MODEL-BASED 3-D RECOGNITION SYSTEM USING GABOR FEATURES AND NEURAL NETWORKS

THESIS

Phung D. Le, Captain, USAF

AFIT/GEO/ENG/90D-05

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GEO/ENG/90D-05

MODEL-BASED 3-D RECOGNITION SYSTEM USING GABOR FEATURES AND NEURAL NETWORKS

THESIS

Phung D. Le, Captain, USAF

AFIT/GEO/ENG/90D-05

AFIT/GEO/ENG/90D-05

MODEL-BASED 3-D RECOGNITION SYSTEM USING GABOR FEATURES AND NEURAL NETWORKS

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Electrical Engineering

Phung D. Le, B.S.E.E, M.S.S.M

Captain, USAF

December 1990

Approved for public release; distribution unlimited

Acknowledgements

I am still as excited about this research as I did when I began the effort. Much of this excitement is owed to Dr. Steven K. Rogers and Dr. Matthew Kabrisky. Beyond the encouragement and support are their enthusiasms and probing questions which made research "fun". I would also like to thank Maj Phil Amburn for his assistance in understanding and using the image generator.

Dave Dahn and Ed Williams made life a little easier with their help in using the Sun computers, getting around UNIX, and programming in C. The PhD students provided great insights into implementing the Gabor transform and neural nets.

Finally, I would like to thank my friends for keeping me sane and putting everything into proper perspective.

Phung D. Le

Table on Contents

List of Figures

List of Tables

List of Text Boxes

Abstract

A different approach to pattern recognition was attempted using Gabor features, artificial neural nets, and an image generator. The Gabor features and artificial neural nets are sound biological-based, and the image generator provides complete access to any view of an object. This research tested the idea that their integration could form a robust 3-D recognition system.

The results of the research showed that the Gabor features together with a neural net were used successfully in classifying objects regardless of their positions, out-of-plane rotations, and to a lesser extent in-plane rotations. The Gabor features were obtained by correlating the image with Gabor filters of varying orientations spaced 15° apart as found in primates' visual systems, and the correlation with each filter was kept separately.

MODEL-BASED 3-D RECOGNITION SYSTEM USING GABOR FEATURES AND NEURAL NETWORKS

I. Introduction

The ability to recognize objects in a scene has great importance in military and industrial applications. The problem is complicated by requiring the system to be invariant to the object's position, scale, orientation, background noise, or occlusion. To date, recognition systems have successfully addressed a few of these issues. However, no single paradigm has solved all levels of the problem. (11:313)

The recognition process consists of segmentation, features extraction, and classification. First, segmentation involves separating the regions of interest from the background. Second, important features pertaining to the objects are extracted to reduce the amount of information necessary for processing. Finally, the classification process involves using these features to identify the objects. (18)

1.1 Background

The Air Force Institute of Technology (AFIT) has on-going projects in all aspects of the recognition problem. Among the most successful segmentation schemes investigated at AFIT is the Gabor transform. Ayer showed that sine wave Gabor functions acted similar to edge detectors, and cosine wave Gabor functions filled in the body details. By combining both functions, complete segmentation (edge detection with details) of FLIR images was accomplished. (2)

Two feature spaces were extensively researched at AFIT. The Kobel-Martin-Horev (KMH) algorithm showed that targets could be located in a high clutter environment through correlation of the transformed scene and template (20). Patterned after similar work done by Casasent and Psaltis, this feature space was made invariant to position, scale, and in-plane rotation (5). Another feature space used was invariant moments (ordinary and Zernike). The moments were calculated for each object using equations derived by Hu and Teague (15; 33) and fed to a decision rule for classification. Robinson and Ruck demonstrated the usability of these features in their optical and artificial neural network systems, respectively (25; 30).

Artificial neural networks as classifiers have been investigated predominantly in the past several years. Both Troxel's and Ruck's implementation of multilayer perceptron networks have shown great accuracy in classifying targets (34; 30). Presently, research involves investigating different types of neural networks including Kohonen, Radial Basis Functions, and Higher-order networks.

With respect to developing a recognition system that is invariant to orientation, the above techniques have only addressed the in-plane rotation and scale problem.

1.2 Problem

Recognition of objects with respect to out-of-plane rotation is unsolved. This thesis will research a solution to identifying objects regardless of their orientations (in or out-of-plane rotations).

1.3 Assumptions

In order to limit the scope of the project, two assumptions are made. First, the object of interest is already segmented. The segmentation problem can be resolved by using the Gabor transforms as described in Ayer's thesis (2). Second, the range to the object is known. This information can be obtained from using range data.

1.4 Approach

The recognition of the object will be accomplished in five steps (see Figure 1). First, the Gabor features will be extracted from the 2-D image. Several Gabor templates with varying orientations will be used. The sum of these templates will be reduced to a more manageable size by retaining only the highest information content Gabor coefficients. The reduced Gabor vector will be fed into a multilayer perceptron network to determine the object's class in step two. The network will use the back-propagation learning algorithm, and the training data will consist of reduced Gabor vectors of different perspective views of the objects.

In step three, the determination of the object's perspective is achieved. Once a class is determined, the same reduced Gabor vector will be fed into another multilayer perceptron network. The training data will be similar to those used in step two except the desired outputs are coordinates of the object's orientation. In step four, these coordinates are entered into an image generator to produce a 2-D image with the estimated orientation. The image generator contains a 3-D model of the object and can generate any 2-D views of that object. Finally, in step five, the 2-D *generated* image will be correlated with the *original* 2-D image to determine a Figure-of-Merit (FOM) for the classification of the object in step two. A max FOM of 1 indicates that the *generated* image is very similar to the *original* image.

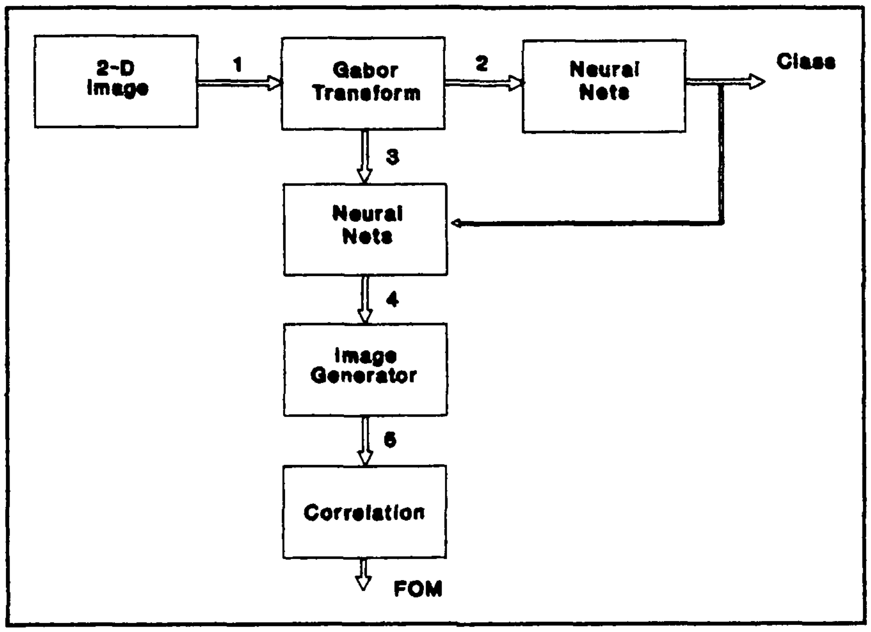

Figure 1: Research approach

1.5 Summary

This chapter has provided a brief background on the work done at AFIT on a recognition system, the problem, assumptions, and an outline of the approach used to investigate the problem. Chapter II provides the background information on related research in solving the recognition problem. Chapter III describes the detail processes of the outlined approach. The results and discussions are in Chapter IV, and Chapter V reviews the research with concluding remarks and recommendations.

II. Literature Review

Real-world image recognition remains a challenging problem for researchers in the field. A robust recognition system must be able to identify objects in the scene regardless of their positions, scales, or orientations (in-plane and out-of-plane rotations). Given a two dimensional image, the system needs to extract important features from the objects and use them for classification. Some of the feature extractors are biologically based and are used successfully for identifying objects in their specific applications. The addition of neural networks to approximate solutions to problems through adaptation and learning provide for more robust and fault tolerant classifiers (21; 32). The combination of a feature extractor and neural network may provide an applicable recognition system for the non-linearity of real-world images.

This chapter will discuss several of these feature spaces: space-variant, moments, and Gabor. Related studies in neural networks are presented. Finally, many 3-D recognition systems require extensive storage space for various views of the object. A model-based approach to alleviate some of the computational and storage burden will also be discussed.

2.1 Space-variant features

In 1977, Casasent and Psaltis proposed a feature space that is invariant to position, scale, or in-plane rotation. This feature space could be used for correlation without loss in signal-to-noise ratio of the correlation peak. Using the Fourier-Mellin properties, they performed the Fourier transform of the scene to obtain position invariance, polar coordinate transform to obtain rotation (in-plane) invariance, and logarithmic transform to the radial axis to achieve the scale invariance. (5:77-84)

Following a similar approach, students at AFIT developed an algorithm (KMH), Figure 2, that allowed for the determination of the correlation peak location and subsequent identification of the target. The unique features of this algorithm were the extraction and reinsertion of the phase information which contains the target's positional information. Digital implementation of the algorithm proved successful in recognizing multiple targets in non-random noise corrupted scenes. However, the computational time was intensive and was considered infeasible for a real-time system. (14; 20)

Optical implementation by subsequent AFIT students proved useful (23; 6; 7), but a number of problems remained in obtaining a real-world image recognition system. The optical implementation of the phase extraction and reinsertion has not been realized. Also, both the KMH and Casasent/Psaltis algorithms did not take into account the out-of-plane rotation problem which is critical in real-world images.

To obtain out-of-plane rotation invariance, Schils and Sweeney proposed the use of a small bank of optical correlation filters that can recognize objects from various perspective views. The holographic filters are generated through iterative procedures. An object is decomposed into a set of fundamental image components. These components called eigenimages contain the essential information from all of the object variations. This fundamental image information is then multiplexed, using phase, into each holographic filter. Detection of target is based on finding points of constant intensity in the correlation plane. Using a bank of 20 holographic filters, they were able to recognize a tank with 16 different perspective views. (31)

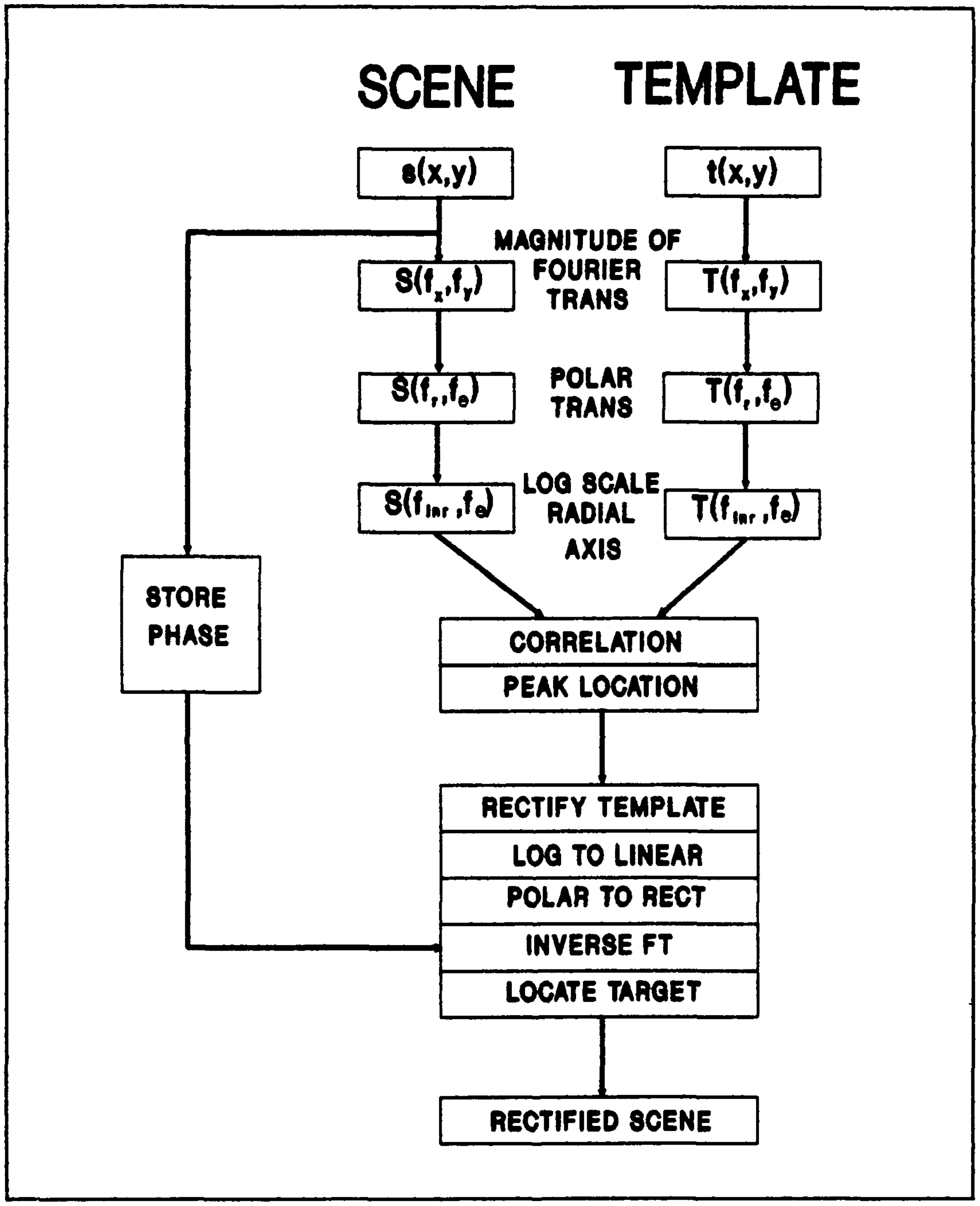

Figure 2: KMH Algorithm

Casasent achieved the out-of-plane rotation invariance by Linear Discriminant Functions (LDFs) calculated from training set images. A LDF Computer Generated Hologram (CGH), consisting of gratings with various modulation levels and orientations, is placed in the feature space. The location formed by the projection of these gratings in the output plane determines the object class. (3:449)

2.2 Moment features

Another feature space widely used in pattern recognition is moments. The use of image moment invariants for two dimensional pattern recognition application was first introduced by Hu in 1962 (15). By using central moments, which are defined with respect to the image centroid, the features are made position invariant. Scale invariance is made through normalization. Derivations for rotation invariant moments were more complex. Teague improved upon Hu's work by deriving a set of moments which are based on Zernike polynomials. The normalized central moments were also used to account for scale and position variances. The rotational effect on the moments was shown to be a multiplication of a phase factor related to the angle of rotation and the degree of the moment. (33:926) These Zernike moments suffer from the same information loss problems as ordinary moments described by Hu, but not from information suppression or redundancy (1:705).

Dudani *et al* applied these ordinary moment features to aircraft identification. Moment parameters calculated from Hu's equations were reduced to the five most important feature components. These components were used for classification in both the Bayes and Distance-Weighted k-Nearest Neighbor decision rules. With a test sample of 132 images from 6 classes obtained at random viewing aspects, they proved that their decision rules worked better than the human counterparts in identifying the various aircrafts. (10) At AFIT, Robinson's research showed a 95.5% accuracy in identifying various objects using these invariant moments in an optical setup (25:25).

2.3 Gabor features

A promising feature space for pattern recognition which fits well with the empirical studies of the visual receptive fields in cats is the Gabor's (11:314). Following the work of Dennis Gabor on data compression in 1946, Turner applied the Gabor functions to image processing and found that by using a non-self similar Gabor function, detailed textural features were retained without sacrificing global features such as periodicity and overall data structure (2:13-14).

The general functional form of the 2-D Gabor filter is (9:1173)

$$g(x,y) = \exp(-\pi[(x-x_o)^2\alpha^2 + (y-y_o)^2\beta^2]) \times \exp(-2\pi i[u_o(x-x_o) + v_o(y-y_o)]) \tag{1}$$

This function describes a complex exponential windowed by a Gaussian envelope. The Gabor transform of an image, i(x,y) is then (27)

$$f(x,y) = i(x,y) \star g(x,y) \tag{2}$$

or

$$F(f_x,f_y) = I(f_x,f_y)\, G^*(f_x,f_y) \tag{3}$$

where $\star$ denotes correlation and * denotes complex conjugate.

Flaton and Toborg applied the Gabor theory to their image recognition system called the Sparse Filter Graph (SFG). Feature vectors were constructed from convolving the image with Gabor filters at different orientations and spatial resolutions. The collection of these features is reduced to "Mini-blocks" to reduce computation time.

"Mini-blocks" are chosen from feature vectors with the highest informational content. The reduction process involved measuring how different one jet is with respect to another, or saliency. Saliency is described as (11:316)

$$S(J^a,J^b) = \frac{J^a \cdot J^b}{(\|J^a\| * \|J^b\|)} * \min\left(\frac{\|J^a\|}{\|J^b\|}, \frac{\|J^b\|}{\|J^a\|}\right) * \min(\|J^a\|, \|J^b\|) \qquad (4)$$

where J^a and J^b are two different jets. The recognition process involved an iterative process of comparing the input block against those stored in a database. The highest set of match values is deemed the correct response. The experiment carried out was successful in identifying a tank with the correct orientation. (11)

The feature spaces discussed above reduce a large quantity of information from an image to a more manageable size. In the past, recognition of an image involved a slow iterative process of matching the image's features to known features in a database. The advent of neural networks provides for a more rapid approach to classification of an image through approximation, and they are more adaptable to variabilities in the input features than past techniques. The next section will discuss related research in using neural networks as classifiers.

2.4 Neural Networks

Formulated from biological research, neural networks provide a heuristic approach to solving problems that could prove quite successful in the areas of speech and image recognition where conventional computers have failed (26). The successful uses of neural networks as classifiers are well documented in literature.

Using a multilayer perceptron (MLP) with back propagation (BP) learning algorithm, Khotansad *et al* showed classification accuracy of 97% for targets and 100% for non-targets (19). At AFIT, Ruck's uses of MLP achieved 86.4% accuracy in classification of tanks, using Zernike moment invariants as features (30:5-4). Similar results were obtained by Troxel's research using features derived from the KMH algorithm (34).

Other uses of neural networks to predict target rotation and to synthesize filters are being investigated at the University of Dayton. Gustafson *et al* used a 5 by 5 array around a correlation peak as inputs into a backpropagation-trained neural net. With a training set of 50 vectors, they were able to predict the rotation angle with less than $\pm$ 2.8° error. The same correlation grid was also used to generate a near-optimum filter for a rotated target. After 78 training cycles, the results showed the synthesized filters in general were superior to a fixed filter. (13)

The next section describes the use of the model-based approach as an alternative to storing numerous templates and filters for various views of an image.

2.5 Model-based

The model-based approach involves the storage of a 3-dimensional model. The consideration to use a model as a tool for the recognition process has two advantages over the common technique of storing a bank of templates or filters. One, the issue of sufficient object representation is greatly reduced since any 2-D views can be generated from the 3-D model. Two, the storage burden is also alleviated.

At AFIT, the model generating process employs two common techniques, cubic spline and surface polygons. Cubic spline provides for a more accurate description of a curve surface than surface polygons. However, the surface polygons technique requires less computational time because the mathematical description is simpler. (35) For this thesis, the surface polygon technique will be used to generate the models.

Though they seemed realistic, the models still do not represent the true "picture of the world". For example, the interaction of light with an object is too complex to model mathematically. One workaround is to use texture wrapping that simulates the reflection of light off of the object. Another is the use of ray tracing where the reflection of one ray is modelled mathematically. Ray tracing improves the way one simulates the reflection of light, however, it requires an enormous amount of computer time. (35) The modelling technology is continually being improved, and despite the lack of realism, the models themselves still offer valid research tools for the recognition process since all of the train and test data are generated from the models.

Casasent and Liebowitz applied this model-based approach to their optical symbolic correlator. Utilizing the capability to generate on-line images at any orientation, they were able to control the filters used in identifying the image. The filters were linear combinations of several reference objects in one or more classes. The peak-to-sidelobe ratio (PSR) filters were used to locate the objects, correlation filters were used to classify objects, and projection filters were used to confirm the object class and determine the object orientation. A test on two classes with varying orientations resulted in approximately 80% correct recognition. (4)

2.6 Summary

A robust recognition system requires the capabilities to extract important features from an image and correctly classify the object regardless of its position, scale, or

The three feature spaces discussed provide working algorithms to reduce the image information to a more manageable size. Neural networks have shown to be flexible classifiers in light of the variabilities in the inputs, and a model-based approach can be used to reduce the storage space. The next chapter will discuss the uses of these items, and how they integrate to form a robust 3-D recognition system.

III. Methodology

The approach discussed in Chapter I outlined five steps to be accomplished in this thesis study (see Figure 1). Steps 1 and 2 comprise the first path of the thesis in which the goal is to see how well the Gabor information can be used to discriminate among classes. Gabor coefficients and their relative positions will be used as features, and a neural network will be used for classification. The output of path 1 will be the probability that a given object within an image belongs to a certain class. Steps 3, 4, and 5 comprise the second path of the thesis in which the goal is to determine the perspective view of the object. The Gabor coefficients and their relative positions will once again be used as features, but a sub neural network will be used to output the estimated coordinates of the object. This sub neural net will be activated based on the winning class in path 1. Using the estimated coordinates, the image generator will produce the corresponding two-dimensional template that will be correlated with the original image. The result of this correlation will be a figure-of-merit for the classification of the object.

This chapter provides specific details on how each path will be accomplished and a summary on the entire process.

3.1 Path 1: Classification of object

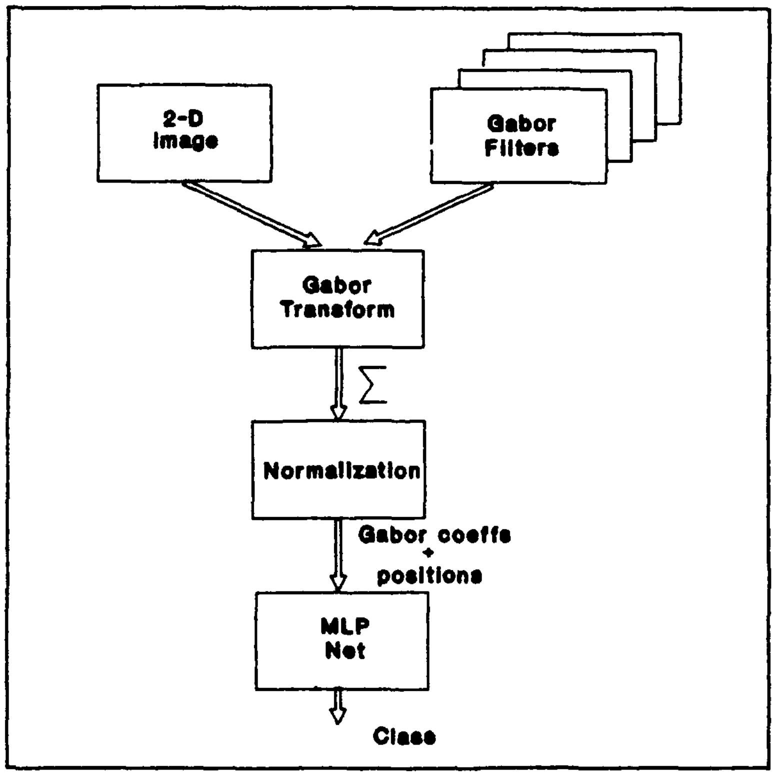

Figure 3: Path 1: Classification of object

3.1.1 2-D images

The 2-D images shown in the Figure 3 will be generated by an image generator using a General Purpose Renderer (gpr) program. A description and its functions are described in Appendix A. The images will consist of various views from three objects, a block, a balloon, and a tank. Examples are shown in Figure 4:

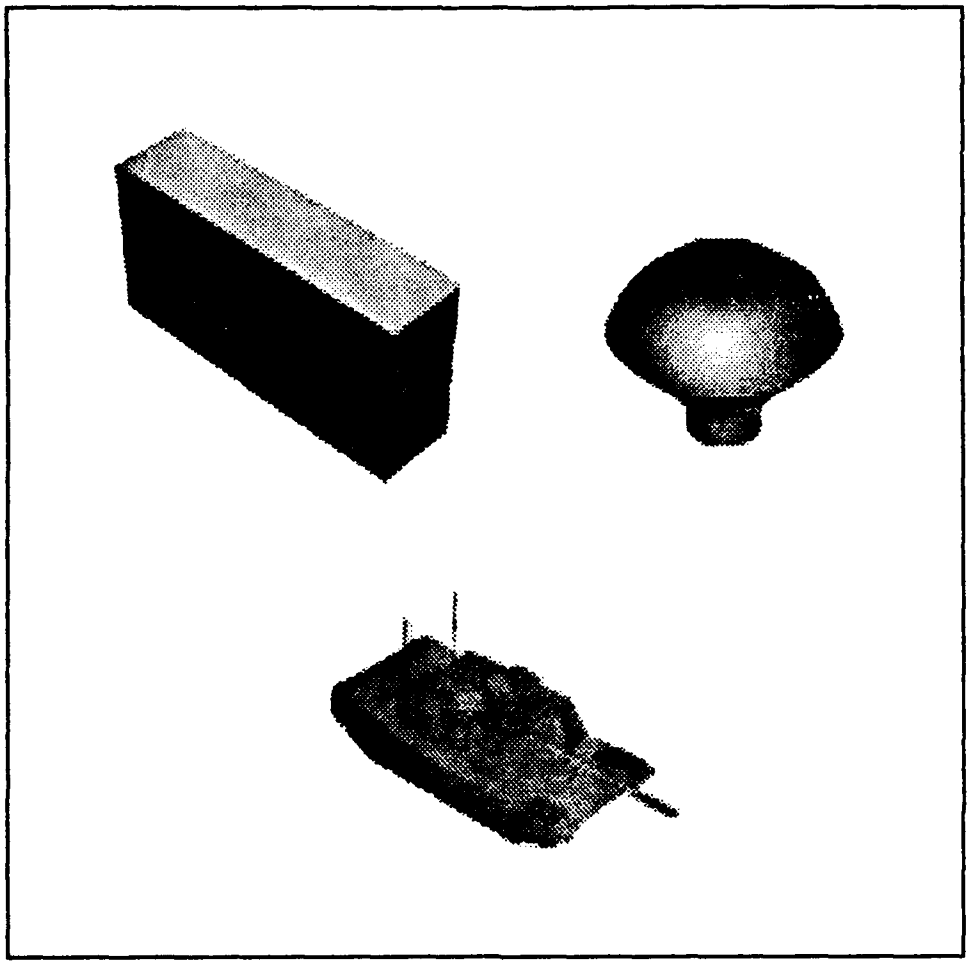
Figure 4: Examples of objects

These images will be correlated with several Gabor filters, and the transform results will be used to form training and testing vectors.

3.1.2 Gabor filters

In order to limit the scope of the study, several parameters relating to the Gabor filter will be fixed for all filters. First, the spatial frequency will be fixed at 1 cycle per Gaussian window. Second, the size of the Gaussian window will be kept constant for all filters, and third, only sinewave Gabors will be used in this study. The decision to use sine over cosine Gabors originated from the curiosity in studying the use of edge information on classification. Ayer's research has shown that sine Gabors can be used to find edges of an object (2) The only parameter that will change will be the orientation. A bank of filters will be generated with the three constant parameters discussed above but with different orientations. The values for the orientations will vary based on experimental results.

3.1.3 Gabor transform

The program "gabortf1.c" in Appendix D will generate the desired filters as well as perform the Gabor transform. The transforms of an image with the bank of filters will be added together in magnitude, pixel-by-pixel. Next, all magnitude values will be converted to positive numbers. Mueller and Fretheim's research has shown that negative Gabor coefficients carry equal importance (information content) as their positive counterparts. (24) The Gabor coefficients will then be sorted in order of their magnitudes, and a "*number*" of the top Gabor coefficients and their relative x-y positions on the image will be kept as features, forming a vector. This "*number*" will vary based on experimental results.

The next step will involved normalizing these numbers to prepare them as inputs into the neural net.

3.1.4 Normalization

The normalization process will involve both horizontal and vertical normalization. In this context, horizontal normalization involves finding the relationship of values within a single vector, whereas vertical normalization is finding the relationship of values among the vectors.

For horizontal normalization, each vector will be recalculated to obtain position invariance. The Euclidean distance will be computed for each coefficient's x-y position to the position of the maximum Gabor coefficient.

$$(distance)\ z_i = \sqrt{(x_i - x_{max})^2 + (y_i - y_{max})^2} \tag{5}$$

where i is the position number of a Gabor coefficient in an array of coefficients sorted in ascending order by their magnitudes. An example of the position invariant algorithm can be shown in Figure 5:

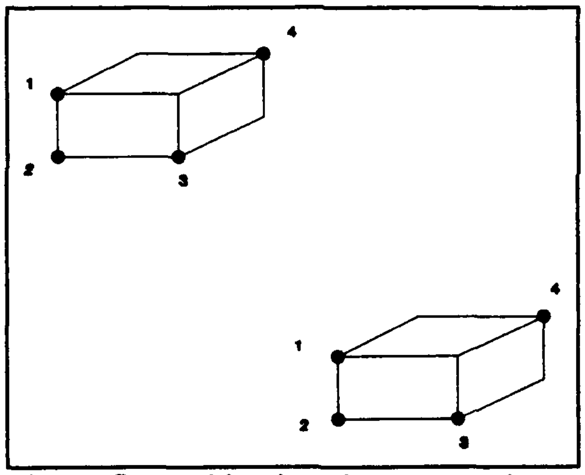

Figure 5: Position invariance example

Given two objects (blocks in this case) of equal size but in different positions within an image, the Gabor wavelets will ring the same points on the objects since the texture information is the same for both objects. Assuming that '1' corresponds to the maximum coefficient, then the Euclidean distances from all other points are relative to the position of the maximum coefficient. Shifting position only changes the x-y locations of the coefficients, not the relative distances to the maximum coefficient.

After normalizing each vector horizontally, all vectors will be normalized vertically. The means and standard deviations will be calculated for each feature with respect to all the vectors, and each value will be normalized by the following equation:

$$(new)\ x_i = \frac{(old)\ x_i - \mu_i}{\sigma_i} \tag{6}$$

where i represents the position of each feature within a vector. μ_i and σ_i are the mean and standard deviation, respectively, for the *ith* position. The program "norm.c" in Appendix D will perform this normalization step.

These normalized features will be the inputs into a neural net for classification. The features will be arranged as coefficient, position, coefficient, position, etc.. Again, the number of features used will vary based on experimental results.

3.1.5 Neural net

The neural net will be a Multi-layer Perceptron (MLP) using Back Propagation (BP) learning algorithm. The make-up of the net will consist of one hidden layer and an output layer with three nodes, one for each class. The number of hidden nodes will vary based on the number of inputs chosen. The output is expected to be a probability that the given inputs belong to a certain class. (28) The node (class) that wins will drive which sub neural net is to be activated in path 2.

The net will be trained using the normalized features of random views of the three objects. The number of training vectors will vary based on the number of input features chosen. Foley's rule suggested that the number of training vectors should be at least three times the number of input features for each class. (28)

Test will be performed on designated data at every 100 iterations through the training data. The test will cover three areas: position invariance, out-of-plane rotation invariance, and in-plane rotation invariance. For position invariance, the test images will consist of a sample of the images from the training data but with the objects shifted within the images. For out-of-plane rotation test, the data will consist of images that the net has not seen but will have the same scale as the training images. The data for the in-plane rotation test will be a set of images from the training data with the objects rotated in-plane.

A "correct" is given if the error from each output node is less than 20%. Training will stop when additional training does not provide significant improvement in the accuracy of the test data. At this point, the net parameters (weights and offsets) will be saved for future use.

With a trained neural net, path 1 is ready for usage. Given an unknown 2-D image, the Gabor transform will be performed, and its normalized features will be input into the neural net. The output of the net will contain the winning node (class) that will activate the corresponding sub neural net in path 2.

3.2 Path 2: Determination of perspective view

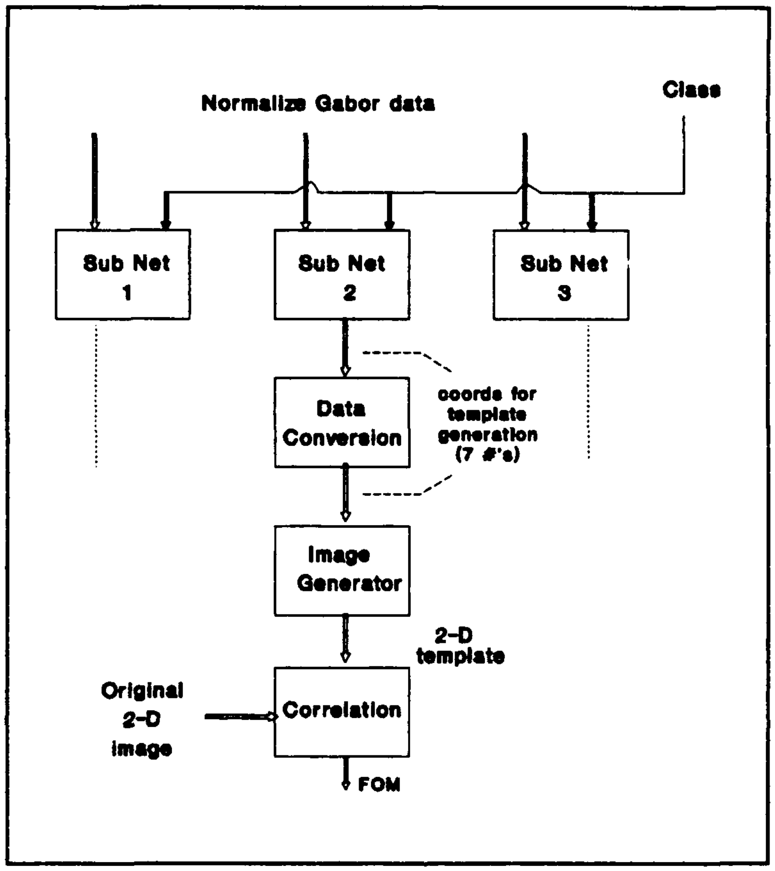

Figure 6: Determination of perspective view

3.2.1 Sub neural net

There will be three sub neural nets corresponding to the three classes under study. A single sub neural net will be activated based on the winning class from path 1. The sub nets will be Multi-layer Perceptron nets using back propagation learning algorithm. Each net will have one hidden layer and an output layer with seven nodes, corresponding to the seven parameters needed for the image generator to produce a 2-D image. The parameters are the x y z coordinates of the camera, the x y z coordinates of the center-of-interest (coi), and the twist angle of the object.

The training data for each net will consist of normalized Gabor information of random views of its class. The number of training vectors will vary based on the number of input features chosen. As in the net in path 1, the test data for each sub net will cover the position, out-of-plane rotation, and in-plane rotation invariance.

3.2.2 Data conversion

The outputs of the net are designed to be between 0 and 1, and the inputs into the image generator could vary between -1500 and 1500 as in the case of the tank. Consequently, it will be necessary to convert the net output numbers to more usable numbers expected by the image generator. The ranges for the outputs will vary between the three objects based on the scales given in their geometry files. Therefore, each sub net will have its own data converter. Among the seven parameters, the ranges will also be different. For example, the x and y coordinates for the camera and the center-of-interest (coi) can vary between a negative "max" and a positive "max", but the z coordinates will vary from 0 to a positive "max". Negative z coordinates will not be used since recognition will not be attempted for the bottom of a tank. For the twist angle, the range from 0 to 1 will correspond to angles from 0 to 360 degrees.

The converted numbers will be used to create a control file for the image generator to use in creating a 2-D template.

3.2.3 Image generator

As mentioned earlier, the image generator creates images using a General Purpose Renderer program. With a 3-D model in its data bank, the image generator can create a 2-D image from any view. In path 2, a 2-D template will be generated with the view specified by the control file from the sub net. The command used to generate the template is given in Appendix A.

3.2.4 Correlation

Correlation is used to determine the similarity between two objects. In this case, the similarity test will be between a 2-D template and the original unknown 2-D image. The correlation peak will be used as a figure-of-merit (FOM) for the classification of the object in path 1. A maximum FOM of 1 means that the two images are very similar.

3.3 Summary

Given the task of classifying an unknown 2-D image, the determination process for this research is as follows: first, the Gabor transform of the image will be taken, and the normalized features will be fed into a trained neural net. The output of the net will be a probability that the given object within the image is of a certain class. Next, using the same normalized features and a sub net activated by the winning class, an image with the estimated view will be generated. This generated image will be correlated with the unknown image to obtain a FOM on the classification.

The next chapter will provide the results and discussions on the research.

IV. Results and Discussions

This chapter provides the results of the approach described in Chapter III in four parts. Part 1 discusses the findings related to the first task of trying to classify the three objects. Part 2 discusses the findings of path 2 which is to determine the perspective view. In part 3, the problems concerning numerical approximation, modelling parameters, and their effects on the gabor features are provided. Finally, a summary of the findings is presented in part 4.

4.1 Path 1: Classification of objects

4.1.1 Test data

The test data for path 1 are given in Table III and Table IV in Appendix C. Each test area consists of 15 test vectors, five for each object. Specific coordinates for the position invariant test data are not available since the test data were obtained using a cut and reposition method as described in Appendix B. This method was used to keep the object at a constant scale. Other methods to reposition the objects were tried that seemed to retain the object size to the observer, but experimental data have shown that the Gabor transform was very sensitive to even the slightest pixel differences and will give different results. The cut and reposition method was the only one guaranteed to keep the scale constant.

4.1.2 Rotation at 45°

The Gaussian windows for all filters were arbitrarily chosen at 8x8 pixels. The first set of orientation angles tested were (0,45,90,135). These angles were effectively used in Ayer's research to segment FLIR images. (2) The Gabor transforms of the three objects are shown in Figure 7. Since the Gabor window was fixed and small compared to the objects' sizes, the sine Gabors picked out the edges of the objects as well as some internal areas as in the case of the tank and balloon.

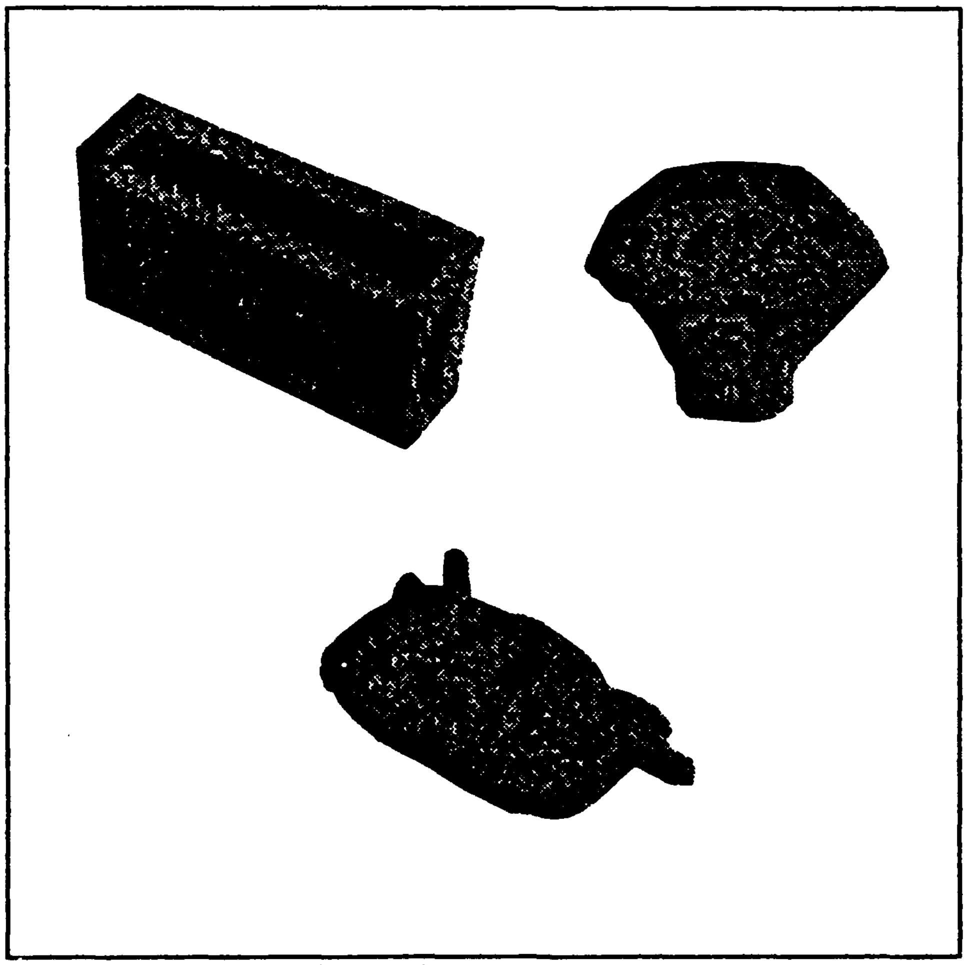
Figure 7: Gabor transforms of the block, balloon, and tank

The net was trained with 60 vectors for each objects. The number of input features were set at 20, 10 Gabor coefficients and their normalized positions, and the number of hidden nodes was set at 10.

The results for the position, in-plane, and out-of-plane tests are given in Figure 8, Figure 9, and Figure 10, respectively. The plots show accuracy of the test data versus the number of iterations ran by the net.

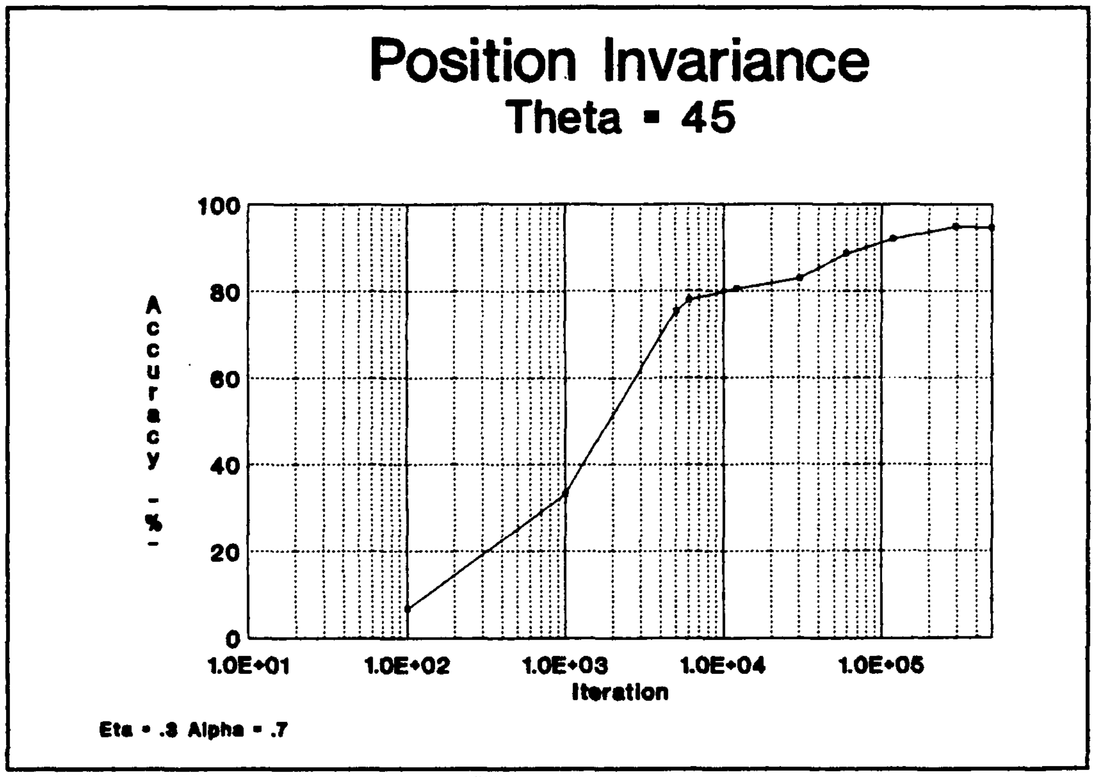

Figure 8: Position test, theta = 45°

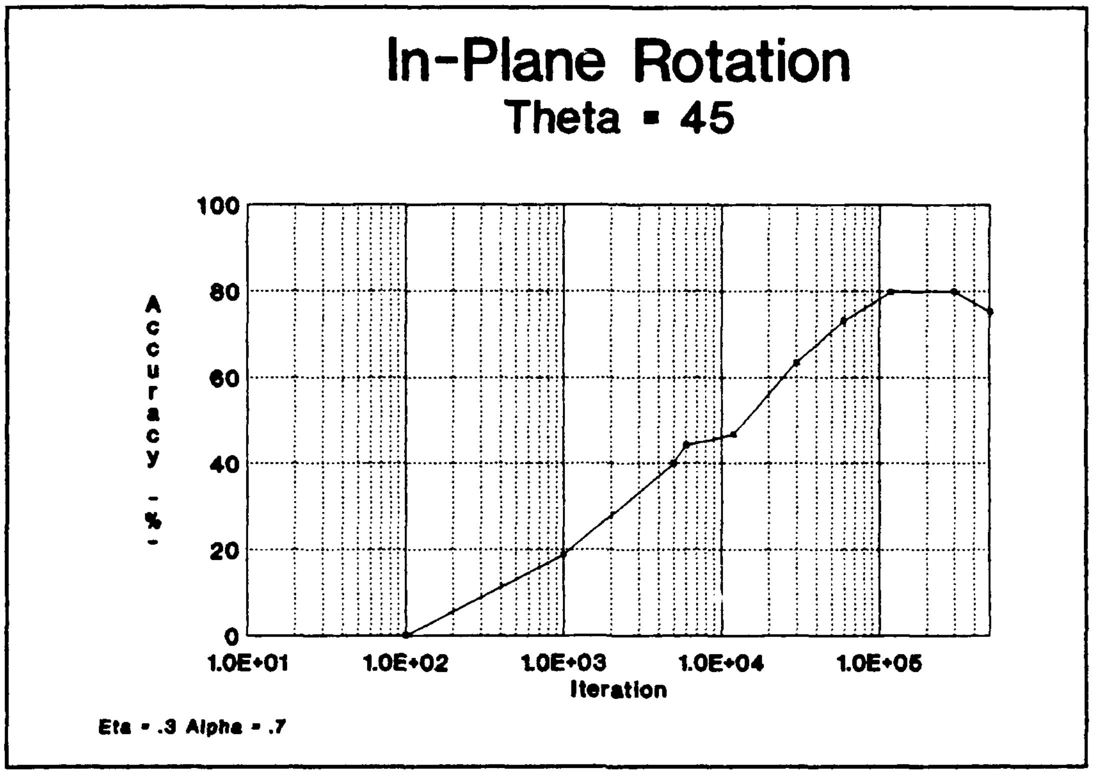

Figure 9: In-plane test, theta = 45°

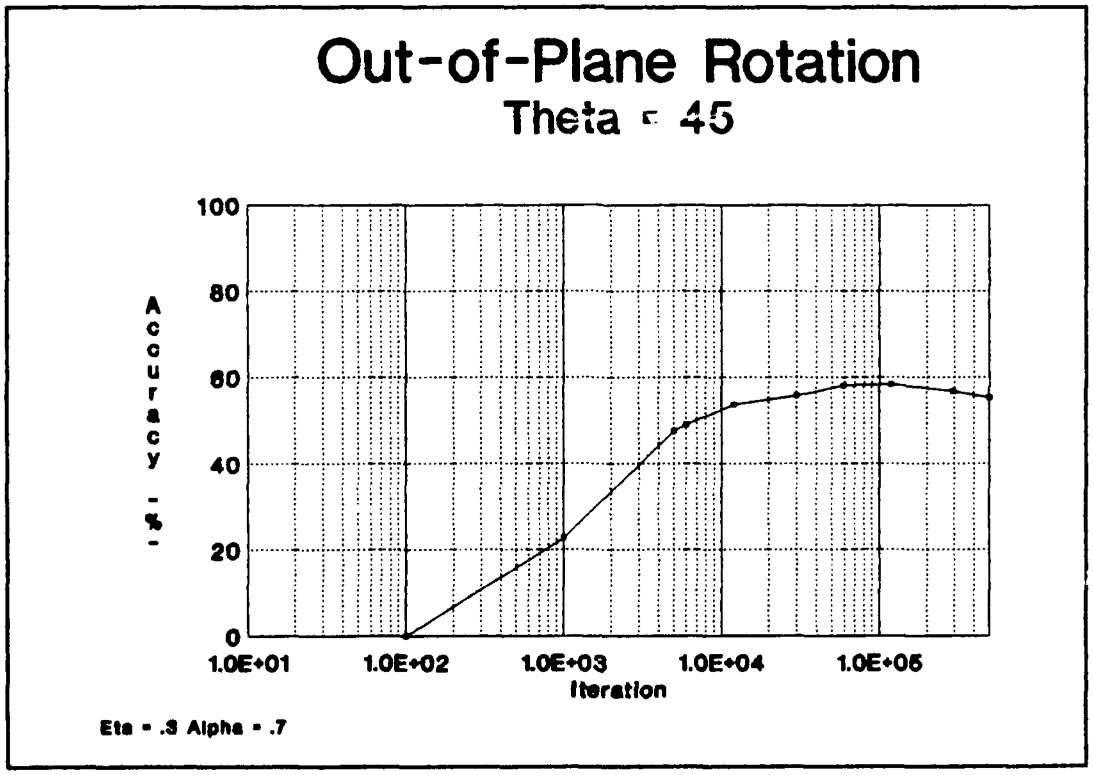

Figure 10: Out-of-plane test, theta = 45°

As expected from the normalization method, the accuracy for the position invariant test was very high. Of the 15 test vectors, there were no misses after 100,000 iterations, and given enough time, the accuracy percentage should be close to 100%.

The in-plane test gave good results with accuracy as high as 79.9% at 295,000 iterations, and the out-of-plane results were fair with accuracy reaching 58.4%. It was difficult to draw any conclusions from these results at this point, except that they showed promising steps in using Gabor features for classification. Further research indicated that perhaps filters at every 15° orientation were more appropriate than at 45°. (16; 12; 17)

4.1.3 Rotation at 15°

Hubel and Wiesel's work on the visual cortex indicated that certain cortical cells responded best to lines with the right tilts, and the most effective orientation varies between the range of 10 to 20 degrees. (16) Recently, Gizzi *et al*'s research on the cat striate and extrastriate visual cortex affirmed the fact that cells in both area 17 and the lateral suprasylvian cortex (LS) response patterns are linked to the orientation of the stimulus' spatial components. Using grating patterns, they found that the orientation modal values for these cells were near 13°. (12) AFIT's research has shown similar results with moving grating patterns and found that the responses occurred at every 15°. (17)

A new test was conducted with 12 filters at orientations 0° to 165° spaced 15° apart. Again the net was trained with 60 vectors for each class, and the number of inputs were set at 20. The results are shown in Figure 11, Figure 12, and Figure 13.

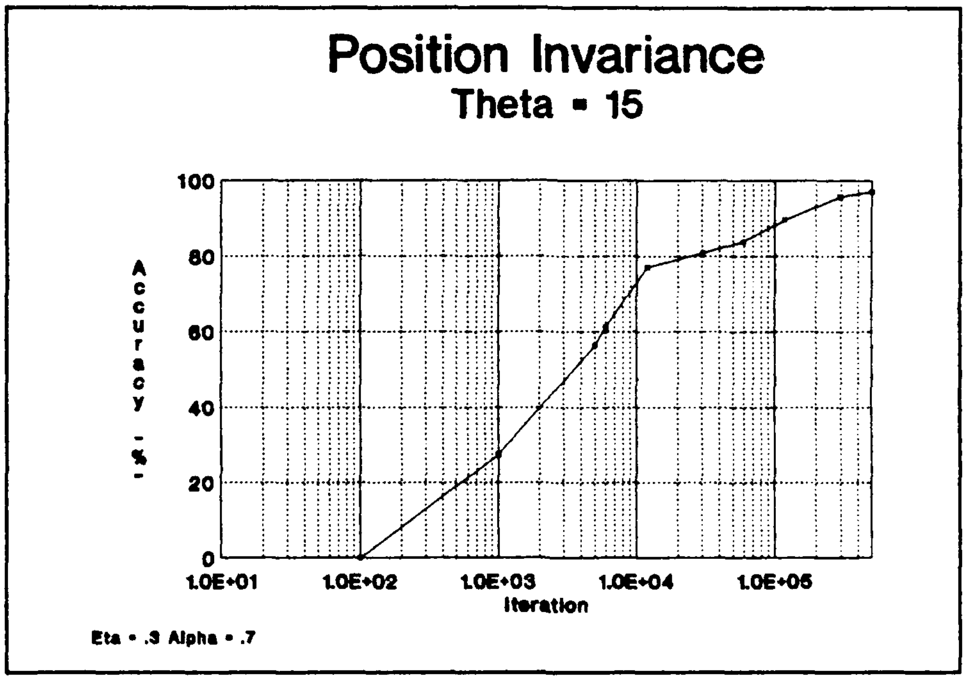

Figure 11: Position test, theta = 15°

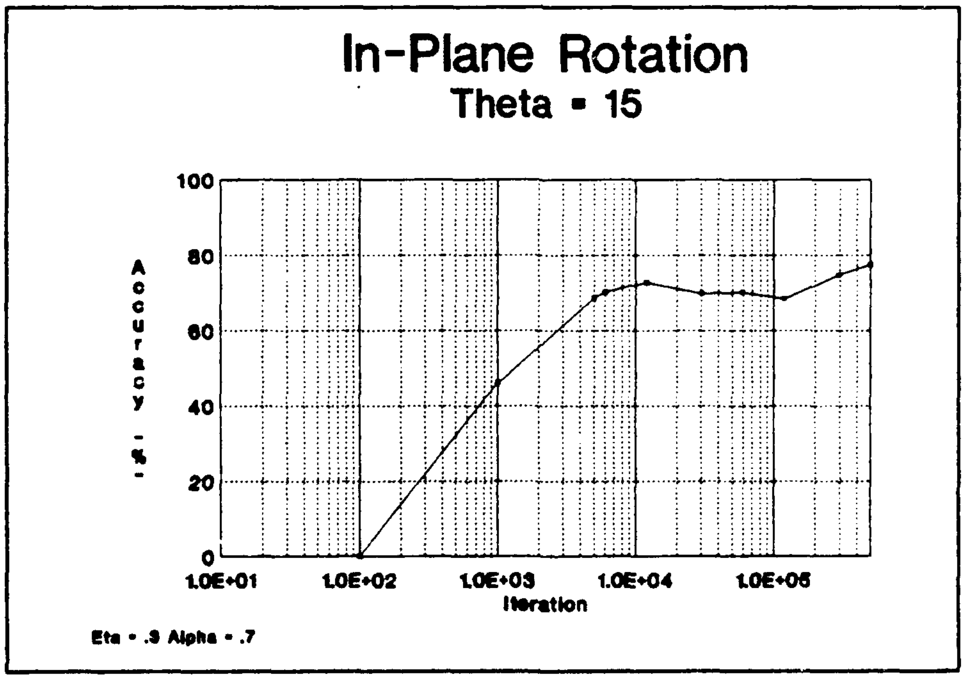

Figure 12: In-plane test, theta = 15°

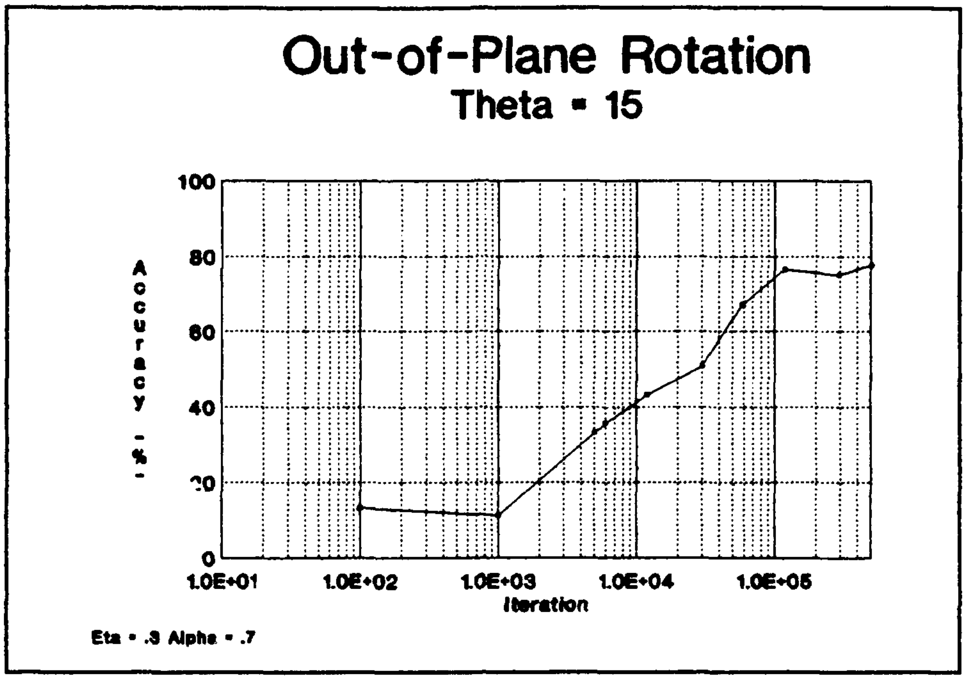

Figure 13: Out-of-plane test, theta = 15°

The position invariant results once more gave high accuracy with no misses after 100,000 iterations. The accuracy for the in-plane test reached as high as 77.4% and for the out-of-plane test, a 77.7% accuracy. There seemed to be an improvement in the out-of-plane test, but no noticeable change was observed in the in-plane test. An observation can be offered to explain these results. Perhaps, the summation of the Gabor transforms could have diluted the information that is important to each filter. Another test with the information for each filter kept separately seemed appropriate.

4.1.4 Rotation at 15°: no summation

The orientation angles were again kept at 15° apart. The number of features selected per filter was reduced to 10, 5 coefficients and their normalized positions. The reduction in the selected features per filter was chosen to limit the number of inputs into the net. The number five was chosen arbitrarily. The top five coefficients in magnitude were selected from each filter. To avoid redundancy in information, if there was a match in the coefficient's positions among the filters, the next highest coefficient was chosen.

The training vectors were increased to 416 for the block, 384 for the balloons, and 392 for the tank. With 12 filters, the number of inputs into the net was 120, and the number of hidden nodes was increased accordingly to 60. The results are shown in Figure 14, Figure 15, and Figure 16.

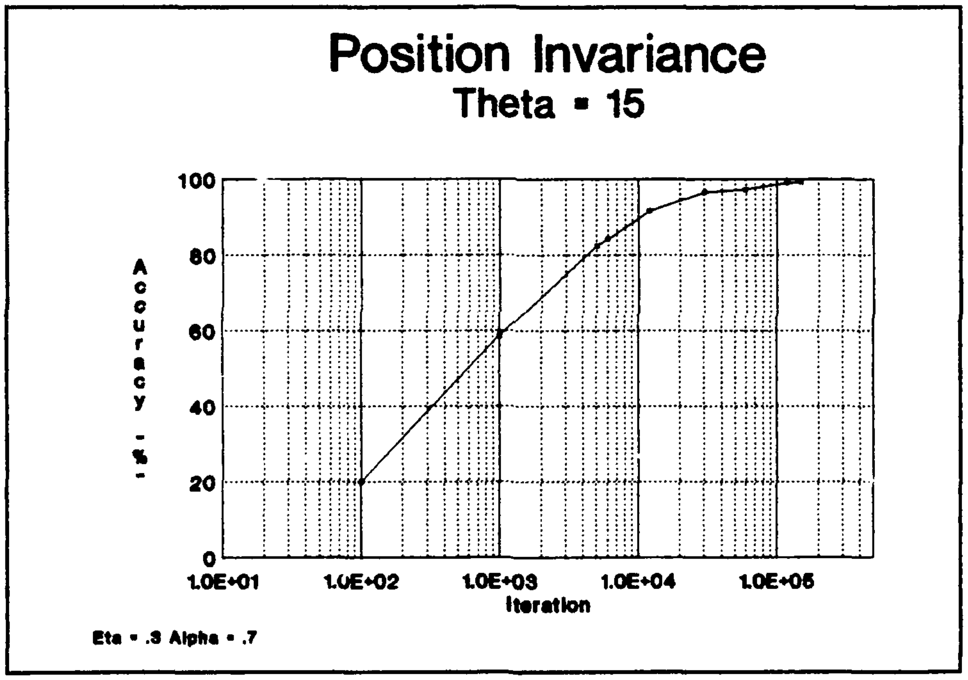

Figure 14: Position test, no summation

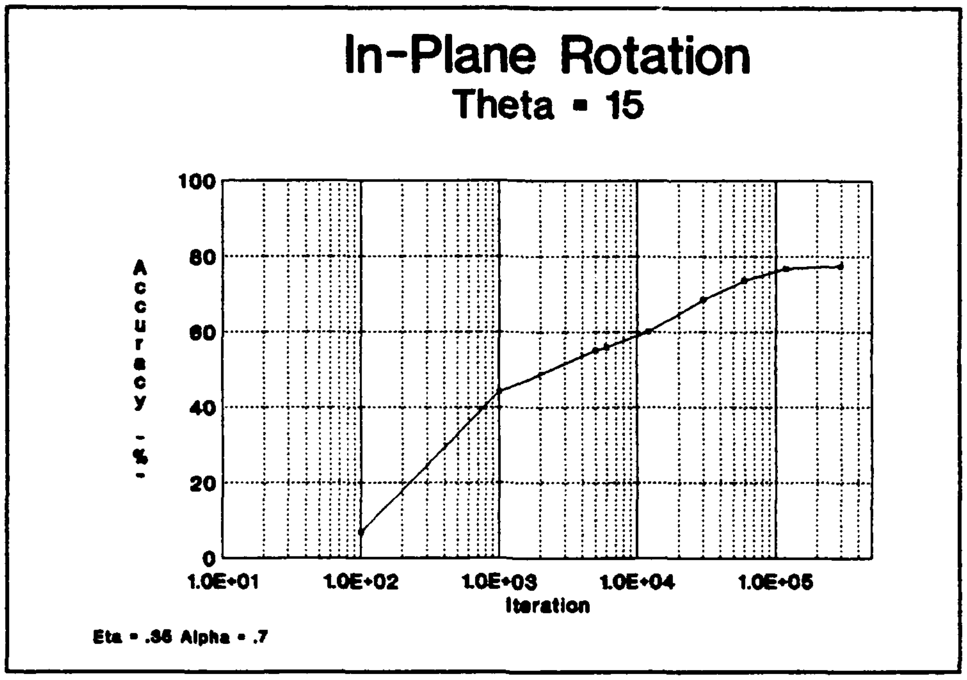

Figure 15: In-plane test, no summation

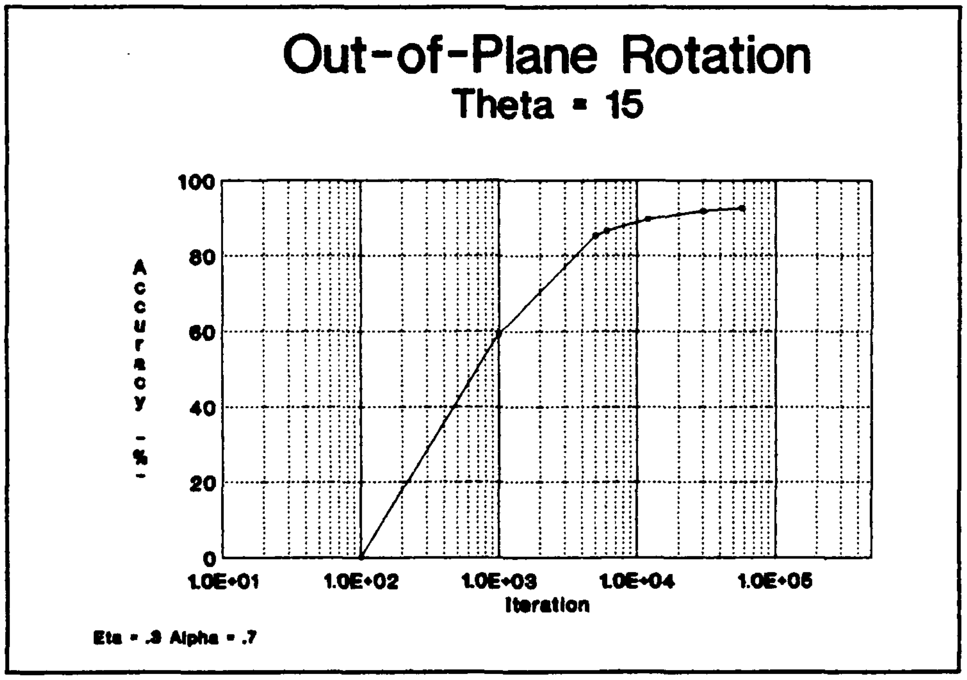

Figure 16: Out-of-plane test, no summation

The same result was obtained for the position invariant test except the convergence was much quicker as compared to previous results. The in-plane test gave comparable results to those with the summation of the transforms. The out-of-plane test, however, showed excellent results with accuracy as high as 92.6% at 57,600 iterations.

Another set of data was tested for comparison with the previous results. Table V and Table VI in Appendix C show the coordinates, chosen randomly, for the in-plane and out-of-plane tests, respectively. Again, the in-plane test gave similar results, Figure 17,to the previous in-plane test data set. The out-of-plane results, Figure 18, did just as well as the previous set with accuracy in the upper 90%.

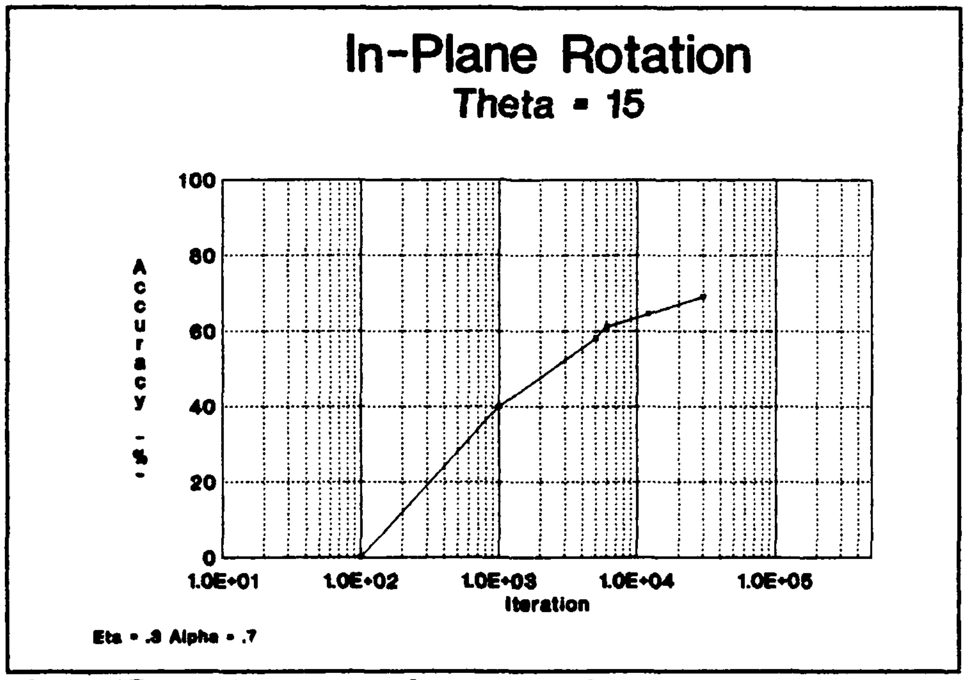

Figure 17: In-plane test, set 2: no summation

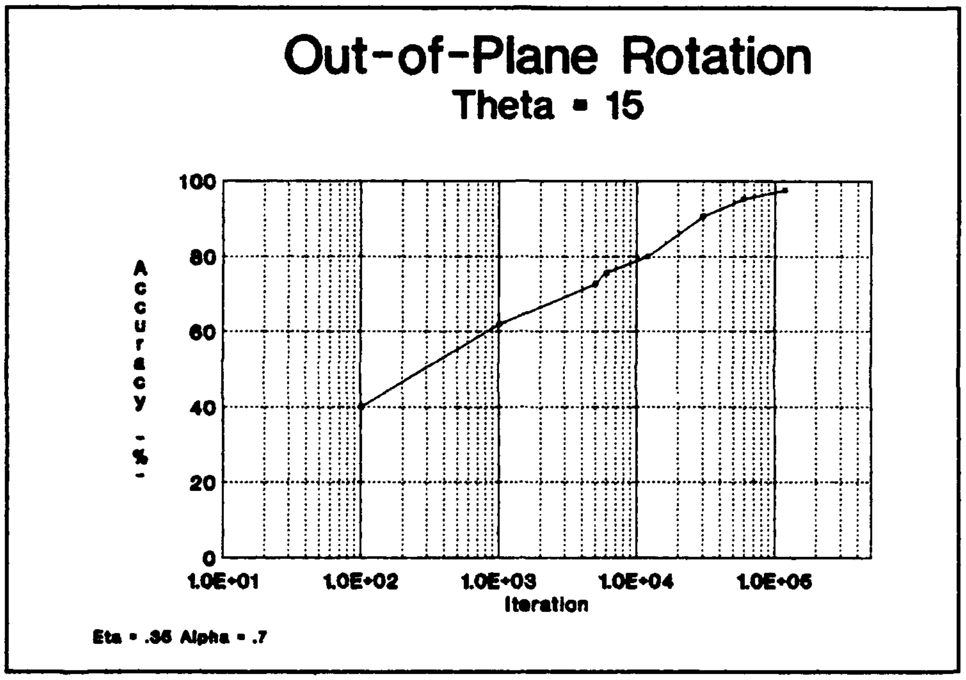

Figure 18: Out-of-plane, set 2: no summation

The similarity between the two sets of results increased the confidence that the net was reporting accurately. The similar results of the in-plane test between keeping the data separately and summation of the transforms were surprising considering the normalization technique used. A review of the normalization technique described in Section 3.1.4 showed that the technique provided not only position invariance, but also in-plane rotation invariance. Since all distance calculations were relative to a maximum coefficient's position, rotations in-plane should give the same summation results at any orientation (in-plane).

If the summation technique provides for in-plane rotation invariance, why are the results not in the upper 90%? The lower accuracy rates can probably be accounted for by two reasons. One, as pointed out earlier, the summation of the transforms could have de-emphasized the important information while emphasizing the less important ones.

Two, error in numerical approximation of the Gabor transforms, which will be discussed in more details in Section 4.3: Problems, seemed to be a likely contributor also. The error in numerical approximation could have caused the same problem for the non-summation technique in-plane test.

From the results, keeping the data separately for each filter at 15° orientation seemed to offer the best results for classifying the objects in path 1. The same vector format was used to determine the perspective view in path 2.

4.2 Path 2: determination of perspective view

Three tests were performed initially to see if the net could learn to output the correct parameters necessary for the image generator to produce a template before moving on to more complex tests as described in Chapter III. Recall that these seven parameters are the xyz coordinates of the camera, the xyz coordinates of the center-of-interest (coi), and the twist angle. The first test involved only the twist angle with the test data taken from the training data set. The second test again involved only the twist angle with test data different from the training set. Finally, the third test varied the xyz coordinates of the camera's position.

4.2.1 Twist angle: similar test data

The training set consisted of 360 vectors of a tank representing 360 degrees of in-plane rotations. Again, the vector format was the same as those in Section 4.1.4 with the data for each filter kept separately and the filter rotation at every 15°. The test data was a subset of the training data. The results of the test are shown in Figure 19.

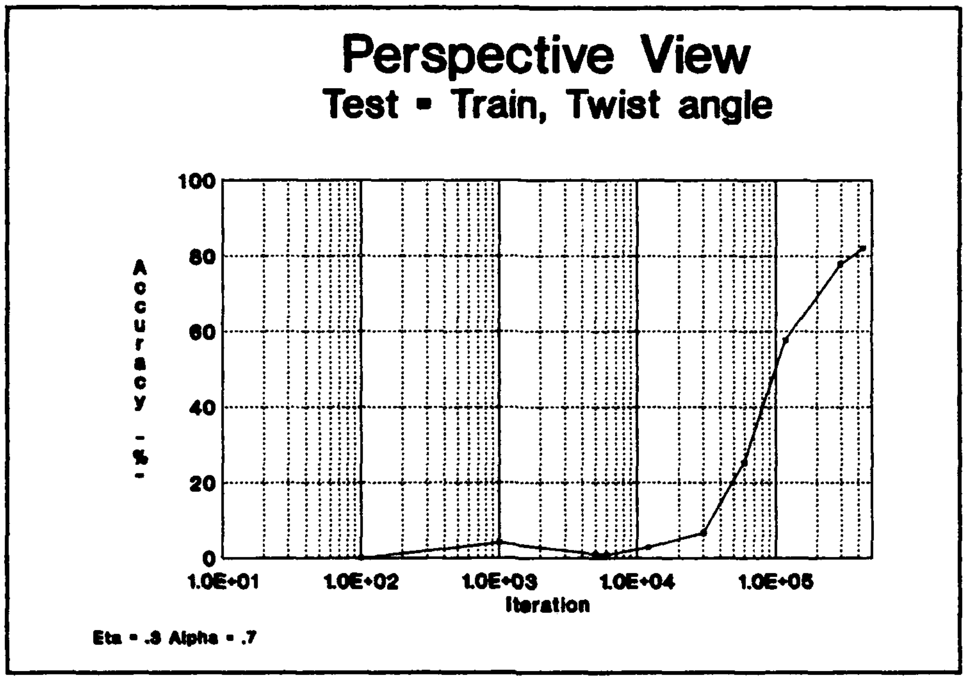

Figure 19: Twist angle: similar test data

The results showed that the net can output the correct parameter with excellent accuracy. Of the 15 test vectors, there were no misses after 100,000 iterations, and given enough time, the accuracy percentage should be in the high 90%.

4.2.2 Twist angle: different test data

Based on the results of the above test, the second test was performed with two changes. One, it seemed unreasonable to train the net at every rotation angle (360) for every view. This number was cut back to 24, 1 for each 15° of in-plane rotation. The number of training vectors was also increased to 408 vectors of different views. The second change involved the test data. The test data consisted of the same 2-D views but with twist angles different from the training data (see Table VII in Appendix C for the listing of the parameters). The results of this test are shown in Figure 20.

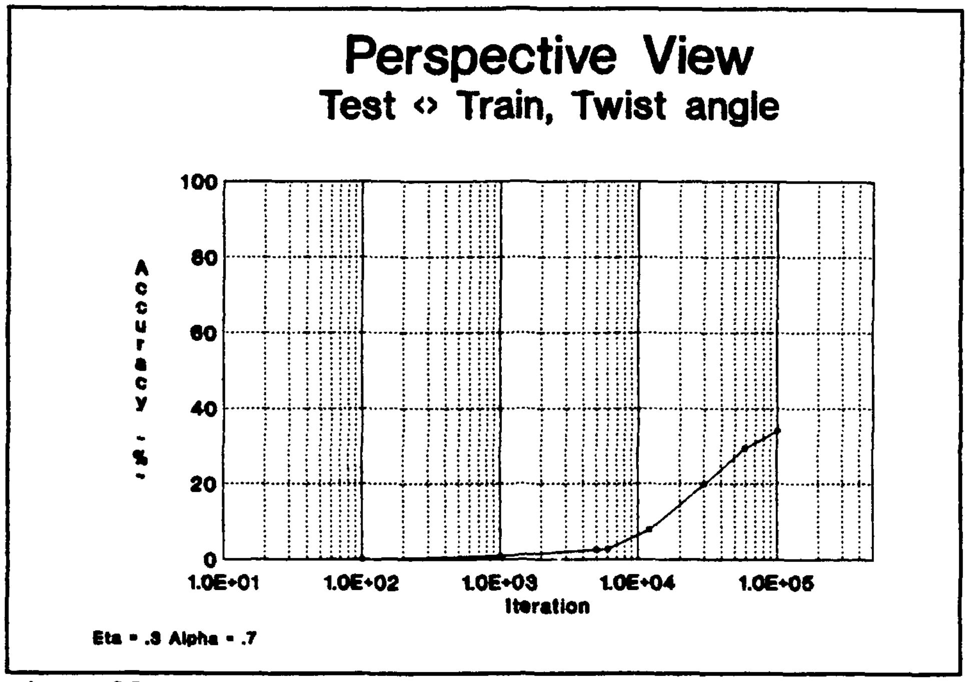

Figure 20: Twist angle: different test data

The results were very poor with the accuracy rate reaching only 34%. Several runs were repeated with longer training time and different step size, eta, but the results were similar. Two conjectures are offered to explain these results. One, the number of training vectors per 2-D view, 24 in this test, were insufficient for the net to correlate between the test data and the training data. The test in Section 4.2.1 showed that the net could learn with 360 training vectors per view. However, if further tests indicate that a larger number, greater than 24, of training vectors per view is needed, the answer would not seem reasonable since the number of training vectors would be enormous. Two, a possible answer to the above results is that the MLP net with these vectors cannot be trained to output the correct parameters within the error constraints. However, the excellent results in Section 4.2.1 do not seem to support this argument. Further testing would have to be performed to back up this answer.

4.2.3 Camera's position: similar test data

This test involved varying the xyz coordinated of the camera's position to see if the net will output the correct parameters. Like Section 4.2.1, the test data was a subset of those from the training set. The results are shown in Figure 21.

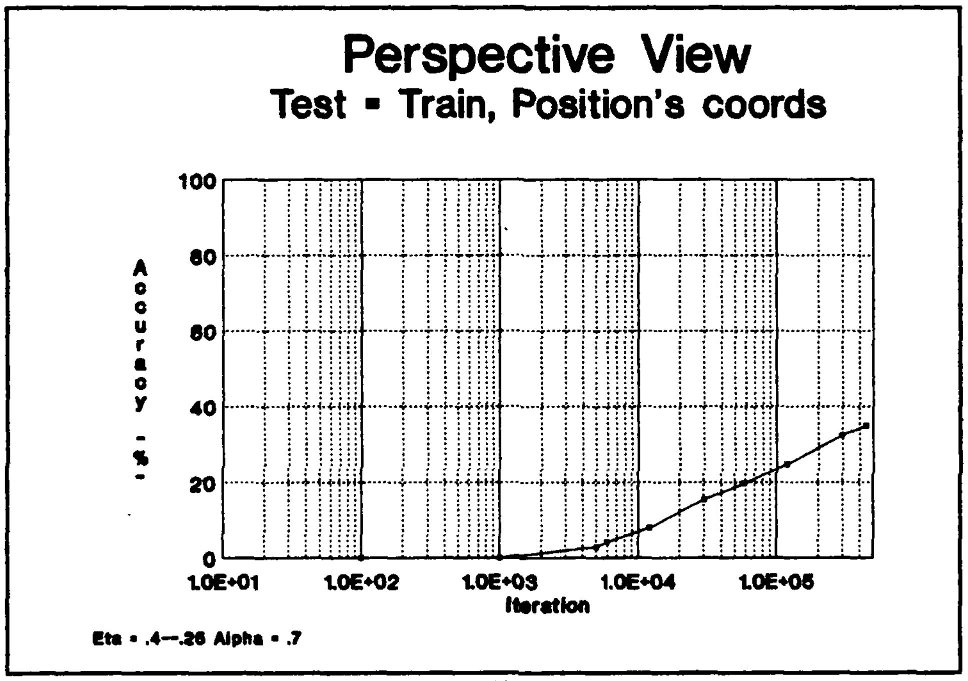

Figure 21: Camera's position: similar test data

Similar to the last test, the results were poor with accuracy reach up to 35%. Again, several runs were repeated, and the results were similar. These results seemed to support the argument that the MLP net configuration with the Gabor features as described cannot be trained to output the correct parameters within the error constraints.

No further tests as outlined in Chapter III were performed at this point since the net cannot be used to provide the necessary parameters to generate a 2-D template. The next section will discuss the problems encountered in this research, and their effect on the Gabor transforms.

4.3 Problems

4.3.1 Numerical approximation

As indicated earlier, the numerical approximation of the Gabor transforms at various angles were not consistent. Theoretically, a target at 0° with a Gabor filter at 0° should give the same transform results as a target at any angle θ with a Gabor filter at the same angle θ. A test was performed to check this theory with one set at 0° and the other at 25°.

Gabor transform at 0°:	15444 260 15444 1 15444 8 15444 119 15442 1
Gabor transform at 25°	14899 395 14889 9 14315 1 14308 9 13788 102

1: Numerical approximation test: 8x8 window

Recall that the vectors' formats are coefficient, position, coeff, pos, etc., starting with the highest coefficient. The first position is the summation of all the other distances to the maximum position. From 1, the corresponding coefficients from both vectors are slightly different. However, the corresponding positions in some cases are not even close. This problem could have contributed to the lower accuracy rate in the in-plane test since the Gabor transforms of various angles did not give the same theoretical results.

The errors in the numerical approximation are caused by the pixelized images being used. Imagine a side of an object covering 5 pixels horizontally being rotate to an angle where the side is aligned diagonally to the pixel array. The length of the side will no longer be the same as the horizontal length, but it will be shorter or longer depending on the modelling toc! used. When this problem is applied to an image, the picture at diagonal rotation will be distorted, and with a small Gabor window (8x8 pixels), slight differences will surface in the correlation results as seen from 1 above. The errors in numerical approximation can be reduced by increasing the size of the Gabor window where the correlation will take place over a larger area, and the slight differences will not be as noticeable as shown in 2 using a 32x32 Gabor window.

Gabor transform at 0°:	12480 82 12480 1 12260 2 12248 1 12217 15
Gabor transform at 25°:	12506 68 12477 1 12311 1 12210 2 12115 15

2: Numerical approximation test 2: 32x32 window

4.3.2 Shading

Different shadings have been found to give different results. For example, from Figure 22. The balloon on the left was generated using FLAT shading, and the balloon on the right was generated using PHONG shading. PHONG shading is generally used on curve surfaces to provide smoother transitions between points on the curve, and FLAT shading, as the name implies, is generally used on flat surfaces.

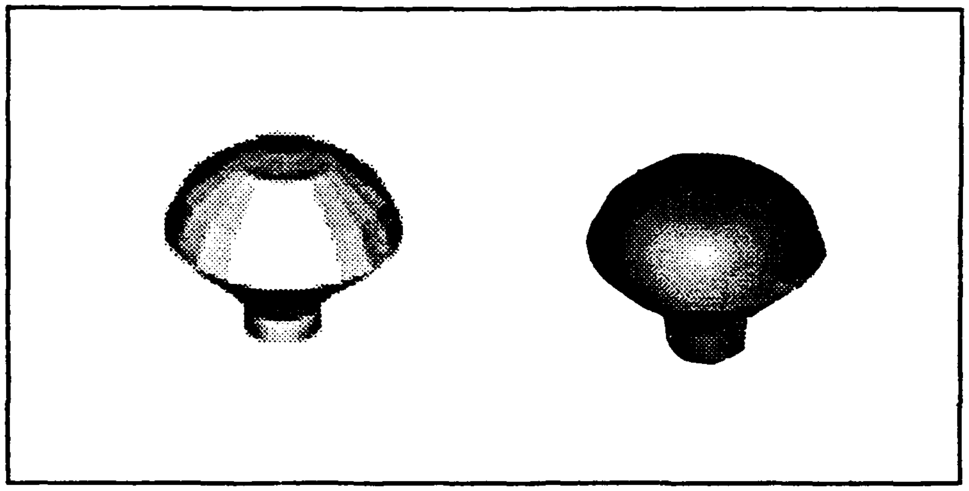
Figure 22: Balloon shadings

As seen from the Gabor transforms results in 3, the corresponding coefficients and positions are slightly different. This problem gave varying results early in the research but was eliminated by keeping the shading for each object constant for all of the training and testing data.

FLAT:	15804	324	15795	9	12371	73	11993	73	11259	73
PHONG:	15804	887	15795	9	14671	115	14106	114	14060	114

3: Shading test

4.3.5 Lighting

Another modelling parameter that made a difference in the results was the position of the lighting. An example is shown in Figure 23 with the lighting for the left block from the camera's position and for the right block, from a point on the x-axis.

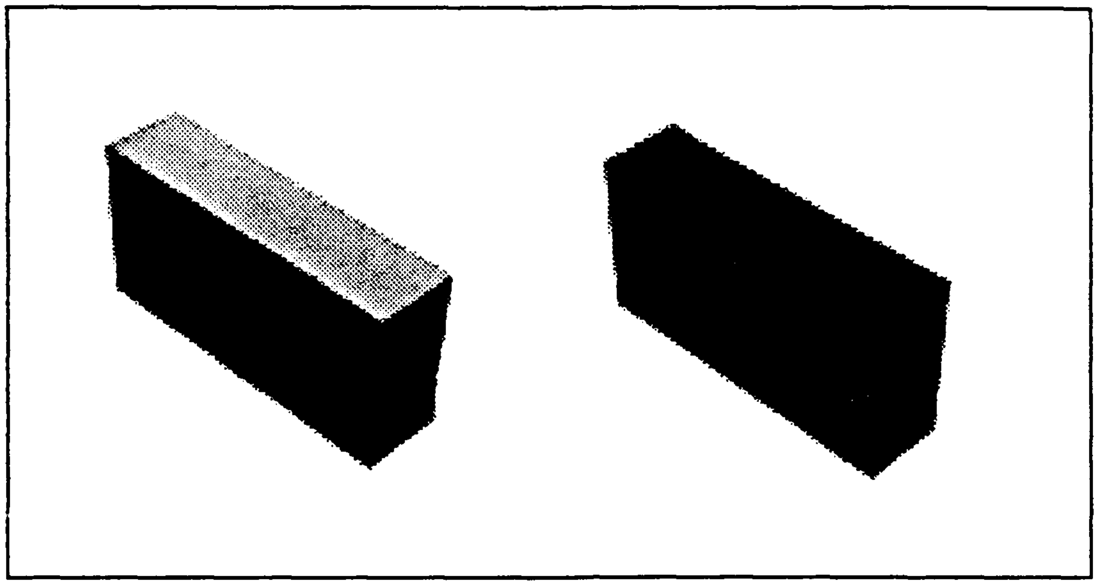
Figure 23: Lighting positions

The results in 4 again show slight differences in the numbers. For the entire test, this problem was mitigated by putting the light at the camera's position.

camera's position:	15804 360 15795 10 15547 43 15368 45 15318 43
on x-axis:	15804 360 15795 10 15587 43 15522 45 15417 42

4: Lighting test

4.3.6 Symmetrical objects

An assumption was made early in the research that symmetrical objects will give the same results at opposite points, and the training data need to cover only one quadrant. However, as seen from the results in 5, this assumption should be made with a caveat. The results came from two views of a block, one at coordinates (14.11,0,1) and the other at the opposite side (-14.11,0,1). The slight differences in the numbers were not understood, but the problem was mitigated by training in all applicable quadrants.

(14.11,0,1):	15804	217	15795	10	10996	1	10996	1	10987	9
(-14.11,0,1):	15804	394	15795	10	11362	47	10996	1	10996	1

5: Symmetrical test

4.4 Summary

In summary, the technique of keeping the transform data for each filter separately provided the best overall results for classification of the objects. The position and out-of-plane tests showed accuracy in the 90's, and the in-plane test showed accuracy in the 70's. Initial results for path 2 showed that the net could be trained to output the correct twist angles. However, subsequent test with different test data gave poor results. Another test to see if the net can output the correct camera's position also failed.

The problems with the modelling parameters and the numerical approximation were discussed. The errors relating to the models were mitigated, but the symmetry problem was not understood. The errors in numerical approximation were shown to effect the Gabor transforms of the objects at various rotation angles.

The next chapter will discuss the implications of these results in the conclusions section and will provide recommendations for further research.

V. Conclusions and Recommendations

5.1 Conclusions

A different approach to pattern recognition was attempted using Gabor features, artificial neural nets and an image generator. The thrust of the research was to use an image generator in the recognition process since it can generate any 2-D templates from the 3-D model. This capability has the advantages of reducing the storage requirement and providing sufficient knowledge over any view of the object.

There were two major goals of this research: one, classify the objects using Gabor information and a neural net; and two, determine the perspective view of the object to create a template for correlation with the original image. The results for the classification of the objects were highly successful. Among the methods tested, the technique of keeping the transform data separately for each filter provide the best overall results. Repeated tests showed similar results and increased the confidence that the Gabor features and the neural net can be used to classify objects.

Initial results with a change in one variable showed that the net can accurately output the desired parameter. However, subsequent tests with increasing levels of difficulty gave poor results. Further research need to be conducted before this combination of MLP net and Gabor features can be dismissed as being unable to correctly output the desired parameters.

Some modelling errors encountered during the research were discussed. Their effects on the transform results were mitigated with respect to the experiments performed.

Future tests must reconsidered the effects of these and other parameters on that particular research.

The numerical approximation error provides a possible explanation for the lower accuracy rate in the in-plane tests. A larger Gabor window has been shown to reduce the error, however, a possible trade-off could be less detail information given to the net.

There are four contributions provided by this thesis. One, the research showed that Gabor features with a neural net can be used to classify objects regardless of their positions, out-of-plane rotations, and to lesser extent in-plane rotations. Two, the use of Gabor features and an MLP net to determine the perspective view of an image provides initial direction for future study in accomplishing this task. Three, the understanding and application of the image generator as a tool for pattern recognition have been advanced. Finally, the fourth contribution of this thesis involved the identification of the problems encountered during the research and provides "warning flags" for future research in this area.

The next section will provide recommendations for future research.

5.2 Recommendations

1. Cosine Gabors could be used instead of sine Gabors or a combination of both.
2. The scale test was not performed in this research. The scale invariance could be achieved by varying the Gaussian windows.

Appendix A: Image Generator

The images used in this thesis for testing were generated using a General Purpose Renderer (gpr) developed at AFIT. Gpr is a general purpose polygon renderer written in C++.

Gpr operates by rendering a geometry file which contains the specific coordinates of the polygons and their attributes like textures and colors. The geometry model is then rendered into an internal buffer based on the parameters specified in the control file. These parameters include size of the image, coordinates that specify the perspective view of the object, lighting, and background color. Once rendering is completed, the buffer is written to an output file in Utah run length encoded (RLE) format. (22) (See Figure 24 below)

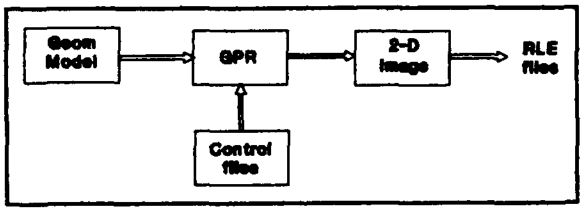

Figure 24: Gpr process

Rendering in this context is described as the transformation of a geometric description into an image.

A.1 Creating an image

As discussed above, the creation of an image requires both the geometry and control files. The coordinates of the geometry model can be input directly if known or generated using the commands: revolve and/or sweep. These commands generate polygonal surfaces based on given profiles and paths of travel, and the outputs of which are geometric description (coordinates and attributes) of the object. Complex objects can be constructed by combining simple models using commands: rotate, scale, translate, and combine. (See manual pages on the Sun for more detailed instructions) The geometry file syntax is given in Table I.

The control files can be patterned after existing ones with modifications to certain controlling parameters relevant to pattern recognition. (See 6 below)

```
buffer ABUFFER size 128 128 background 0 0 0
viewport 0 127 0 127 ka 0.1 color 1 1 1
numcameras 1 numlights 1

position 20 20 20 coi 0 0 0 twist 0 near 0.1 far 10000.0 fovx 50 fovy 50

direction 20 20 20 color 1 1 1 status ON type INFINITE
```

6. Sample control file

The size of the images generated were kept at 128x128 pixels to minimize the calculation time yet maintain enough resolution for visual display. The size can be adjusted by changing the numbers on the first line of the control file. Note that an increase in size will also increase the processing time (doubling the image size will increase the processing time by 4). The perspective view of the object is set by changing the coordinates of the camera's position, the center of interest (coi), and the twist angle. The coordinates are arranged as x y z, and the twist angle is in degree of rotation about the line of sight from the camera's position to the center of interest. These seven numbers are the most important parameters relative to the pattern recognition research since they control the perspective views (in-plane and out-of-plane rotations) of the object as well as their relative sizes and positions. The last line specifies the lighting profile. The direction of the light was kept from the camera's coordinates to the coi's coordinates. Research has found that changing the light direction can affect the Gabor transforms as indicated in Chapter IV. The remaining parameters in the file control other lighting characteristics, colors, and the field of view which can remain unchanged. Table II contains the control file syntax.

A.2 Example

The following steps show how a balloon model and image are created:

step 1: Generate a profile

The coordinates are obtained by plotting the profile of the object on graph paper and creating a profile file with these numbers. The first line of the file specifies the number of points to be plotted from the profile. In this example, there are eight points. The next eight lines define the points further in three dimensional space. The format of this definition is x y z $\bar{n}_x$ $\bar{n}_y$ $\bar{n}_z$, where $\bar{n}_x$, $\bar{n}_y$, and $\bar{n}_z$ are the corresponding normals. An attribute line follows with description of shading for the polygon surfaces.

<balloon.pro>

```
points 8
1 0 0 1 0 0              (x y z n̄x n̄y n̄z)
1 0 0.75 1 0 0
1.5 0 1.5 1 0 -0.5
2.5 0 2.5 1 0 -1
```

```
3 0 3 1 0 0
2.5 0 4 1 0 0.5
1.25 0 4.9 0.1 0 0.9
0 0 5 0 0 1
shading FLAT reflectance FLAT kd 0.1 ks 0.3 n 10 color 1 0 0 opacity 1
```

NOTE: Addition information for texture can be included in the file but is omitted for simplicity. See line 6+ of Table I for the syntax of the shading parameters.

step 2: Create geometry file

using command:

```
revolve <filename.pro> sector degree > <outfile.geom>
```

revolve **balloon.pro** 20 18 > **balloon.geom** will rotate the balloon profile about the z axis creating 20 sectors spaced 18 degrees apart.

step 3: Generate control file

a. Generate a file similar to the file in 6

b. Change the relevant parameters as discussed above

<balloon.control>

```
buffer ABUFFER size 128 128 background 0 0 0
viewport 0 127 0 127 ka 0.1 color 1 1 1
numcameras 1 numlights 1

position 20 20 20 coi 0 0 0 twist 0 near 0.1 far 10000.0
fovx 50 fovy 50

direction 20 20 20 color 1 1 1 status ON type INFINITE
```

This control file will limit the image size to 128x128 pixels, and the object will be viewed from the coordinates (20,20,20) to the center of interest (0,0,0) with no twist.

step 4: Generate RLE file

using command:

```
gpr -c controlfile -g geometryfile -o outputfile [-l listfile] [-i infolevel]
```

where -l specifies a list of geometry files to be rendered, and -i specifies the amount of debugging information to be displayed. The -l and -i flags are optional.

gpr -c **balloon.control** -g **balloon.geom** -o **balloon.rle** will create an image in the file <**balloon.rle**>.

The RLE files can be displayed on several workstations:

TAAC	gettaac <**filename.rle**>
SUN	getsun <**filename.rle**>
SGI 4D	get4D <**filename.rle**>
SGI 3130	getiris <**filename.rle**>

Table I: Geometry File Syntax (8)

Line	Keywords	Description
1	Comments	Anything up to 1024 characters
2	[ccw][cw][purge][nopurge]	Geometry Parameters
3+	points <# of points>	Object component/attribute counts
	patches <# of patches>	
	[parts <# of parts> <list>]	
	[bsp <#of nodes>]	
	[attributes <# of attributes>]	
	[textures <# of textures>]	
4+	<x> <y> <z>	Vertex lines
	[normal <i> <j> <k>]	
	[color <r> <g> <b>]	
	[tindex <u> <v>]	
5+	<n> <pt 1>...<pt n>	Polygon/Patch lines
	[attribute <n>]	
	[texture <n>]	
	[type{PLAIN,COLOR,TEXTURE,STEXTURE}]	
6+	[shading{FLAT,PHONG}]	Attribute Lines
	[reflectance{FLAT,PHONG,COOK}]	
	[kd <n>]	diffuse component
	[ks <n>]	specular reflection component
	[n <n>]	specular profile $cos^n\phi$
	[opacity <n>]	< 1.0 for transparent surfaces
	[color <r> <g> <b>]	base color of object
	[m <n>]	microfacet orientation distribution
	[material <filename>]	name of material file
7+	[filename]	texture map filename
	[WOODGRAIN <scale value>]	solid texture parameters

Table II: Control File Syntax (8)

Line	Keywords	Description
1	[buffer{ABUFFER,ZBUFFER, SZBUFFER}]	Buffer Specification
	[size <xsize> <ysize>]	desired picture size
	[background <r> <g> <b>]	background color
2	[viewport<vxl><vxr><vyb> <vyt>]	Global Environment Parameters
	[ka <n>]	ambient light coefficient
	[color <r> <g> <b>]	ambient light color
3	[numcameras <n>]	number of cameras
	[numlights <n>]	number of lights
4+	[position <x> <y> <z>]	Camera Lines
	[coi <x> <y> <z>]	center of interest
	[twist <n>]	twist angle
	[near <n>]	distance to near clipping plane
	[far <n>]	distance to far clipping plane
	[fovx <n>]	field of view in x
	[fovy <n>]	field of view in y
5+	[position <x> <y> <z>]	Light Lines
	[direction <i> <j> <k>]	direction light is pointing
	[color <r> <g> <b>]	color of light
	[status{ON,OFF}]	turns light ON and OFF
	[type{INFINITE,POINT,SPOT}]	type of light source
	[exponent <n>]	specular profile
	[intensity <n>]	strength of light source
	[solidangle <n>]	spread angle for POINT and SPOT Lights and COOK shading model

Appendix B: Repositioning Method

The data used for the position invariant test were obtained using a cut and reposition method. For the test, the data needed to consist of objects in different locations within the image while maintaining the same scale. Of the various methods tried, the cut and reposition method was the only one that could accomplished this task.

The cut and reposition method employs four commands:

rlebg - create a background

crop - segment the object

repos - reposition the object

comp - combine the repositioned object with the background

B.1 Example

The following steps show how a tank is moved to a new location while keeping the scale constant:

step 1: Create a background

A background is generated to place the object on after it is repositioned.

Using the command:

rlebg **red green blue [alpha]** [-l] [-v[top[bottom]]] [-s **xsize ysize**] **[outfile]**

rlebg 0 0 0 -s 128 128 > **bg.rle** creates a black background of 128x128 pixels and gives the output to the file **bg.rle**. For explanation of the options, see the manual pages on the Sun 4.

step 2: Segment the object

Given an image in Figure 25, cut out the object as shown by the dotted lines.

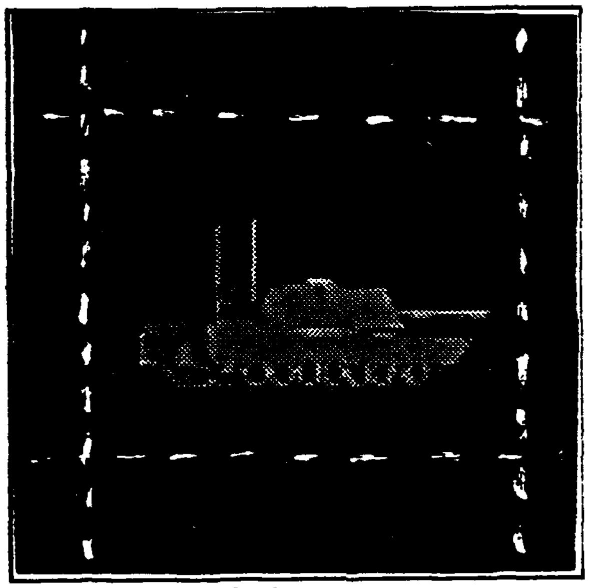
Figure 25: Tank.rle

Using the command:

crop **xmin ymin xmax ymax [infile]** > **outfile**

crop 10 25 118 108 **tank.rle** > **t.rle** cuts out the object at xmin-10, ymin-25, xmax-118, and ymax-108. These coordinates are determined by how large the original image is. In this example, the image is 128x128 pixels.

step 3: Reposition the object

Let's supposed the new desired location for the object is in the bottom left corner of the image.

Using the command:

repos [-p **xpos ypos**] [-i **xinc yinc**] [**infile**] > **outfile** repos -p 0 0 t.rle > t1.rle moves the lower corner of the cut out area represented by (xmin,ymin) to a new coordinates (0,0). Explanations of the options can be found in the manual pages on the Sun 4.

step 4: Combine

By combining the repositioned object with the background, a new image is created with a repositioned object having the same scale as the object in Figure 25.

Using the command:

```
comp [-o op] Afile Bfile > outfile
```

comp -o atop t1.rle bg.rle > **tank1.rle** will output the results in **tank1.rle** as shown in Figure 26. Other op commands can be found in the manual pages on the Sun 4.

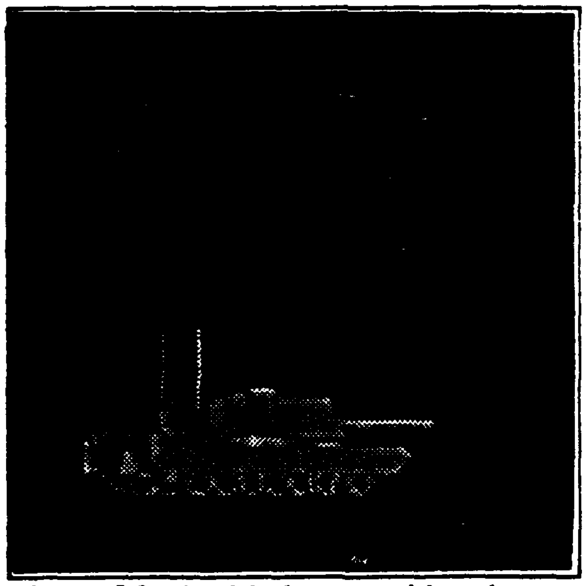
Figure 26: Tank1.rle: repositioned

Appendix C: Test Data

Table III: Out-of-Plane test data

Vectors	Position's coordinates
Test.1	(9.77,9.77,3)
Test.2	(0,-13.82,3)
Test.3	(-7.71,-7.71,9)
Test.4	(-10.91,0,9)
Test.5	(6.28,6,28,11)
Test.6	(19.92,19.92,5)
Test.7	(0,-27.53,9)
Test.8	(24.28,0,17)
Test.9	(-15.71,-15.71,20)
Test.10	(-14.13,0,27)
Test.11	(724.87,850,600)
Test.12	(-401.95,587,1000)
Test.13	(-509.02,-800,550)
Test.14	(806.05,-775,125)
Test.15	(806.75,980,0)

Table IV: In-Plane test data

Vectors	Position's coordinates	Twist angle
Test.1	(8.89,0,-11)	15
Test.2	(-11.22,-7,-5)	30
Test.3	(4.69,-3,13)	40
Test.4	(-5.48,-13,1)	-20
Test.5	(-8.37,9,7)	53
Test.6	(23.28,-16,4)	-15
Test.7	(-22.67,-16,8)	45
Test.8	(19.64,18,12)	20
Test.9	(16.55,8,24)	-60
Test.10	(-15.42,16,20)	-27
Test.11	(-1156.61,300,0)	-30
Test.12	(-1153.77,100,240)	15
Test.13	(988.87,-400,480)	-20
Test.14	(-801.54,600,720)	-40
Test.15	(-374.58,700,960)	17

Table V: Out-of-Plane test data2

Vectors	Position's coordinates
Test.1	(10,10,0)
Test.2	(12.65,2,6)
Test.3	(-9.54,10,-3)
Test.4	(-9.98,-9.98,-1)
Test.5	(8.66,5,10)
Test.6	(23.97,15,2)
Test.7	(19.85,19.85,6)
Test.8	(-22.42,-5,19)
Test.9	(17.02,-2,25)
Test.10	(-22.78,15,10)
Test.11	(853.68,-210,800)
Test.12	(-764.69,790,625)
Test.13	(-725.83,1000,313)
Test.14	(-558.56,-840,487)
Test.15	(990.20,650,210)

Table VI: In-Plane test data2

Vectors	Position's coordinates	Twist angle
Test.1	(9.77,9.77,3)	15
Test.2	(0,-13.82,3)	30
Test.3	(-7.71,-7.71,9)	40
Test.4	(-10.91,0,9)	-20
Test.5	(6.28,6,28,11)	53
Test.6	(19.92,19.92,5)	-15
Test.7	(0,-27.53,9)	45
Test.8	(24.28,0,17)	20
Test.9	(-15.71,-15.71,20)	-60
Test.10	(-14.13,0,27)	-27
Test.11	(724.87,850,600)	-30
Test.12	(-401.95,587,1000)	15
Test.13	(-509.02,-800,550)	-20
Test.14	(806.05,-775,125)	-40
Test.15	(806.75,980,0)	17

Table VII: Twist angle test coordinates

Vectors	Position's coordinates	Twist angle
Test.1	(439.57,-1000,0)	10
Test.2	(439.51,-1000,0)	53
Test.3	(1038.83,-500,0)	25
Test.4	(1038.83,-500,0)	100
Test.5	(1199.51,0,0)	5
Test.6	(1199.51,0,0)	70
Test.7	(1133.52,500,0)	93
Test.8	(1133.52,500,0)	125
Test.9	(783.49,1000,0)	7
Test.10	(783.49,1000,0)	26
Test.11	(332.57,-1000,300)	33
Test.12	(332.57,-1000,300)	97
Test.13	(1000.33,-500,300)	48
Test.14	(1000.33,-500,300)	113
Test.15	(1166.46,0,300)	277

Appendix D: Source Codes

```
** The following codes were used to run a MLP neural network */

/* This is the main program that calls other functions to perform the
neural net input, calculations, and outputs.                    */

#include "macros.h"
#include <stdio.h>
#include "globals.h"

void menu(float [],float [],float [],float []);
void makeinput(float [],float []);
void ran_gen(int []);
void feed(float [],float [],float [],float [],float [],float [],float []);
void backprop(float [],float [],float [],float [],float [],float [],float [],
              float [],int *,float *);
void save_params(float [],float [],float [],float []);
void initialize(float [],float [],float [],float []);
void test(float *,int *,int *);

main()
{
    int seq[VECTORS+BVECTORS+TVECTORS];
    int i,j,iteration,num;

    iteration = 0;
    correct = 0;
    test_acc = 0;
    eta = ETA;
    initialize(w1,w2,t1,t2);
    makeinput(feats,classes);     /* generate an array of inputs */
    while (iteration < 150000)
    {
        ran_gen(seq);                   /* generate random number */
        loopi(VECTORS+BVECTORS+TVECTORS)
        {
            iteration++;
            loopj(FEATURES)           /* select a random vector */
                vector[j] = feats[idx(seq[i]-1,j,FEATURES)];
            loopj(OUTPUT_LAYER)    /* assign appropriate output */
            {
                if ((j+1) == classes[seq[i]-1])
                    desired_output[j] = 1.0;
                else
                    desired_output[j] = 0.0;
            }
```

```
                    feed(y_hidden,y_out,w1,w2,t1,t2,vector);
                    /* check error every 100 iterations */
                    if ((iteration % 100) == 0)
                    {
                        num = iteration;
                        printf("%d\n",iteration);
                        test(&test_acc,&correct,&iteration);
                    }
                    backprop(vector,w1,w2,t1,t2,y_hidden,y_out,
                        desired_output,&num,&eta);
                }
        }
        save_params(w1,w2,t1,t2);
}

/* This function randomly assigns values to the weights and thetas with
   values between -1 and 1                                          */

#include <time.h>
#include <stdio.h>
#include <math.h>
#include "macros.h"

int ran_gen(int []);

initialize(float w1[],float w2[],float t1[],float t2[])
{
        int i,j,sign;
        int seq[VECTORS+BVECTORS+TVECTORS];

        srand((unsigned)time(NULL));
        ran_gen(seq);
        loopi(OUTPUT_LAYER)
        {
                loopj(LAYER_ONE)
                {
                        sign = (int) pow((double) -1,(double) seq[j]);
                        w2[idx(i,j,LAYER_ONE)] = rand() % 100;
                        w2[idx(i,j,LAYER_ONE)] = sign *
                                        (w2[idx(i,j,LAYER_ONE)]/100);
                }
                sign = (int) pow((double) -1,(double) seq[j]);
                t2[i] = rand() % 100;
                t2[i] = sign * (t2[i]/100);
        }
        loopi(LAYER_ONE)
        {
                loopj(FEATURES)
                {
                        sign = (int) pow((double) -1,(double) seq[j]);
                w1[idx(i,j,FEATURES)] = rand() % 100;
                        w1[idx(i,j,FEATURES)] = sign *
                                        (w1[idx(i,j,FEATURES)]/100);
                }
                sign = (int) pow((double) -1,(double) seq[j]);
                t1[i] = rand() % 100;
                t1[i] = sign * (t1[i]/100);
        }
}


/* This function reads in the data vectors and creates arrays for further
   processing                                                         */

#include <stdio.h>
#include "macros.h"

makeinput(float feats[],float classes[])
{
    FILE *fp;
    int i,j;
    char filename[20];
    float c,f;
    float vect[VECTORS*FEATURES],bvect[BVECTORS*FEATURES],
        tvect[TVECTORS*FEATURES],testvect[TESTS];
    float class[VECTORS],bclass[BVECTORS],tclass[TVECTORS],testclass[TESTS];

    loopi(VECTORS)        /* store block data and class */
    {
        sprintf(filename,"../normdata/vectors.%d",i+1);
       OPEN_FILE(fp,filename,filename);  /* open vector files for */
                         /* reading */

        loopj(FEATURES)              /* storing features */
        {
            fscanf(fp,"%f ",&f);
            vect[idx(i,j,FEATURES)] = f;
        }
        fscanf(fp,"%f\n",&c);            /* storing classes */
        class[i] = c;
        fclose(fp);
}
loopi(BVECTORS)       /* store balloon data and class */
{
        sprintf(filename,"../normbdata/bvectors.%d",i+1);
       OPEN_FILE(fp,filename,filename);  /* open vector files for */
                                   /* reading */

        loopj(FEATURES)                 /* storing features */
        {
             fscanf(fp,"%f ",&f);
             bvect[idx(i,j,FEATURES)] = f;
        }
        fscanf(fp,"%f\n",&c);               /* storing classes */
        bclass[i] = c;
        fclose(fp);
}
loopi(TVECTORS)       /* store tank data and class */
{
        sprintf(filename,"../normtdata/tvectors.%d",i+1);
       OPEN_FILE(fp,filename,filename);  /* open vector files for */
                                   /* reading */

        loopj(FEATURES)                 /* storing features */
        {
             fscanf(fp,"%f ",&f);
             tvect[idx(i,j,FEATURES)] = f;
        }
        fscanf(fp,"%f\n",&c);               /* storing classes */
        tclass[i] = c;
        fclose(fp);
}
loopi(TESTS)      /* store test data and class */
{
       sprintf(filename,"../testdata/test.%d",i+1);
       OPEN_FILE(fp,filename,filename);  /* open vector files for */
                                          /* reading */

       loopj(FEATURES)                 /* storing features */
       {
            fscanf(fp,"%f ",&f);
            testvect[idx(i,j,FEATURES)] =  f;
       }
       fscanf(fp,"%f\n",&c);                /* storing classes */
        testclass[i] = c;
        fclose(fp);
    }

    loopi(VECTORS)        /* combine all data and classes */
    {
        loopj(FEATURES)
            feats[idx(i,j,FEATURES)] = vect[idx(i,j,FEATURES)];
        classes[i] = class[i];
    }
    loopi(BVECTORS)
    {
        loopj(FEATURES)
            feats[idx(i+VECTORS,j,FEATURES)] = bvect[idx(i,j,FEATURES)];
        classes[i+VECTORS] = bclass[i];
    }
    loopi(TVECTORS)
    {
        loopj(FEATURES)
            feats[idx(i+(VECTORS+BVECTORS),j,FEATURES)] =
                                    tvect[idx(i,j,FEATURES)];
        classes[i+(VECTORS+BVECTORS)] = tclass[i];
    }
    loopi(TESTS)
    {
        loopj(FEATURES)
            feats[idx(i+(VECTORS+BVECTORS+TVECTORS),j,FEATURES)]=
                                    testvect[idx(i,j,FEATURES)];
        classes[i+(VECTORS+BVECTORS+TVECTORS)] = testclass[i];
    }


}


/* This function provides a random selection of the input vectors used
   to train the neural nets                                        */

#include <stdio.h>
#include <time.h>
#include "macros.h"

ran_gen(int seq[VECTORS+BVECTORS+TVECTORS])

{
      int count=0,i, randnum;
```

```
          srand((unsigned)time(NULL));
          while(count<(VECTORS+BVECTORS+TVECTORS))
          {
                randnum = (rand() % (VECTORS+BVECTORS+TVECTORS)) +1;
                for (i=0;i<count;i++)
                {
                      if(randnum == seq[i]) break;
                }
                if(i>=count)
                {
                      seq[count] = randnum;
                      count++;
                }
          }
}


/* This function takes an input vector and feeds it through the net using
   assigned parameters and provides the calculated outputs.

Original author:  Greg Tarr
Revised by:  Phung Le                                                  */

#include "macros.h"
#include <math.h>

float calcy(float [],float [],float [],int,int);


feed(float y_hidden[],float y_out[],float w1[],float w2[],float t1[],float t2[],
          float input[])
{
          int i;

          loopi(LAYER_ONE)   /* propagate inputs thru first layer */
                y_hidden[i] = calcy(input,w1,t1,i,FEATURES);
          loopi(OUTPUT_LAYER)  /* propagate from first to output layer */
                y_out[i] = calcy(y_hidden,w2,t2,i,LAYER_ONE);
}

float calcy(float x[],float weight[],float theta[],int upper,int lower)
{
          int i;
          float y;
```

```
        y = 0.0;              /* initialize output value */
        loopi(lower)          /* calculate summation of inputs x weights */
                y = y + (x[i] * weight[upper * lower +i]);

        y = y + theta[upper];       /* subtract the offset, theta */
        if (y < 60 && -y > -60)
                y = 1.0 / (1.0 + exp (-y)); /* return output as a function */
        else                                /* of a sigmoid */
        {
                if (y >= 60)                    /* boundary conditions */
                        y = 1.0;
                if (y <= -60)
                        y = 0.0;
        }
        return (y);

}


/* This function uses the recursive backpropagation training algorithm
   to update the weights and thetas

Original author:   Greg Tarr
Revised by:  Phung Le                                                   */

#include "macros.h"
#define MAX 1e34

void dely(float [],float [],float []);
void delx(float [],float [],float [],float []);


backprop(float x[],float w1[],float w2[],float t1[],float t2[],
         float y_hidden[],float y_out[],float d[],int *N,float *eta)
{
        int i,j;
        float del_upper[OUTPUT_LAYER],del_lower[LAYER_ONE];
        f           l           o           a           t
new_w1[LAYER_ONE*FEATURES],new_w2[OUTPUT_LAYER*LAYER_ONE];
        float new_t1[LAYER_ONE],new_t2[OUTPUT_LAYER];
        s     t     a     t     i     c           f     l     o     a     t
old_w1[LAYER_ONE*FEATURES],old_w2[OUTPUT_LAYER*LAYER_ONE];
        static float old_t1[LAYER_ONE],old_t2[OUTPUT_LAYER];

        if ((*N) > 30000)
        (*eta) = (float) .3;
/* calculate error terms and adjust weights from hidden to output
   layer */
loopi(OUTPUT_LAYER)        /* initialize */
{
        loopj(LAYER_ONE)
                new_w2[idx(i,j,LAYER_ONE)] = 0.0;
        del_upper[i] = 0.0;
        new_t2[i] = 0.0;
}

dely(y_out,d,del_upper);     /* get error terms to output layer */
loopi(OUTPUT_LAYER)
{
        loopj(LAYER_ONE)
        {
                new_w2[idx(i,j,LAYER_ONE)] = w2[idx(i,j,LAYER_ONE)] +
                  ((*eta) * del_upper[i] * y_hidden[j]) +
                  (ALPHA * (w2[idx(i,j,LAYER_ONE)] -
                  old_w2[idx(i,j,LAYER_ONE)]));
                old_w2[idx(i,j,LAYER_ONE)] = w2[idx(i,j,LAYER_ONE)];
        }
        new_t2[i] = t2[i] + ((*eta) * del_upper[i]) +
                        (ALPHA * (t2[i] - old_t2[i]));
        old_t2[i] = t2[i];
}
loopi(OUTPUT_LAYER)    /* reassign w2 and t2 with updated values */
{
        loopj(LAYER_ONE)
        {
                if (new_w2[idx(i,j,LAYER_ONE)] > MAX)
                        new_w2[idx(i,j,LAYER_ONE)] = MAX;
                if (new_w2[idx(i,j,LAYER_ONE)] < -MAX)
                        new_w2[idx(i,j,LAYER_ONE)] = -MAX;
                w2[idx(i,j,LAYER_ONE)] = new_w2[idx(i,j,LAYER_ONE)];
        }
        if (new_t2[i] > MAX) new_t2[i] = MAX;
        if (new_t2[i] < -MAX) new_t2[i] = -MAX;
        t2[i] = new_t2[i];
}

/* calculate error terms and adjust weight from input to hidden
   layer */
loopi(LAYER_ONE)           /* initialize */
{
```

```
            loopj(FEATURES)
                  new_w1[idx(i,j,FEATURES)] = 0.0;
            del_lower[i] = 0.0;
            new_t1[i] = 0.0;
      }
      delx(y_hidden,w2,del_upper,del_lower);  /* get error terms to hidden */
                                              /* layer */
      loopi(LAYER_ONE)
      {
            loopj(FEATURES)
            {
                  new_w1[idx(i,j,FEATURES)] = w1[idx(i,j,FEATURES)] +
                    ((*eta) * del_lower[i] * x[j]) +
                    (ALPHA * (w1[idx(i,j,FEATURES)] -
                    old_w1[idx(i,j,FEATURES)]));
                  old_w1[idx(i,j,FEATURES)] = w1[idx(i,j,FEATURES)];
            }
            new_t1[i] = t1[i] + ((*eta) * del_lower[i]) +
                        (ALPHA * (t1[i] - old_t1[i]));
            old_t1[i] = t1[i];
      }
      loopi(LAYER_ONE) /* reassign w1 and t1 with updated values */
      {
            loopj(FEATURES)
            {
                  if (new_w1[idx(i,j,FEATURES)] > MAX)
                        new_w1[idx(i,j,FEATURES)] = MAX;
                  if (new_w1[idx(i,j,FEATURES)] < -MAX)
                        new_w1[idx(i,j,FEATURES)] = -MAX;
                  w1[idx(i,j,FEATURES)] = new_w1[idx(i,j,FEATURES)];
            }
            if (new_t1[i] > MAX) new_t1[i] = MAX;
            if (new_t1[i] < -MAX) new_t1[i] = -MAX;
            t1[i] = new_t1[i];
      }
}

void dely(float y[],float d[],float del_upper[])
{
      int i;

      loopi(OUTPUT_LAYER)
            del_upper[i] = y[i] * (1 - y[i]) * (d[i] - y[i]);
}
```

```
void delx(float y[],float w[],float del_upper[],float del_lower[])
{
        int i,j;
        float sum;

        sum = 0.0;
        loopi(LAYER_ONE)
        {
                loopj(OUTPUT_LAYER)
                        sum = sum + (del_upper[j] * w[idx(i,j,OUTPUT_LAYER)]);
                del_lower[i] = y[i] * (1 - y[i]) * sum;
        }
}


/* This function checks the testdata and assigns the error as well as
the accuracy associated with the test                              */

#include "macros.h"
#include <stdio.h>
#include "globals.h"

void feed(float [],float [],float [],float [],float [],float [],float []);
int check_error(float [],float [],int *);

test(float *test_acc,int *correct,int *num_check)
{
        int i,j,vector_sum;

        vector_sum = VECTORS+BVECTORS+TVECTORS;
        loopi(TESTS)
        {
                loopj(FEATURES)         /* select a random vector */
                        vector[j] = feats[idx(i+vector_sum,j,FEATURES)];
                loopj(OUTPUT_LAYER)     /* assign appropriate output */
                {
                        if ((j+1) == classes[i+vector_sum])
                                desired_output[j] = 1.0;
                        else
                                desired_output[j] = 0.0;
                }
                feed(y_hidden,y_out,w1,w2,t1,t2,vector);
                check_error(y_out,desired_output,correct);
        }
        /* acc = total correct/num check */
        *test_acc = (float) (*correct)/(((*num_check)/100)*(TESTS));
        printf("acc: %f\n",*test_acc);
}

/* This function calculates and displays the error at the output layer */

#include "macros.h"
#include <math.h>

check_error(float y[],float d[],int *correct)
{
        int i;
        float err;
        float error[OUTPUT_LAYER];

        err = 0;
        loopi(OUTPUT_LAYER)
        {
                error[i] = fabs((float) d[i] - y[i]);
                err += SQR(error[i]);
        }
        err = .5*err;
        printf("%f ",err);
        if (error[0] < .2 && error[1] < .2 && error[2] <.2)
                (*correct)++;
        printf("%d\n",*correct);

}


/* This function saves the neural net parameters (weights, offsets, layer
numbers) for later use.                                                   */

#include "macros.h"
#include <stdio.h>

save_params(float w1[],float w2[],float t1[],float t2[])
{
        int i,j;
        char select;
        FILE *fp;
        char filename[20] = "gabor.wgt";

/*      printf("Save parameters? ");
        select = getchar();
        if (select == 'n')
                exit(1);
        else
        {
                printf("\nEnter filename: ");
        scanf("%s",filename);*/
        CREATE_FILE(fp,filename,filename);
        fprintf(fp,"%d %d %d \n",OUTPUT_LAYER,LAYER_ONE,FEATURES);
        loopij(OUTPUT_LAYER,LAYER_ONE)
                fprintf(fp,"%f ",w2[idx(i,j,LAYER_ONE)]);
        loopij(LAYER_ONE,FEATURES)
                fprintf(fp,"%f ",w1[idx(i,j,FEATURES)]);
        loopi(OUTPUT_LAYER)
                fprintf(fp,"%f ",t2[i]);
        loopi(LAYER_ONE)
                fprintf(fp,"%f ",t1[i]);
        fclose(fp);

}


/* This function provides a menu for initializing or assigning existing
   values to all weights and thetas                                    */

#include <stdio.h>

void initialize(float [],float [],float [],float []);
void read_params(char [],float [],float [],float [],float []);

menu(float w1[],float w2[],float t1[],float t2[])
{
        int select;
        char filename[20];

        printf("\nMENU\n");
        printf("1.  Initilize parameters \n");
        printf("2.  Retrieve parameter file \n");
        printf("e.  EXIT \n");
        select = getchar();
        switch(select)
        {
                case 'e': exit(0);
                        break;
                case '1': initialize(w1,w2,t1,t2);
                        break;
                case '2': printf("\nEnter filename: ");
                          scanf("%s",filename);
                          read_params(filename,w1,w2,t1,t2);
                          printf("Parameters installed\n");
                          break;
        }
}


/* This function reads in parameter values from an existing file and
   assign them to the weights and thetas appropriately                  */

#include "macros.h"
#include <stdio.h>

read_params(char filename[],float w1[],float w2[],float t1[],float t2[])
{
        int i,j;
        FILE *fp;
        float temp1,temp2,temp3,temp4;
        int output_nodes,hidden_nodes,input_nodes;

        OPEN_FILE(fp,filename,filename);
        fscanf(fp,"%d %d %d",&output_nodes,&hidden_nodes,&input_nodes);
                                        /* assigning weight values */
        loopij(output_nodes,hidden_nodes)
        {
                fscanf(fp,"%f ",&temp1);
                w2[idx(i,j,hidden_nodes)] = temp1;
        }
        loopij(hidden_nodes,input_nodes)
        {
                fscanf(fp,"%f ",&temp2);
                w1[idx(i,j,input_nodes)] = temp2;
        }
                                        /* assigning theta values */
        loopi(output_nodes)
        {
                fscanf(fp,"%f ",&temp3);
                t2[i] = temp3;
        }
        loopi(hidden_nodes)
        {
                fscanf(fp,"%f ",&temp4);
                t1[i] = temp4;
        }
        fclose(fp);
}

/* Global variables for the neural network              */
float y_hidden[LAYER_ONE],y_out[OUTPUT_LAYER];
float vector[FEATURES],desired_output[OUTPUT_LAYER];
float w1[LAYER_ONE*FEATURES],w2[OUTPUT_LAYER*LAYER_ONE];
float feats[(VECTORS+BVECTORS+TVECTORS+TESTS)*FEATURES];
float classes[VECTORS+BVECTORS+TVECTORS+TESTS];
float t1[LAYER_ONE],t2[OUTPUT_LAYER];
float dzidxj;
float test_acc;
float eta;
int correct;


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

Convenient Macros for Perceptron Package

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

/*** MACROS ***/

#define MAX_PIC 128
#define Boolean int
#define FALSE 0
#define TRUE  1

/** Mask Definitions **/
#define OFF 0.0
#define ON  1.0

#define rloopi(A) for(i=(A)-1;i>=0;i--)
#define rloopj(A) for(j=(A)-1;j>=0;j--)
#define rloopk(A) for(k=(A)-1;k>=0;k--)
#define rloopl(A) for(l=(A)-1;l>=0;l--)
#define rloopm(A) for(m=(A)-1;m>=0;m--)
#define rloopn(A) for(n=(A)-1;n>=0;n--)
#define rloopp(A) for(p=(A)-1;p>=0;p--)
#define rloopij(A,B) for(i=(A)-1;i>=0;i--) for(j=(B)-1;j>=0;j--)
#define loopi(A) for(i=0;i<A;i++)
#define loopj(A) for(j=0;j<A;j++)
#define loopk(A) for(k=0;k<A;k++)
```

```
#define loopl(A) for(l=0;l<A;l++)
#define loopm(A) for(m=0;m<A;m++)
#define loopn(A) for(n=0;n<A;n++)
#define loopp(A) for(p=0;p<A;p++)
#define loopij(A,B) for(i=0;i<A;i++) for(j=0;j<B;j++)
#define MALLOC(A,B,C,D) if((A=(C *)malloc((B)*sizeof(C)))==NULL) { \
         fprintf(stderr, strcat(D,": insufficient memory\n") ); \
         exit(-1); }
#define CREATE_FILE(A,B,C) if((A=fopen(B,"w")) == NULL) { \
         printf(strcat(C,": can't open for writing - %s.\n", 24),B); \
         exit (-1); }
#define OPEN_FILE(A,B,C) if((A=fopen(B,"r")) == NULL) { \
         printf(strcat(C,": can't open for reading - %s.\n", 24),B); \
         exit (-1); }
#define idx(I,J,N) (I)*(N)+(J)
#define FEQ(A,B) ( ( ((A)-(B)) < 1E-6 ) ? 1 : 0 )
#define SQR(X) (X)*(X)

#define TRAIN    0
#define TEST     1

#define VECTORS 416 /* corresponds to number of views of an object */
#define BVECTORS 384 /* number of views of balloon */
#define TVECTORS 392  /* number of views of tank */
#define TESTS 15  /* number of test vectors */
#define FEATURES 120 /* number of Gabor coeffs, positions, and class */
#define OUTPUT_LAYER 3 /* number of output nodes */
#define LAYER_ONE 60 /* number of hidden nodes */
#define ETA .5  /* gain term */
#define ALPHA  .7 /* momentum term */


** The following codes were used to generate gabor filters and
   to normalize the feature vectors **

/**************************************************************************
   DATE:  18 Jun 90
   DESCRIPTIONS:  This function performs: 1) conversion of image file
   from RLE to double format, 2) gabor transform of the image, and
   3) reduction of gabor coefficients to top NUM_COEFF (defined in macros.h).

   FILES READ:  1.  image.rle - image file in RLE format
                2.  image.sum - summation of gabor transforms
   FILES WRITTEN:  1.  vectors - contains top NUM_COEFF gabor coefficients and
                       their respective positions.
```

```
  AUTHOR:  Phung Le
*************************************************************************/

#include <malloc.h>
#include <stdio.h>
#include "macros.h"
#include <math.h>

unsigned char *read_rle();
void transform();

main()
{

        FILE *fp1,*fp2,*fp3,*fp4;          /* file pointers */
        int i,j,ncolors,rows,cols,num;
        unsigned char *tempfile;
        double *dblfile;
        double temp;
        char filename[15];

        OPEN_FILE(fp1,"image.rle","image.rle");  /* get image */

        /* read rle file and return unsigned char file */

        tempfile = read_rle(fp1,&ncolors,&rows,&cols);
        MALLOC(dblfile,rows*cols,double,"dblfile");

        loopij(rows,cols)                      /* cast to double */
                dblfile[idx(i,j,cols)] = (double) tempfile[idx(i,j,cols)];
        fclose(fp1);

        CREATE_FILE(fp2,"image.dbl","image.dbl");

        loopij(rows,cols)           /* write new double file */
                fprintf(fp2,"%f\n",dblfile[idx(i,j,cols)]);
        fclose(fp2);
        transform();            /* performs gabor transform of image */


}

/*************************************************************************
```

```
 DATE:  8 Jun 90

 DESCRIPTION:  This function performs the Gabor transform of an image and
 arranges the transforms with each Gabor filter separately.

 FILES READ:  1.  gabordat1 - contains frequency, orientation, and width
                  of the Gaussian window
              2.  image.dbl - image file in 'double' format
 FILES WRITTEN: vectors

 AUTHOR:  Eric Fretheim
 HISTORY:  revised by Phung Le and Al L'Homme
***************************************************************************/

#include <stdio.h>
#include <math.h>
#include <string.h>
#include <ctype.h>
#include "macros.h"
#include "fft.c"


#define FORWARD 1
#define BACKWARD -1
#define PI 3.141592653589793

void indexx();

transform()
{
int low_freq,high_freq,freq_cnt;
int max_rot,rot_step,rot;
int width,length,number,count;
int i,j,k,filter;
int indx[MAX_PIC*MAX_PIC];
int x[NUM_COEFF],y[NUM_COEFF];
static int indx_file[NUM_COEFF*NUM_FILTERS];
double z[NUM_COEFF];
s    t    a    t    i    c              d    o    u    b    l    e
coeff[NUM_COEFF*NUM_FILTERS],pos[NUM_COEFF*NUM_FILTERS];
double e[MAX_PIC*MAX_PIC];
int temp_file[NUM_COEFF];

double **a;
double **b;
double **c;
double **d;
double *xtra;
double rtemp, itemp;

char file_name[15];

FILE *ofp,*gd;
j=0;
a=(double **)calloc(MAX_PIC,sizeof(xtra));
b=(double **)calloc(MAX_PIC,sizeof(xtra));
c=(double **)calloc(MAX_PIC,sizeof(xtra));
d=(double **)calloc(MAX_PIC,sizeof(xtra));
if(!a || !b || !c || !d) j=1;
for(i=0;i<MAX_PIC;i++)
{
        a[i]=(double *)calloc(MAX_PIC,sizeof(double));
        b[i]=(double *)calloc(MAX_PIC,sizeof(double));
        c[i]=(double *)calloc(MAX_PIC,sizeof(double));
        d[i]=(double *)calloc(MAX_PIC,sizeof(double));

        if(!a[i] || !b[i] || !c[i] || !d[i]) j=1;
}

if(j)
{
        printf("Error! Calloc can't allocate enough memory\n");
        exit();
}

/* open data file for reading */

OPEN_FILE(gd,"gabordat1","gabordat1");
fscanf(gd,"%d",&low_freq);
fscanf(gd,"%d",&high_freq);
fscanf(gd,"%d",&max_rot);
fscanf(gd,"%d",&rot_step);
fscanf(gd,"%d",&width);
fscanf(gd,"%d",&length);

gget_file(a,b,width,length);     /* get image file */
filter = 0;

for (freq_cnt = low_freq; freq_cnt < high_freq + 1; freq_cnt++)
{
```

```
for (rot = 0; rot <= max_rot; rot+=rot_step)
{
        printf("Calculating Freq %d, Rotation %d --- ",freq_cnt,rot);

        rotate(c,rot,width,length,freq_cnt);/*generate Gabor filter*/

        printf(" --");
        loopij(MAX_PIC,MAX_PIC)    /* initialize arrays    */
        {                          /* to zero */
                d[i][j] = 0;
                e[idx(i,j,MAX_PIC)] = 0;
        }

        printf(" transform --\n");

        fft2(c,d,MAX_PIC,FORWARD);   /* Fourier T of filter */
        loopij(MAX_PIC,MAX_PIC)
                                /* Image x (filter)* */
        {
                rtemp = c[i][j] * a[i][j] - b[i][j] * d[i][j];
                itemp = d[i][j] * a[i][j] + c[i][j] * b[i][j];
                c[i][j] = rtemp;
                d[i][j] = itemp;
        }

        fft2(c,d,MAX_PIC,BACKWARD);   /*  inverse FT to get */
                                      /* correlation results */

        loopij(MAX_PIC,MAX_PIC)               /* convert to + values */
                if (c[i][j] < 0) c[i][j] = -c[i][j];

        loopij(MAX_PIC,MAX_PIC)
                e[idx(i,j,MAX_PIC)] = c[i][j];

        indexx(MAX_PIC*MAX_PIC,e,indx);       /* sort coeff by magnitude */
        count=0;
        number=0;
        if (filter > 0)
        {
        while (count<NUM_COEFF)
        {
        loopi(NUM_COEFF*filter)  /* avoid redundant info */
        {
                while (indx_file[i] == indx[MAX_PIC*MAX_PIC-number])
                {
```

```
                number++;
                i=-1;
            }
        }
        temp_file[count] = indx[(MAX_PIC*MAX_PIC)-number];
        count++;
        number++;
        }/* end while */
        }/* end if */

        if (filter==0)
        {
        loopi(NUM_COEFF)   /* storing initials positions */
            temp_file[i] = indx[MAX_PIC*MAX_PIC-i];
        }

        loopi(NUM_COEFF)         /* find x,y values of indx */
        {
            y[i+1] = (int) temp_file[i]/128;
            x[i+1] = temp_file[i] % 128;
            if ((temp_file[i] % 128) == 0)
            {
                x[i+1] = 128;
                y[i+1] = y[i+1] - 1;
            }

        }
        loopi(NUM_COEFF)
        {
            /* normalize indx with respect to the position */
            /* of max Gabor coeff by Euclidean distance */
            z[i+1] =  sqrt((double) SQR(x[i+1] - x[1]) +
                          (double) SQR(y[i+1] - y[1]));
            z[1] += z[i+1];
        }

        loopi(NUM_COEFF)         /* save coeffs and positions */
        {
            indx_file[(NUM_COEFF*filter)+i] = temp_file[i];
            coeff[(NUM_COEFF*filter)+i] = e[temp_file[i]];
            pos[(NUM_COEFF*filter)+i] = z[i+1];
        }
        filter++;
    }
}
```

```
/* write out the file  */

printf("Ready to write ");

CREATE_FILE(ofp,"vectors","vectors");

loopj(NUM_FILTERS)
{
loopi(NUM_COEFF)
{

        fprintf(ofp,"%f  ", (float) coeff[(j*NUM_COEFF)+i]);
        fprintf(ofp,"%f ", (float) pos[(j*NUM_COEFF)+i]);
}
}
fprintf(ofp,"%d\n",1);
fclose(ofp);
printf("\nFile write complete\n");
return;
}

gget_file(real,imag,width,length)
double **real;
double **imag;
int width,length;
{
        double foo;
        int i,j;
        FILE *fp;

        fp = fopen("image.dbl", "r");

        printf("File opened -- reading ...\n");

        loopij(MAX_PIC,MAX_PIC)
        {
                fscanf(fp,"%lf",&foo);
                real[i][j] = foo;
                imag[i][j] = 0;
        }

        for (j = MAX_PIC - length; j < MAX_PIC; j++)
                for (i = MAX_PIC - width; i < MAX_PIC; i++)
                        real[i][j] = 0;
        printf("Done reading - transforming....\n");

        fclose(fp);

        fft2(real,imag,MAX_PIC,FORWARD);  /* FT of image */

        return;
}

rotate(gab_array,degrees,x_size,y_size,omega)
double **gab_array;
int degrees, x_size, y_size;
int omega;
{
        int i,j;
        int l,m;
        double theta;
        double x,y,tx,ty;
        double x_prime,y_prime;
        double sig_x, sig_y;
        double gtf();

        theta = (((double) degrees) /180)  * PI;

        sig_x = x_size/8;     /* covers +-4 standard devs */
        sig_y = y_size/8;

        loopij(MAX_PIC,MAX_PIC)
                gab_array[i][j] = 0;

        tx = x_size;
        ty = y_size;

        for (j = y_size/2; j < y_size + y_size/2; j++)
                for (i = x_size/2; i < x_size + x_size/2; i++)
                {
                        l = x_size - i;
                        if (l < 0) l += MAX_PIC;
                        m = y_size - j;
                        if (m < 0) m += MAX_PIC;
                        x = i; y = j;
                        /* coordinates transformation */
                        x_prime = x*cos(theta) - y*sin(theta)
                                - tx*cos(theta) + ty*sin(theta);
                        y_prime = x*sin(theta) + y*cos(theta)
```

```
                                        - tx*sin(theta) - ty*cos(theta);

                gab_array[l][m] = gtf(x_prime,y_prime,sig_x,sig_y,omega)*255;
                }
}

double gtf(x,y,sigma_x,sigma_y,cycles)
double x,y;
double sigma_x, sigma_y;
int cycles;
{
        double f_of_x;

        f_of_x = (1/(2*PI*sigma_x*sigma_y))
        *exp(-0.5*((x/sigma_x)*(x/sigma_x)+(y/sigma_y)*(y/sigma_y)));

        f_of_x *= sin((x*PI* (double) cycles)/(2*sigma_x));  /* sine Gabor */

        return (f_of_x);

}


/*  This function performs the fourier transform of the given image */

#include "macros.h"

fft2(picc,ipicc,n,dir)
double **picc;
double **ipicc;
int n;
int dir;

{

        double pic[MAX_PIC+1];
        double ipic[MAX_PIC+1];
        int i,j;


/*      printf("Start Fourier transform --- rows.\n"); */

        for (i = 0; i < n; i++)
                {
                for (j = 0; j < n; j++)
```

```
                        {
                        pic[j+1] = picc[i][j];
                        ipic[j+1] = ipicc[i][j];
                        }
                fft(pic,ipic,n,dir);

                for (j = 0; j < n; j++)
                        {
                        picc[i][j] = pic[j+1];
                        ipicc[i][j] = ipic[j+1];
                        }
                }


        for (j = 0; j < n; j++)
                {
                for (i = ); i < n; i++)
                        ,

                        pic[i+1] = picc[i][j];
                        ipic[i+1] = ipicc[i][j];
                        }
                fft(pic,ipic,n,dir);

                for (i = 0; i < n; i++)
                        {
                        picc[i][j] = pic[i+1];
                        ipicc[i][j] = ipic[i+1];
                        }
                }
        return;
}


fft(fr,fi,n,dir)
double *fr;
double *fi;
int n;
int dir;

{
        double tr=0, ti=0;
        double wr=0, wi=0;
        double el=0, a=0;
        int i=0, j=0;
        int mr=0;
int l=0;
int k=0;
int m=0, nn=0;
int step=0;


if (dir < 0 )
        for (i = 1; i < n+1; i++)
                fi[i] = -fi[i];
mr = 0;
nn = n - 1;
for (m = 1; m <= nn; m++)
        {
        l = n/2;
        while (l + mr > nn) l = l/2;
        mr = mr%l + l;
        if (mr > m )
                {
                tr = fr[m+1];
                fr[m+1] = fr[mr+1];
                fr[mr+1] = tr;

                ti = fi[m+1];
                fi[m+1] = fi[mr+1];
                fi[mr+1] = ti;
                }
        }
l = 1;
while (l < n)
        {
        step = 2*l;
        el = l;
        for (m = 1; m <= l; m++)
                {
                a = 3.1415926535 * (double) (1-m)/ el;
                wr = cos(a);
                wi = sin(a);
                for (i = m; i <= n; i += step)
                        {
                        j = i + l;
                        k = i;
                        tr = wr*fr[j] - wi*fi[j];
                        ti = wr*fi[j] + wi*fr[j];
                        fr[j] = fr[k] - tr;
                        fi[j] = fi[k] - ti;
                        fr[k] = fr[k] + tr;
                        fi[k] = fi[k] + ti;
                        }
                    }
                l = step;
                }

        if ( dir < 0)
                for (i = 1; i < n+1; i++)
                    {
                    fr[i] = fr[i]/n;
                    fi[i] = -fi[i]/n;
                    }

        return;
}


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

  rle.c: Routines to read and write RLE files.

Dennis W. Ruck, AFIT/ENG
DS-90D

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

#include <stdio.h>
#include <malloc.h>

#include <svfb_global.h>
#include "macros.h"

#define MAX_CHANNELS 4

unsigned char * read_rle();
void save_rle();

unsigned char * read_rle( image_file, ncolors, rows, cols )
FILE        *image_file;
int         *ncolors, *rows, *cols;
{
/**************************************************
 Input: image_file
```

```
  Output: image, ncolors, rows, cols;

  The routine will allocate memory for the image. It
  is assumed
  that the file has already been opened for reading.

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

  unsigned char      *image;
  rle_pixel         **scan;
  unsigned char      *pixdata;
  int                 i, j, k, lncolors, lrows, lcols, ro  , y, offset;
  struct sv_globals   globals;

  /** Read the header **/
  globals.svfb_fd = image_file;
  rle_get_setup( &globals );
  *rows = lrows = globals.sv_ymax-globals.sv_ymin+1;
  *cols = lcols = globals.sv_xmax-globals.sv_xmin+1;
  *ncolors = lncolors = globals.sv_ncolors;

#ifdef DEBUG
  fprintf(stderr, "read_rle: image has %d rows and %d columns\
 and %d channels\n", lrows, lcols, lncolors );
  fprintf(stderr,"read_rle: xmin = %d, ymin = %d, xmax = %d, ymax = %d\n",
    globals.sv_xmin, globals.sv_ymin, globals.sv_xmax,
    globals.sv_ymax);
#endif

  /** Allocate space for the image **/
  if((image=(unsigned char *)malloc(lrows*lcols*lncolors))==NULL){
    fprintf(stderr, "read_rle: insufficient memory, %d bytes requested\n",
            lrows*lcols*lncolors );
    exit(-1);
  }
  if(rle_row_alloc( &globals, &scan ) != 0){
    fprintf(stderr, "read_rle: insufficient memory\n");
    exit(-1);
  }
  /** read in the image **/
  while( (y=rle_getrow(&globals, scan)) <= globals.sv_ymax) {
    /** Transfer scanline information to image **/
    /** The returned image format has the channel information grouped in
        contiguous memory by row major format starting at the top of the
        image. Remember RLE returns information from the bottom of the
```

```
      image first.                                **/
    row  = globals.sv_ymax - y;
    loopk(lncolors) {
      offset = k*lrows*lcols;
      pixdata = scan[k];
      loopj(lcols) image[idx(row,j,lcols)+offset] = pixdata[j+globals.sv_xmin];
    }
  }

  return image;

}

void save_rle( image_file, image, ncolors, rows, cols )

FILE          *image_file;
unsigned char  *image;
int            ncolors, rows, cols;
{
/*****************************************************************

  save_rle: Routine to save image in  an RLE format file. It is
assumed the file has already been opened for writing.

*****************************************************************/
  rle_pixel        **scan, *srows[MAX_CHANNELS];
  int              i, j, k, offset;
  struct sv_globals   globals;

  /** Set up the header **/
  globals.svfb_fd = image_file;
  globals.sv_ymax = rows-1;
  globals.sv_ymin = 0;
  globals.sv_xmax = cols-1;
  globals.sv_xmin = 0;
  globals.sv_ncolors = ncolors;
  globals.sv_background = 0;
  globals.sv_comments = NULL;
  globals.sv_ncmap = 0;
  loopi(ncolors) SV_SET_BIT (globals, i);
  sv_setup ( RUN_DISPATCH, &globals);

  if(rle_row_alloc( &globals, &scan ) != 0){
    fprintf(stderr, "save_rle: insufficient memory\n");
    exit(-1);
```

```
 }

 /** save the image data **/
 loopi(rows) {
  /** Transfer image information to scan **/
  /** The source image format has the channel information grouped in
      contiguous memory by row major format starting at the top of the
      image. Remember RLE stores information from the bottom of the
      image first.                                          **/
  loopk(ncolors) {
   offset = k*rows*cols;
   srows[k] = &image[idx((rows-1)-i,0,cols)+offset];
  }
  sv_putrow( srows, cols, &globals );
 }

 /** put EOF mark **/
 sv_puteof( &globals );

}


/* This function sorts the data from the array 'arrin' in ascending order
   and provides the order in the array 'indx'                        */

void indexx(n,arrin,indx)
int n,indx[];
float arrin[];
{
        int l,j,ir,indxt,i;
        float q;

        for (j=1;j<=n;j++) indx[j]=j;
        if (n == 1) return;
        l=(n >> 1) + 1;
        ir=n;
        for (;;) {
                if (l > 1)
                        q=arrin[(indxt=indx[--l])];
                else {
                        q=arrin[(indxt=indx[ir])];
                        indx[ir]=indx[1];
                        if (--ir == 1) {
                                indx[1]=indxt;
                                return;
                        }
                }
                i=l;
                j=l << 1;
                while (j <= ir) {
                        if (j < ir && arrin[indx[j]] < arrin[indx[j+1]]) j++;
                        if (q < arrin[indx[j]]) {
                                indx[i]=indx[j];
                                j+= (i=j);
                        }
                        else j=ir+1;
                }
                indx[i]=indxt;
        }
}


/* This function normalizes the data vectors for further processing */


#include <stdio.h>
#include "macros.h"
#include <math.h>

void distribution(float[],float[],float [],int);

main()
{
     FILE *fp,*fp1;
     int i,j,c;
     char filename[20];
     float f;
     float mean[FEATURES],cmean[FEATURES],bmean[FEATURES],
                              tmean[FEATURES];
     float std_dev[FEATURES],cstd_dev[FEATURES],bstd_dev[FEATURES],
                              tstd_dev[FEATURES];
     float x_cube[VECTORS*FEATURES],x_bulb[BVECTORS*FEATURES],
               x_tank[TVECTORS*FEATURES],x[VECTORS*FEATURES];
     float feats[FEATURES],classes[VECTORS];
     float m[FEATURES],s[FEATURES];


     loopi(VECTORS)
     {
          sprintf(filename,"../data/vectors.%d",i+1);
```

```
    OPEN_FILE(fp,filename,filename);  /* open vector files for */
                                      /* reading */
     loopj(FEATURES)
     {
         fscanf(fp,"%f ",&f);
         x_cube[idx(i,j,FEATURES)] = f;
     }
     fclose(fp);
}
distribution(m,s,x_cube,VECTORS);  /* get mean & std dev */
loopj(FEATURES)
{
      cmean[j] = m[j];
      cstd_dev[j] = s[j];
}
loopi(BVECTORS)
{
     sprintf(filename,"../bdata/bvectors.%d",i+1);
    OPEN_FILE(fp,filename,filename);  /* open vector files for */
                                      /* reading */
     loopj(FEATURES)
     {
         fscanf(fp,"%f ",&f);
         x_bulb[idx(i,j,FEATURES)] = f;
     }
     fclose(fp);

}
distribution(m,s,x_bulb,BVECTORS);   /* get mean & std dev */
loopj(FEATURES)
{
     bmean[j] = m[j];
     bstd_dev[j] = s[j];
}
loopi(TVECTORS)
{
     sprintf(filename,"../tdata/tvectors.%d",i+1);
    OPEN_FILE(fp,filename,filename);  /* open vector files for */
                                      /* reading */
     loopj(FEATURES)
     {
         fscanf(fp,"%f",&f);
         x_tank[idx(i,j,FEATURES)] = f;
     }
     fclose(fp);
```

```
}
distribution(m,s,x_tank,TVECTORS);  /* get mean & std dev */
loopj(FEATURES)
{
    tmean[j] = m[j];
    tstd_dev[j] = s[j];
}

loopj(FEATURES)        /* sum all means and std_devs */
{
    mean[j] = cmean[j] + bmean[j] + tmean[j];
    std_dev[j] = cstd_dev[j] + bstd_dev[j] + tstd_dev[j];

}

CREATE_FILE(fp,"mean.dat","mean.dat");
CREATE_FILE(fp1,"std_dev.dat","std_dev.dat");
loopj(FEATURES)        /* save means and std_devs */
{
    fprintf(fp,"%f ",mean[j]);
    fprintf(fp1,"%f ",std_dev[j]);
}
fclose(fp);
fclose(fp1);



loopi(VECTORS)         /* normalize all data by average max value */
{
    sprintf(filename,"../normdata/vectors.%d",i+1);
    OPEN_FILE(fp,filename,filename); /* open vector files for */
                                     /* reading */
    loopj(FEATURES)         /* storing features */
    {
        fscanf(fp,"%f ",&f);
        feats[j] = f;
    }
    fscanf(fp,"%d",&c);              /* storing classes */
    classes[i] = (float) c;
   fclose(fp);

    loopj(FEATURES)         /* calculate new values */
        feats[j] = (feats[j] - mean[j])/std_dev[j];
    CREATE_FILE(fp,filename,filename)  /* open file for writing */
    loopj(FEATURES)
        fprintf(fp,"%f ",feats[j]);
    fprintf(fp,"%f\n",classes[i]);
    fclose(fp);
}
loopi(BVECTORS)      /* normalize all data by average max value */
{
    sprintf(filename,"../normbdata/bvectors.%d",i+1);
    OPEN_FILE(fp,filename,filename);  /* open vector files for */
                                      /* reading */
    loopj(FEATURES)          /* storing features */
    {
         fscanf(fp,"%f ",&f);
         feats[j] = f;
    }
    fscanf(fp,"%d",&c);                /* storing classes */
    classes[i] = (float) c;
   fclose(fp);

    loopj(FEATURES)          /* calculate new values */
        feats[j] = (feats[j] - mean[j])/std_dev[j];
    CREATE_FILE(fp,filename,filename)   /* open file for writing */
    loopj(FEATURES)
        fprintf(fp,"%f ",feats[j]);
    fprintf(fp,"%f\n",classes[i]);
    fclose(fp);
}
loopi(TVECTORS)      /* normalize all data by average max value */
{
    sprintf(filename,"../normtdata/tvectors.%d",i+1);
    OPEN_FILE(fp,filename,filename);  /* open vector files for */
                                      /* reading */
    loopj(FEATURES)          /* storing features */
    {
         fscanf(fp,"%f ",&f);
         feats[j] = f;
    }
    fscanf(fp,"%d",&c);                /* storing classes */
    classes[i] = (float) c;
   fclose(fp);

    loopj(FEATURES)          /* calculate new values */
        feats[j] = (feats[j] - mean[j])/std_dev[j];
    CREATE_FILE(fp,filename,filename)   /* open file for writing */
    loopj(FEATURES)
```

```
                fprintf(fp,"%f ",feats[j]);
            fprintf(fp,"%f\n",classes[i]);
            fclose(fp);
        }
}

void distribution(float m[],float s[],float values[],int size)
{
        int i,j;
        float sum[FEATURES],sum1[FEATURES];
        double var[FEATURES];

        loopj(FEATURES)             /* initialize values */
        {
                sum[j] = 0;
                sum1[j] = 0;
        }
        loopj(FEATURES)             /* calculate mean */
        {
                loopi(size)
                        sum[j] += values[idx(i,j,FEATURES)];
                m[j] = sum[j]/(VECTORS+BVECTORS+TVECTORS);

        }

        loopj(FEATURES)             /* calculate standard deviation */
        {
                loopi(size)
                        sum1[j] += SQR(values[idx(i,j,FEATURES)] - m[j]);
                var[j] = sum1[j]/((VECTORS+BVECTORS+TVECTORS) - 1);
                if (var[j] == 0) s[j] = 1;
                else s[j] = (float) sqrt(var[j]);
        }

}
```

Bibliography

1. Abu-Mustafa, S. Yaser, and Demetri Psaltis. "Recognitive Aspects of Moment Invariants," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol PAMI-6, No 6: 698-706 (November 1984).

2. Ayer, Capt Kevin W. Gabor Transforms for Forward Looking Infrared Image Segmentation. MS Thesis, AFIT/GEO/ENG/89D-1. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1989 (AD-A215046).

3. Casasent, David. "Scene Analysis Research: Optical Pattern Recognition and Artificial Intelligence," SPIE Optical and Hybrid Computing, Vol 634: 439-56 1986.

4. Casasent, David and Suzanne A. Liebowitz. "Model-based Knowledge-based Optical Processors," Applied Optics, Vol 26, No 10: 1935-42 (15 May 87).

5. Casasent, David and Demetri Psaltis. "New Optical Transforms for Pattern Recognition," Proceedings to the IEEE, Vol 65, No 1: 77-84 (January 1977).

6. Childress, Capt Timothy G. and J. Thomas Walrond. Position, Scale, and Rotation Invariant Optical Pattern Recognition for Target Extraction and Identification. MS Thesis, AFIT/GE/ENG/88D-4. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1988 (AD-A202600).

7. Cline, Capt John D. Hybrid Optical/Digital Architecture for Distortion Invariant Pattern Recognition. MS Thesis, AFIT/GEO/ENG/89D-2. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1989 (AD-A215628).

8. Dahn, Capt David A. "GPR User's Manual," Air Force Institute of Technology (AU), Wright-Patterson AFB OH, (14 July 1990).

9. Daugman, John G. "Complete Discrete 2-D Gabor Transforms by Neural Networks for Image Analysis and Compression," IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol 36, No 7: 1169-79 (July 1988).

10. Dudani, Sahibsingh A., Kenneth J. Breeding, and Robert B. McGhee. "Aircraft Identification by Moment Invariants," IEEE Transactions on Computers, Vol C-26, No 1: 39-45 (January 1977).

11. Flaton, Kenneth A. and Scott T. Toborg. "An Approach to Image Recognition using Sparse Filter Graphs," 1989 IEEE INNS International Joint Conference on Neural Networks, Vol I: 313-20 1989.

12. Gizzi, Martin S., Ephrain Katz, Robert A. Schumer, and J. Anthony Movshon. "Selectivity for Orientation and Direction of Motion of Single Neurons in Cat Striate and Extrastriate Visual Cortex," Journal of Neurophysiology, Vol 63, No 6: 1529-43 (June 1990).

13. Gustafson, Steven C., David L. Flannery, and Darren M. Simon. "Neural Networks with Optical Correlation Inputs for Recognizing Rotated Targets." University of Dayton, Research Institute, Dayton OH, 1990.

14. Horev, Moshe. Picture Correlation Model for Automatic Machine Recognition. MS Thesis, AFIT/GE/EE/80D-25. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1980 (AD-A100765).

15. Hu, Ming-Kuei. "Visual Pattern Recognition by Moment Invariants," IRE Transactions on Information Theory, IT-8: 179-87 (February 1962).

16. Hubel, David H. and Torsten N. Wiesel. "Brain Mechanisms of Vision," Scientific American, Vol 241, No 3: 150-163 (1979).

17. Kabrisky, Matthew. Personal interviews. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, October 1990.

18. Kabrisky, Matthew. Lecture notes taken in EENG 620, Pattern Recognition I. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 1990.

19. Khotanzad, A., J. H. Lu, and M. D. Srinath. "Target Detection using a Neural Network-based Passive Sonar System," IEEE INNS International Joint Conference on Neural Networks, Vol I: 335-40 1989.

20. Kobel, Capt William G. and Timothy Martin. Distortion Invariant Pattern Recognition in Non-Random Noise. MS Thesis, AFIT/GE/ENG/86D-20. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1986 (AD-A177598).

21. Lippman, Richard P. "An Introduction to Computing with Neural Nets," IEEE ASSP Magazine: 4-22 (April 1987).

22. Manual pages on Sun 4's pertaining to "gpr".

23. Mayo, 2Lt Michael W. Computer Generated Holograms and Magneto-Optic Spatial Light Modulator for Optical Pattern Recognition. MS Thesis, AFIT/GEO/ENG/87D-1. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1987 (AD-A189553).

24. Mueller, Michael, Erik J. Fretheim, Matthew Kabrisky, and Steven K. Rogers. "Gabor Transforms to Preprocess Video Images for Back-Propagation,"

25. Robinson, Capt Vicky L. Hybrid Calculation of Invariant Moments. MS Thesis, AFIT/GEO/ENG/88D-3. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1988 (AD-A202594).

26. Rogers, Maj Steven K. and Matthew Kabrisky. Biological and Artificial Neural Networks for Pattern Recognition. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 1990.

27. Rogers, Maj Steven K. Lecture notes taken in EENG 527, Fourier Optics. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 1989.

28. Ruck, Capt Dennis W. Characterization of MLP and their Application to Multi-sensor Automatic Target Detection. PhD Disertation, AFIT/GE/ENG/DS-90D. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1990.

29. Ruck, Capt Dennis W., Steven K. Rogers, and Matthew Kabrisky. "Feature Selection Using a Multilayer Perceptron," Submitted for publication in The Journal of Neural Network Computing.

30. Ruck, Capt Dennis W. Multisensor Target Detection and Classification. MS Thesis, AFIT/GE/ENG/87D-56. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1987 (AD-A177598).

31. Schils, George F. and Donald W. Sweeney. "Optical Processor for Recognition of Three-Dimensional Targets Viewed from any Direction," Journal of the Optical Society of America A, Vol 5, No 8: 1309-21 (August 1988).

32. Tarr, Capt Gregory L. Dynamic Analysis of Feedforward Neural Networks Using Simulated and Measured Data. MS Thesis, AFIT/GE/ENG/88D-54. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1988 (AD-A202573).

33. Teague, Michael R. "Image Analysis via the General Theory of Moments," Journal of the Optical Society of America, Vol 70, No 8: 920-30 (August 1980).

34. Troxel, 1Lt Steven E. Position, Scale, and Rotation Invariant Target Recognition using Range Imagery. MS Thesis, AFIT/GE/ENG/87D-3. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1987 (AD-A188828).

35. Dahn, Capt David A. Personal interview. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, September 1990.

Vita

Captain Phung D. Le was born on 2 Feb 1964 in Da Nang, Viet Nam. He immigrated to the United States in the summer of 1975 and lived in Illinois then California where he graduated from Homestead high school in 1982. In June 1982, he entered the United States Air Force Academy and received his commission in May 1986. Upon graduation, he was assigned to the Ballistic Missile Office, Norton AFB CA, where he was a Guidance and Control Project Officer. While there, he obtained a Master of Science in Systems Management at the University of Southern California.

REPORT DOCUMENTATION PAGE

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE December 1990	3. REPORT TYPE AND DATES COVERED Final

4. TITLE AND SUBTITLE
Model-Based 3-D Recognition System Using Gabor Features and Neural Networks

5. FUNDING NUMBERS

6. AUTHOR(S)
Phung D. Le, Capt, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
School of Engineering
Air Force Institute of Technology (AFIT)
Wright-Patterson AFB, OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER

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

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; Distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

A different approach to pattern recognition was attempted using Gabor features, artificial neural nets, and an image generator. The Gabor features and artificial neural nets are sound biological-based, and the image generator provides complete access to any view of an object. This research tested the idea that their integration could form a robust 3-D recognition system.
The results of the research showed that the Gabor features together with a neural net were used successfully in classifying objects regardless of their positions, out-of-plane rotations, and to a lesser extent in-plane rotations. The Gabor features were obtained by correlating the image with Gabor filters of varying orientations spaced 15 degrees apart as found in primates' visual systems, and the correlation with each filter was kept separately.

14. SUBJECT TERMS
Pattern recognition, Gabor, Neural networks, Models

15. NUMBER OF PAGES 98

16. PRICE CODE

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