-2.1516009) * Xout[45] + (float)(-1.2516009) * Xout[46] +
               (float)(-1.2149009) * Xout[47];
    Xout[49] = 1.0 / (1.0 + exp( -Xout[49] ));
}

/* Generating code for PE 2 in layer 5 */
if ( (1) > RRand( 0.0, 1.0 ) ) {
    Xout[50] = (float)(-1.5560194) + (float)(-1.9922764) * Xout[40] +
               (float)(-4.1056643) * Xout[41] + (float)(-1.6468688) * Xout[42] +
               (float)(-1.9186729) * Xout[43] + (float)(-2.9839637) * Xout[44] +
               (float)(-2.1863382) * Xout[45] + (float)(-2.0252335) * Xout[46] +
               (float)(-2.2691226) * Xout[47];
    Xout[50] = 1.0 / (1.0 + exp( -Xout[50] ));
}

/* Generating code for PE 3 in layer 5 */
if ( (1) > RRand( 0.0, 1.0 ) ) {
    Xout[51] = (float)(-7.5012159) + (float)(-1.8578643) * Xout[40] +
               (float)(-2.6704893) * Xout[41] + (float)(4.0227356) * Xout[42] +
               (float)(-1.1177913) * Xout[43] + (float)(-8.2507019) * Xout[44] +
               (float)(-1.6269013) * Xout[45] + (float)(3.5883842) * Xout[46] +
               (float)(-1.6637592) * Xout[47];
    Xout[51] = 1.0 / (1.0 + exp( -Xout[51] ));
}

/* Generating code for PE 4 in layer 5 */
if ( (1) > RRand( 0.0, 1.0 ) ) {
    Xout[52] = (float)(-7.5216393) + (float)(-2.6641499) * Xout[40] +
               (float)(4.4571595) * Xout[41] + (float)(4.1911469) * Xout[42] +
               (float)(3.7276466) * Xout[43] + (float)(0.99506444) * Xout[44] +
               (float)(-1.5021159) * Xout[45] + (float)(-3.2832842) * Xout[46] +
               (float)(2.708878) * Xout[47];
    Xout[52] = 1.0 / (1.0 + exp( -Xout[52] ));
}

/* Generating code for PE 5 in layer 5 */
if ( (1) > RRand( 0.0, 1.0 ) ) {
    Xout[53] = (float)(-7.6800332) + (float)(-2.6641499) * Xout[40] +
               (float)(3.526284) * Xout[41] + (float)(2.6704893) * Xout[42] +
               (float)(-1.9922764) * Xout[43] + (float)(-3.4571595) * Xout[44] +
               (float)(-1.6468688) * Xout[45] + (float)(4.0227356) * Xout[46] +
               (float)(-1.1177913) * Xout[47];
    Xout[53] = 1.0 / (1.0 + exp( -Xout[53] ));
}
(float)(3.0341754) * Xout[41] + (float)(-1.1334498) * Xout[42] + 
(float)(-1.1303492) * Xout[43] + (float)(4.0503893) * Xout[44] + 
(float)(-1.750948) * Xout[45] + (float)(-1.606297) * Xout[46] + 
(float)(-2.9442151) * Xout[47];
Xout[53] = 1.0 / (1.0 + exp(-Xout[53]));
}

/* Generating code for PE 6 in layer 5 */
if ( (1) > RRand( 0.0, 1.0 ) ) {
    Xout[54] = (float)(-2.477226) + (float)(-3.1324854) * Xout[40] + 
(float)(4.3384471) * Xout[41] + (float)(-6.950294) * Xout[42] + 
(float)(-2.3266387) * Xout[43] + (float)(-4.3290095) * Xout[44] + 
(float)(-2.7724161) * Xout[45] + (float)(1.1221293) * Xout[46] + 
(float)(-0.60269499) * Xout[47];
    Xout[54] = 1.0 / (1.0 + exp(-Xout[54]));
}

/* Generating code for PE 7 in layer 5 */
if ( (1) > RRand( 0.0, 1.0 ) ) {
    Xout[55] = (float)(-1.3425812) + (float)(-0.80393267) * Xout[40] + 
(float)(-2.5950315) * Xout[41] + (float)(0.067171589) * Xout[42] + 
(float)(-0.90884262) * Xout[43] + (float)(-1.8293939) * Xout[44] + 
(float)(-0.81607509) * Xout[45] + (float)(-2.5554676) * Xout[46] + 
(float)(-0.81596851) * Xout[47];
    Xout[55] = 1.0 / (1.0 + exp(-Xout[55]));
}

/* De-scale and write output from network */
Yout[0] = Xout[48] * (1) + (0);
Yout[1] = Xout[49] * (1) + (0);
Yout[2] = Xout[50] * (1) + (0);
Yout[3] = Xout[51] * (1) + (0);
Yout[4] = Xout[52] * (1) + (0);
Yout[5] = Xout[53] * (1) + (0);
Yout[6] = Xout[54] * (1) + (0);
Yout[7] = Xout[55] * (1) + (0);
return( 0 );
#include <sign10.h>

#if __STDC__
#define ARGS(x) x
#else
#define ARGS(x) ()
#endif /* __STDC__ */

/* --- External Routines --- */
extern double exp ARGS((double));
/* *** MAKE SURE TO LINK IN YOUR COMPILER's MATH LIBRARIES *** */

#if __STDC__
int sinnet( void *NetPtr, float Yin[100], float Yout[2] )
#else
int sinnet( NetPtr, Yin, Yout )
#endif /* __STDC__ */
{
    float Xout[134]; /* work arrays */
    long ICmpT; /* temp for comparisons */
    int i,k;

/* *** WARNING: Code generated assuming Recall = 0 *** */

/* Read and scale input into network */
for(i=0; i<100; i++)
{
    Xout[i+2] = Yin[i];
}

LAB110:

    Xout[102] = Xout102[0];
    for(k=1; k<cXout102; k++)
    {
        Xout[102] = Xout[102] + Xout102[k]*Xout[k+1];
    }
    Xout[102] = 1.0 / (1.0 + exp(-Xout[102]));

    Xout[103] = Xout103[0];
    for(k=1; k<cXout103; k++)
    {
        Xout[103] = Xout[103] + Xout103[k]*Xout[k+1];
    }
    Xout[103] = 1.0 / (1.0 + exp(-Xout[103]));

    Xout[104] = Xout104[0];
    for(k=1; k<cXout104; k++)
{ 
    Xout[104] = Xout[104] + Xout104[k]*Xout[k+1];
} 
Xout[104] = 1.0 / (1.0 + exp(-Xout[104]));

Xout[105] = Xout105[0];
for(k=1; k<cXout105; k++)
{
    Xout[105] = Xout[105] + Xout105[k]*Xout[k+1];
}
Xout[105] = 1.0 / (1.0 + exp(-Xout[105]));

Xout[106] = Xout106[0];
for(k=1; k<cXout106; k++)
{
    Xout[106] = Xout[106] + Xout106[k]*Xout[k+1];
}
Xout[106] = 1.0 / (1.0 + exp(-Xout[106]));

Xout[107] = Xout107[0];
for(k=1; k<cXout107; k++)
{
    Xout[107] = Xout[107] + Xout107[k]*Xout[k+1];
}
Xout[107] = 1.0 / (1.0 + exp(-Xout[107]));

Xout[108] = Xout108[0];
for(k=1; k<cXout108; k++)
{
    Xout[108] = Xout[108] + Xout108[k]*Xout[k+1];
}
Xout[108] = 1.0 / (1.0 + exp(-Xout[108]));

Xout[109] = Xout109[0];
for(k=1; k<cXout109; k++)
{
    Xout[109] = Xout[109] + Xout109[k]*Xout[k+1];
}
Xout[109] = 1.0 / (1.0 + exp(-Xout[109]));

Xout[110] = Xout110[0];
for(k=1; k<cXout110; k++)
{
    Xout[110] = Xout[110] + Xout110[k]*Xout[k+1];
}
Xout[110] = 1.0 / (1.0 + exp(-Xout[110]));

Xout[111] = Xout111[0];
for(k=1; k<cXout111; k++)
{
    Xout[111] = Xout[111] + Xout111[k]*Xout[k+1];
}
Xout[111] = 1.0 / (1.0 + exp(-Xout[111]));
Xout[112] = Xout[112][0];
for(k = 1; k < cXout[112]; k++)
{
    Xout[112] = Xout[112] + Xout[112][k]*Xout[k+1];
}
Xout[112] = 1.0 / (1.0 + exp(-Xout[112]));

Xout[113] = Xout[113][0];
for(k = 1; k < cXout[113]; k++)
{
    Xout[113] = Xout[113] + Xout[113][k]*Xout[k+1];
}
Xout[113] = 1.0 / (1.0 + exp(-Xout[113]));

Xout[114] = Xout[114][0];
for(k = 1; k < cXout[114]; k++)
{
    Xout[114] = Xout[114] + Xout[114][k]*Xout[k+1];
}
Xout[114] = 1.0 / (1.0 + exp(-Xout[114]));

Xout[115] = Xout[115][0];
for(k = 1; k < cXout[115]; k++)
{
    Xout[115] = Xout[115] + Xout[115][k]*Xout[k+1];
}
Xout[115] = 1.0 / (1.0 + exp(-Xout[115]));

Xout[116] = Xout[116][0];
for(k = 1; k < cXout[116]; k++)
{
    Xout[116] = Xout[116] + Xout[116][k]*Xout[k+1];
}
Xout[116] = 1.0 / (1.0 + exp(-Xout[116]));

Xout[117] = Xout[117][0];
for(k = 1; k < cXout[117]; k++)
{
    Xout[117] = Xout[117] + Xout[117][k]*Xout[k+1];
}
Xout[117] = 1.0 / (1.0 + exp(-Xout[117]));

Xout[118] = Xout[118][0];
for(k = 1; k < cXout[118]; k++)
{
    Xout[118] = Xout[118] + Xout[118][k]*Xout[k+1];
}
Xout[118] = 1.0 / (1.0 + exp(-Xout[118]));

Xout[119] = Xout[119][0];
for(k = 1; k < cXout[119]; k++)
{
    Xout[119] = Xout[119] + Xout[119][k]*Xout[k+1];
}
\[ X_{out}[119] = \frac{1.0}{1.0 + \exp(-X_{out}[119])}; \]

\[ X_{out}[120] = X_{out}[120][0]; \]
for (k = 1; k < \text{cXout120}; k++)
{
    \[ X_{out}[120] = X_{out}[120] + X_{out}[120][k] \times X_{out}[k + 1]; \]
}\n
\[ X_{out}[120] = \frac{1.0}{1.0 + \exp(-X_{out}[120])}; \]

\[ X_{out}[121] = X_{out}[121][0]; \]
for (k = 1; k < \text{cXout121}; k++)
{
    \[ X_{out}[121] = X_{out}[121] + X_{out}[121][k] \times X_{out}[k + 1]; \]
}\n
\[ X_{out}[121] = \frac{1.0}{1.0 + \exp(-X_{out}[121])}; \]

\[ X_{out}[122] = X_{out}[122][0]; \]
for (k = 1; k < \text{cXout122}; k++)
{
    \[ X_{out}[122] = X_{out}[122] + X_{out}[122][k] \times X_{out}[k + 1]; \]
}\n
\[ X_{out}[122] = \frac{1.0}{1.0 + \exp(-X_{out}[122])}; \]

\[ X_{out}[123] = X_{out}[123][0]; \]
for (k = 1; k < \text{cXout123}; k++)
{
    \[ X_{out}[123] = X_{out}[123] + X_{out}[123][k] \times X_{out}[k + 1]; \]
}\n
\[ X_{out}[123] = \frac{1.0}{1.0 + \exp(-X_{out}[123])}; \]

\[ X_{out}[124] = X_{out}[124][0]; \]
for (k = 1; k < \text{cXout124}; k++)
{
    \[ X_{out}[124] = X_{out}[124] + X_{out}[124][k] \times X_{out}[k + 1]; \]
}\n
\[ X_{out}[124] = \frac{1.0}{1.0 + \exp(-X_{out}[124])}; \]

\[ X_{out}[125] = X_{out}[125][0]; \]
for (k = 1; k < \text{cXout125}; k++)
{
    \[ X_{out}[125] = X_{out}[125] + X_{out}[125][k] \times X_{out}[k + 1]; \]
}\n
\[ X_{out}[125] = \frac{1.0}{1.0 + \exp(-X_{out}[125])}; \]

\[ X_{out}[126] = X_{out}[126][0]; \]
for (k = 1; k < \text{cXout126}; k++)
{
    \[ X_{out}[126] = X_{out}[126] + X_{out}[126][k] \times X_{out}[k + 1]; \]
}\n
\[ X_{out}[126] = \frac{1.0}{1.0 + \exp(-X_{out}[126])}; \]

\[ X_{out}[127] = X_{out}[127][0]; \]
for (k = 1; k < \text{cXout127}; k++)
{
Xout[127] = Xout[127] + Xout127[k]*Xout[k+1];
}
Xout[127] = 1.0 / (1.0 + exp(-Xout[127]));

Xout[128] = Xout128[0];
for(k=1; k < cXout128; k++)
{
    Xout[128] = Xout[128] + Xout128[k]*Xout[k+1];
}
Xout[128] = 1.0 / (1.0 + exp(-Xout[128]));

Xout[129] = Xout129[0];
for(k=1; k < cXout129; k++)
{
    Xout[129] = Xout[129] + Xout129[k]*Xout[k+1];
}
Xout[129] = 1.0 / (1.0 + exp(-Xout[129]));

Xout[130] = Xout130[0];
for(k=1; k < cXout130; k++)
{
    Xout[130] = Xout[130] + Xout130[k]*Xout[k+1];
}
Xout[130] = 1.0 / (1.0 + exp(-Xout[130]));

Xout[131] = Xout131[0];
for(k=1; k < cXout131; k++)
{
    Xout[131] = Xout[131] + Xout131[k]*Xout[k+1];
}
Xout[131] = 1.0 / (1.0 + exp(-Xout[131]));

Xout[132] = Xout132[0];
for(k=1; k < cXout132; k++)
{
    Xout[132] = Xout[132] + Xout132[k]*Xout[cXout102+k];
}
Xout[132] = 1.0 / (1.0 + exp(-Xout[132]));

Xout[133] = Xout133[0];
for(k=1; k < cXout133; k++)
{
    Xout[133] = Xout[133] + Xout133[k]*Xout[cXout102+k];
}
Xout[133] = 1.0 / (1.0 + exp(-Xout[133]));

/* De-scale and write output from network */
Yout[0] = Xout[132];
Yout[1] = Xout[133];
return(0);
Appendix C -- Neural Network Training Set
Development Programs
/*
 ** Main Program COLORNET -- Get the RGB values of specified pixels and there neighboring pixels
 ** and convert them into natural color values and then specify what the
 ** color of the pixel appears to be
 */

/* Written by David Kellmeyer, November 10, 1991 */

#include <stdio.h>
#include <math.h>
#include <errno.h>
#include <targraf.h>

void Mosaic(int x, int y, int side);
void RGBtoHSI(RGBColor pixelcolor, double *Hue, double *Sat, double *Inten);
void HSItoRGB(double Hue, double Sat, double Inten, RGBColor *color);
void SatEnhance(int c, int d);
void RGBtoNAT(RGBColor pixelcolor, float *input1, float *input2, float *input3);
/* void RGBtoHSI(RGBColor pixelcolor, float *Hue, float *Sat, float *Int); */

main()
{
    unsigned int hSize, vSize;  /* the horizontal and vertical resolution of the image */
    int seg;  /* the width of the sample color swatches */
    int side;  /* the length of the averaging square used in Mosaic, the resolution will be reduced by this value */
    int neighborhood;  /* the number of pixels in each direction that will be included in the neighborhood */
    /* int primary[10], secondary[5]; */  /* the primary and secondary colors that will be assigned to the designated pixel */
    SegColor color;
    int cont, again, index, circle, k, m, n, q, r, s, t, v, w, step, colorcounter[10];  /* counters */
    register int i, j;
    unsigned int dispMode, buttonState;  /* the display mode, the state of the buttons on the mouse */
    Point position;  /* the pixel designated by the pointer */
    RGBColor tempcolor[20];  /* temporary location to store a pixel color when the mouse is on top of it */
    RGBColor pixelcolor;  /* the color of the designated pixel */
    /* RGBColor dotcolor; */  /* RGBColor dotcolor; */
    double Hue, Sat, Inten;

    /* values of the specified colors */
    RGBColor redviolet = {55295,2048,38911};
    RGBColor red = {0xffff,0,0};
    RGBColor redorange = {0xffff,24576,0};
    RGBColor orangered = {0xffff,32768,0};
    RGBColor orange = {0xffff,38911,0};
    RGBColor orangeyellow = {0xffff,45055,0};
    RGBColor yelloworange = {0xffff,51199,0};
    RGBColor yellow = {0xffff,0,0xffff};
    RGBColor yellowgreen = {51199,0xffff,4096};
    RGBColor greenyellow = {38911,0xffff,10240};
    RGBColor green = {0,0xffff,0};
    RGBColor greenblue = {2048,43007,26624};
RGBColor bluegreen = \{0,45055,49151\};
RGBColor blue = \{0,0,0xffff\};
RGBColor blueviolet = \{8192,0,45055\};
RGBColor violetblue = \{32768,6144,0xffff\};
RGBColor violet = \{28672,0,40959\};
RGBColor violetred = \{36863,0,40959\};
RGBColor white = \{0xffff,0xffff,0xffff\};
RGBColor gray = \{32500,32500,32500\};
RGBColor black = \{0,0,0\};
RGBColor brown = \{25000,15000,0\};

Rect imagesize; /* the size of the image that the mouse will have access to so that enough neighbor pixels will be available around each pixel */
float input1, input2, input3; /* the natural color inputs computed from the RGB values */
FILE *ColorFile, *HSIFile, *AnswerFile; /* the file the pixel and neighboring pixel colors are stored in, the file the color value assigned to each designated pixel is stored in */
char colorname[13], HSIname[13], answername[13]; /* variable filenames */

/* checks to see if a proper Targa board and driver are available and if so initializes the graphics capabilities */
if (InitGraphics() < 0)
{
    puts("TARGA Driver not available");
    exit(-1);
}

seg = 7; /* set the width of the sample color swatches */
again = 1; /* sets again to TRUE */

/* initialize all color counters to 0 */
for(w=0; w <=8; w++)
{
    colorcounter[w] = 0;
}

/* choose the file names to be used and open the proper files in the append mode */
printf("Enter Color File Name: ");
scanf("%s", colorname);
printf("Enter HSI Color File Name: ");
scanf("%s", HSIname);
printf("Enter Answer File Name: ");
scanf("%s", answername);
ColorFile = fopen(colorname, "a");
HSIFile = fopen(HSIname, "a");
AnswerFile = fopen(answername, "a");

/* continue the program until again is set to FALSE */
while(again >= 1)
{
    cont = 1; /* set cont to TRUE */
    flushall(); /* flushes all buffers */
    SetZoom(FloatToFixed(1.0),FloatToFixed(1.0)); /* set Zoom back to 1 */
    SetPanPos(0,0); /* pan to lower left corner */
GetDisplaySize(&hSize, &vSize);  /* gets the horizontal and vertical resolution of the image */
printf("\n\nImage resolution is %d by %d\n", hSize, vSize);

SelectLiveSource(vidInputSVidw);  /* set Targa board to accept composit video */
printf("\nS-Vidw Selected\n");
GetDisplayMode(&dispMode);  /* stores the present display mode */
EnableGenlockO;  /* set the Targa board to GenLock */
SetDisplayMode(dispModeLive);  /* set the display mode to live video */

/* Grabs the live video frame when ENTER is pushed */
printf("\n\nP...Frame has been Grabbed\n");
SetDisplayMode(dispMode);  /* set the board back to it original display mode */

/* choose the amount of image reduction to take place */
printf("\nEnter Desired Resolution(1 = %dx%d, 2 = %dx%d, 4 = %dx%d or 8 = %dx%d): ", hSize, vSize, hSize/2, vSize/2, hSize/4, vSize/4, hSize/8, vSize/8);
scanf("%d", &side);

if(side == 2 || side == 4 || side == 8)
Mosaic(hSize, vSize, side);  /* do resolution reduction is desired */
else
printf("No Resolution Reduction -- Image Resolution is still %d by %d\n", hSize, vSize);

/* Saturation Enhancement routine */
printf("\nExecuting Saturation Enhancement using natural log...\n");
SatEnhance((hSize/side), (vSize/side));
printf("...Saturation Enhancement Completed\n");

/* the neighborhood to be included in the color values */
printf("\nEnter neighborhood in pixels: ");
scanf("%d", &neighborhood);

/* limit the mouse boundaries in accordance with the neighborhood choosen and the resolution size */
SetRect(&imagesize,neighborhood,neighborhood,(hSize/side-(neighborhood + 1)),(vSize/side-(neighborhood + 1)));
OpenPntDev(microsoftMouse);
LimitPntDev(imagesize);

seg = hSize/side/22;  /* set the width of the 22 sample color swatches */

/* draws the 22 sample color swatches along the bottom of the screen */
for(index = 1; index <= 22; index++)
{
for(s=0; s <= (vSize/side/12); s++)
{
for(r=((index-1)*seg); r <= ((index-1)*seg+seg-1); r++)
{
if( r == ((index-1)*seg+seg-1) && fmod( (double)index, 3.0 ) == 0.0 )
SetRGBPixel(r,s,black);
else if(index == 1)
    SetRGBPixel(r,s,redviolet);
else if(index == 2)
    SetRGBPixel(r,s,red);
else if(index == 3)
    SetRGBPixel(r,s,redorange);
else if(index == 4)
    SetRGBPixel(r,s,orangered);
else if(index == 5)
    SetRGBPixel(r,s,orange);
else if(index == 6)
    SetRGBPixel(r,s,orangeyellow);
else if(index == 7)
    SetRGBPixel(r,s,yelloworange);
else if(index == 8)
    SetRGBPixel(r,s,yellow);
else if(index == 9)
    SetRGBPixel(r,s,yellowgreen);
else if(index == 10)
    SetRGBPixel(r,s,greenyellow);
else if(index == 11)
    SetRGBPixel(r,s,green);
else if(index == 12)
    SetRGBPixel(r,s,greenblue);
else if(index == 13)
    SetRGBPixel(r,s,bluegreen);
else if(index == 14)
    SetRGBPixel(r,s,blue);
else if(index == 15)
    SetRGBPixel(r,s,blueviolet);
else if(index == 16)
    SetRGBPixel(r,s,violetblue);
else if(index == 17)
    SetRGBPixel(r,s,violet);
else if(index == 18)
    SetRGBPixel(r,s,violetred);
else if(index == 19)
    SetRGBPixel(r,s,white);
else if(index == 20)
    SetRGBPixel(r,s,gray);
else if(index == 21)
    SetRGBPixel(r,s,black);
else
    SetRGBPixel(r,s,brown);
}
}

/* allows you to keep selecting points until cont is set to FALSE */

printf("Select Answer Color Using RIGHT Mouse\n");
printf("Select Pixel Using Left Mouse\n");
printf("Press RIGHT Mouse above colors to Exit\n");
while(cont > = 1)
{
    GetPDevPos(&position);  /* gets the current mouse postion and stores it in position */
    k = 0;

    /* puts a cross red cross hair on the current mouse position and stores any pixels colors that are
    overwritten */
    for(i=position.x-3; i<=position.x+3; i++)
    {
        if(i != position.x)
        {
            GetRGBPixel(i,position.y,&tempcolor[k]);
            SetRGBPixel(i,position.y,red);
            k = k+1;
        }
    }
    for(j=position.y-3; j<=position.y+3; j++)
    {
        if(j != position.y)
        {
            GetRGBPixel(position.x,j,&tempcolor[k]);
            SetRGBPixel(position.x,j,red);
            k = k+1;
        }
    }
    k = 0;

    /* resets the overwritten pixels to their original colors */
    for(i=position.x-3; i<=position.x+3; i++)
    {
        if(i != position.x)
        {
            SetRGBPixel(i,position.y,tempcolor[k]);
            k = k+1;
        }
    }
    for(j=position.y-3; j<=position.y+3; j++)
    {
        if(j != position.y)
        {
            SetRGBPixel(position.x,j,tempcolor[k]);
            k = k+1;
        }
    }
}

Buttons(&buttonState);  /* checks if any mouse buttons are being pressed */

/* if left button is down selects that pixel and its neighbors and stores their color values anlong with
the currently selected primary and secondary color */
if(buttonState & leftButton)
{
    while(buttonState & leftButton)  /* waits for left button to be released */
    {
        Buttons(&buttonState);
    }
for($n = position.y-neighborhood; $n <= position.y+neighborhood; $n++)
{
    for($m = position.x-neighborhood; $m <= position.x+neighborhood; $m++)
    {
        GetRGBPixel($m,$n,&pixelcolor); /* gets the RGB values of a pixel */
        RGBtoNAT(pixelcolor,&input1,&input2,&input3); /* changes the RGB to natural color values */
        RGBtoHSI(pixelcolor,&Hue,&Sat,&Inten); /* writes the natural color values to a file */
        printf("input1 = %f, input2 = %f, input3 = %f\n", input1, input2);
        fprintf(ColorFile, "%f %f %f\n", input1, input2, input3); /* writes the natural color values to a file */
    }
}

/* determines which primary color was chosen and increments the number is times it has been selected */
for($v = 1; $v <= 8; $v++)
{
    if($v == color.primary)
        colorcounter[$v]++;
}

/* prints the primary color to a file */
/* prints the secondary color to a file */
/* prints the number of total number of times each primary color has been selected */
printf("Primary Color = %d\n", color.primary);
printf("Secondary Color = %d\n", color.secondary);
fprintf(AnswerFile, "%d %d %d\n", color.primary, color.secondary, color.suplimentary);

/* prints the number of total number of times each primary color has been selected */
printf("Red(3) = %d\n", colorcounter[1]);
printf("Orange(7) = %d\n", colorcounter[2]);
printf("Yellow(13) = %d\n", colorcounter[3]);
printf("Green(10) = %d\n", colorcounter[4]);
printf("Blue(5) = %d\n", colorcounter[5]);
printf("Acromatic(10) = %d\n", colorcounter[7]);
printf("Brown(2) = %d\n", colorcounter[8]);
}
/* if right button is pressed: either ends the pixel selection routine for that image or chooses another primary/secondary color combination */
if(buttonState & rightButton)
{
    while(buttonState & rightButton) /* waits for right button to be released */
    {
        Buttons(&buttonState);
    }
    step = 1; /* sets primary color to red */
}

/* increments through each primary color */
for(q=seg*3; q<seg*3*8; q=q+seg*3)
{
    /* if far enough above the color swatches assumes you wish quit this image */
    if(position.y > 20)
    {
        cont = 0; /* set cont to FALSE */
        break; /* quits the for loop */
    }
    if(position.x < q) /* checks if mouse is within current primary color */
    {
        for(t=1; t<8; t++) /* determines which primary color this is */
        {
            if(t == step)
            {
                /* primary[t] = 1; */ /* sets the proper primary color to one */
                color.primary = t;
            }
            else /*
                /* primary[t] = 0; */ /* sets the rest to zero */
            }
        /* determines which secondary color is choosen within the selected primary color and sets it to one */
        if(color.primary == 8)
        {
            /* secondary[1] = 0;
            secondary[2] = 1;
            secondary[3] = 0; */
            color.secondary = 0;
        }
        else if(position.x < q-2*seg)
        {
            /* secondary[1] = 1;
            secondary[2] = 0;
            secondary[3] = 0; */
            color.secondary = -1;
        }
        else if(position.x < q-seg)
        {
            /* secondary[1] = 0;
            secondary[2] = 1;
            secondary[3] = 0; */
            color.secondary = 0;
        }

}
else
{
    /* secondary[1] = 0;
    secondary[2] = 0;
    secondary[3] = 1; */
    color.secondary = 1;
}
color.suplementary = 0;

    /* printf("Primary is set to 1 = %d 2 = %d 3 = %d 4 = %d 5 = %d 6 = %d 7 = %d 8 = %d\n", primary[1],primary[2],primary[3],primary[4],primary[5],primary[6],primary[7],primary[8]); */
    /* printf("Secondary is set to 1 = %d 2 = %d 3 = %d", secondary[1],secondary[2],secondary[3]); */
    printf("Primary is set to %d\n", color.primary);
    printf("Secondary is set to %d\n", color.secondary);

    break; /* ends the for loop upon finding a color */
}
step++; /* increments to the next primary color */
}
}

circle = 1; /* sets circle to TRUE */
printf("Left mouse for NEW IMAGE -- Right mouse to QUIT\n");
/* determines if you wish to get a new image or quit, stays in this loop until circle if FALSE */
while(circle > = 1)
{
    Buttons(&buttonState); /* checks mouse button status */

    /* when left button is choosen gets out of this loop and goes back to begining of the program to get a new image */
    if(buttonState & leftButton)
    {
        while(buttonState & leftButton) /* waits until left button is released */
        {
            Buttons(&buttonState);
        }
        circle = 0; /* sets circle to FALSE */
    }

    /* when right button is choosen gets out of this loop and gets out of main program loop so you can quit */
    if(buttonState & rightButton)
    {
        while(buttonState & rightButton) /* waits until right button is released */
        {
            Buttons(&buttonState);
        }
        circle = 0; /* sets circle to FALSE */
    }
again = 0;    /* sets again to FALSE */
}
/*
 ** Main Program SIGNNET -- Used to develop training sets for the sign recognition
 ** Neuronal Network
 */

/* Written by David Kellmeyer, November 10, 1991 */

#include <stdio.h>
#include <targraf.h>

void ImageOut(int hSize, int vSize, int minsize, int maxsize, RGBColor color, int *signnum, Rect SignRect[30]);
void FinalSignFrame(int side, int RectNum, Rect PossibleRect[30]);

/* void NATtoRGB(RGBColor *pixelcolor, float input1, float input2, float input3); */

main()
{
    int a,c,d;
    register int i,j;
    int side;
    unsigned int hSize, vSize;
    RGBColor yellow = {0xffff,0xffff,0};
    RGBColor orange = {0xffff,38911.0,0};
    RGBColor crazycolor = {65000.0,30000,15000};
    RGBColor crazycolor2 = {30000,15000,65000.0};
    RGBColor crazycolor3 = {15000,65000.0,30000};
    RGBColor black = {0,0,0};
    RGBColor pixelcolor;
    FILE *SignFile;
    char compactname[20], rgbname[20], signname[20]; /* variable filenames */
    Point upperleft = {0,120};
    int original;
    Rect Ysignrect[30], Osignrect[30], temprect;
    Rect outrect;
    int yellownum, orangenum;
    int SignSizeX, SignSizeY;
    int Xdif, Ydif, yellownum, nullcount, Xindex, Yindex;
    int answer;
    int minsize,maxsize;
    /* float in1, in2, in3; */

    /* checks to see if a proper Targa board and driver are available and if so initilizes the graphics capabilities */
    if(InitGraphics() < 0)
    {
        puts("TARGA Driver not available");
        exit(-1);
    }

    SetDisplayMode(dispModeIndependent);
    SetZoom(FloatToFixed(1.0),FloatToFixed(1.0));
SetPanPos(0,0);

flushall();    /*flushes all buffers*/

GetDisplaySize(&hSize, &vSize);    /*gets the horizontal and vertical resolution of the image*/
printf("the image resolution is %d by %d
", hSize, vSize);

/*choose the file names to be used and open the proper files in the append mode*/

printf("Enter Compact File Name: ");
scanf("%s", compactname);

printf("Loading -- %s...", compactname);
if( GetPic(compactname,0,upperleft) >= 0)
    printf("%s successfully loaded
", compactname);
else
    printf("There was an ERROR in loading %s !!!!!!!!!!
", compactname);

printf("Enter zoom factor:");
scanf("%d", &side);

SetZoom(FloatToFixed((float)side),FloatToFixed((float)side));
SetPanPos(0,0);

printf("Enter minimum sign size: ");
scanf("%d", &minsize);
printf("Enter maximum sign size: ");
scanf("%d", &maxsize);

ImageOut(hSize/side, vSize/side, minsize, maxsize, yellow, &yellownum, Ysignrect);
ImageOut(hSize/side, vSize/side, minsize, maxsize, orange, &orangenum, Osignrect);

original = 0;
printf("Enter 1 to View Original Picture:");
scanf("%d", &original);

if(original == 1)
{
    SetZoom(FloatToFixed(1.0),FloatToFixed(1.0));
    SetPanPos(0,0);

    printf("Enter RGB File Name:");
    scanf("%s", rgbname);
    printf("Loading -- %s...", rgbname);
    if( GetPic(rgbname,0,upperleft) >= 0)
        printf("%s successfully loaded
", rgbname);
    else
        printf("There was an ERROR in loading %s !!!!!!!!!!
", rgbname);
SetRGBForeColor(crazycolor);

FinalSignFrame(side, yellownum, Ysignrect);

FinalSignFrame(side, orangenum, Osignrect);
}
/* Function ImageOut -- Outline areas given by the color variable and then frames them */

/* Written by David Kellmeyer, September 17, 1991 */

#include "targrah.h"

int SignIn(int hSize, int vSize, RGBColor color, Rect PossibleRect);
void ImageOut(int hSize, int vSize, int minsize, int maxsize, RGBColor color, int *SignNum, Rect SignRect[30])

/*@ Begin */
{
  int j;
  register int c,d;
  int b,z,dir,a,w,turn;
  int xmin,xmax,ymin,ymax;
  RGBColor pixelcolor, corner1, corner2;
  Point startpoint, prev;
  Rect ColorRect[50];
  Rect WrongRect[50];
  /* RGBColor crazycolor = {10000,5000,50000.0}; */
  /* int answer; */

  b = 0;
  z = 0;
  w = 0;

  /************************ OUTLINES THE COLOR AREAS***********/

  for(j=1; j<vSize-1; j++)
  {
    dir = 2;
    d = j;

    xmin = hSize;
    xmax = 0;
    ymin = vSize;
    ymax = 0;

    for(c=0; c<hSize-1; c++)
    {
      GetRGBPixel(c,d,&pixelcolor);

      /**********************Check to see if this pixel is already within a bounded rectangle***********/

      for(a=1; a<=b; a++)
      {
        if(c<ColorRect[a].x2 && & c) = ColorRect[a].x1 && d < ColorRect[a].y2 &&
          d = ColorRect[a].y1)
        {
          if(ColorRect[a].x2 > c)
            c = ColorRect[a].x2;

          /* Another check for the other limit */
          /* limit2 = ColorRect[a].y2; */
        }
      }
    }
  }
}
for(a = 1; a <= w; a++)
{
    if(c < WrongRect[a].x2 && c > WrongRect[a].x1 && d < WrongRect[a].y2 &&
        d > WrongRect[a].y1)
    {
        if(WrongRect[a].x2 > c)
            c = WrongRect[a].x2;
    }
}

/****************************************************************************
For all Yellow Pixels
*****************************************************************************/

if(pixelcolor.red == color.red && pixelcolor.green == color.green &&
pixelcolor.blue == color.blue)
{
    startpoint.x = c;
    startpoint.y = d;

    if(c < xmin)
        xmin = c;
    if(c > xmax)
        xmax = c;
    if(d < ymin)
        ymin = d;
    if(d > ymax)
        ymax = d;

    prev.x = c;
    prev.y = d;
    d++; dir = 1;

    while(c != startpoint.x || d != startpoint.y)
    {
        if(c < xmin)
            xmin = c;
        if(c > xmax)
            xmax = c;
        if(d < ymin)
            ymin = d;
        if(d > ymax)
            ymax = d;
        GetRGBPixel(c,d,&pixelcolor);

        if(pixelcolor.red == color.red && pixelcolor.green == color.green &&
pixelcolor.blue == color.blue)
        {
            if(c == prev.x || d == prev.y)
        {
turn = 1;
prev.x = c;
    prev.y = d;
}  
else
{
    GetRGBPixel(c, prev.y, &corner1);
    GetRGBPixel(prev.x,d,&corner2);
    if( (corner1.red == color.red && corner1.green == color.green &&
     corner1.blue==color.blue) || (corner2.red == color.red && corner2.green == color.green &&
     corner2.blue==color.blue) )
    {
        turn = 1;
        prev.x = c;
        prev.y = d;
    }  
    else  
        turn = 0;
}
else
    turn = 0;

if(turn == 1)  /* Left Turn */
{
    if(dir == 1)
    {
        c--;
        dir = 4;
    }
    else if(dir == 2)
    {
        d++;
        dir = 1;
    }
    else if(dir == 3)
    {
        c++;
        dir = 2;
    }
    else
    {
        d--;
        dir = 3;
    }
}
else  /* Right Turn */
{
    if(dir == 1)
    {
        c++;
        dir = 2;
    }
    else if(dir == 2)
    {
        d++;
        dir = 1;
    }
else if(dir == 2)
{
    d--;   
    dir = 3;
}
else if(dir == 3)
{
    c--;   
    dir = 4;
}
else
{
    d++;   
    dir = 1;
}
}

/******Upon return to the startpoint***************/

if(c == startpoint.x && d == startpoint.y)
{
    d--;   
    if(d < ymin)
        ymin = d;
}

/******For regions of proper size***************

if( (xmax-xmin > minsize) && (ymax-ymin > minsize) && (xmax-xmin <= maxsize) &&
(ymax-ymin <= maxsize) )
{
    b++;   
    SetRect(&ColorRect[b],xmin,ymin,xmax+1,ymax+1);
    printf(\"Possible Sign (%d) -- Min = (%d, %d) Max = (%d, %d)\n\", b,xmin, ymin, xmax,
ymax);
    SignIn(hSize, vSize, color, ColorRect[b]);
    SignRect[b] = ColorRect[b];
    d = j;
    c = xmax + 1;
    xmin = hSize;
    xmax = 0;
    ymin = vSize;
    ymax = 0;
}

/******For regions too large**********************/

else if( (xmax-xmin > maxsize) || (ymax-ymin > maxsize) )
{
    w++;   
    SetRect(&WrongRect[w],xmin,ymin,xmax+1,ymax+1);
printf("Wrong Size (%d, %d) -- Min = (%d, %d) Max = (%d, %d)\n", w, xmin, ymin, xmax, ymax);

d = j;
c = xmax + 1;
xmin = hSize;
xmax = 0;
ymin = vSize;
ymax = 0;
}

/**************** For regions too small **************/

else
{
    d = j;
c = xmax + 1;
xmin = hSize;
xmax = 0;
ymin = vSize;
ymax = 0;
}

}

/* SetRGBForeColor(crazyColor);

for(a = 1; a <= b; a++)
{
    FrameRect(ColorRect[a]);
    SignInfSize, vSize, color, ColorRect[a]);
    printf("In this a sign: ");
    scanf("%d", &answer);
    if(answer == 1)
    {
        z++;
        SignRect[z] = ColorRect[a];
    }
} */

*SignNum = b;
}
Appendix D -- User's Guide
USER'S GUIDE

Hardware Requirements

386/486 computer (math co-processor highly recommended for 386)
Targa+ vision board
Multi-Sync or Dual Monitor System
Video input device (could just be a *.TGA file)
3 MB of hard disc space

Installation

Step 1: Create directory to hold program and desired images

(to place in a directory called sign in the c drive type the following)
  type: md sign press return
  type: cd sign press return
  type: copy b:\color4s.exe press return

Running the Program

Step 1: Go into the sign directory

(if at the c:\> prompt)
  type: cd sign press return

Step 2: Start the program

  type: color4s press return

Step 3: Select image input format

  a) For Video input select:
     1. Select Video format
        0 = Composite Video
        1 = S-Video
        2 = RGB Video

     (for S-Video)
        type: 1 press return

     2. Digitize (Grab) Frame
type: 1 when you want to grab the image
press return

b) For input from a targa image file (*.TGA):

1. Select input from a file option
   type: 3 press return

2. Enter the file name
   (for a file in the c:\sign directory called image.TGA)
   type: image.TGA press return

   (for a file on a floppy disc in the b:\ drive called image.TGA)
   type: b:\image.TGA press return

3. The program should tell you it is loading this file.

   *In either case the program should store the digitized image as Original.TGA

Step 4: Select a resolution reduction option (the smaller the resolution the faster the program will run)

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Approx. Run Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 = 640 x 480</td>
<td>40 min</td>
</tr>
<tr>
<td>2 = 320 x 240</td>
<td>10 min</td>
</tr>
<tr>
<td>4 = 160 x 120</td>
<td>2.5 min</td>
</tr>
<tr>
<td>8 = 80 x 60</td>
<td>40 sec</td>
</tr>
</tbody>
</table>

(usually use 160 x 120)

   type: 4 press return

Should see a small version of the image being drawn in the lower left hand corner of the video screen and it will blow up to fill the screen when resolution reduction is completed. (50 sec)

Step 5: Select a saturation enhancement bias. This is most useful when the background around the sign is brown or yellowish. By increasing the saturation bias only the sign is saturated and the background is left alone.

   0 = full saturation enhancement
   1.0 = black and white image
(usually 0.05 provides the best bias)

\[ type: \ 0.05 \ \ \ \ \ \ \ \ \ \ \ \ press \ return \]

Should see the image alter in colorfullness from bottom to top.  
(10 sec)

The program now saves this image as Sat.TGA.

The program should now begin to automatically perform color segmentation. It should be counting down as it completes the segmentation for each vertical line. When the countdown hits zero the program should represent the image in only eight colors on the video monitor.  (1 min 30 sec)

The program now saves this image as Seg.TGA.

**Step 6:** Choose any size restriction for the sign to be searched for.

(for a 160 x 120 image min = 5 and max = 30 are usually good)

\[ type: \ 5 \ \ \ \ \ \ \ \ press \ return \]

\[ type: \ 30 \ \ \ \ \ \ \ \ press \ return \]

(for no size restrictions)

\[ type: \ 0 \ \ \ \ \ \ \ \ press \ return \]

\[ type: \ 10000 \ \ \ \ \ \ \ \ press \ return \]

**Step 7:** Select Colors to search for. Since this program is for finding warning signs, usually want to choose yellow and/or orange.

(for yellow)

\[ type: \ 3 \ \ \ \ \ \ \ \ press \ return \]

Program shows any possible sign locations using the yellow color

lower-left corner  \ min = (x,y) \]

upper-right corner \ max = (x,y) \]

Also provides output from the sign recognition neural network

Sign out = probability from 0-1 that the region is a warning sign

Non-Sign out = probability from 0-1 that the region is not a warning sign

**** Note: The two outputs do not have to equal zero!!!!! ****

**** If the Sign out > 0.5 and Non-Sign out < 0.5 then the region is referred to as a probable sign region. ****
Repeat **Step 7** for any other desired colors.

**Step 8:** When finished scanning different colors enter 0.

```
type: 0  press return
```

Program should outline all probable sign regions in blue.

**Step 9:** To see these framed regions in the original image enter 1, if you do not want to see the original image enter 0.

```
type: 1 or 0  press return
```

if you type in 1:
Program loads the Original.TGA file
Provides locations in 640 x 480 coordinates of probable signs
Saves the image as Final.TGA

**Output**

Four images saved: Original.TGA, Sat.TGA, Seg.TGA and Final.TGA

For each detected sign region: saves the coordinates, sign color, shape and gray-scale values from 0-256 in Signs.TXT

Sample of a Signs.TXT file

```
20 60 30 70  (lower-left x,y and upper right x,y coordinates in 640x480)
yellow  (color)
diamond  (shape)
234 200 230 ... 190
233 250 234 ... 195  (Gray-Scale values reading across from left to right and down from top to bottom)
:  :  :
:  :
145 34 100 ... 55
```