ELECTRICAL ENGINEERING

OBJECT RECOGNITION USING FUZZY MEMBERSHIP FUNCTION RULES

JONATHAN EDWARD HUNTER

Thesis under the direction of Professor Mitch Wilkes

Object recognition and learning algorithms are huge areas of robotics research with many different methods in use by various researchers. A common result of using complex recognition methods is the loss of meaning (for humans) in the subsequent processing of the data. When programs incorrectly identify objects, the reason why is often lost in the data analysis. If the researchers can understand what the robot sees, they are better able to develop a system that has limited image understanding. Fuzzy models in object recognition are one of the better methods for achieving such a learning system.

Our desire is to develop a system that is quickly and easily trained, a system that can relate the decision of objects through the feature vector (vector of measured characteristics about the object), and a system that is relatively simple in its calculation of results.

The research was applied in conjunction with an experiment done by the Psychology Department. This system was applied to recorded videos of tasks done by their subjects. The system proves to be quite effective in object recognition and provides many options for more advanced data processing.

Approved_____Date_____

OBJECT RECOGNITION USING FUZZY MEMBERSHIP FUNCTION RULES

By

Jonathan Edward Hunter

Thesis
Submitted to the Faculty of the
Graduate School of Vanderbilt University
in partial fulfillment of the requirements
for the degree of
MASTER OF SCIENCE
in
Electrical Engineering
May, 2005
Nashville, Tennessee

Approved:
Professor Mitch Wilkes
Professer Alan Peters II

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

First, I want to thank God for the blessings. I would like to thank the Vanderbilt University Electrical Engineering Department for the financial support. I would like to thank Dr. Mitch Wilkes for his ample patience and guidance throughout my graduate career. I would like to thank Dr. Cordelia Brown for her guidance, supportive words, helpful planning, useful advice and friendship throughout my time at Vanderbilt. I also thank Dr. Alan Peters for his guidance through this thesis. I want to thank my parents (James and Carol Hunter) and brothers (Jaime and Jason Hunter) for their love and support throughout my life. I also want to thank my college friends both in and outside CIS for being there when I needed them.

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Object recognition learning algorithms that are easily trained, accurate, and capable of relating decisions to the feature vector are difficult to create. Fuzzy membership rules provide a possible solution. Object recognition and learning algorithms are huge areas of robotics research with many different methods in use by various researchers. Object recognition is identifying objects using stored data representations of the objects [16]. Some object recognition algorithms rely on complete three-dimensional information about objects to identify them. Others require less information about the object but use complex detection systems. For these object recognition methods, it is difficult to establish a learning algorithm. Learning algorithms are adaptive systems that use data arrays to establish and adjust what is characteristic about objects [17]. For learning systems, it is not reasonable to assume that there exist three-dimensional models of every object in the robot's environment, especially with objects that do not have exactly the same shape (e.g., trees). A common result of using complex recognition methods is the loss of meaning (for humans) in the subsequent processing of the data. When programs incorrectly identify objects, the reason why is often lost in the data analysis. This loss prevents researchers from helping to understand what the robot sees and improving the recognition process. If the researchers can understand what the robot sees, they are better able to develop a system that has limited image understanding. Fuzzy models in object recognition are one of the better methods for achieving such a learning system.

The development of a system that uses fuzzy rules to deduce general objects is the method we have chosen to examine. Our desire is to develop a system that is quickly and easily trained, a system that can relate the decision of objects through the feature vector (vector of measured characteristics about the object), and a system that is relatively simple in its calculation of results. The system should be robust which is why its complexity lies in the amount of features in the vector rather than the computation method itself. This reason is also why the system should be quickly and easily trained so the robot can be transported to different environments and can quickly learn the objects of the new environment. The ability for an algorithm to relay the decision it makes in a understandable fashion will aid in the selection of

future features that may be added. The ease of adding features to the feature vector makes the system very flexible. The Psychology Department's research can be used to show these characteristics. They are studying the motion of subjects demonstrating set tasks for different types of audiences.

CHAPTER II

BACKGROUND

A major component of the feature vector is the color histogram. The fuzzy sets used to analyze the images are very important. Both need background information to be explained to make clear the future information.

Color

All the images and videos are taken from a camcorder that stores each frame as a RGB (red green blue) color image. Each image consists of its height, width, and three bands for each of the primary colors. For each coordinate in the image, three numbers exist. The first number represents the intensity of red, the second represents green, and the third represents blue. The RGB color model is an additive model, which means by adding the primary colors in various amounts (intensities) all other colors of the spectrum can be created. When the primary colors are added together at maximum intensity, the color created is white as shown in Figure 1.

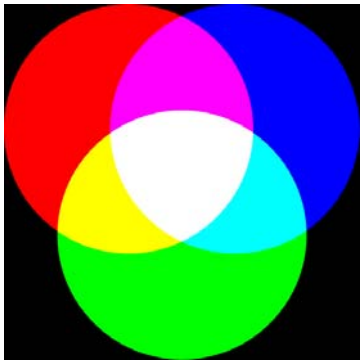

Figure 1: Additive RGB color model [9]

The images are then converted to the HSV (hue saturation value) color space for use also. Hue is the color. The range of the colors is normalized from 0 to 1 in the latter chapters. As you cycle through the hue values starting at red, it continues through yellow, green, cyan, blue, magenta, and back to red. The saturation is considered to be the vibrancy of the color. It represents the purity of a color like the "redness" of red. The less saturation in a color the more

pale it looks (washed out). The value is the brightness of the color. Figure 2 is a pictorial representation of the HSV color space.

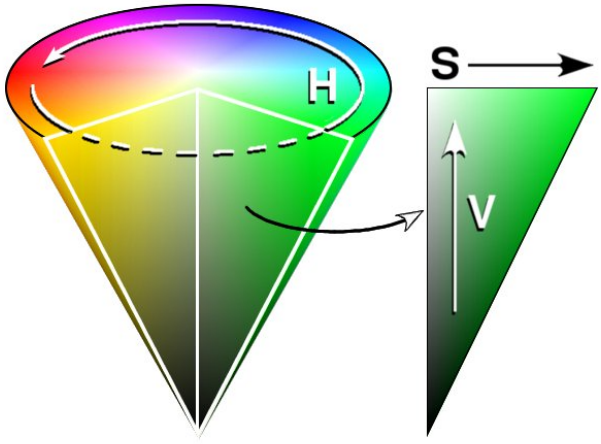

Figure 2: HSV Color Space [10]

Assuming the RGB values are all ranged from 0 to 1, the conversion to HSV is done in the following manner:

Let *MAX* equal the maximum of the (*R*, *G*, *B*) values, and *MIN* equal the minimum of those values. The formula can then be written as:

$$H = \begin{cases} \left(0 + \frac{G-B}{MAX-MIN}\right) \times 60, & \text{if } R = MAX \\ \left(2 + \frac{B-R}{MAX-MIN}\right) \times 60, & \text{if } G = MAX \\ \left(4 + \frac{R-G}{MAX-MIN}\right) \times 60, & \text{if } B = MAX \end{cases}$$

$$S = \frac{MAX - MIN}{MAX}$$

$$V = MAX$$

Note also that the formulas given here reflect some other properties of HSV:

- **If *MAX* = *MIN* (i.e. *S* = 0), then *H* is undefined.** This is apparent, considering the diagrams of HSV space above. If *S* = 0 then the color lies along the central line of grays, so naturally it has no hue, and the angular coordinate is meaningless.

- **If *MAX* = 0 (i.e. if *V* = 0), then *S* is undefined.** This is best illustrated in the conical diagram above. If *V* = 0, then the color is pure black, so naturally it has neither hue, nor saturation. Thus the conical diagram collapses to a single point and both the angle and radius coordinates are meaningless at that point.[10]

After the formulas, the hue values range from 0 to 360 and the saturation and value values range from 0 to 1. The hue values are normalized from 0 to 1 in Matlab's rgb2hsv function.

Fuzzy information

Fuzzy information is an imprecise set of descriptions or instructions. The fuzziness of information varies given the description [12,18]. "The object is close" is less precise than "The object is that is about 5 feet away." Fuzzy sets are developed to establish a set of values that meet a certain property to varying amounts. A fuzzy set (F) of "the object is about 20 feet away" would contain a set of distances that are "about 20 feet". Membership functions (m_F) map the fuzzy set to an interval of 0 to 1. Figure 3 shows an example of a fuzzy membership plot. The membership function contains a value (grade of membership) for each distance in the fuzzy set. If the numbers 18,19,20,21,and 22 were in the fuzzy set for "about 20 feet", their grade would be in the range of 0 to 1. The closer to 1 the grade is, the more the distance is "about 20 feet". The grades of membership that are given come from data and people. Membership functions that are measured from data have some calculation using the data to obtain the grade. Others use people's opinions to determine the "closeness" of a value to the given fuzzy information.

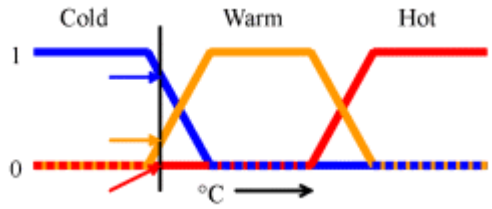

Figure 3: Fuzzy membership plot example [15]

The standard fuzzy operations for membership functions are as follows [12]:

(=) *Equality* $\mathrm{A} = \mathrm{B} \Leftrightarrow \mathrm{m}_{\mathrm{A}}(x) = \mathrm{m}_{\mathrm{B}}(x)$ (1)

($\subset$) *Containment* $\mathrm{A} \subset \mathrm{B} \Leftrightarrow \mathrm{m}_{\mathrm{A}}(x) \le \mathrm{m}_{\mathrm{B}}(x)$ (2)

(~) *Complement* $\mathrm{m}_{\bar{\mathrm{A}}}(x) = 1 - \mathrm{m}_{\mathrm{A}}(x)$ (3)

($\cap$) *Intersection* $\mathrm{m}_{A \cap B}(x) = \min\{\mathrm{m}_{\mathrm{A}}(x), \mathrm{m}_{\mathrm{B}}(x)\}$ (4)

($\cup$) *Union* $\mathrm{m}_{A \cup B}(x) = \max\{\mathrm{m}_{\mathrm{A}}(x), \mathrm{m}_{\mathrm{B}}(x)\}$ (5)

These are used to combine membership functions with each other. For this system's rule processing, it uses classification trees to determine the minterms for its rule. Since the rule processing was done in this manner, it was decided that using probability axioms and Bayes' intersection rule was necessary for the best results (discussed later). The operations used are [11,13,14]:

$P(A \cup B) = P(A) + P(B) - P(A \cap B)$ union (6)

$\overline{P(E)} = 1 - P(E)$ complement (7)

$P(A \cap B) = P(A) \cdot P(B)$ Bayes' intersection (8)

There has been much research done in color segmentation [2,3,4,7,8]. The use of fuzzy rules in image analysis seems to be most simular with the use of seed points. The seed points are points used to be a basis for pixels in the image color analysis. For the research done by Chamorro-Martinez et al., the seed points also provide an approximate edge to the object in the image [1]. The color features are fuzzified and the position of the seeds are stored. The algorithm uses a cost function that is determined by the distance of one seed point to the next consecutive seed point. By using the fuzzy data as a membership function, paths that go through a different region than what the seed point represents have high costs. By minimizing the cost, the program can segment the pictures.

CHAPTER III

LEARNING FUZZY MEMBERSHIP FUNCTIONS

In this chapter, the processing of information to provide data sets and method of extracting fuzzy rule sets are discussed. The process for this object recognition algorithm consists of training the system on certain objects, processing these objects to determine rule trees, extracting the rules and studying the effectiveness of the rules. This area focuses on training and analyzing the rules. All programming is done in MATLAB which uses ".m" files.

Feature Extraction

During the training step, sections of images are selected and given a label (class) according to what the object is. This is accomplished using the script file `trainingthevideo.m`. The file creates a structure array called Imemory and allows the user to provide the objects to be trained. In the `addtoImemory` function, the name of the picture/ frame of video, upper left coordinate of the selection rectangle, the width of the rectangle, the height of the rectangle, and the class of the selected object are added to the Imemory structure, as shown in Figure 4. These areas are processed to extract the features of each object. The function used is called `getfeaturesfromImemory`. This function takes the name of the picture and selected coordinates and extracts the features of these areas. The feature vector structure was originally 433 characteristics. As testing continued, some of the features were removed because of processing costs.

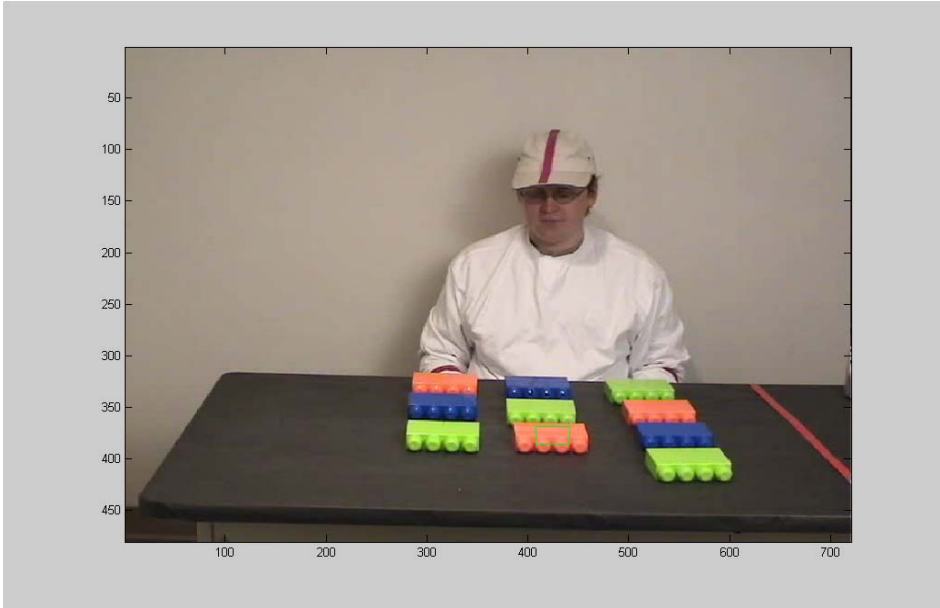

Figure 4: Acquiring a selection region

The first 250 of the 433 features are a histogram (or equivalently a probability density function, pdf) of color measurements for each training object. The image was converted from its original RGB format to HSV. The hue is broken into 10 bins, [0.00 0.05 0.14 0.22 0.28 0.45 0.54 0.75 0.81 0.92 1.00]. Bin 1 (orange with a bit of red) is 0.00 to 0.05, bin 2 (yellow) is 0.05 to 0.14, bin 3 (yellow-green) is 0.14 to 0.22, bin 4 (green) is 0.22 to 0.28, bin 5 (blue-green) is 0.28 to 0.45, bin 6 (blue) is 0.45 to 0.54, bin 7 (blue-violet) is 0.54 to 0.75, bin 8 (purple) is 0.75 to 0.81, bin 9 (red-violet) is 0.81 to 0.92, and bin 10 (red) is 0.92 to 1.00. The saturations and values are evenly distributed in to 5 bins each ranging from 0.00 to 1.00. The bin values are [0.00 0.20 0.40 0.60 0.80 1.00]. Each color is represented by one bin for each of the three bands. All possibilities of one bin selected from every band equals 250 different color features. The program looks at each color feature and finds the number of pixels that are defined in that feature. After doing this for all the color features, there are 250 numbers, each representing the number of pixels of a certain color that is in the selection region. The total number of pixels in the selection region divides these 250 numbers.

The next feature is a standard texture measurement. This feature measures the "roughness" of the area. The texture is measured from the grayscale of the image. The grayscale image is then filtered with a simple 3 by 3 Laplacian edge detector. Since there are possible negative values, the absolute value is taken of the normalized value (divided by 256

because of the 256 gray levels). The sum normalized across the entire selection region (divided by the number of pixels in the selection region) is the texture measure. The next feature is a texture measurement based on the Canny edge detector. This feature was not used in any experiments discussed in this work.

A motion feature was added to replace the Canny texture feature for some experiments. The motion feature used video feed to determine the amount of movement in an object. This feature also uses the 256 level grayscale image. The motion was calculated by taking the absolute value of the difference of consecutive frames in the selection region. The difference matrix now contains numbers from 0 to 256. The matrix is normalized from 0 to 1 and is summed to give the motion feature measure.

The next feature was a phase symmetry feature. This feature measures how symmetrical the graylevels in the region are. This feature was not used in any experiments discussed in this work.

The final 180 were line features. This feature measures how much of the selection area contains lines at varying degrees. By using the radon function in MATLAB over the selection region, each degree for 180 degrees for each pixel is calculated. The maximum value for each row is taken and normalized over the selection region. These features were removed eventually, but some experiments were preformed using this feature. All the features are shown in Figure 5.

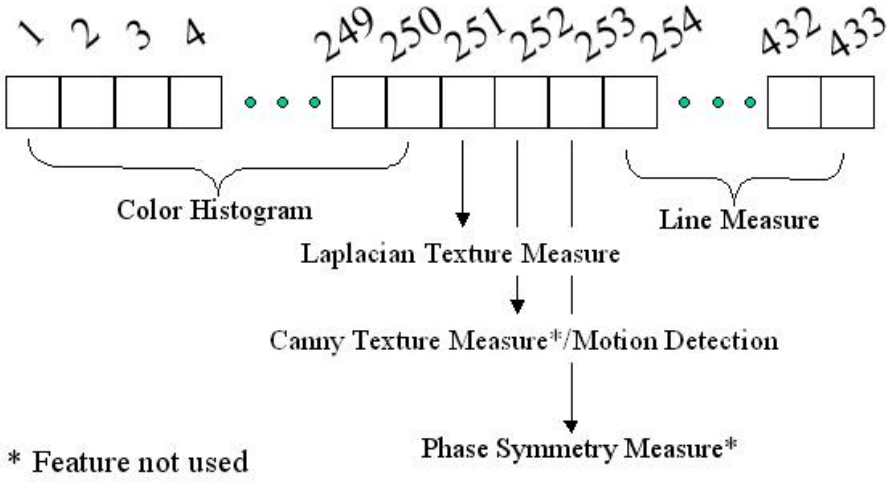

Figure 5: Feature Vector

Creating the Classification Tree

After all of the samples from the training objects are taken, the program now has a matrix of features for each sample and an array of class labels. The matrix rows and the label array's index correspond to each other (as shown below).

$$FV = \begin{bmatrix} n_{1,1} & n_{1,2} & n_{1,3} & n_{1,4} & n_{1,5} & n_{1,6} & ..... \\ n_{2,1} & n_{2,2} & n_{2,3} & n_{2,4} & n_{2,5} & n_{2,6} & \\ n_{3,1} & n_{3,2} & n_{3,3} & n_{3,4} & n_{3,5} & n_{3,6} & \\ n_{4,1} & n_{4,2} & n_{4,3} & n_{4,4} & n_{4,5} & n_{4,6} & \\ . & & & & & & \\ . & & & & & & \\ . & & & & & & \end{bmatrix} \qquad L = \begin{bmatrix} class_1 \\ class_2 \\ class_3 \\ class_4 \\ . \\ . \\ . \end{bmatrix}$$

feature matrix (a x 433) class vector (a x 1)

a = the total number of training samples for all objects

Note: Each coordinate, n_{sf}, contains the measure of feature number (f) in the sample number (s).

Both of these are used by the MATLAB provided `treefit` function to create a classification tree. Using these arrays, the function returns a structure array containing all of the information about the created tree. For the classification process, you need four main items: sample set, splitting criteria, stopping criteria and pruning. These are all parameters that can be set in `treefit`, but we use the default settings. All the samples together are the parent node. The most discriminatory feature is used to split the parent node into two child nodes. The process is continued to create several more child nodes using the splitting function (Gini's index of diversity was used as the splitting criterion).

The splitting criterions are a measure of impurities [9]. Impurities are the amount of indecisiveness in a probability function. Lets say event t has 2 possibilities, higher or lower than threshold, in Node q are possible. If there were no impurity, the $p_{t/q}$ (probability of t given q) would either be 1 or 0. For most situations, there exists an impurity. The maximum impurity would be $p_{t/q}$ would be 0.5. This indicates the maximum amount of indecisiveness. The probability that event t is higher or lower than the threshold given Node q are equally likely. This is a negative condition when considering using Node q to divide samples of object 1 and object 2. The Gini improvement measure uses the Gini impurity function to find the optimal

(causes the least amount of impurity) Node j (parent node) to split a collection of samples. The Gini diversity index is:

$$\text{Diversity index} = 1 - \sum p_{i/j}^{2} = 2p_{1/j}(1 - p_{1/j}). \quad (10)$$

This finds the impurity for Node j.

Then, the impurity of the two new groups of samples (child nodes) created from Node j is calculated using the Gini diversity index. The weighted averages of impurities for the child nodes are calculated.

$$\text{Weighted avg. of impurity} = [(p1)(\text{diversity index1})] + [(p2)(\text{diversity index2})] \quad (11)$$

The impurity is considered weighted because the amount of the parent node in child 1 (p1) and the amount of parent node in child 2 (p2) are taken into account. The final step is to compare the impurity of the parent node with the impurities of the child nodes to receive the improvement measure.

$$\text{Improvement measure} = \text{impurity(parent node)} - \text{weighted impurity(child nodes)} \quad (12)$$

Matlab finds all nodes that contain more samples than splitmin (discussed later) and are impure (having more than one class of samples in the node). On these nodes, the feature that gives the best improvement measure is chosen and the samples are split.

The stopping criteria that treefit uses is a coefficient called splitmin. This number, with default value of 10, determines the minimum number of samples that must be present in a node to able to be split again. This limits the size of the classification tree.

The pruning of a classification tree cuts branches to the most optimal with the least size difference in the tree. The branches are the splits that occur in the nodes when a tree is created. The branches lead to other nodes that can be split or to nodes that cannot be split (leaves). The treeprune function begins by finding all of the branches that split into two leaves (twigs) and calculates the cost of the leaves. Cost is a measure of misclassification. Unless the matrix is pre-defined, the cost of each misclassification in each node is 1. It also calculates the cost of the entire tree. It systematically combines twigs that decrease the overall cost of the tree. The function works to minimize

$$\text{Cost} + \text{alpha}*\text{treesize} \quad (13)$$

$$\text{alpha} = \max(0, \text{nodecosts} - \text{sumbranchcost}) / \max(\text{eps}, \text{numnodes})$$

$$\text{cost} = \begin{bmatrix} 0 & 1 & 1 & 1 & \cdots \\ 1 & 0 & 1 & 1 & \\ 1 & 1 & 0 & 1 & \\ 1 & 1 & 1 & 0 & \\ \vdots & & & & \end{bmatrix} \text{ if no previous cost is entered}$$

cost (n x n)

n = the number of classes

nodecosts = the array of costs for each node

sumbranchcost = the sum of all the costs for the branches

numnodes = number of nodes

Note: the max functions in the numerator and denominator are used to ensure the alpha is never negative or undefined.

treesize = the number of branches and nodes.

The function continues to prune until all optimal improvement measures (taken from the minimized alpha) are less than the error costs of the prune.

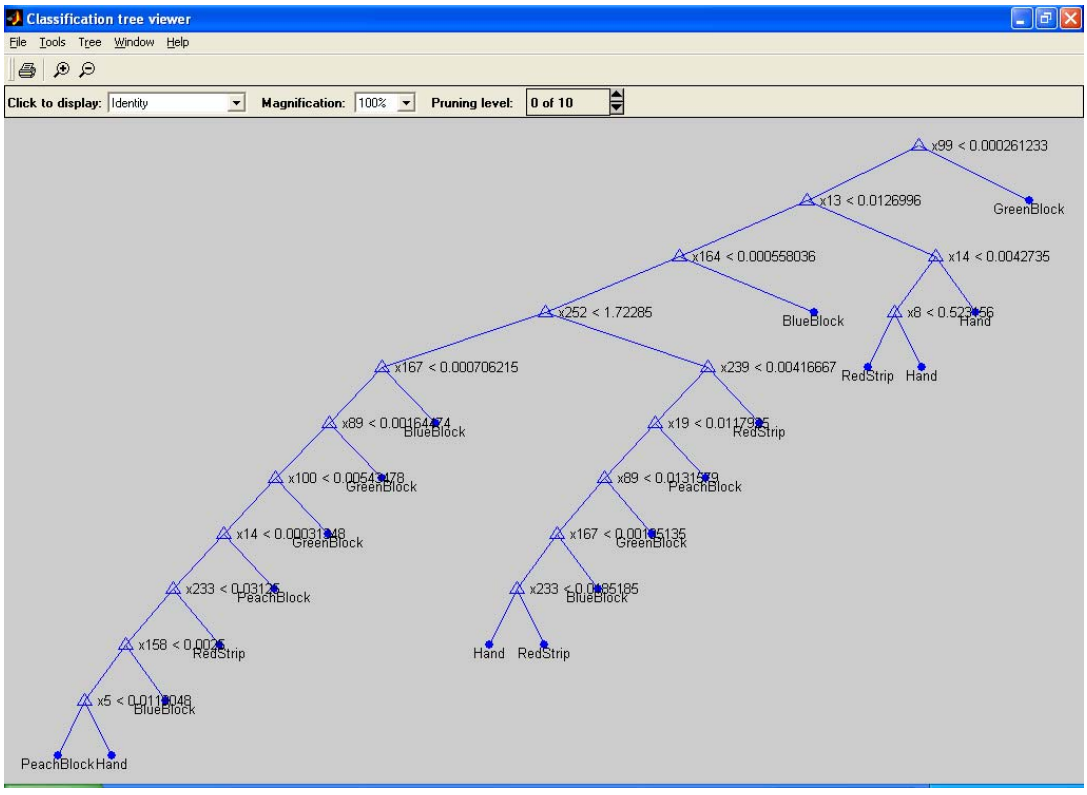

Figure 6: Sample classification tree

Rule Extraction

The method used for our system actually creates fifteen classification trees. The original data is used to create a tree. The `analyzetreeseq.m` file creates the loop to process the fifteen trees. Once the rules (discussed later) are extracted from the original tree, the most significant discriminating feature is removed and a new tree is formed. In Figure 6, the feature at the top of the tree, feature 99, would be set to zero for all vectors and the modified vectors would then be used to produce a new tree. This is repeated fifteen times. The reasoning behind this is that objects have more than one discriminating feature. A single tree relies strongly on one set of features, which may differentiate most of an object from the rest of the sample set. The "lesser" trees help to strengthen other discriminatory features that may be present in an object that the main tree does not use. In effect, the "lesser" trees ensure greater use of the other features available, making the algorithm more robust.

The trees have rules that are extracted using the script file `analyzetree.m`. Using the structure array returned from treefit, the percentage of each class in each leaf node is extracted. After taking the maximum percentage and its class of all the leaf nodes, they are sorted from largest to smallest. The sorted leaf nodes can tell what are the final features that determine the last splits in the tree. When the trees are created, the end splits lead to one leaf and another impure group or two leaves. If the leaf is in the first column, then the leaf is decided to have less than the threshold amount of the deciding feature. Each feature is used as a fuzzy membership function. An array is created to hold whether a leaf node has less or more than the threshold amount with a –1 or 1. In Figure 6, the bottommost set of leaves contains two classifications; PeachBlock and Hand. The deciding feature for this particular split is feature 5. As the program moves up the tree, -1 would be assigned for PeachBlock because it is less than the threshold at feature 5 and 1 for Hand. All of the decision features are multiplied by a –1 or 1 and saved for each leaf starting from the leaf with the highest percentage. These decision feature arrays start with the last decision and end at the first. The result is an array of feature numbers that are either positive or negative listed from least significant to most. This resulting array is called a minterm (a collection of grades of membership) for that specific case of a class. Several minterms added together make a rule. Each tree creates a number of minterms each for one of the object classes. The minterms for the same class are added together to create a rule.

An example of part of the rule for Hand in Figure 6:

$$\text{Hand Rule} = ((\overline{X_{99}}) \cap (X_{13}) \cap (X_{14})) \cup ((\overline{X_{99}}) \cap (X_{13}) \cap (\overline{X_{14}}) \cap (X_{08})) \quad (14)$$

Above is an example of a possible rule taken from a set of trees. Each term is considered a minterm. Each variable is referring to a feature. The "< threshold" next to the feature number in Figure 6 notes the manner in which the classification tree makes the decision. The program changes the minterm to be interpreted in as a fuzzy set. The equation 14 above would read: not a lot of feature 99 AND a lot of feature 13 AND a lot of feature 14 OR not a lot of feature 99 AND a lot of feature 13 AND not a lot of feature 14 AND a lot of feature 8 and so on. As discussed earlier, the matrix of feature numbers would represent the example as follows:

Table 1: Minterm storage example

	Features				
Minterm 1	14	13	-99	0	0
Minterm 2	8	-14	13	-99	0

The minterms (rows) would continue till all the Hand classes for this tree are taken. The number of features that can be held for each minterm (columns) is set at 40 currently but this limitation is arbitrary and can easily be increased. Notice that the features closest to the leaf nodes are in the early columns of the matrix, this is due to the direction taken through the tree to collect the features. The complete rule for this tree will encompass all of the Hand leaf nodes in the classification tree.

Rule Analysis

After extracting the rules, a method was developed to observe the effectiveness of the rules amongst the other training vectors. The reason was to study the minterms to determine if there were any methods to decrease incorrect classifications. After processing the classification trees, extracting a matrix of minterms (as described above) and an array of numbers that corresponds to the minterm matrix indicating the class that a specific minterm is describing, the minterms are compared to each other to determine how object specific each one is.

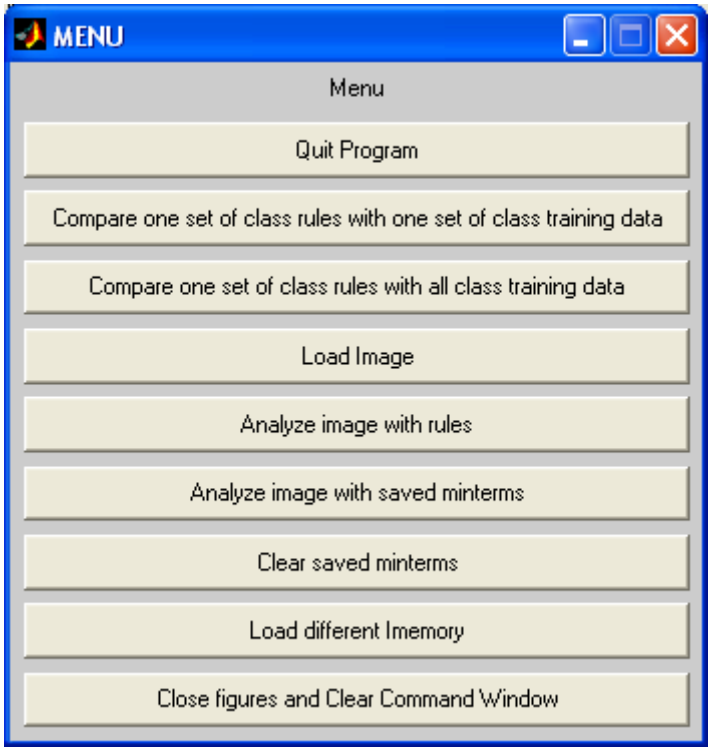

Figure 7: Initial menu for minterm analysis

The overall setup of the program is to allow the comparison of minterms between classes. The main window of this observation program is shown in Figure 7. By studying the discrimination graphs, specific minterms, individually selected or via a specified threshold, can be observed if desired. The program allows for these minterms to be observed with their properties (colors, descriptive terms) and applied to images to give visual perspective on the minterm's discriminatory properties (discussed later). To make studying different training sets more robust, the program allows for minterms of one dataset to be compared to training data of another dataset.

Once the desired class (chosen class) is identified, the program takes the minterm set of that class and compares it with each of the feature vectors of other classes (comparison classes). The result is a matrix whose columns correspond to the chosen class' minterms and the rows correspond to the comparison classes. Each coordinate in this matrix contains the summed values of the comparison class' feature vectors for that minterm.

Visual representation of previous paragraphs using classification tree example:

$$MT_{sample} = \begin{bmatrix} 99 & 0 & 0 & 0 & 0 & 0 & ..... \\ 14 & 13 & -99 & 0 & 0 & 0 & \\ 8 & -14 & 13 & -99 & 0 & 0 & \\ -8 & -14 & 13 & -99 & 0 & 0 & \\ . & & & & & & \\ . & & & & & & \\ . & & & & & & \end{bmatrix} \qquad L_{sample} = \begin{bmatrix} GreenBlock \\ Hand \\ Hand \\ \text{Re}\, dStrip \\ . \\ . \\ . \end{bmatrix}$$

entire minterm matrix (m x 40) class of minterm(m x 1)

m = the number of minterms gathered from all the trees of all the classes

If the chosen class is Hand, Hand's minterms are extracted from the entire matrix (MT_{sample}) using the class array.

$$MT_{hand} = \begin{bmatrix} & & & & & & \\ 14 & 13 & -99 & 0 & 0 & 0 & ..... \\ 8 & -14 & 13 & -99 & 0 & 0 & \\ . & & & & & & \\ . & & & & & & \\ . & & & & & & \\ . & & & & & & \end{bmatrix} \qquad L_{hand} = \begin{bmatrix} \\ Hand \\ Hand \\ . \\ . \\ . \\ . \end{bmatrix}$$

Hand minterms (n x 40) array of minterm's classes (n x 1)

n = the number of minterms defining the rule for the class Hand

Objects have the contribution from multiple features to establish their characteristics, just as the rules created suggest. The operations used for standard fuzzy membership functions would negate the many features from each minterm. For this reason, Bayesian rules are used. For example,

If fuzzy operations are used,

$$\text{Hand Rule} = ((\overline{X_{99}}) \cap (X_{13}) \cap (X_{14})) \cup ((\overline{X_{99}}) \cap (X_{13}) \cap (\overline{X_{14}}) \cap (X_{08})) \qquad (14)$$

$$\Downarrow$$

$$\text{Hand Rule} = \min\{\max\{(1-X_{99}),(X_{13}),(X_{14})\}, \max\{(1-X_{99}),(X_{13}),(1-X_{14}),(X_{08})\}\} \quad (15)$$

This simply results in the finding the maximum feature value for each minterm and then finding the minimum of the two maximums. The result would only contain the value of one feature rather than be compiled of all the features. This method partially negates the reason for using classification trees.

If the probability axioms and Bayes' intersection rule given independence are used,

$$\text{Hand Rule} = ((\overline{X_{99}}) \cap (X_{13}) \cap (X_{14})) \cup ((\overline{X_{99}}) \cap (X_{13}) \cap (\overline{X_{14}}) \cap (X_{08})) \quad (14)$$

$$\Downarrow$$

$$\text{Hand Rule} = (1 - X_{99})(X_{13})(X_{14}) + (1 - X_{99})(X_{13})(1 - X_{14})(X_{08}) \quad (16)$$

All the features of minterms that have value are represented. The union rule is [14]:

$$P(A \cup B) = P(A) + P(B) - P(A \cap B) \quad (6)$$

When the union rule is implemented, the final term ($-P(A \cap B)$) is not used. This part was ignored for ease of implementation. The reason this can be removed is it can be argued that the final term will become incredibly small upon calculation. Removing this term has not caused a noticeable decrease in the quality of results as compared to leaving it in.

Depending on whether the user chose to compare the chosen class to one comparison class or the chosen class to all comparison classes, the program collects the feature vectors of the chosen class and one or more comparison classes. The features values are extracted according to the minterms for processing. A value for each minterm is calculated for all the feature vectors. As the minterm calls for different features, these features are normalized from 0 to 1 with all the values of that feature in the class. Depending on whether the feature number in the minterm is positive or negative, the value of the feature is interpreted as either normalized $n_{i,j}$ or 1 – normalized $n_{i,j.}$ Once all the values are collected for each feature in the minterm, the values are multiplied together. At this point, the value of a minterm for all the samples of one class has been collected. The program continues to do this for every minterm in the chosen class minterm matrix. The values are summed and saved for each class in a matrix called sumvectdata.

Lets assume there are 4 feature vectors of BlueBlock (feature vectors 7, 10, 25, 103) that are extracted from the complete feature matrix (FV). Lets also assume that the first 2 minterms of class Hand are the only minterms.

$$FV_{blue} = \begin{bmatrix} n_{7,1} & n_{7,2} & n_{7,3} & n_{7,4} & n_{7,5} & n_{7,6} & \cdots \\ n_{10,1} & n_{10,2} & n_{10,3} & n_{10,4} & n_{10,5} & n_{10,6} & \cdots \\ n_{25,1} & n_{25,2} & n_{25,3} & n_{25,4} & n_{25,5} & n_{25,6} & \cdots \\ n_{103,1} & n_{103,2} & n_{103,3} & n_{103,4} & n_{103,5} & n_{103,6} & \cdots \\ \end{bmatrix}$$

Feature vectors of the class BlueBlock (4 x 433)

Note: The feature vectors of a class are normalized by feature (column wise) from 0 to 1 by dividing by the maximum value of the feature before being calculated.

$$MT_{hand} = \begin{bmatrix} 14 & 13 & -99 & 0 & 0 \\ 8 & -14 & 13 & -99 & 0 \end{bmatrix}$$

First 2 minterms for class Hand

The measure of each minterm is calculated for each feature vector (MT_{hand} applied to FV_{blue}).

$$S = \begin{bmatrix} (n_{7,14})(n_{7,13})(1-n_{7,99}) & (n_{7,8})(1-n_{7,14})(n_{7,13})(1-n_{7,99}) \\ (n_{10,14})(n_{10,13})(1-n_{10,99}) & (n_{10,8})(1-n_{10,14})(n_{10,13})(1-n_{10,99}) \\ (n_{25,14})(n_{25,13})(1-n_{25,99}) & (n_{25,8})(1-n_{25,14})(n_{25,13})(1-n_{25,99}) \\ (n_{103,14})(n_{103,13})(1-n_{103,99}) & (n_{103,8})(1-n_{103,14})(n_{103,13})(1-n_{103,99}) \end{bmatrix}$$

$$S = \begin{bmatrix} val_{7,1} & val_{7,2} \\ val_{10,1} & val_{10,2} \\ val_{25,1} & val_{25,2} \\ val_{103,1} & val_{103,2} \end{bmatrix}$$

Now $val_{7,1}$ means the value of feature vector 7 for minterm 1. To calculate the summed value of the BlueBlock class for Hand minterm 1 the first column of the score matrix (S) is summed.

$$Ssum = \begin{bmatrix} (val_{7,1} + val_{10,1} + val_{25,1} + val_{103,1}) & (val_{7,2} + val_{10,2} + val_{25,2} + val_{103,2}) \end{bmatrix}$$

If there were only 2 Hand minterms and these 4 samples for BlueBlock, this 1x2 row vector (Ssum) would represent the BlueBlock feature vectors analyzed with Hand minterms of sumvectdata.

$$\text{Sumvectdata} = \begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} & v_{15} & \ldots\ldots & v_{1s} \\ v_{21} & v_{22} & v_{23} & v_{24} & v_{25} & & \\ v_{31} & v_{32} & v_{33} & v_{34} & v_{35} & & \\ . & & & & & & \\ . & & & & & & \\ . & & & & & & \\ v_{r1} & & & & & & \end{bmatrix}$$

sumvectdata (r x s)

r = the comparison class and the chosen class (if only one comparison class is chosen), or all of the classes (if comparison with all classes is chosen)

s = the number of minterms for the chosen class

For example, v_{23} is the summed values of the third minterm for all the samples of class 2

Now using these values, the program can evaluate how well each minterm applies to its class. Sumvectdata is divided into one row vector and a matrix. The row vector is the values of the chosen class with for minterms (sumvectcorrect). The matrix has the values of the incorrect classes, which are summed down each column to create a new row vector (vectwrong). The elements of vectcorrect are divided by the corresponding elements of vectwrong to show the properties of each minterm.

This process is shown below.

$$\text{vectwrong} = \begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} & v_{15} & \ldots\ldots & v_{1s} \\ v_{31} & v_{32} & v_{33} & v_{34} & v_{35} & & \\ . & & & & & & \\ . & & & & & & \\ . & & & & & & \\ v_{r1} & & & & & & \end{bmatrix}$$

matrix of incorrect class values (r –1 x s)

$$\text{sumvectcorrect} = \begin{bmatrix} v_{21} & v_{22} & v_{23} & v_{24} & v_{25} & \ldots\ldots \end{bmatrix}$$

array of correct class values (1 x s)

Note: Assuming class 2 is the chosen class

After summing down the columns for vectwrong, sumvectwrong is produced.

$$\begin{bmatrix} v_{11}+v_{31}+v_{41}\cdots & v_{12}+v_{32}+v_{42}\cdots & v_{13}+v_{33}+v_{43}\cdots & v_{14}+v_{34}+v_{44}\cdots & v_{15}+v_{35}+v_{45}\cdots & \ldots\ldots \end{bmatrix}$$

sumvectwrong(1 x s)

$$\begin{bmatrix} \dfrac{v_{21}}{v_{11}+v_{31}+v_{41}\cdots} & \dfrac{v_{22}}{v_{12}+v_{32}+v_{42}\cdots} & \dfrac{v_{23}}{v_{13}+v_{33}+v_{43}\cdots} & \dfrac{v_{24}}{v_{14}+v_{34}+v_{44}\cdots} & \dfrac{v_{25}}{v_{14}+v_{34}+v_{44}\cdots} & \ldots\ldots \end{bmatrix}$$

sumvectcorrect / sumvectwrong (1 x s)

The results of sumvectcorrect / sumvectwrong (discrimratio) are plotted revealing the more powerful discriminating features, which appear as the higher peaks.

There are three sets of data that are being used in the following examples and plots. The first two sets are taken from an outdoor setting. The classes for both data sets are Tree, Grass, AL (Artificial Landmark), and Pathway. One main difference of the two sets is in the feature vectors. Both contain vectors that are 433 long. The first set (Dataset 1) uses the 180 line features while the second set (Dataset 2) does not have these features. Instead of changing the number of the column indices in the vector for the unused features, these indices are set to zero. Both contain the 250 color features and the standard texture measure. Both sets also do not have

the Canny texture measure and the phase symmetry measure. Another main difference is the size of the sample set, Dataset 2 contains 2079 samples for all of its classes while Dataset 1 only contains 191 samples for all of the classes. The numbers of samples for each class are given in Table 2.

Table 2: Sample quantities for Dataset 1 and Dataset 2

	# of samples in Dataset 1	# of samples in Dataset 2
TreeTrunk	66	115
Grass	61	1048
Pathway	47	899
Artificial Landmark	17	17
Total Samples	191	2079

The third set (Dataset 3) of data is taken from a video of a joint project with the psychology department. The classes in this set are BlueBlock, GreenBlock, PeachBlock, Hand, and RedStrip. This data set also contains 250 color features and the standard texture measure. This data set contains the motion feature but does not have the 180 line features. Dataset 3 has 250 samples for its classes (50 samples for each class). Lets examine what is shown for datasets and what can be deduced from the plots.

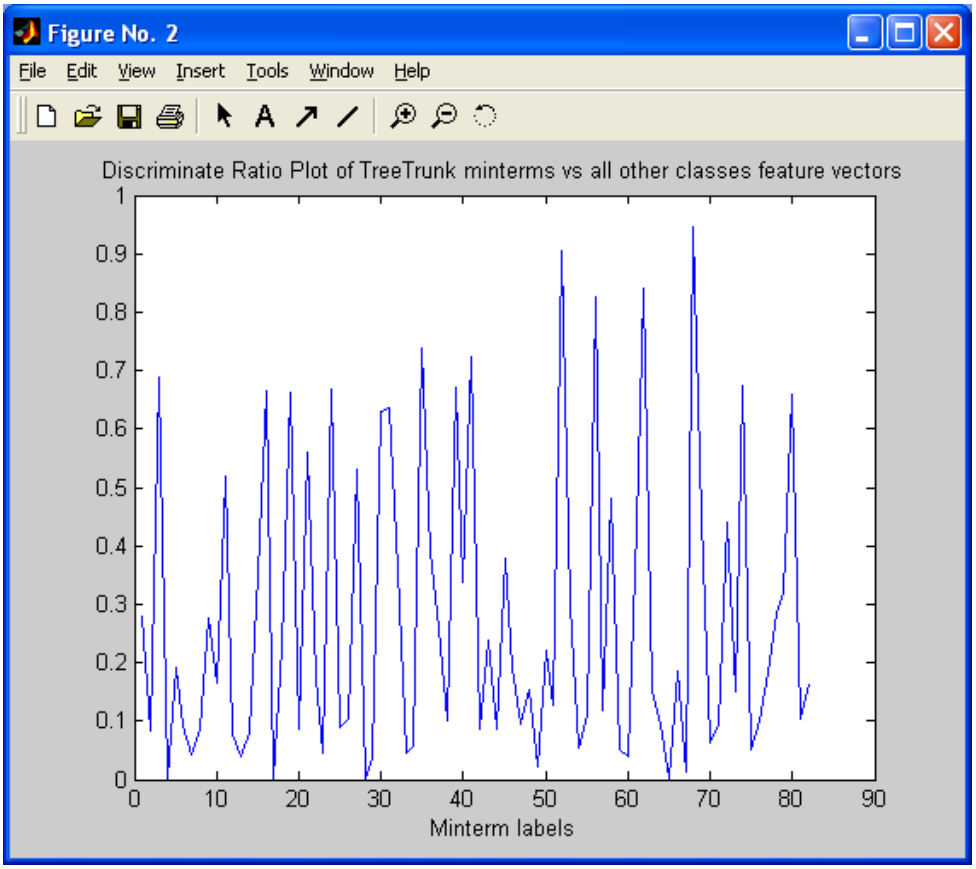

Figure 8: Comparison plot for TreeTrunk using Dataset 1

The minterm labels on the x-axis correspond to the rows in the chosen class' extracted matrix. These minterm labels are used to find the actual minterm content when analyzing minterms. The y-axis is the ratio of minterm strength for the correct class over the minterm strength for the sum of the incorrect classes. The minterms that are the most discriminating have higher peaks. A problem with the way these plots are calculated is its dependency on the number of samples in its class. The method used does not adjust for class 1 having more samples than class 2. The values on the y-axis do not give a measure that can be equated to minterm comparisons of different classes. Nevertheless, the calculation is still useful for determining the most discriminatory minterms. Figure 9 shows the secondary menu after a class comparison is evaluated.

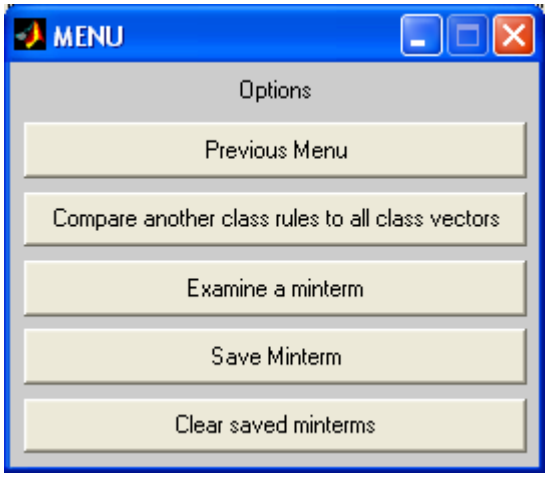

Figure 9: Secondary Menu after Class comparison

For example lets look at minterm label 68 in Figure 8 since it has a large spike. Figure 10 shows the information for this minterm, which we summarize as:

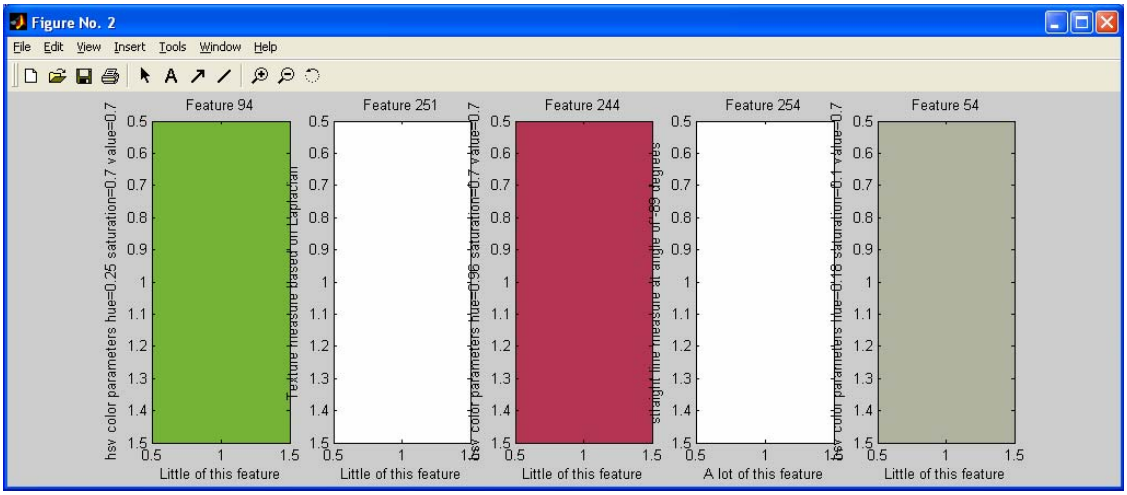

Figure 10: TreeTrunk minterm label 68 from Dataset 1

Table 3: TreeTrunk minterm 68 from Dataset 1

Feature 94
hsv color parameters hue=0.25 saturation=0.7 value=0.7
Little of this feature

Feature 251
Texture measure based on Laplacian
Little of this feature

Feature 244
hsv color parameters hue=0.96 saturation=0.7 value=0.7
Little of this feature

Feature 254
straight line measure at angle of -89 degrees
A lot of this feature

Feature 54
hsv color parameters hue=0.18 saturation=0.1 value=0.7
Little of this feature

By selecting "Examine a Minterm", the minterm description image and description text appear after indicating the minterm label. We have observed that the algorithm has tendencies to evaluate a class by what features it does not have versus the stronger features it does have. The reason for this may be that the feature vectors are mostly zero throughout making it much more likely to choose features that do not exist rather than the few features that do exist. Due to this fact and the preferred method of description for objects, special notice is shown for features that are stronger (used in the classification tree) and are significant due to the large measures taken from the feature.

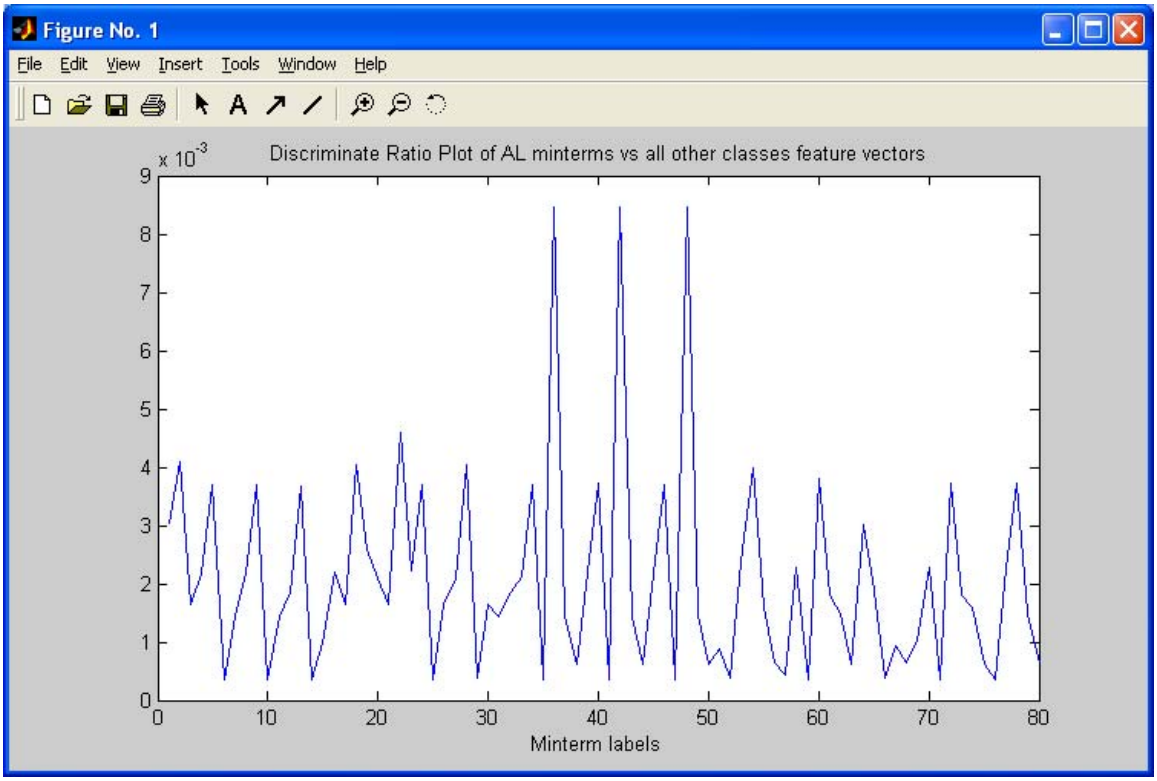

Figure 11: Comparison plot from Dataset 2

In the case for the Artifical Landmark, there are 3 very prominent peaks in Figure 11. The artificial landmark is a 3-color cylinder (neon pink, neon orange, neon green) created for object recognition. Obviously, its main purpose is to stand out from its surroundings. The y-axis values are very small. This reinforces the previous explanation about the number of samples in the set distorting the outcome of the y-axis in the plots. The artificial landmarks have only 17 samples out of 2079. The program still does well determining the landmarks. Even though the minterms determine the artifical landmarks well, the description of the minterm is not easily understood. The description a human would give would be likely to focus on the three bright colors in the landmarks. This is not the case in our algorithm as shown below in Figure 12.

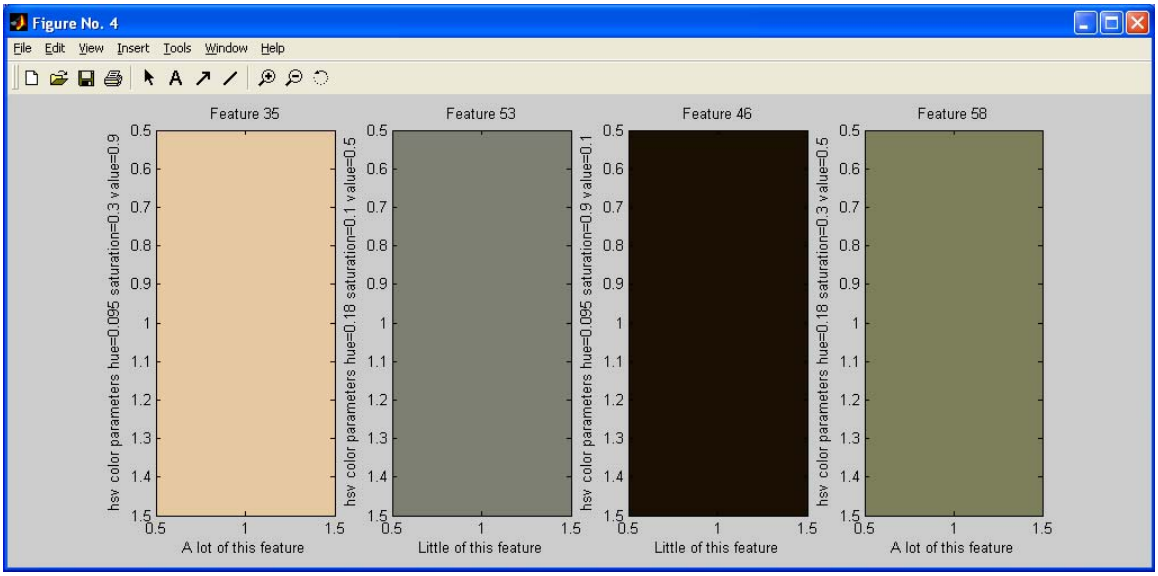

Figure 12: AL minterm label 48 from Dataset 2

Table 4: AL minterm 48 from Dataset 2

Feature 35
hsv color parameters hue=0.095 saturation=0.3 value=0.9
A lot of this feature

Feature 53
hsv color parameters hue=0.18 saturation=0.1 value=0.5
Little of this feature

Feature 46
hsv color parameters hue=0.095 saturation=0.9 value=0.1
Little of this feature

Feature 58
hsv color parameters hue=0.18 saturation=0.3 value=0.5
A lot of this feature

Features 35 and 46 have the same hue value. This value indicates orange for the color. Feature 35 is a faded orange that is very bright (strongly resembles peach). Feature 58 is a vibrant orange that has very little brightness (strongly resembles black). Features 53 and 58 have the same hue as well. Their color is yellow-green. Feature 53 is faded to the point that it is difficult to detect any color and has some brightness (strongly resembles gray). Feature 58 is faded and has some brightness (strongly resembles brownish-green). This minterm is looking for a lot of the peach and brownish-green and little of the gray and black. The representation does not quite meet the expectations of the description in human terms. Though this is just one of the spiking

minterms, all of the highest 3 have similar representations. These representations give a good grasp of the reasons for the program's decisions and may lead to selection of stronger additional features.

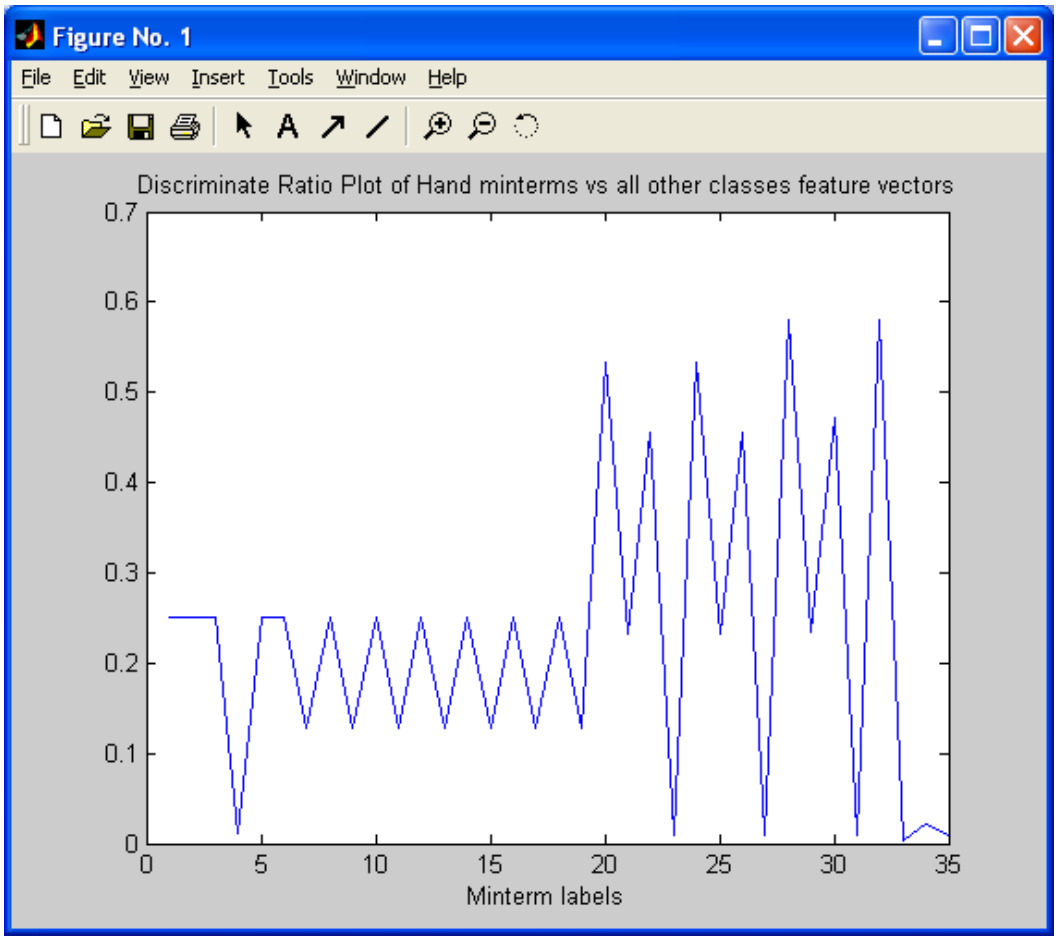

Figure 13: Comparison plot of Dataset 3

Due to the nature of the experiments performed with the Psychology Department (C.F. Chapter V), the motion feature was added to aid in segmenting the hand. Figure 13 shows a comparison of the Hand minterms to the other classes, and Figure 14 examines minterm label 20.

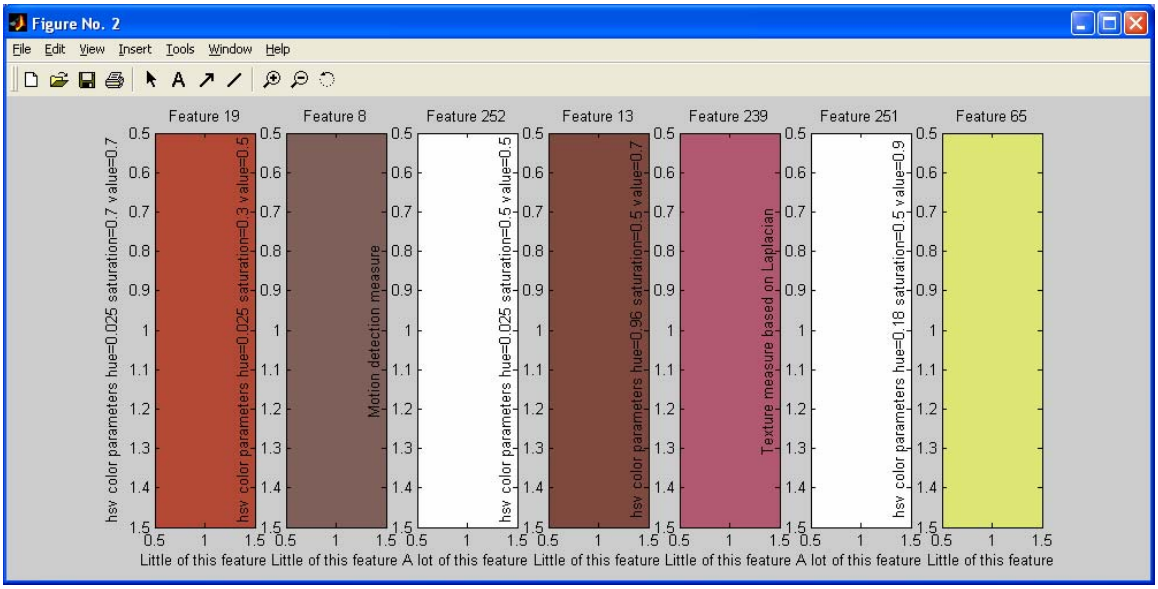

Figure 14: Hand minterm label 20 from Dataset 3

Table 5: Hand minterm label 20 from Dataset 3

Feature 19
hsv color parameters hue=0.025 saturation=0.7 value=0.7
Little of this feature

Feature 8
hsv color parameters hue=0.025 saturation=0.3 value=0.5
Little of this feature

Feature 252
Motion detection measure
A lot of this feature

Feature 13
hsv color parameters hue=0.025 saturation=0.5 value=0.5
Little of this feature

Feature 239
hsv color parameters hue=0.96 saturation=0.5 value=0.7
Little of this feature

Feature 251
Texture measure based on Laplacian
A lot of this feature

Feature 65
hsv color parameters hue=0.18 saturation=0.5 value=0.9
Little of this feature

This minterm describes a bunch of colors that should not be included with the hand. Features 19, 8, and 13 are all orange-red with varying saturations and mid-range values (resembles orangish-brown, brown, and light brown). Feature 239 is red with mid-range saturation and high value (resembles reddish-violet). Feature 65 is a yellow-green with mid-range saturation and very high value (resembles a bright yellow). Little of the colors are expected in this minterm. The minterm also contains a lot of feature 251 and 252. Feature 251 is the Laplacian texture measure. Feature 252 is the new motion feature. The additional motion feature was significant for the hand detection, allowing the motion patterns of people to be tracked for study.

The features that were removed were because of computation costs and/or lack of significance in the classification of objects. For a majority of the samples, color features are the strongest. In the case for TreeTrunk between Dataset 1 and Dataset 2, Dataset 1 consistently has

the line measure feature in its minterms and does a better job of classifying tree trunks in images (shown later). For outdoor datasets, the line feature appears to be a useful feature.

CHAPTER IV

APPLYING THE MEMBERSHIP FUNCTIONS TO IMAGES

This chapter's purpose is to discuss applying the rules to a new image, and displaying the results of the image processed with the rules.

After the training vectors have been entered and the rules have been extracted, the next step is to apply these rules to images. First, the image is loaded using the function `imread` in the Matlab Signal Processing Toolkit. Now the features of the image need to be extracted.

Extracting Image Features

The process of extracting a feature set from a new image is the same method used to gather the training sample's features. The main difference is the moving window used to define areas in the image from which the features are extracted. Currently, the window is 15 by 15 pixels and it increments by 10 pixels in the row and column directions. These values are arbitrary and can be changed easily, but a decrease in window increment (increase in resolution) increases the processing time. The window increments are constrained not to exceed the image when the window reaches the borders of the image. The features of each window are extracted and put into a matrix form. This matrix is similar to the feature matrix in the training data except the number of rows is now the number of windows in the new image. This process is shown below.

$$I = \begin{bmatrix} im_{1,1} & im_{1,2} & im_{1,3} & im_{1,4} & im_{1,5} & im_{1,6} & \cdots \\ im_{2,1} & im_{2,2} & im_{2,3} & im_{2,4} & im_{2,5} & im_{2,6} & \\ im_{3,1} & im_{3,2} & im_{3,3} & im_{3,4} & im_{3,5} & im_{3,6} & \\ im_{4,1} & im_{4,2} & im_{4,3} & im_{4,4} & im_{4,5} & im_{4,6} & \\ \vdots & & & & & & \end{bmatrix}$$

image (r x c)

$\Downarrow$

$$\text{WinF} = \begin{bmatrix} wf_{1,1} & wf_{1,2} & wf_{1,3} & wf_{1,4} & wf_{1,5} & wf_{1,6} & \cdots \\ wf_{2,1} & wf_{2,2} & wf_{2,3} & wf_{2,4} & wf_{2,5} & wf_{2,6} & \\ wf_{3,1} & wf_{3,2} & wf_{3,3} & wf_{3,4} & wf_{3,5} & wf_{3,6} & \\ wf_{4,1} & wf_{4,2} & wf_{4,3} & wf_{4,4} & wf_{4,5} & wf_{4,6} & \\ \vdots & & & & & & \end{bmatrix}$$

window feature matrix (n x 433)

r = rows in the image

c = columns in the image

n = number of 15x15 windows in image.

Applying the Rules to the Image

For each tree formed, we obtain one rule (in a sum of minterms form) for each object class. Thus, if the total number of trees formed is given by N_{trees}, then there will be N_{trees} rules for each object class. For a given object class, C_i, each rule is applied to the window feature matrix, resulting in N_{trees} result matrices for class C_i. A result matrix gives the degree of membership estimated for each region by the applied rule. What remains to be done to combine the N_{trees} results for the given class into a single result. Viewing the individual results as fuzzy sets, we may combine them using the union of the N_{trees} results using either Equation (5) or (6). However, this approach yielded poor results.

We use another approach to combining the results in to switch to another paradigm of interpretation. If we take each result vector and divide it by the total sum of all values in the result vector, the new normalized vector now has a sum of values equal to 1. Thus, the vector can be viewed as a probability density function (pdf) describing the distribution of the objects of class C_i as estimated by the associated rule, R_i, j=1,…,N_{trees}. This pdf is denoted by

$$f((x,y) \in C_i \,|\, R) \text{ where } R = R_1,\ldots,R_{Ntrees} \tag{17}$$

Then using a probabilistic assumption of equally likely rules.

$$P(R_j) = \frac{1}{N_{trees}} \quad j = 1,2,\ldots,N_{trees} \tag{18}$$

The pdfs may be combined as

$$f((x,y) \in C_i) = \sum_{k=1}^{Ntrees} f((x,y) \in C_i \,|\, R_k)\, P(R_k) \tag{19}$$

This method of combining the rule outputs yielded good results. The fuzzy membership functions seem to give a more robust evaluation of the data when gathering features. The Bayesian methods give a more robust expression of the features after they are combined. So by adding the probability density function of fuzzy membership rules across the entire image, the strength of both fuzzy rules and Bayesian probability rules can be observed.

These values are represented visually as a saliency map by applying a grayscale colormap to display the results. Using the function reshape, Matlab can manipulate this array of summed values back into the relative positions of each window in the image. The image has brighter spots for areas in the image that values are higher compared to the other values of the set. The minterms for a class worked well if the values (brightness) of the object contrast highly with the values (brightness) of objects that are not of the class. The resolution decrease observed in the output images is due to the reduction of each increment of the window down to one pixel representation.

Lets observe the differences between Dataset 1 and Dataset 2. Please note that the following steps were done numerous times with different images. The examples are representing the general discoveries of analysis.

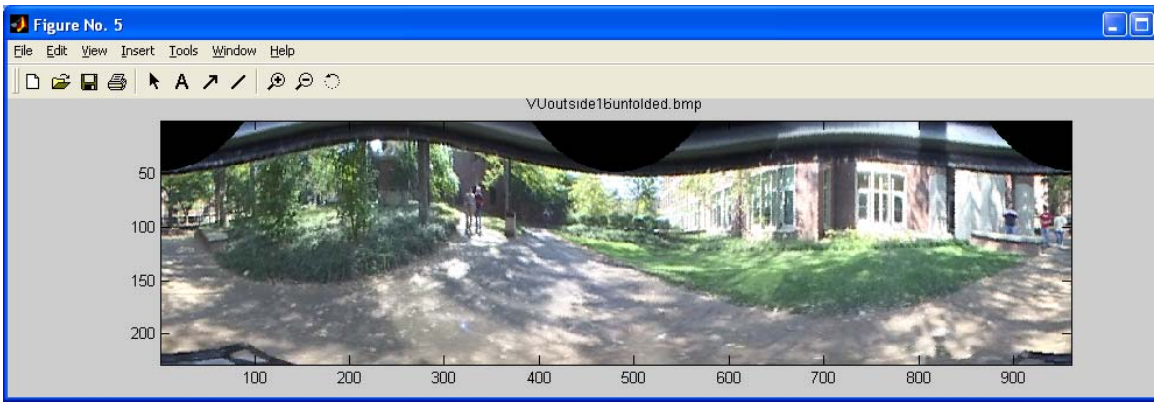

Figure 15: Original Image

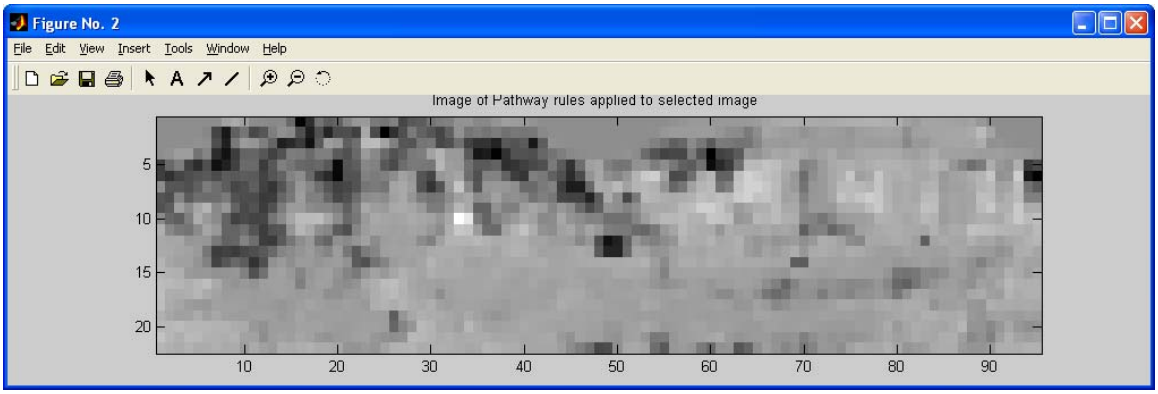

Figure 16: Pathway detection from Dataset 1

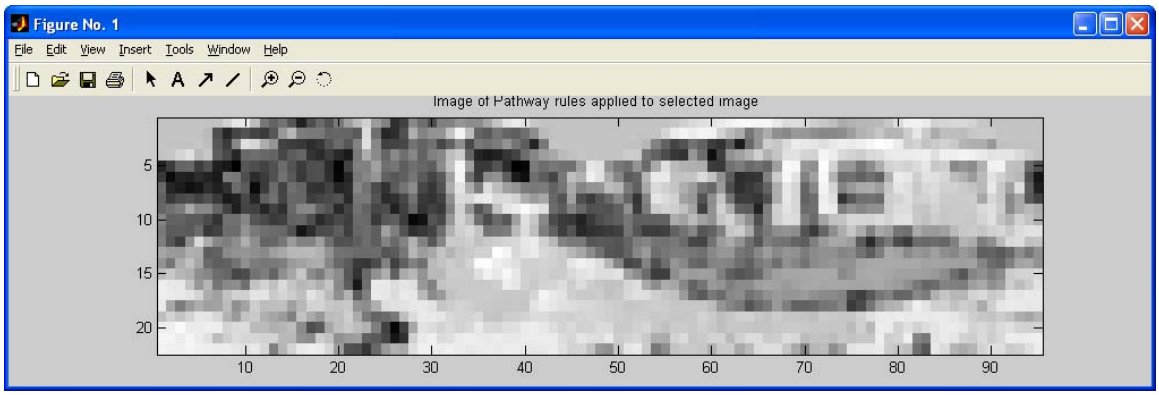

Figure 17: Pathway detection from Dataset 2

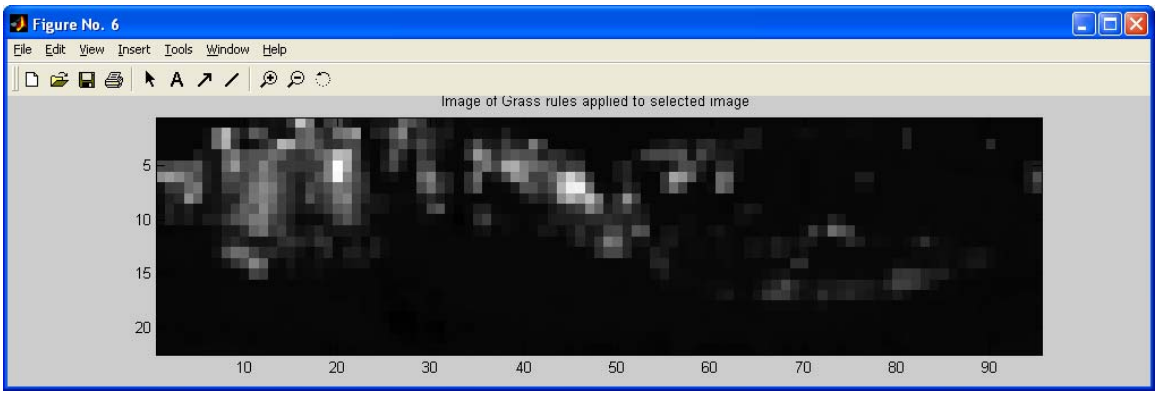

Figure 18: Grass detection from Dataset 1

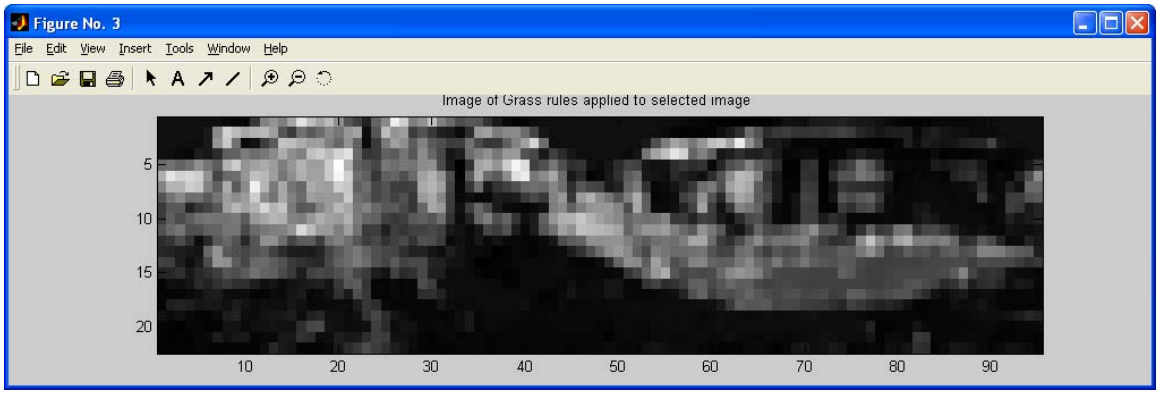

Figure 19: Grass detection from Dataset 2

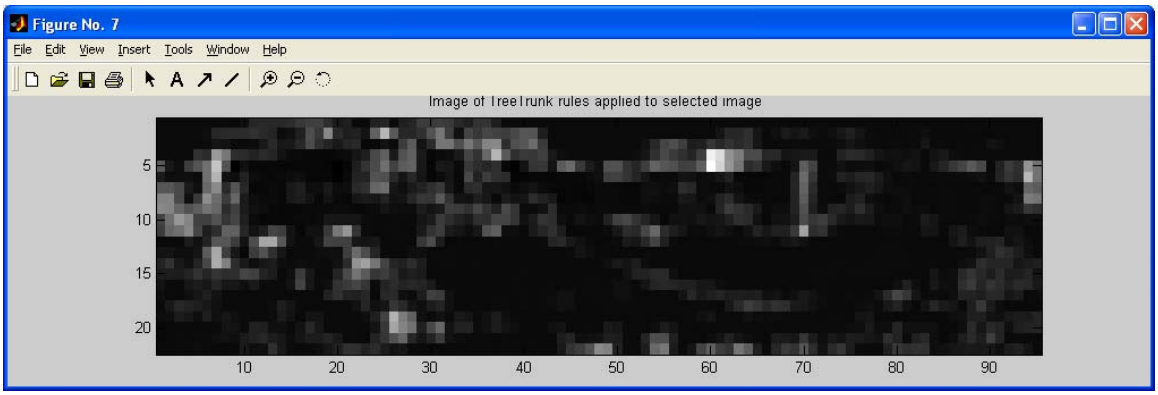

Figure 20: TreeTrunk detection from Dataset 1

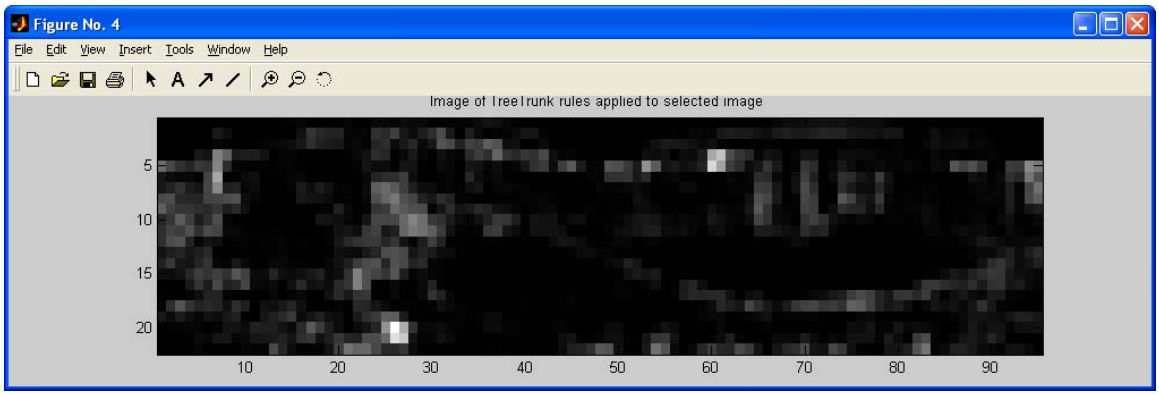

Figure 21: TreeTrunk detection from Dataset 2

The original image was created with a 360 degree panoramic mirror and unfolded as shown in Figure 15. This is the reason for the black areas at the top of the image. The main observation to note is the significant increase in rule strength due to increase in the number of training vectors,

especially for the grass (Figure 18 and 19) and pathway (Figure 16 and 17). Due to the manner in which the images are created, the stronger the contrast between the pixels means the better the rule was at detecting the object. Remember that Dataset 1 contains only 191 vectors and Dataset 2 contains 2079. The improvement in the pathway and grass detection is apparent. The main reason behind the improvement is the vast increase in samples. The TreeTrunk detection did not noticeably improve even though the number of samples increased (Figure 20 and 21). These observations lead to more analysis to try and determine the reason. The option may be chosen to select a pixel and display the percentages of how much each minterm devoted to the total value. In the areas that were correctly decided as TreeTrunk, the pixels were selected to determine if any more information could be extracted about the minterms. Below is the bright pixel from Dataset 1 at row 5 column 8 from Figure 20:

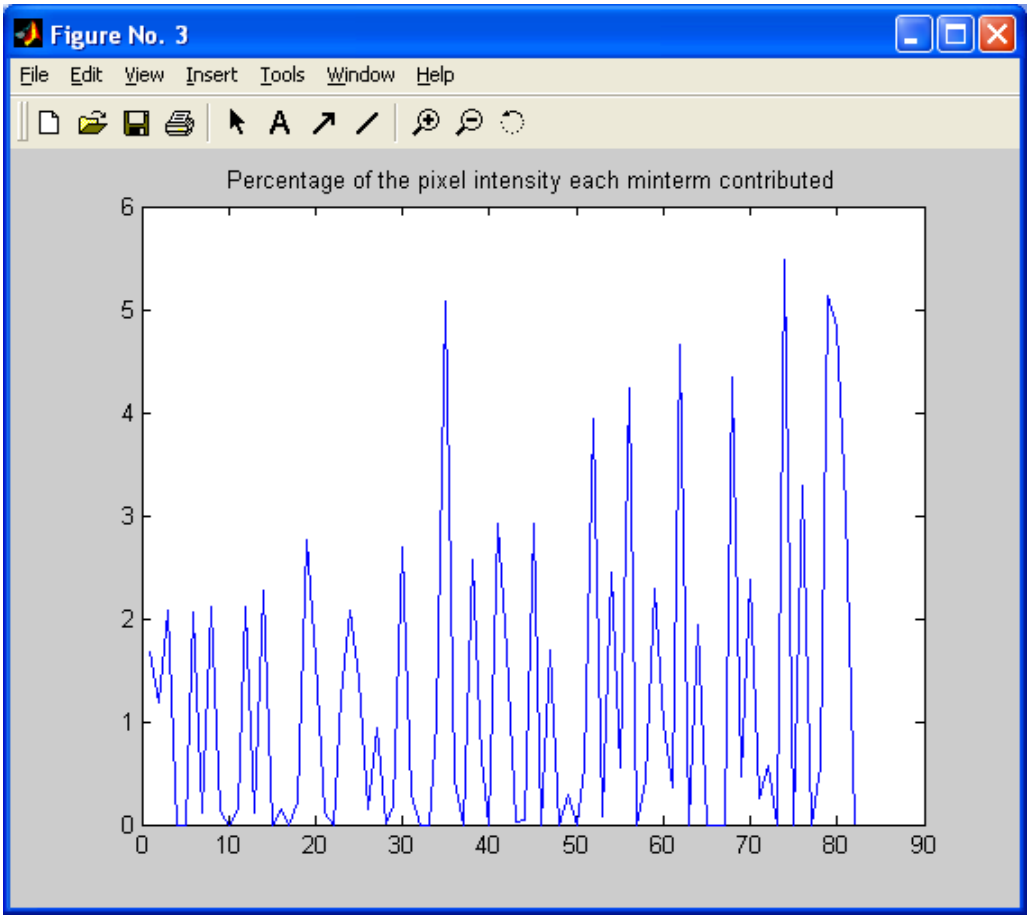

Figure 22: Minterm percentage contribution to pixel (5,8)

The numbers on the y-axis represent the percentage each minterm contributed to the selected pixel and the x-axis is the minterm labels in Figure 22. The reasoning behind creating the pixel selection was to allow for the minterms above certain thresholds to be saved and used to reanalyze the image determine if the results would improve. For Figure 23, a threshold of greater than or equal to 2% was set to determine whether these minterms would provide a better detection.

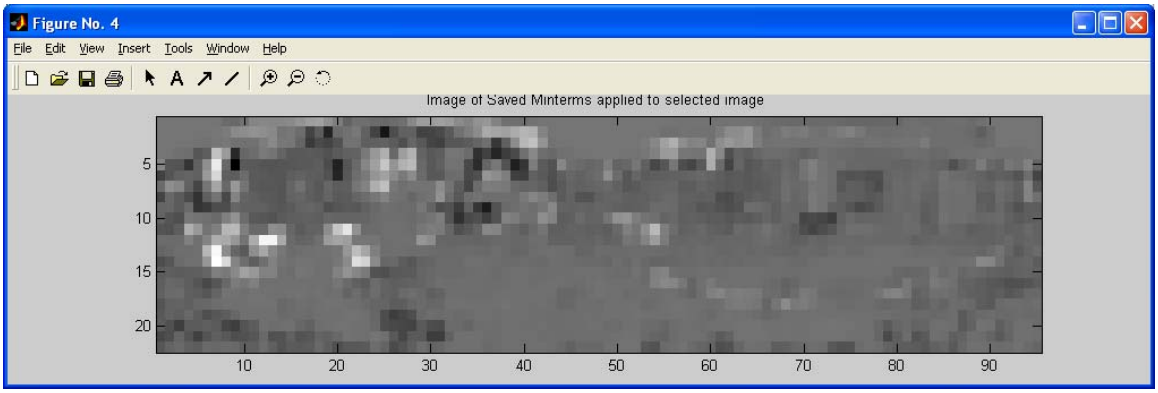

Figure 23: TreeTrunk detection with minterms above 2% contribution at pixel (5,8)

When compared to the original detection of TreeTrunk from Dataset 1, the use of the most contributing minterms did not improve. The overall certainty (contrast) in the image decreased and some of the areas where these minterms were not as strong, that were actually TreeTrunks, were lost as well. The minterms seem to work best as a complete rule. The minterms contribute to the overall certainty of the detection. The difference between the detection of Trees for Dataset 1 and Dataset 2 are very small. Dataset 1 had only 66 samples while Dataset 2 had 115. The only thing Dataset 1 had was the line measure feature. It may be deduced that either the line measure feature caused Dataset 1 to equally detect with Dataset 2 or the increase in the number of samples from Dataset 1 to Dataset 2 was not enough to be significant in detection.

The next step is to allow the detection of multiple objects in one picture. Now the program must be able to decide whether a pixel is of a certain class or of no class at all. Now the program is taking the window feature matrix and extracting the rules for all the classes. The resulting matrix (ImSc) has n (number of windows in image) rows by y (number of classes) columns.

$$\text{ImSc} = \begin{bmatrix} sc_{1,1} & sc_{1,2} & sc_{1,3} & sc_{1,4} & sc_{1,5} & sc_{1,6} & \cdots \\ sc_{2,1} & sc_{2,2} & sc_{2,3} & sc_{2,4} & sc_{2,5} & sc_{2,6} & \\ sc_{3,1} & sc_{3,2} & sc_{3,3} & sc_{3,4} & sc_{3,5} & sc_{3,6} & \\ sc_{4,1} & sc_{4,2} & sc_{4,3} & sc_{4,4} & sc_{4,5} & sc_{4,6} & \\ \vdots & & & & & & \end{bmatrix}$$

score matrix (n x y)

n = number of windows in image

y = number of classes

Each index in score matrix is the value of that windows score with the class' minterms. These scores are normalized from 0 to 1 along classes (columns). The scores are then compared to each other in the same row. The window is decided to be a certain class if it has the largest score in the rows and it is greater than 0.15 (a threshold chosen empirically by observing graphs of the scores). This value can be increased to ensure only the higher scores to be recognized as different classes. The sample image in Figure 24 was taken from the psychology department experiments. The goal is to study the motion of humans when given different descriptions of the audience. It is necessary to be able to track the hand for this experiment.

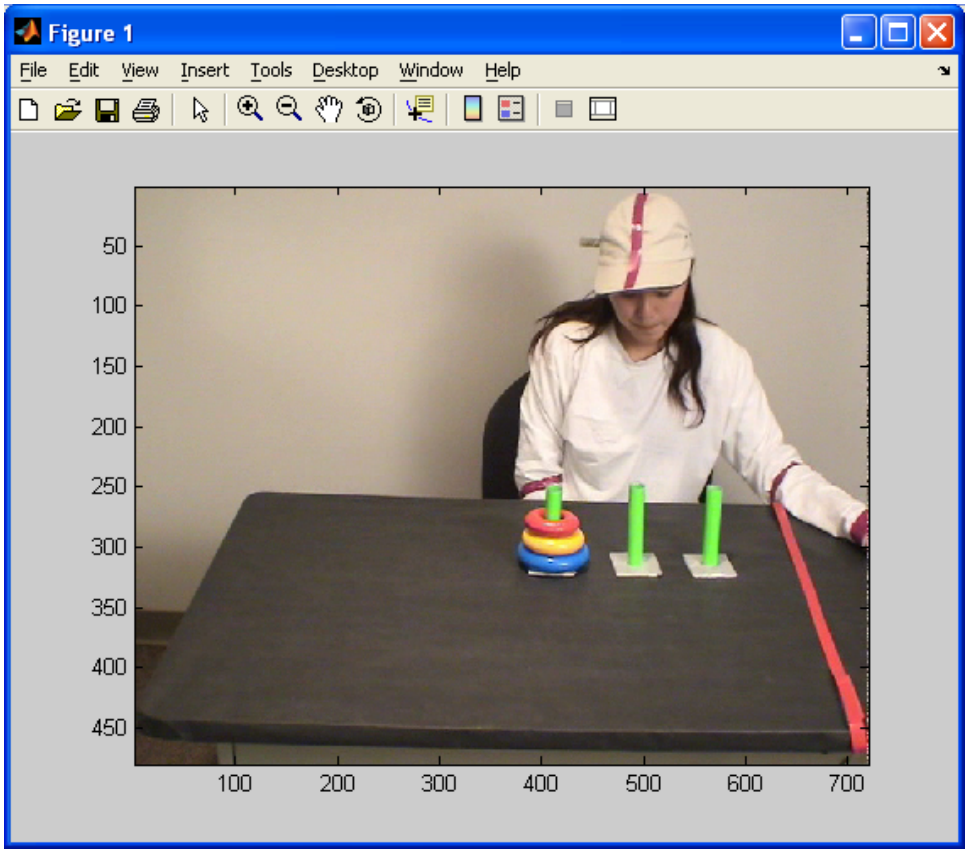

Figure 24: Sample Image

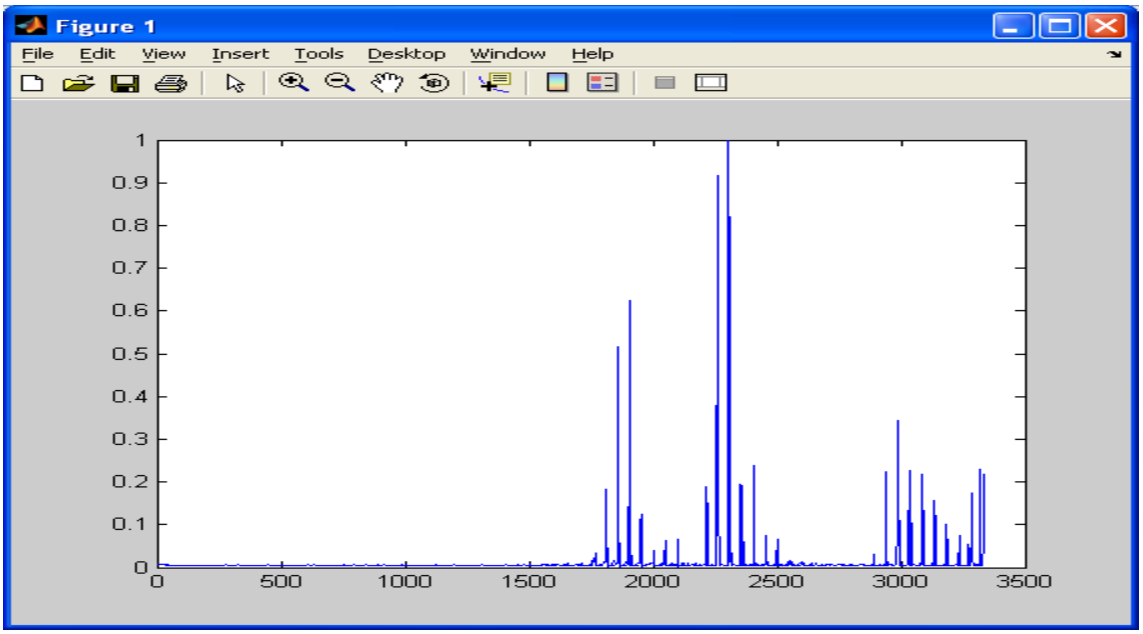

Figure 25: Redstrip score plot for Sample Image

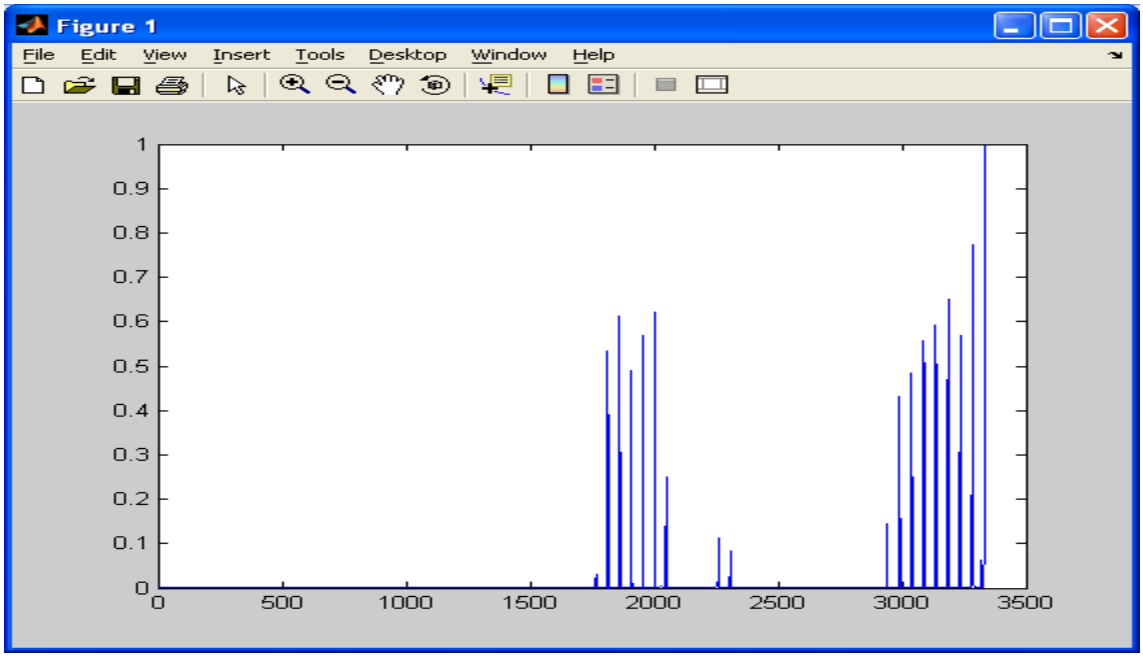

Figure 26: RedRing score plot for Sample Image

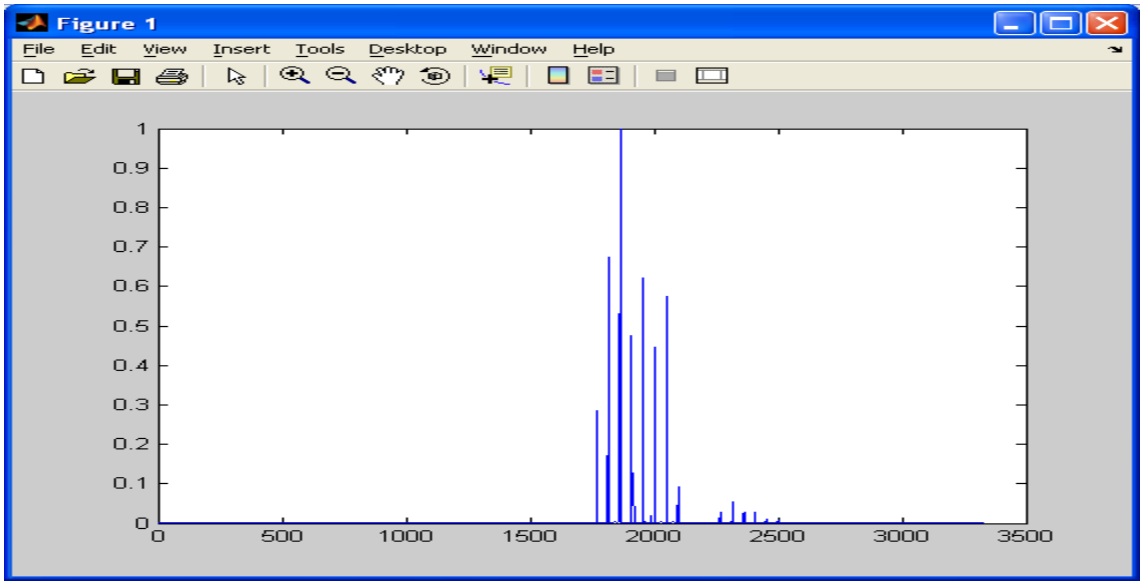

Figure 27: YellowRing score plot for Sample Image

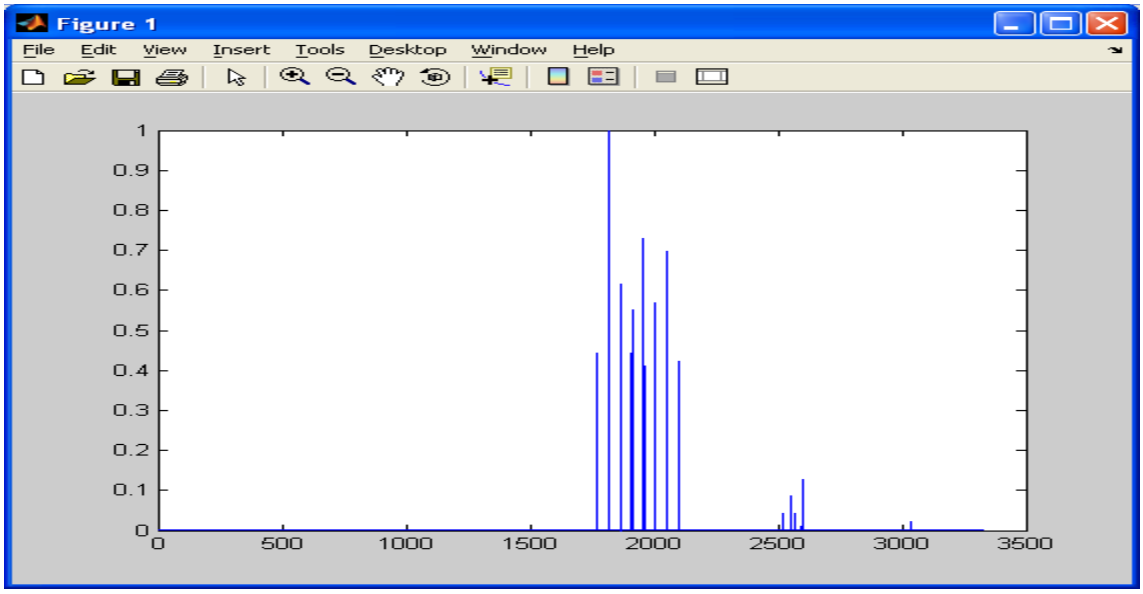

Figure 28: BlueRing score plot for Sample Image

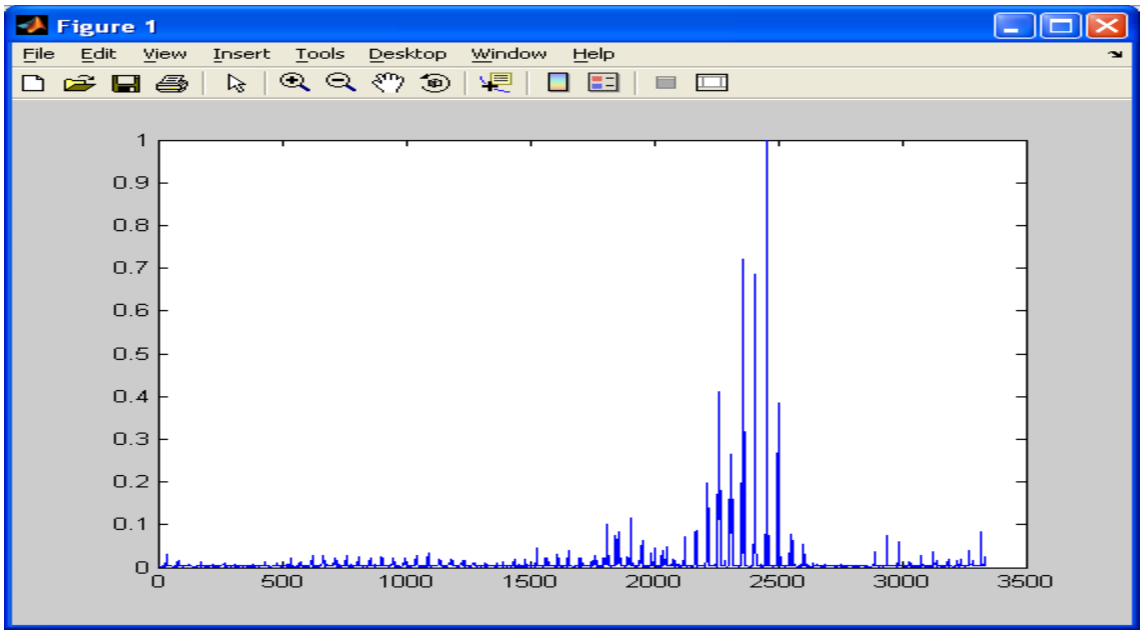

Figure 29: Hand score plot for Sample Image

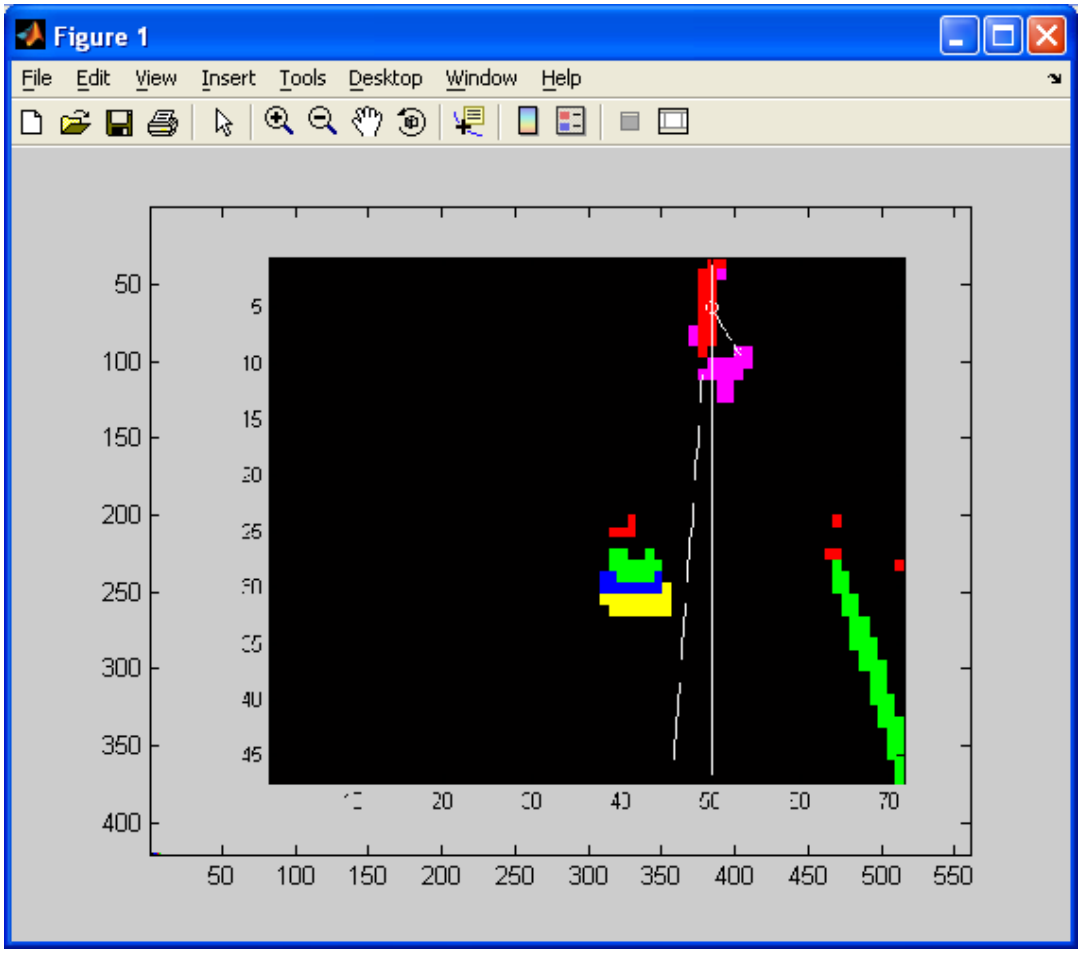

Figure 30: Segmentation Image for Sample Image

The result of the sample image (Figure 30) shows the segmentation (yellow = BlueRing, green = RedRing, blue = YellowRing, purple = Hand, and red = RedStrip). The plots of the BlueRing (Figure 28) and YellowRing (Figure 27) definitely show that the group of peaking values represents the object in the video. The RedRing plot (Figure 26) has two groups of peaking values for the object and the red tape. The Hand plot (Figure 29) indicates detection in the image though there is no hand in view. The face of the person is represented as Hand, which is logical since the face has about the same color as the hand. The plot of the RedStrip (Figure 25) indicates detection as well. As shown in the result, the strip on the hat is correctly identified.

CHAPTER V

APPLICATIONS AND RESULTS

This chapter discusses the use of the fuzzy membership recognition algorithm to discern the objects and movements in instructional task videos.

Objective of Experiment

The experiment is to use the fuzzy membership recognition algorithm to discern objects in videos from the psychology department. One of the objects is the person's hand. As stated before, the ultimate goal is to study the differences the in motion of a person when demonstrating a task for different entities (humans or robots). The main objective is to segment movies created by their department and be able to track the hand as well as the head angle. Each participant was to wear a hat with a red stripe at the center of the hat. This was to aid in discerning the direction the participant was looking. By tracking these two movements, they are able to store data on where the subject was looking and hand motion patterns. They speculated that the hand motion of the subject when told they were performing for a robot would be slower and more rigid. The information will also reveal the effectiveness of the fuzzy rules deduction and demonstrate the speed of training to new environments.

The immediate goal is to show that the recognition algorithm is effective in segmenting the objects in videos created for study by the psychology department. It can track the movements of the hand and the angle of the head, and the system is easily trained and implemented with a relatively small number of samples.

Experiment Setup

The participants are told to show either a human or a robot how to move some objects on a table to complete a task. The three tasks shown here are the card arrangement, towers of Hanoi, and tower building tasks. Each of the tasks has objects of different colors.

Figure 31: Card arrangement task initial setup

Figure 32: Card arrangement task completed

The card arrangement task shown in Figures 31 and 32 is to take the cards arranged by color and shift them to be arranged by the number of white squares on them. The only limitations are that the person may only use one hand and the cards are not allowed to pass over each other.

Figure 33: Towers of Hanoi task initial setup

Figure 34: Towers of Hanoi task completed

The Tower of Hanoi task shown in Figures 33 and 34 is to move the stack of colored rings from the spindle on the most right side of the image to the left. A ring can only be moved to an empty spindle or on top of a larger ring. Again, the task is to be completed with one hand.

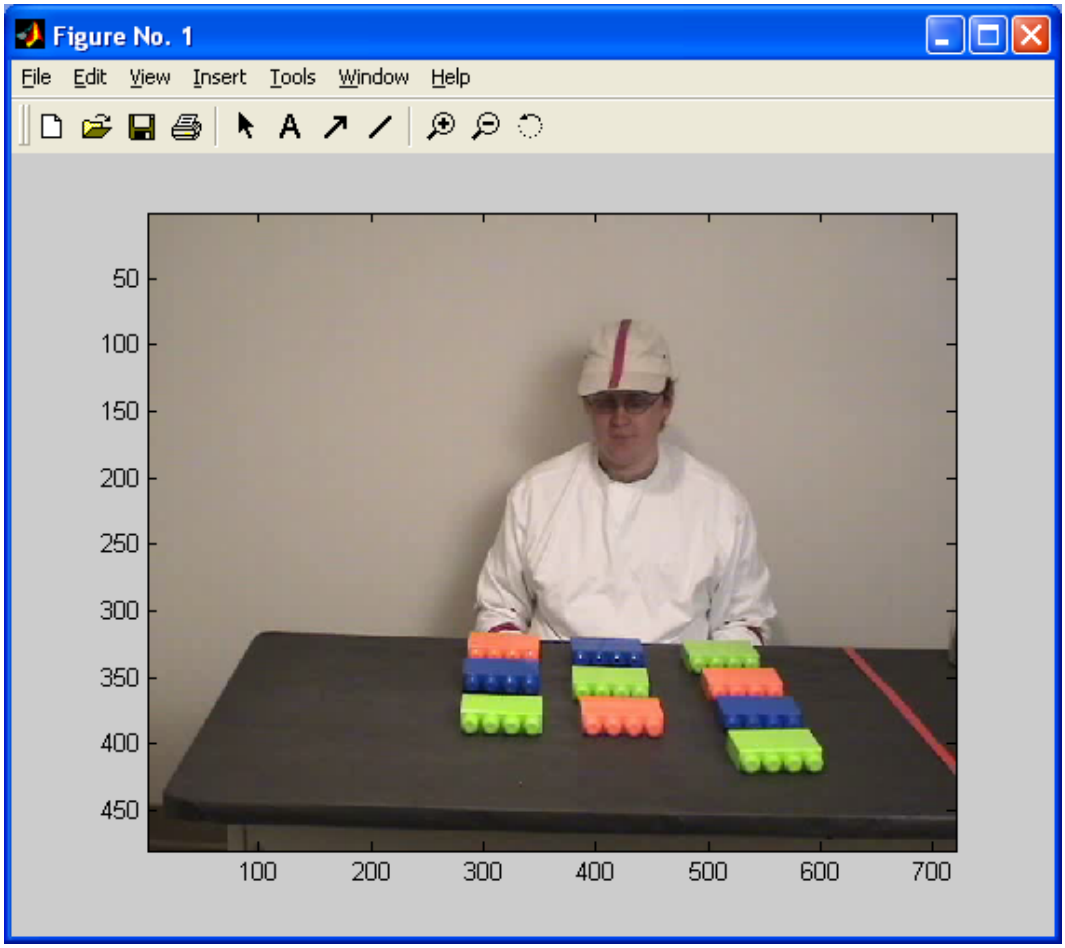

Figure 35: Tower building task initial setup

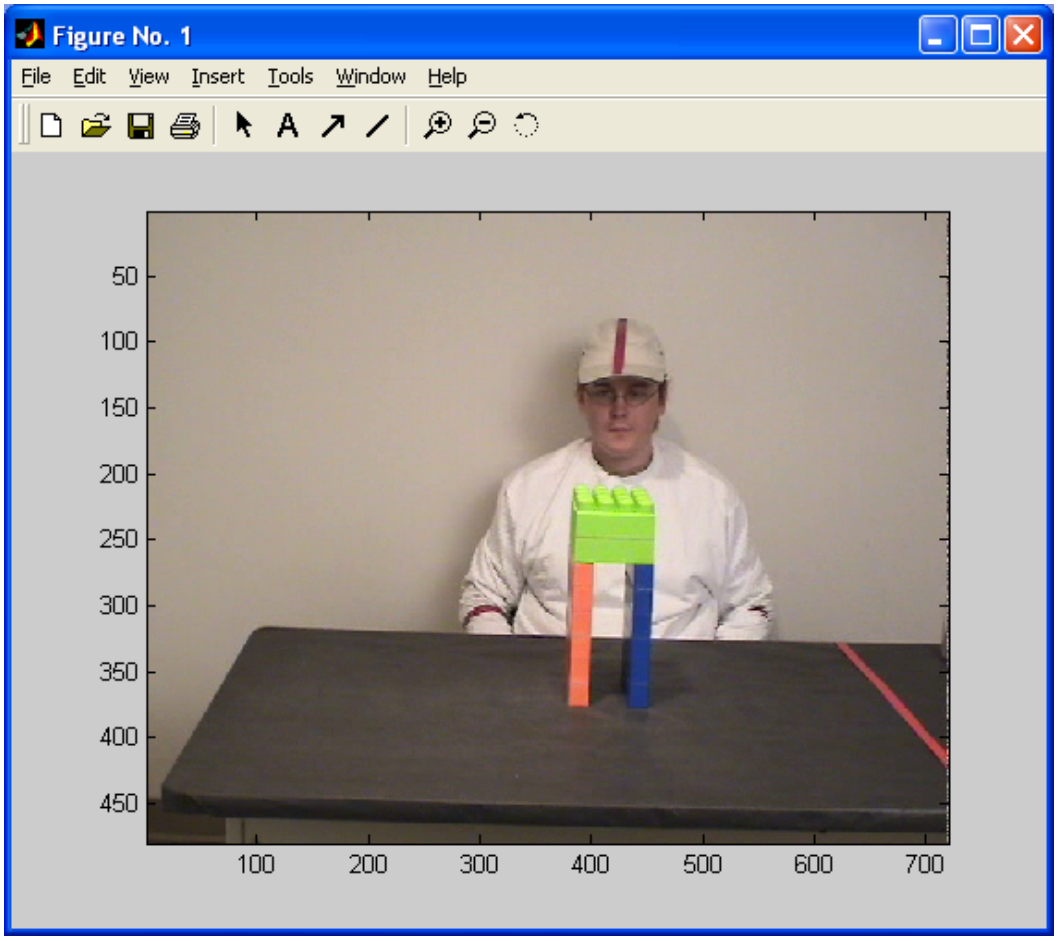

Figure 36: Tower building task completed

The tower building task is to take the blocks in Figure 35 and construct them to the structure in Figure 36. The participant is to attempt to build the tower with one hand but may use both if necessary.

Two new script files were created to easily train and process the videos by consolidating the current functions. A small algorithm is used to plot an estimated line for the center of the head, an estimated line for the direction of the gaze, and a marker to determine the hand position. The first two use the image segmentation of the red strip on the head. The strip is separated from the rest of the image by only considering the region where the head of the participant is. The center of the strip is considered the center of the head. A line is plotted going straight down from the center. The estimation of gaze uses what the algorithm determines is the red strip at the head region, and puts a best fit line to the points to determine the line of the gaze. The line of the gaze is plotted along with a verticals centerline, thus an angle of gaze from the center can be

estimated. The hand marking finds the range of hue values that flesh tones fall in for the image and multiplies them by the amount of motion detected in the same image. So the coordinates match, the flesh-tone/ motion product image is down sampled to match the segmented image coordinates. The hand position is simply the max value in the down sampled flesh-tone/motion image.

The processed videos segmented very well. Each video was created using a training vector of only 50 samples for each object. The collection of the 250 samples takes about 30-45 minutes to collect for each task. A huge feature space (252 dimensional) needs many samples to create solid clusters. Although the numbers of samples are only 250, the recognition of the videos is reasonably accurate. Table 6 shows the object classes for each task.

Table 6: Class names for the three separate tasks

Towers of Hanoi task	Card arrangement task	Tower building task
BlueRing	BluePaper	GreenBlock
YellowRing	GreenPaper	PeachBlock
RedRing	PinkPaper	BlueBlock
RedStrip	RedStrip	RedStrip
Hand	Hand	Hand

Results

The results had some noise but this is probably due to the limited number of samples. The videos themselves segmented very well considering the small number of samples. The two main areas of noise are from "inconsistent flashes" and the skin tone items in the image (i.e., PinkPaper). The inconsistent flashes are areas that are not the object it is identified to be but due to minor feature characteristics of the image scoring high enough to be considered an object. Remember the object is decided from its score on the rules. Since the score is normalized from 0 to 1 across all the windows of the frame, some noise (i.e., lighting differences) causes a fluctuation in the scores of all the windows giving incorrect results. Also, the color features are not fine enough to separate the colors of the skin and pink. Integrating more samples into the training matrix can reduce the noise from the inconsistent flashes. Increasing the number of bins in the color feature space will separate very close colors in the hue space to different bins. The

noise reduction would probably be at least half as noisy if the samples doubled. The small number of samples allows for decent detection, the pixels that were correctly detected can be saved and used as additional training vectors. This allows for the training vector samples to increase dramatically with each frame results. This change can allow for quick analysis on the effectiveness of the fuzzy rule. If the object detection does not improve with samples, the most likely action is to adjust the feature vector itself.

In all of the videos, the hand marking and gaze estimation worked rather well. By storing these numbers, the hand motions can be analyzed to reveal possible motions of the hand. By taking these vectors of positions in the hand, it is possible to extract changes in the direction of the hand and/or pauses. Using these to segment one motion of the hand from another, the frames where the changes occur can be displayed and analyzed. Figure 37 is an example of motion display for Figure 33.

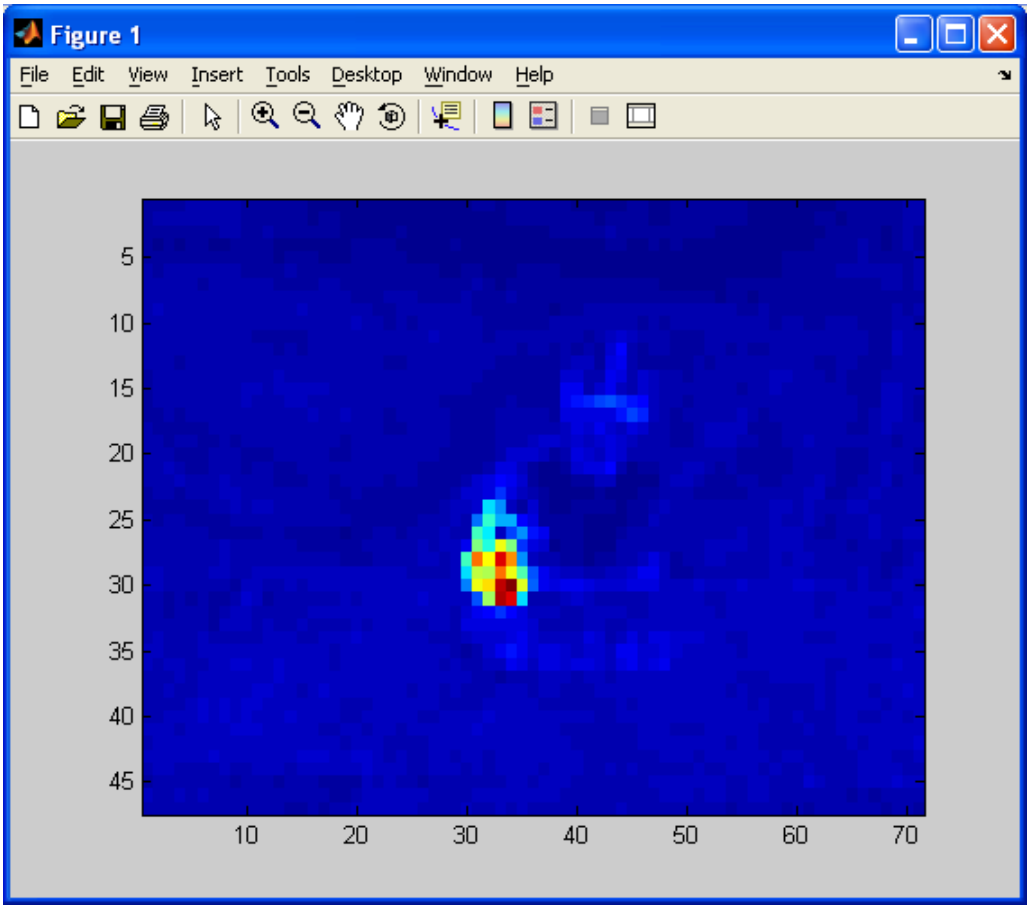

Figure 37: Motion image of Figure 33

The collected hand positions give the following plots. Figure 38 is the median-filtered x-axis (column) position of the hand plotted vs frame number. As the numbers increase, the farther to the right side of the image is the hand location. Figure 39 is a derivative of the median-filtered x-axis hand motion used to aid in extracting the pauses in the x directions. Figure 40 is the median-filtered y-axis (row) position of the hand. As the numbers increase the closer to the bottom of the image is the hand location. Figure 41 is the derivative of the median-filtered y-axis hand motion used to aid in extracting the pauses in the y directions.

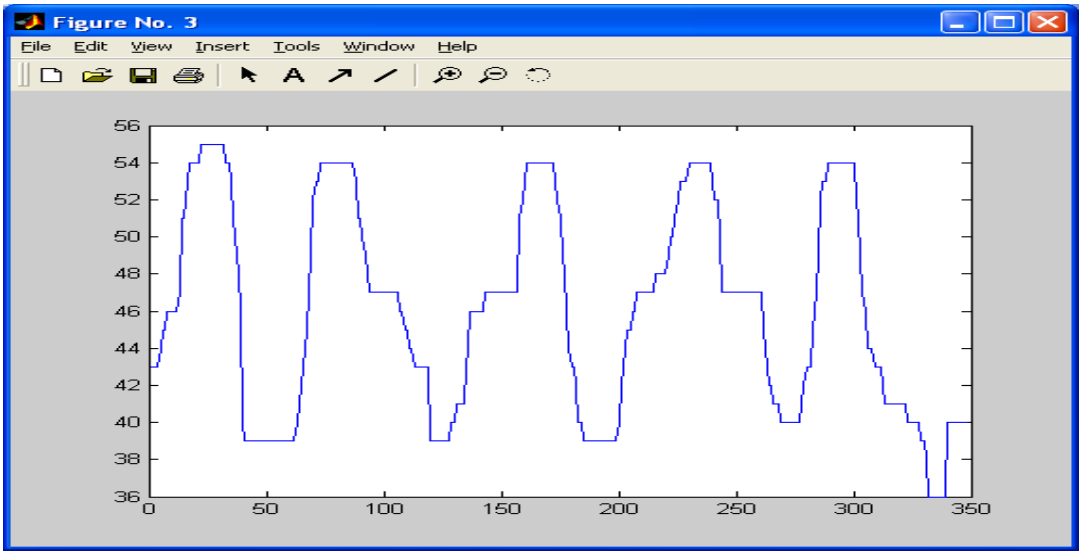

Figure 38: X-axis hand position

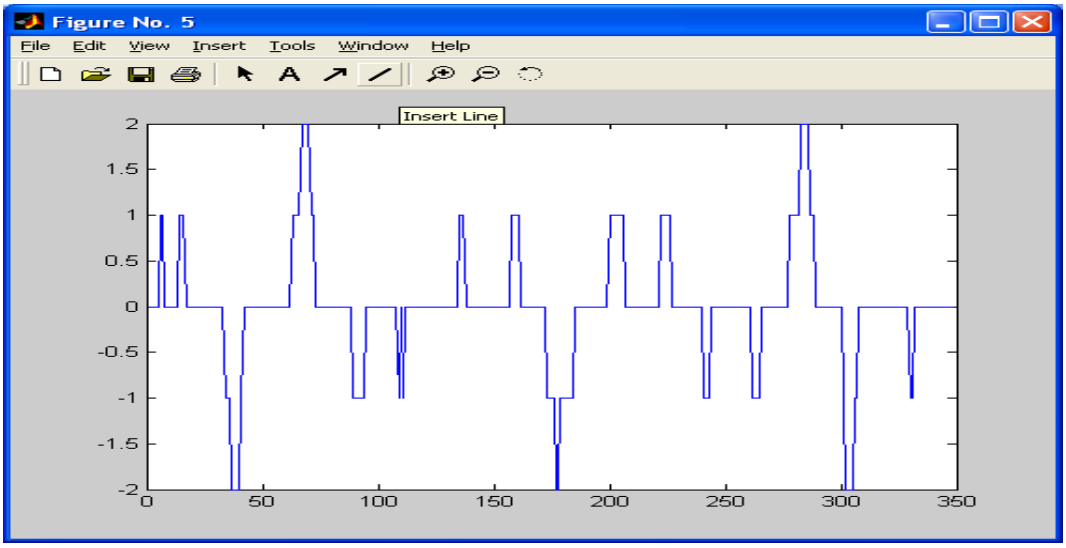

Figure 39: Derivative of x-axis hand position

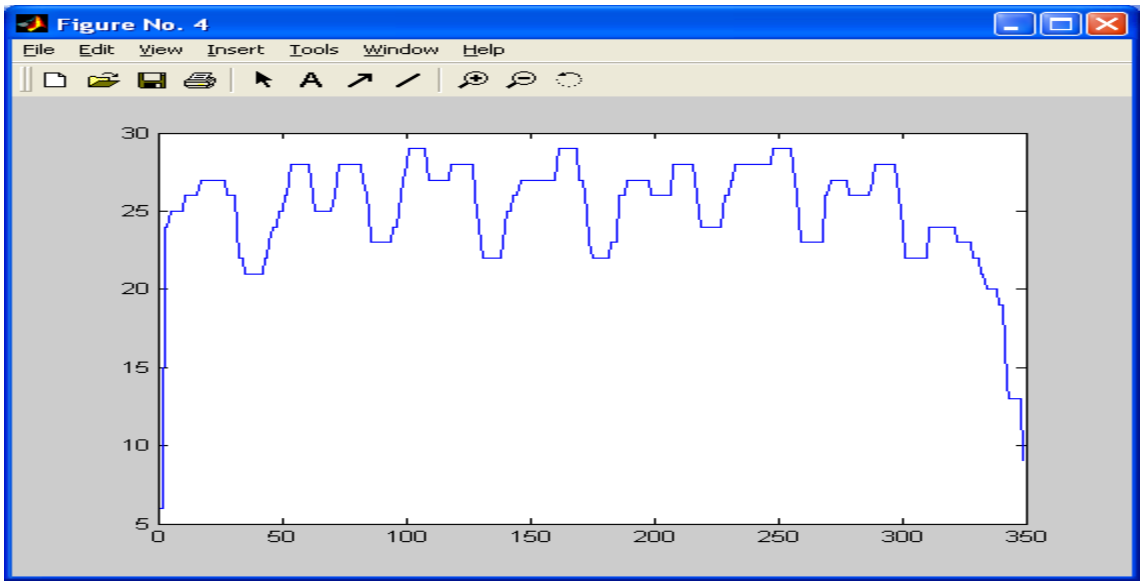

Figure 40: Y-axis hand position

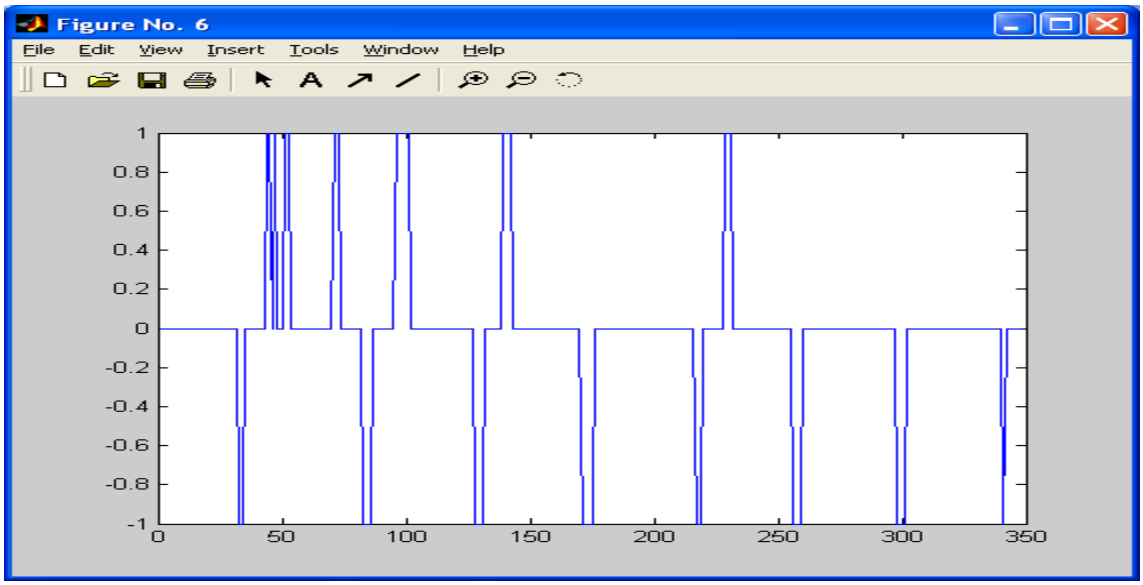

Figure 41: Derivative of y-axis hand position

Figures 42 through 56 are the frames estimated to be the transition points from one motion to the next for the Tower 2 video:

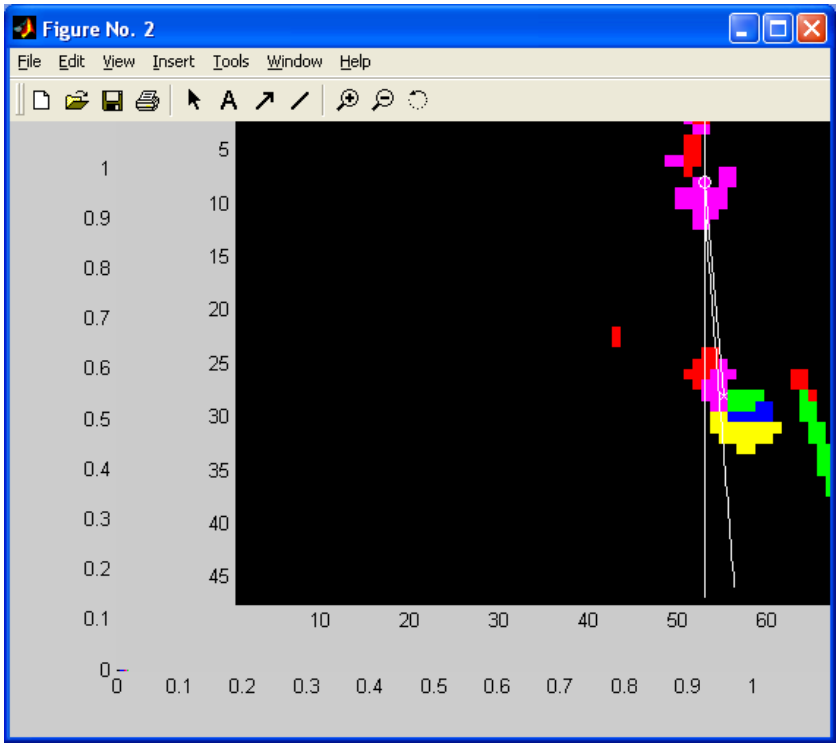

Figure 42: End - Reach to RedRing, Begin - Place

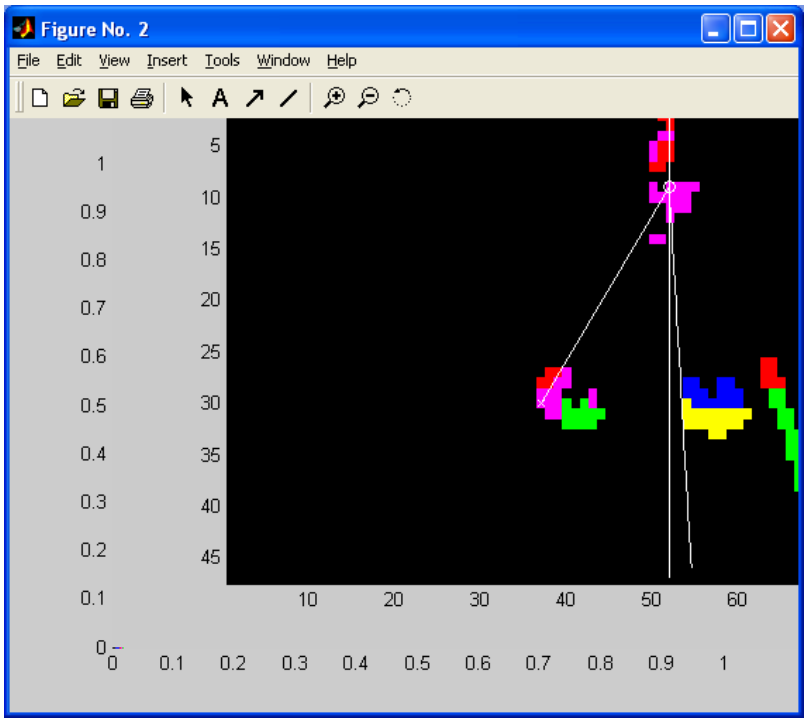

Figure 43: End – Place RedRing, Begin - Reach

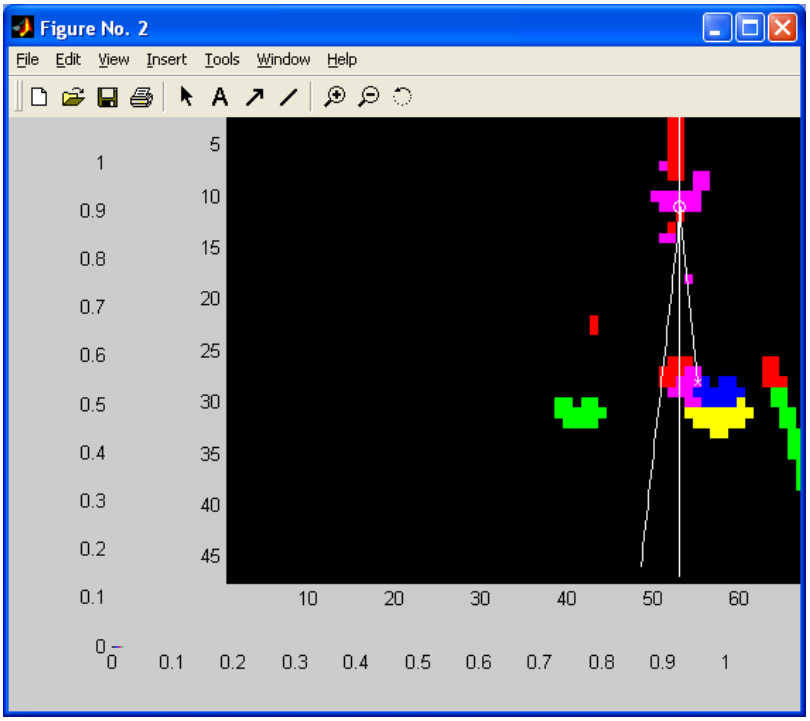

Figure 44: End – Reach YellowRing, Begin - Place

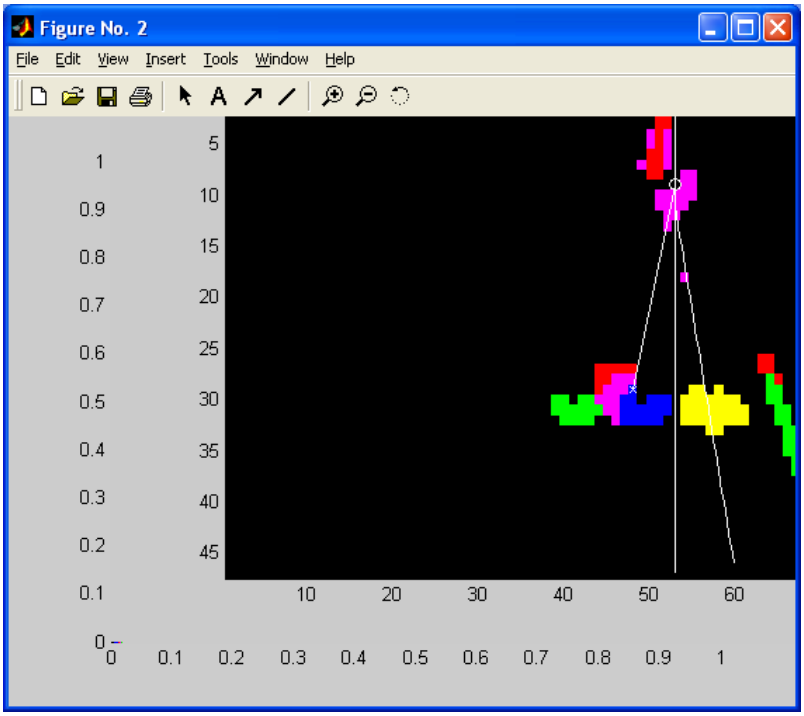

Figure 45: End – Place YellowRing, Begin – Reach

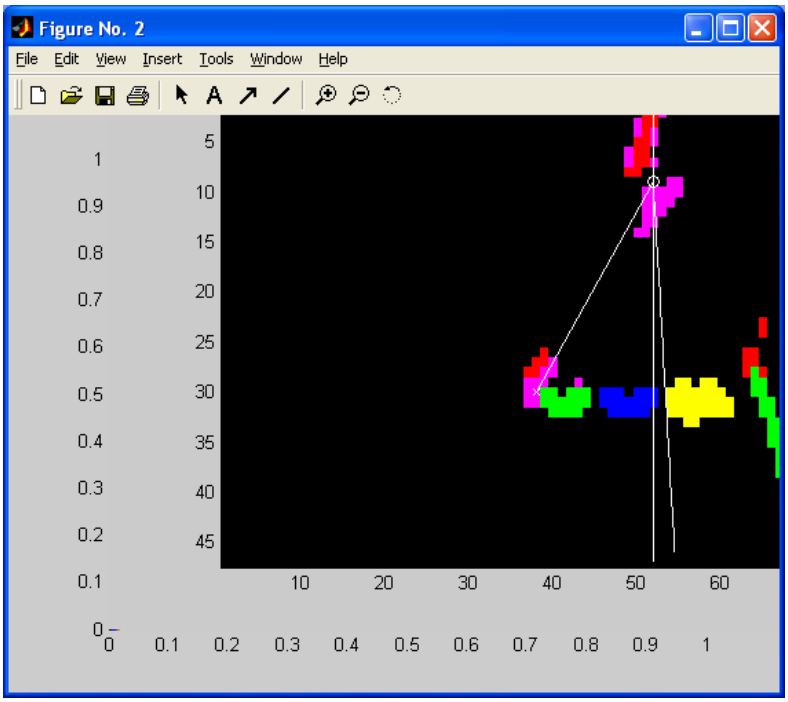

Figure 46: End – Reach RedRing, Begin – Place

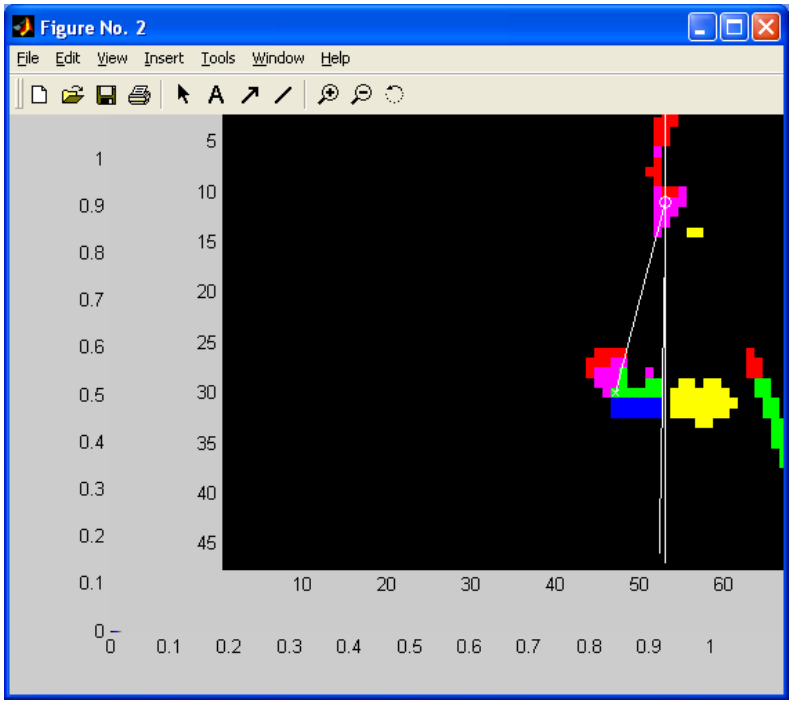

Figure 47: End – Place Redring, Begin – Reach

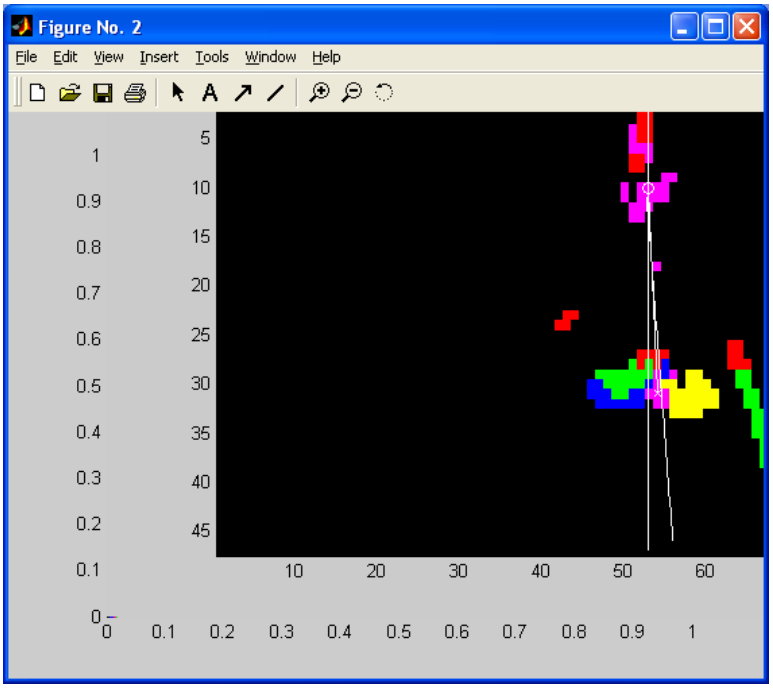

Figure 48: End – Reach BlueRing, Begin - Place

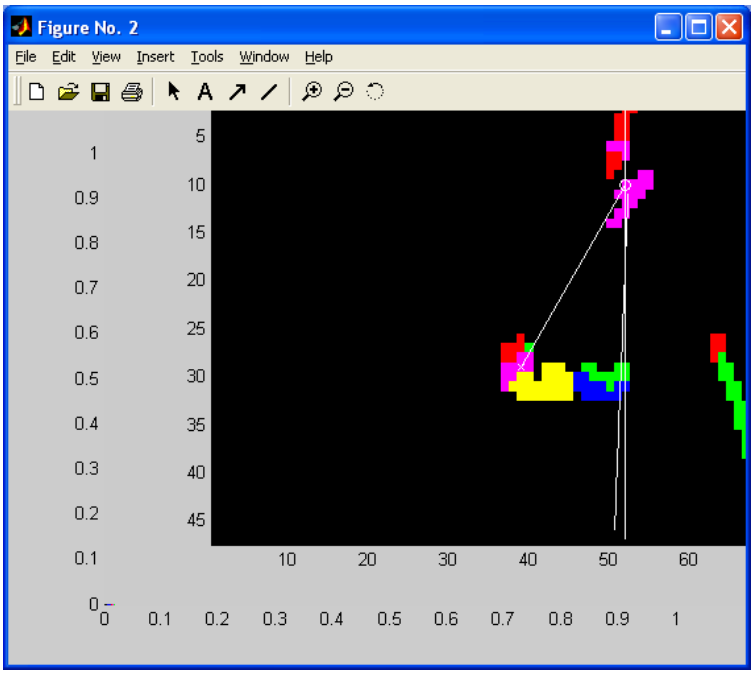

Figure 49: End – Place BlueRing, Begin - Reach

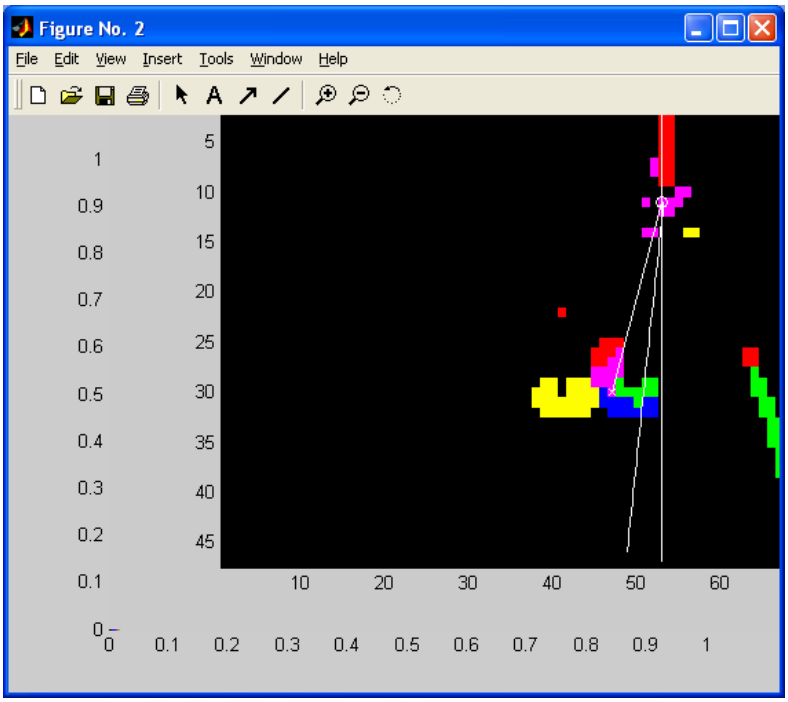

Figure 50: End – Reach RedRing, Begin - Place

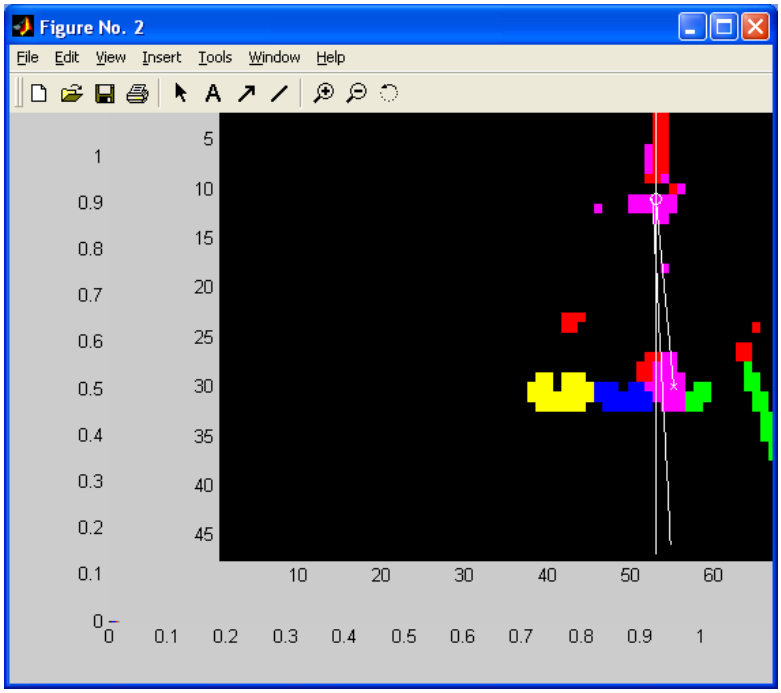

Figure 51: End – Place RedRing, Begin - Reach

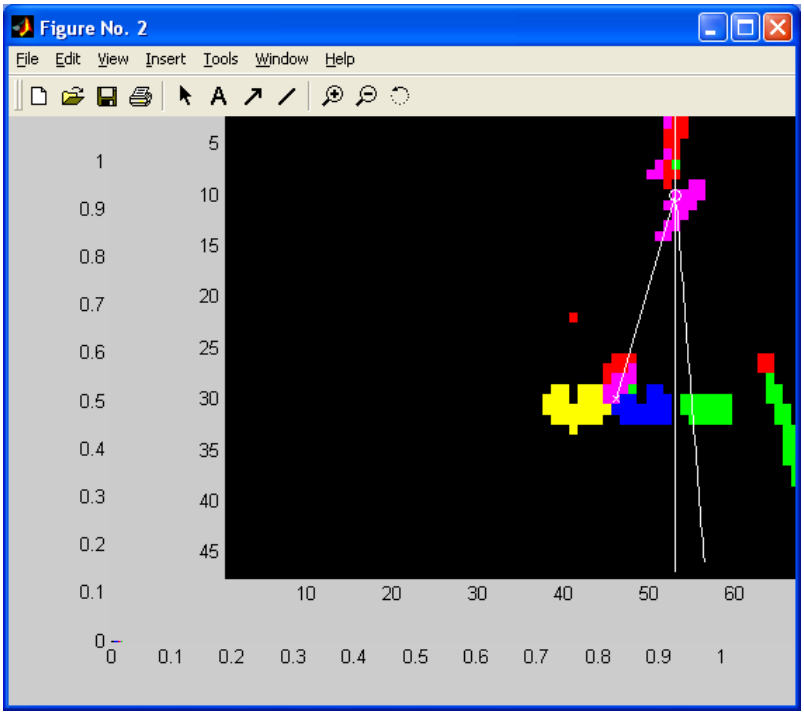

Figure 52: End – Reach YellowRing, Begin – Place

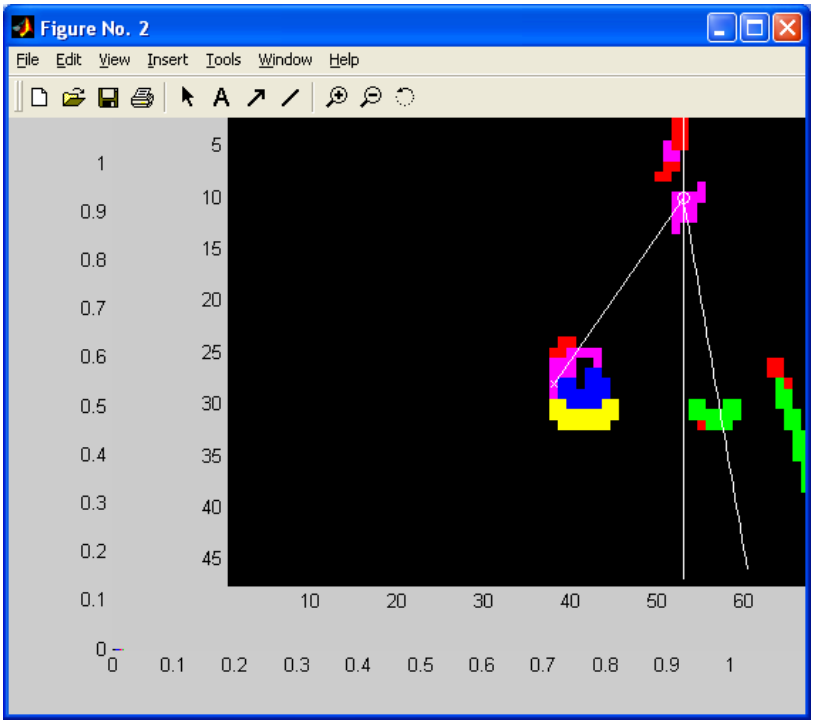

Figure 53: End – Place YellowRing, Begin - Reach

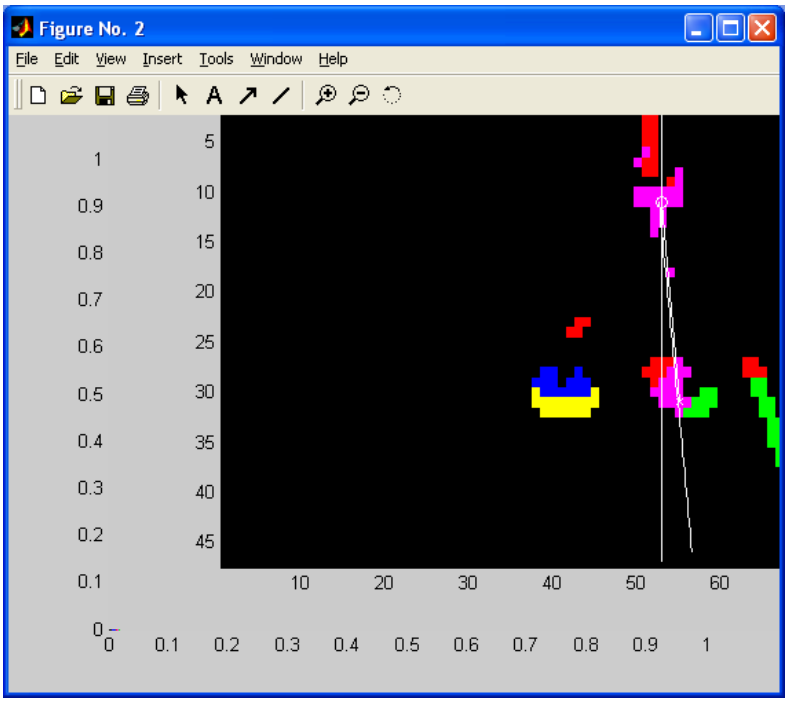

Figure 54: End - Reach RedRing, Begin – Place

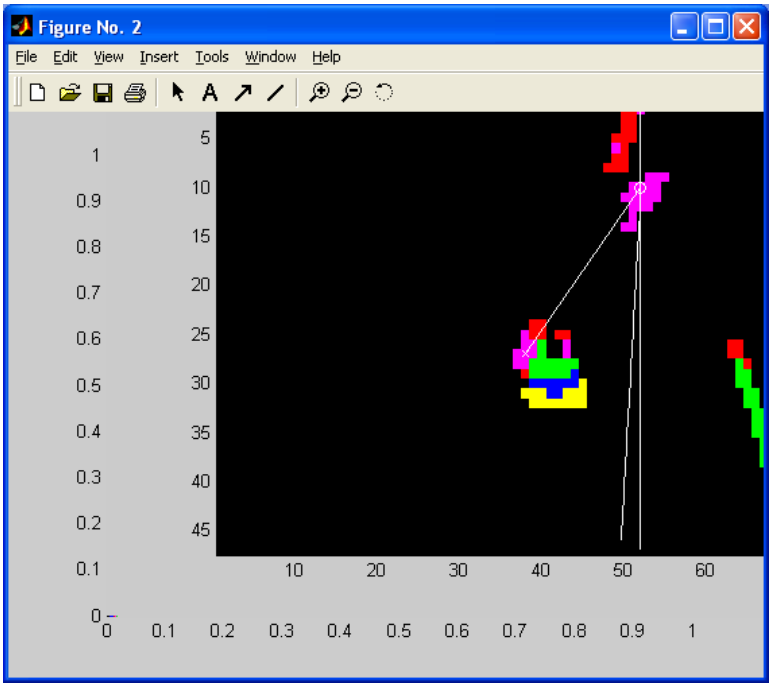

Figure 55: End – Place RedRing, Begin – Move to rest

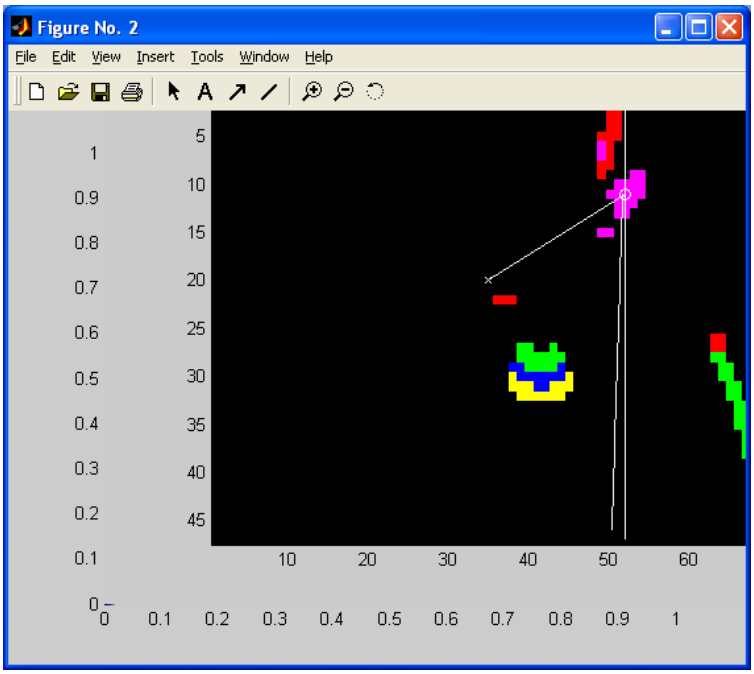

Figure 56: End – Move to Rest

CHAPTER VI

CONCLUSION

The use of fuzzy membership functions for object recognition is one of the better ways to develop a learning object recognition system. This system takes advantage of the classification trees, which allows it to relate to objects through the feature vectors in a manner that is easy to understand. The simple training process allows for many different features to be added. These features are can be used in many interesting manners, due to the way they are stored. The motion image can be used to find when motion below the hand is at a maximum to determine if the hand is moving a object. Just as the hand's position can be stored, other objects (i.e., a red strip) can be used to track other significant motions in the image.

The processing of the videos takes a long time. The goal of the system is to run in real-time eventually. This slowness may be due to MATLAB not being the optimal system to run the image processing. The code will be ported to C++ to determine how much faster it can be processed.

The low resolution does not allow for the detection of small hand movements. Increasing the resolution increases the process time quite a bit. This problem may also be improved if the port to C++ will be a significant decrease in processing time.

The object recognition of images can be used to add to the training vector by taking the window samples that are correctly identified. Using this method, the number of samples will increase dramatically. As the number of training vectors grows larger, storage will become an issue. Since the vectors are mostly zero, they can be stored as a sparse vector to solve the problem immediately. In the long run, it will be necessary to create a general model for objects to conserve space.

Morphology could be used to reduce noise in recognition. This process would help reduce the random noise spots that occur in the recognition process.

Combining the training sets could allow for subject identification for the same tasks. The subjects that are asked to do the same tasks may have different enough skin tones to be identified. It is likely that the color histogram would need to be able to represent a high number of colors in the skin tone.

The ability for a robot to create descriptions of a feature vector and motions may lead to instruction. These instructions can be used to teach other robots how to do certain tasks. Their feature vectors will identify objects. The human's hand can be tracked to observe the behavior. The robot can track its own hand and mimic the behavior observed from the human. In terms of navigation, a robot can observe the features of a pathway and use a simple cost/reward system to learn about objects in its environment.

There are many different paths to explore with this system. A possible application is to be used as the perception module in a working memory system.

APPENDIX

addtoImemory.m

```
function Imemory = addtoImemory(Imemory, picturename, classlabel, picnum)

nextvec = length(Imemory) + 1;
X = aviread([picturename],picnum);
x = X.cdata;
figure(1), image(x)
[xcrop, xrect] = imcrop;
xrect = round(xrect);
Imemory(nextvec).picture = picturename;
Imemory(nextvec).picturenum = picnum;
Imemory(nextvec).upperleftrow = xrect(2) ;
Imemory(nextvec).upperleftcol = xrect(1);
Imemory(nextvec).width = xrect(3);
Imemory(nextvec).height = xrect(4);
Imemory(nextvec).class = classlabel;

set(rectangle('Position',[ Imemory(nextvec).upperleftcol Imemory(nextvec).upperleftrow
Imemory(nextvec).width Imemory(nextvec).height]), 'Edgecolor', [0 1 0])
```

addtoVideostruct.m

```
function Videostruct = addtoVideostruct(Videostruct, picturename)

nextvec = length(Videostruct)+1;
Videostruct(nextvec).picturename = picturename;
```

analyzetree.m

```
t = treefit(FV, Landmarks);

classtotals = t.classcount(1,:);        % total numbers of training vectors for each class
leaves = find( t.children(:,1) == 0);   % list of leaf nodes

leafpercentages = t.classcount(leaves,:) ./ (ones(size(leaves))*classtotals); %percentage of each
class in each leaf node

[maxpercentage, maxpercentageclass]= max( leafpercentages' );

% Find max percentage class for each leaf node and sort from largest to
% smallest
[maxpercentage, sortorder] = sort(maxpercentage);
maxpercentage = fliplr(maxpercentage);
sortorder = fliplr(sortorder);
maxpercentageclass = maxpercentageclass(sortorder);
leaves = leaves(sortorder);

decisionvars = t.var(t.parent(leaves));  % feature number of each parent node of the leaf nodes
for i=1:length(leaves)
   if leaves(i) == t.children(t.parent(leaves(i)), 1)
     lotorlittle(i) = -1;
   else
     lotorlittle(i) = 1;
   end
end

%[leaves maxpercentage' decisionvars lotorlittle']

ruledepth = 40;
rulebase = [];
for i=1:length(leaves)
   temprule = zeros(1, ruledepth);
   leaf = leaves(i);
   temprule(1) = decisionvars(i)*lotorlittle(i);
   currparent = t.parent(leaf);
   ruleindex = 1;
   while currparent ~= 1
      currnode = currparent;
      currparent = t.parent(currnode);
      children = t.children(currparent) == currnode;
      ruleindex = ruleindex + 1;
      if children(1) == 1
         temprule(ruleindex) = -1*t.var(currparent);
```

```
        else
            temprule(ruleindex) = t.var(currparent);
        end
    end
    rulebase = [rulebase ; temprule];
end

[leaves maxpercentage' rulebase]
t.classname(t.class(leaves))
```

analyzetreeseq.m

```
FV(:,253) = 0;

leavestotal = [];
maxpercentagetotal = [];
rulebasetotal = [];
classnamestotal = [ ];
classtotal = [];
loopnumbertotal = [];

for loopcount = 1:15

analyzetree

[numberofrules, coltemp]=size(rulebase);

leavestotal = [leavestotal ; leaves];
maxpercentagetotal = [maxpercentagetotal ; maxpercentage'];
rulebasetotal = [rulebasetotal ; rulebase];
classnamestotal = [classnamestotal ; t.classname(t.class(leaves)) ];
classtotal = [classtotal ; t.class(leaves) ];
loopnumbertotal = [loopnumbertotal ; loopcount*ones(numberofrules,1)];

FV(:, t.var(1) ) = 0;

end
```

applyimagerule2.m

```
function [Decisionmat2, mask2] = applyimagerule2(features, rulevec, nrows, ncols)

[numrules, coltmp]= size(rulevec);
mask2 = zeros(size( features(:,1) ));
[a b]=size(features);
Decisionmat2 = [];
Rule=[];
for rule = 1:numrules
  mask = ones(size( features(:,1) ));
  i=1;
  while (rulevec(rule,i) ~= 0) %while loop processes one minterm at a time
    temp = features(:, abs(rulevec(rule,i)) );
    if sum(temp) ~= 0
      if rulevec(rule,i) > 0
        tempmask = temp/max(temp);

      else
        tempmask = max(temp)/max(temp) - temp/max(temp);

        %tempmask = ones(size(temp));
      end
      mask = mask .* tempmask;
    end

    % Decisionmat2 = [Decisionmat2 mask];
    % Now mask contains one minterm's results
    i = i+1;
  end
  Decisionmat2 = [Decisionmat2 mask];
  mask2 = mask2 + mask;
end
```

applyrules2.m

```
classmasks=[];
classmasksnorm=[];
maxoverall=zeros(max(size(features)),1);
CLASSoverall=zeros(max(size(features)),1);
for incre=1:length(t.classname)
   classnum=incre;
Numterms = loopnumbertotal(length(loopnumbertotal));

III = find(classtotal == classnum);
classrules = rulebasetotal(III, :);
loopclass = loopnumbertotal(III);

%mask = ones(numrows*numcols, 1);
[featrows, featcols] = size(features);
mask = zeros(featrows, 1);
Decisionmatim=[];
Rulematrixim=[];
for lcount = 1:Numterms
  % get rules from tree number lcount
  I = find(loopclass == lcount);
  looprules = classrules(I,:);

  [Decisionmat2, mask2] = applyimagerule2(features, looprules, 1, 1);
  Rulematrixim=[Rulematrixim; looprules];
  mask2 = mask2 / sum(mask2);
  Decisionmatim=[Decisionmatim Decisionmat2];
  mask = mask + mask2;
end
classmasks=[classmasks mask];
classmasksnorm=[classmasksnorm mask./max(mask)];
classmasksnorm2 = mask./max(mask) > 0.15;
classmasksnorm2 = mask./max(mask) .* classmasksnorm2;
CLASSoverall= classnum .* (classmasksnorm2 > maxoverall) + CLASSoverall .*
(classmasksnorm2 <= maxoverall);
maxoverall=classmasksnorm2 .* (classmasksnorm2 > maxoverall) + maxoverall .*
(classmasksnorm2 <= maxoverall);
end

% remaxoverall = reshape(maxoverall,numrows,numcols);
% reclassoverall = reshape(classoverall,numrows,numcols);


cmap = [
   0 0 0
```

```
1 0 0
0 1 0
0 0 1
1 1 0
1 0 1
0 1 1
0.5 0.5 0.5
1 1 1];
```

getclassesfromImemory.m

```
function Landmarks = getclassesfromImemory(Imemory)

for i=1:length(Imemory)
   Landmarks(i) = {Imemory(i).class};
end
```

getfeaturesfromImemory.m

```
function FV = getfeaturesfromImemory(Imemory)

FV = [];

for i=1:length(Imemory)

   Im = Imemory(i);
   Irect = [Im.upperleftcol Im.upperleftrow Im.width Im.height];
   picnum = Im.picturenum;
   M = aviread([Im.picture],picnum);
   x = M.cdata;
   M = aviread([Im.picture],picnum-1);
   xim1 = M.cdata;
   features = getregionfeaturesfromImemory(x, Irect ,xim1);
   FV = [FV ; features];
   i
end
```

getregionfeaturesfromImemory.m

```
function features = getregionfeaturesfromImemory(x,rect,xim1)


rect = round(rect);
xsec = x( rect(2):(rect(2)+rect(4)) , rect(1):(rect(1)+rect(3)) , : );
xim1sec = xim1( rect(2):(rect(2)+rect(4)) , rect(1):(rect(1)+rect(3)) , : );
y = rgb2hsv(xsec);
yh = y(:,:,1);
ys = y(:,:,2);
yv = y(:,:,3);

yg = rgb2gray(x);
ygsec = rgb2gray(xsec);

Nfilt = 15;
Ndec = 10;

havg = ones(Nfilt, Nfilt)/(Nfilt^2);

[nrows, ncols] = size(yh);

hues = [0 .05 .14 .22 .28 .45 .54 .75 .81 .92 1];
sats = 0:(1/5):1;
vals = 0:(1/5):1;

features = [];
for h = 1:length(hues)-1
  for s = 1:length(sats)-1
    for v = 1:length(vals)-1
      if  (h==1)
        tmph = (hues(h)<=yh)&(yh<=hues(h+1));
      else
        tmph = (hues(h)<yh)&(yh<=hues(h+1));
      end
      if  (s==1)
        tmps = (sats(s)<=ys)&(ys<=sats(s+1));
      else
        tmps = (sats(s)<ys)&(ys<=sats(s+1));
      end
      if  (v==1)
        tmpv = (vals(v)<=yv)&(yv<=vals(v+1));
      else
        tmpv = (vals(v)<yv)&(yv<=vals(v+1));
      end
```

```
        colorseg = tmph .* tmps .* tmpv;
        numcolors = sum(colorseg(:));
        features = [features numcolors/(nrows*ncols)];

      end
    end
end

hlap = [-1 -1 -1
    -1 8 -1
    -1 -1 -1]/8;whos
texture1 =  abs(filter2(hlap, ygsec)/256);
texture1 = sum(texture1(:))/(nrows*ncols);
features = [features texture1];


motsec = abs(double(rgb2gray(xsec)) - double(rgb2gray(xim1sec)));
% motsec2 = filter2(havg, abs(filter2(hlap, motsec)/256));
motsec2 = sum(motsec(:)/(nrows*ncols));
features = [features motsec2];

%texture2 = edge(yg, 'canny');
%texture2 = texture2( rect(2):(rect(2)+rect(4)) , rect(1):(rect(1)+rect(3)) );
%texture2 = sum(texture2(:))/(nrows*ncols);
% texture2 = 0;
% features = [features texture2];

%disp('Adding phase symmetry feature');
%Phase Symmetry Feature - not using - just give zero
%[phaseSym orientation] = phasesym(yg);
%phaseSym = phaseSym( rect(2):(rect(2)+rect(4)) , rect(1):(rect(1)+rect(3)) );
features = [features 0];

%disp('Adding line feature');
%features2 = [];
%eg = edge(yg, 'canny');
%xwin= eg( rect(2):(rect(2)+rect(4)) , rect(1):(rect(1)+rect(3)) );
%[xwinr,xp]=radon( xwin ,0:179);
%line = max(xwinr)/(nrows*ncols);
%features2 = line;
features2 = zeros(1, 180);
features = [features features2];
```

processthevideo.m

```
% You must choose the set of rules you want the video to be processed by
disp('Chose the set of rules to be used in processing');
theselectedrules = uigetfile('*.mat')
load ([theselectedrules])

disp('Chose the object to be followed for angle analysis');
clnum=menu('Object for angle analysis',t.classname);

% The video to be processed
disp('What is the video to be processed (name.avi):');
videoname = uigetfile('*.avi')
MaxFrame = input('What is the section of frames in the video ([begin# end#]): ');

% The name of the video to be created
processedvideoname = input('What is the name of the new video (name.avi): ','s');

Videostruct = struct('picturename',{'nil'},'features',0);
if MaxFrame(1) <= 1
   MaxFrame(1)=2;
end
for place=2:MaxFrame(2)-MaxFrame(1)+2 %2 till number of frames in video
   sfilename=videoname;
   Videostruct = addtoVideostruct(Videostruct, sfilename);
   Videostruct(place).picturenum=MaxFrame(1)+place-2;
   place
end

for place=2:length(Videostruct)
   Iobj = Videostruct(place);
   X = aviread([Iobj.picturename],Iobj.picturenum);
   xi = X.cdata;
   x = xi;

   X = aviread([Iobj.picturename],(Iobj.picturenum)-1);
   xim1 = X.cdata;
   psychimagefeatures4
   Iobj.features=sparse(features);
   Videostruct(place)=Iobj;
   place
   clear features x xi xim1 yq yhq ysq ys yh yvq yg texture2 texture1 features2 colorfilt y yv
ctmp tmp havg colsamp rowsamp hues hlap vals stats
end
```

```
Videostruct = Videostruct(2:length(Videostruct));
processvideo2
```

processvideo2.m

```
rowhand = [];
colhand = [];
loadnum=0;
lastframe=0;
cmap = [
   0 0 0
   1 0 0
   0 1 0
   0 0 1
   1 1 0
   1 0 1
   0 1 1
   0.5 0.5 0.5
   1 1 1];

% load ([theselectedrules])
% ovid=avifile('originalvid','compression','None');
% mvid=avifile('motionvid','colormap',gray(256),'compression','None');
% hvid=avifile('handtrackingvid','colormap',gray(256),'compression','None');
rvid=avifile([processedvideoname],'colormap',cmap(1:length(t.classname)+1,:),'compression','no
ne')
% fig1=figure(1)
% fig2=figure(2)
% fig3=figure(3)
fig4=figure(4);
% set(fig1,'DoubleBuffer','on')
% set(fig2,'DoubleBuffer','on')
% set(fig3,'DoubleBuffer','on')
set(fig4,'DoubleBuffer','on')
rowshand = [];
colshand = [];
frames = [];
gazeangle = [];
handobjmot = [];
for i=1:(MaxFrame(2) - MaxFrame(1)) + 1 % one minus the total number of frames in video

%     if i >= lastframe
%         loadnum=loadnum+1;
%         vidnstr=sprintf('%0.3d',loadnum);
%         load (['videostruct' vidnstr]);

features=Videostruct(i).features;

i
```

```
frames = [frames Videostruct(i).picturenum];

X = aviread([Videostruct(i).picturename],Videostruct(i).picturenum);
xi = X.cdata;

X = aviread([Videostruct(i).picturename],(Videostruct(i).picturenum)-1);
xim1 = X.cdata;

yi = rgb2hsv(xi);

% figure(1), image(xi)
% set(fig1,'Erasemode','xor')
% ovid=addframe(ovid,uint8(xi));

% mot = abs(double(rgb2gray(xi)) - double(rgb2gray(xim1)));
% fig2, imagesc(uint8(mot)), colormap(gray(256))
% set(fig2,'Erasemode','xor')
% gframe2=getframe(fig2);
% mvid=addframe(mvid,gframe2);

% mask = (yi(:,:,1)>0.03) & (yi(:,:,1) < 0.09);
% skin = yi(:,:,2).*yi(:,:,3).*mask;
% hand = skin .* mot;
%
% fig3, imagesc(uint8(hand)), colormap(gray(256))
%
% havg2 = ones(15,15)/(15^2);
% % handblur = filter2(havg, hand);
% handblur=filter2(havg2, abs(filter2((ones(3,3)/9), mot)/256));
% handmax = max(max(handblur));
% [II,JJ]= find( handblur == handmax );
%
% hold on
% plot(JJ,II,'rx')
% hold off
%
% set(fig3,'Erasemode','xor')
% gframe3=getframe(fig3);
% mvid=addframe(hvid,gframe3);

[nrows ncols]=size(xi(:,:,1));
Nfilt = 15;
Ndec = 10;
rad = (Nfilt+1)/2;
rowsamp = rad:Ndec:(nrows - rad);
colsamp = rad:Ndec:(ncols - rad);
```

```
numrows = length(rowsamp);
numcols = length(colsamp);

mot = abs(double(rgb2gray(xi)) - double(rgb2gray(xim1)));
% fig2, imagesc(uint8(mot)), colormap(gray(256))
% set(fig2,'Erasemode','xor')
% gframe2=getframe(fig2);
% mvid=addframe(mvid,gframe2);

mask = (yi(:,:,1)>0.03) & (yi(:,:,1) < 0.09);
skin = yi(:,:,2).*yi(:,:,3).*mask;
hand = skin .* mot;
%
% fig3, imagesc(uint8(hand)), colormap(gray(256))
%
havg2 = ones(15,15)/(15^2);
% handblur = filter2(havg, hand);
handblur=filter2(havg2, abs(filter2((ones(3,3)/9), hand)/256));
handblur=handblur(rowsamp,colsamp);
handmax = max(max(handblur));
[II,JJ]= find( handblur == handmax );
II=II(1);
JJ=JJ(1);
rowshand=[rowshand II];
colshand=[colshand JJ];


motionimg = reshape(features(:,252),numrows,numcols);
distbelowhand = 4;
heightbox = 4;
halfwidthbox = 3;
if ((II + distbelowhand + heightbox) <= numrows) & ((JJ - halfwidthbox) >= 1) & ((JJ +
halfwidthbox) <=numcols)
   handobjmot = [handobjmot sum(sum(motionimg((II + distbelowhand):(II + distbelowhand +
heightbox),(JJ - halfwidthbox):(JJ + halfwidthbox))))];
else
   if (JJ - halfwidthbox) < 1
      if ((II + distbelowhand + heightbox) > numrows)
         if (II + distbelowhand) > numrows
            handobjmot = [handobjmot sum(sum(motionimg((numrows:numrows),1:(JJ +
halfwidthbox))))];
         else
            handobjmot = [handobjmot sum(sum(motionimg((II + distbelowhand):numrows,1:(JJ
+ halfwidthbox))))];
         end
      else
```

```
        handobjmot = [handobjmot sum(sum(motionimg((II + distbelowhand):(II +
distbelowhand + heightbox),1:(JJ + halfwidthbox))))];
      end
   elseif (JJ + halfwidthbox) > numcols
      if ((II + distbelowhand + heightbox) > numrows)
         if (II + distbelowhand) > numrows
            handobjmot = [handobjmot sum(sum(motionimg((numrows:numrows),(JJ -
halfwidthbox):numcols)))];
         else
            handobjmot = [handobjmot sum(sum(motionimg((II + distbelowhand):numrows,(JJ -
halfwidthbox):numcols)))];
         end
      else
         handobjmot = [handobjmot sum(sum(motionimg((II + distbelowhand):(II +
distbelowhand + heightbox),(JJ - halfwidthbox):numcols)))];
      end
   else
      if ((II + distbelowhand + heightbox) > numrows)
         if (II + distbelowhand) > numrows
            handobjmot = [handobjmot sum(sum(motionimg((numrows:numrows),(JJ -
halfwidthbox):(JJ + halfwidthbox))))];
         else
            handobjmot = [handobjmot sum(sum(motionimg((II + distbelowhand):numrows,(JJ -
halfwidthbox):(JJ + halfwidthbox))))];
         end
      end
   end
end



applyrules2
reshapepic=reshape(CLASSoverall, numrows, numcols);
[ycoor,xcoor]=find(reshapepic==clnum);
xcoor=xcoor(find(ycoor < 19));
ycoor=ycoor(find(ycoor < 19));
p=polyfit(ycoor,xcoor,1);
xp=[1:numcols];
linep=p(1)*xp + p(2);
linep=linep(find(xp > 10 & xp < numrows));
xp=xp(find(xp > 10 & xp < numrows));


centerpixel=[round(sum(xcoor)/(length(xcoor))) round(sum(ycoor)/(length(ycoor)))];
xp2 = [1:numrows];
midhatline = 0*xp2 + centerpixel(1);
```

```
vecstrip =[xp(length(xp))-centerpixel(2) ; linep(length(linep))-centerpixel(1) ;];
vecmidhat = [xp2(length(xp2))-centerpixel(2) ;midhatline(length(midhatline))-centerpixel(1)];
vechand = [ II-centerpixel(2) ; JJ-centerpixel(1)];
angletostrip = acos((vecstrip' * vecmidhat)/(sqrt(sum(vecstrip.^2)) * sqrt(sum(vecmidhat.^2))));
angletohand = acos((vechand' * vecmidhat)/(sqrt(sum(vechand.^2)) * sqrt(sum(vecmidhat.^2))));
angletostrip = angletostrip*180/pi;
angletohand = angletohand*180/pi;
gazeangle = [gazeangle angletostrip];


fig4, imagesc(uint8(reshapepic)), colormap(cmap(1:classnum+1,:))
hold on
plot(linep,xp,'w')
plot(centerpixel(1),centerpixel(2),'wo')
plot([JJ; centerpixel(1)] , [II; centerpixel(2)], 'w')
plot(midhatline,xp2,'w')
plot(JJ,II,'wx')
hold off
% fig4,title(['Hand position  x:',num2str(JJ),'  y:',num2str(II),'  Angle to Hand:
',num2str(angletohand),'  Angle to Strip: ', num2str(angletostrip)]);
drawnow
gframe=getframe(fig4);
rvid=addframe(rvid,gframe);

% if i > 5
%    rowhand = [rowhand II];
%    colhand = [colhand JJ];
% end
% if i == 2
%     pause
% end
% pause( 0.05 )

end
% ovid=close(ovid);
% mvid=close(mvid);
% hvid=close(hvid);
dlmwrite('Frames.xls',frames,'\n')
dlmwrite('Angle_of_Gaze.xls',gazeangle,'\n')
dlmwrite('Y-axis_Hand_Values.xls',rowshand,'\n')
dlmwrite('X-axis_Hand_Values.xls',colshand,'\n')
dlmwrite('HandObjectMotion.xls',full(handobjmot),'\n')
dlmwrite('RowHands.xls',rowshand,'\n')
dlmwrite('ColHands.xls',colshand,'\n')
save HandObjectMotion handobjmot
```

```
save RowHands rowshand
save ColHands colshand
rvid=close(rvid);
```

psychimagefeatures4.m

```
y = rgb2hsv(x);
yh = y(:,:,1);
ys = y(:,:,2);
yv = y(:,:,3);

yg = rgb2gray(x);

[nrows, ncols] = size(yh);

hues = [0 .05 .14 .22 .28 .45 .54 .75 .81 .92 1];
sats = 0:(1/5):1;
vals = 0:(1/5):1;

Nfilt = 15;
Ndec = 10;

havg = ones(Nfilt, Nfilt)/(Nfilt^2);

features = [];

rad = (Nfilt+1)/2;

rowsamp = rad:Ndec:(nrows - rad);
colsamp = rad:Ndec:(ncols - rad);
numrows = length(rowsamp);
numcols = length(colsamp);


disp('Adding color features');

yhq = zeros(size(yh));
for i=2:10
  yhq = yhq + ((yh > hues(i))&(yh <= hues(i+1)))*i;
end
yhq = yhq + ((yh >= hues(1))&(yh <= hues(2)))*1;

ysq = zeros(size(yh));
for i=2:5
  ysq = ysq + ((ys > sats(i))&(ys <= sats(i+1)))*i;
end
ysq = ysq + ((ys >= sats(1))&(ys <= sats(2)))*1;

yvq = zeros(size(yh));
for i=2:5
```

```
  yvq = yvq + ((yv > vals(i))&(yv <= vals(i+1)))*i;
end
yvq = yvq + ((yv >= vals(1))&(yv <= vals(2)))*1;

yq = (yhq-1)*25 + (ysq-1)*5 + (yvq-1);

        rowindx = 0;
        colorfilt = [];
        for jcol = colsamp
           for irow = rowsamp
              tmp = yq( (irow-((Nfilt-1)/2)):(irow+((Nfilt-1)/2)) , (jcol-((Nfilt-1)/2)):(jcol+((Nfilt-
1)/2)) );
              ctmp = hist(tmp(:), 0:249);
              colorfilt = [colorfilt ; ctmp];
           end
        end



features = colorfilt/(Nfilt^2);

[reducedrows, reducedcols] = size(colorfilt);

disp('Adding texture feature');
%Laplacian Texture Feature
hlap = [-1 -1 -1
   -1 8 -1
   -1 -1 -1]/8;
texture1 = filter2(havg, abs(filter2(hlap, yg)/256));
texture1 = texture1(rowsamp, colsamp );
features = [features texture1(:)];

disp('Adding motion detection feature');
%Canny Edge Feature
mot = abs(double(rgb2gray(xi)) - double(rgb2gray(xim1)));
% mot2 = filter2(havg, abs(filter2(hlap, mot)/256));
mot2h=[];
for jcol = colsamp
   for irow = rowsamp
      mot2 = mot((irow-((Nfilt-1)/2)):(irow+((Nfilt-1)/2)) , (jcol-((Nfilt-1)/2)):(jcol+((Nfilt-
1)/2)));
      mot2 = sum(mot2(:)/(length(mot2(:))));
      % texture2 = zeros(size(texture1));
      mot2h=[mot2h; mot2];
   end
end
```

```
features = [features mot2h];

% disp('Adding canny edge feature');
% %Canny Edge Feature
% %texture2 = filter2(havg, edge(yg, 'canny'));
% %texture2 = texture2(rowsamp, colsamp);
% texture2 = zeros(size(texture1));
% features = [features texture2(:)];

disp('Adding phase symmetry feature');
%Phase Symmetry Feature - removed - just add zeros here
features = [features zeros(numrows*numcols,1)];

%disp('Adding line feature');
%features2 = [];
%eg = edge(yg, 'canny');
%for j = colsamp
%    for i = rowsamp
%        xwin = eg((i-rad+1):(i-rad+Nfilt),(j-rad+1):(j-rad+Nfilt));
%        [xwinr,xp]=radon( xwin ,0:179);
%        line = max(xwinr)/(Nfilt^2);
%        features2 = [features2
%                line];
%    end
%end

features2 = zeros(numrows*numcols, 180);
features = [features features2];

[numrows,numcols]=size(texture1);
```

trainingthevideo.m

```
Imemory =
struct('picture',0,'picturenum',0,'upperleftrow',0,'upperleftcol',0,'width',0,'height',0,'class',0,'featur
es',0);
newname = input('What is the new rule matrix name (name.mat): ','s');

disp('What is the video to be used for gathering training data:');
videoname = uigetfile('*.avi')


MaxFrame = input('What is the maximum number of frames in the video (number): ');
NUMobject = input('What is the number of objects (number): ');

for NUMob = 1:NUMobject
   classlabel=input('What is the name of the object (i.e BluePaper, Hand): ','s');
   samptemp=input('How many samples for the object (50 or so): ');
   numimage = round(2:MaxFrame/samptemp:MaxFrame);
   for ntemp=1:samptemp
      Imemory = addtoImemory(Imemory, videoname, classlabel, numimage(ntemp));
   end
   close all
end
Imemory=Imemory(2:length(Imemory));
FV = getfeaturesfromImemory(Imemory);
Landmarks = getclassesfromImemory(Imemory);
analyzetreeseq
save (newname)
```

processtrain.m

```
load newfeaturesImemory8
choice=0;
choice2=0;
choice3=0;
choice4=0;
choice5=0;
savedminterms=[];
Isimageloaded=0;
Ismintermsaved=0;
scrsz = get(0,'ScreenSize');
while choice ~= 1
   choice=menu('Menu','Quit Program','Compare one set of class rules with one set of class
training data','Compare one set of class rules with all class training data','Load Image','Analyze
image with rules','Analyze image with saved minterms','Clear saved minterms','Load different
Imemory','Close figures and Clear Command Window');
   if choice==2
      while(choice2 ~= 1)
         choice2=0;
         sumvectdata=[];
         classnum=menu('Rules for which class',t.classname);
         classvecs=menu('Feature vectors of which class',t.classname);
         mintermmat
         % figure, imagesc(Decisionmat)
         % title(['Image of ',char(t.classname(classnum)),' rules vs ', char(t.classname(classvecs)),'
feature vectors'])
         % xlabel('Minterm labels')
         % ylabel('Feature vectors')
         % figure, plot(sum(Decisionmat))
         % title(['Sum of Decisionmat for ',char(t.classname(classnum)),' rules vs ',
char(t.classname(classvecs)),' feature vectors'])
         % xlabel('Minterm labels')
         figure('Position',[10 50 scrsz(3)*.985 scrsz(4)*2/6]), plot(mean(Decisionmat))
         title(['Average of Decisionmat for ',char(t.classname(classnum)),' minterms vs ',
char(t.classname(classvecs)),' feature vectors'])
         xlabel('Minterm labels')
         % figure, plot(max(Decisionmat))
         % title(['Max of Decisionmat for ',char(t.classname(classnum)),' rules vs ',
char(t.classname(classvecs)),' feature vectors'])
         % xlabel('Minterm labels')
         % figure, plot(median(Decisionmat))
         % title(['Median of Decisionmat for ',char(t.classname(classnum)),' rules vs ',
char(t.classname(classvecs)),' feature vectors'])
         % xlabel('Minterm labels')
         while choice2 ~= 1 & choice2 ~= 2
```

```
            choice2=menu('Options', 'Previous Menu' , 'Compare another set of rules', 'Examine a
minterm','Save Minterm','Clear saved minterms');
            if choice2 == 3
               mintermdisplay(Rulematrix);
            elseif choice2 == 4
               savedminterms=savemintermsauto(Rulematrix,-1,savedminterms)
               Ismintermsaved=1;
            elseif choice2 == 5
               savedminterms=[]
               Ismintermsaved=0;
            end
         end
      end
   elseif choice == 3
      while choice3 ~= 1
         choice3=0;
         % Chooses the landmark
         classnum=menu('Rules for which class',t.classname);
         sumvectdata=[];
         % Loop extracts all of the Landmarks to compare with chosen Landmark
         for X1=1:length(t.classname)
            classvecs=X1;
            mintermmat
            % figure, imagesc(Decisionmat)
            % title(['Image of ',char(t.classname(classnum)),' rules vs ',
char(t.classname(classvecs)),' feature vectors'])
            % xlabel('Minterm labels')
            % ylabel('Feature vectors')
            % figure, plot(sum(Decisionmat))
            % title(['Sum of Decisionmat for ',char(t.classname(classnum)),' rules vs ',
char(t.classname(classvecs)),' feature vectors'])
            % xlabel('Minterm labels')
            % figure, plot(mean(Decisionmat))
            % title(['Average of Decisionmat for ',char(t.classname(classnum)),' rules vs ',
char(t.classname(classvecs)),' feature vectors'])
            % xlabel('Minterm labels')
            % functifigure, plot(max(Decisionmat))
            % title(['Max of Decisionmat for ',char(t.classname(classnum)),' rules vs ',
char(t.classname(classvecs)),' feature vectors'])
            % xlabel('Minterm labels')
            % figure, plot(median(Decisionmat))
            % title(['Median of Decisionmat for ',char(t.classname(classnum)),' rules vs ',
char(t.classname(classvecs)),' feature vectors'])
            % xlabel('Minterm labels')
         end
         discrimratio=discriminateratio(sumvectdata,classnum,length(t.classname));
```

```
        figure, plot(discrimratio)
        title(['Discriminate Ratio Plot of ',char(t.classname(classnum)),' minterms vs all other
classes feature vectors'])
        xlabel('Minterm labels')
        while choice3 ~= 1 & choice3 ~= 2
           choice3=menu('Options', 'Previous Menu' , 'Compare another class rules to all class
vectors', 'Examine a minterm','Save Minterm','Clear saved minterms');
           if choice3 == 3
              mintermdisplay(Rulematrix)
           elseif choice3 == 4
              savedminterms=savemintermsauto(Rulematrix,discrimratio,savedminterms)
              Ismintermsaved=1;
           elseif choice3 == 5
              savedminterms=[]
              Ismintermsaved=0;
           end
        end
     end
  elseif choice == 4
     choice5 = menu('Image Options', 'Feature extracted', 'New image');
     if choice5 == 1;
        load (uigetfile('*.mat'))
        figure('Position',[10 50 scrsz(3)*.985 scrsz(4)*2/6]), image(x), title(sfilename)
     elseif choice5 == 2
        sfilename = uigetfile('*.bmp');
        x = imread(sfilename);
        %unfold, x = y;
        [xnrows, xncols, xnplanes] = size(x);
        %x = imresize(x, 256/xnrows);
        figure('Position',[10 50 scrsz(3)*.985 scrsz(4)*2/6]), image(x), title(sfilename)
        imagefeatures3  %main output is features, a matrix of features vectors stored as rows
     end
     Isimageloaded=1;
  elseif choice == 5 & Isimageloaded == 1
     process
     while choice4 ~= 1
        choice4 = menu('Options','Exit','Analyze pixel minterms and percentage','Save
Minterms','Analyze Minterms');
        if choice4 == 2

[foundarray,foundarrayperc]=getpixelminterms(mask,Decisionmatim,Rulematrixim,numrows,nu
mcols);
        elseif choice4 == 3
           savedminterms=savemintermsauto(Rulematrixim,foundarrayperc,savedminterms);
           Ismintermsaved=1;
        elseif choice4 == 4
```

```
            mintermdisplay(Rulematriximi)
          end
        end
    elseif choice == 6 & Isimageloaded == 1 & Ismintermsaved == 1
        process
    elseif choice == 7
        savedminterms=[]
        Ismintermsaved=0;
    elseif choice == 8
        load (uigetfile('*.mat'))
    elseif choice == 9
        clc
        close all
        if Isimageloaded == 1
            figure('Position',[10 50 scrsz(3)*.985 scrsz(4)*2/6]), image(x), title(sfilename)
        end
    else
        if Isimageloaded == 0
            disp('You must load an image first');
        end
        if Ismintermsaved == 0 & choice == 6
            disp('You must save minterms before applying them');
        end
    end
    choice3=0;
    choice2=0;
    choice4=0;
    choice5=0;
end
```

mintermmat.m

```
% Finds the row positions of the Landmark number (classvecs)
Iclass = find(strcmp(t.classname(classvecs) , Landmarks));


% Extracts the features of the Landmark number
featuresT=FV(Iclass,:);

% Numterms = the number of trees that were analyzed to make the rules
Numterms = loopnumbertotal(length(loopnumbertotal));

% Extracts the row positions of the chosen Landmark
III = find(classtotal == classnum);

% Extracts the minterms for all of the chosen Landmark
classrules = rulebasetotal(III, :);

% Extracts the tree number that each minterm was created
loopclass = loopnumbertotal(III);

%mask = ones(numrows*numcols, 1);
% Initialize variables
[featrows, featcols] = size(featuresT);
mask = zeros(featrows, 1);
Decisionmat=[];
Rulematrix=[];

% Get rules from tree number lcount
for lcount = 1:Numterms

  I = find(loopclass == lcount);
  looprules = classrules(I,:);

  [Decisionmat2, mask2] = applyimagerule2(featuresT, looprules, 1, 1);
  Rulematrix=[Rulematrix; looprules];
  mask2 = mask2 / sum(mask2);
  Decisionmat = [Decisionmat  Decisionmat2];
  mask = mask + mask2;
end
sumvectdata=[sumvectdata; sum(Decisionmat)];
%figure(2), imagesc( reshape(mask, numrows, numcols) ), colormap(gray(256))
```

mintermdisplay.m

```
function mintermdisplay(Rulematrix)

mindisplay=input('What is the Minterm label:  ');
minindex=find(Rulematrix(mindisplay,:) ~= 0);
Rulematrix(mindisplay,minindex)
figure
for(NN=1:length(Rulematrix(mindisplay,minindex)))
   [DESC,COLORHSV] = describefeature(abs(Rulematrix(mindisplay,NN)));
   if Rulematrix(mindisplay,NN) > 0
      if abs(Rulematrix(mindisplay,NN))>250

subplot(1,length(Rulematrix(mindisplay,minindex)),NN),image(255),colormap(gray(256))
      else
         ImA(1,1,:)=hsv2rgb(COLORHSV);
         subplot(1,length(Rulematrix(mindisplay,minindex)),NN), image(ImA)
      end
      subplot(1,length(Rulematrix(mindisplay,minindex)),NN), title(['Feature
',num2str(abs(Rulematrix(mindisplay,NN)))])
      subplot(1,length(Rulematrix(mindisplay,minindex)),NN), ylabel(DESC)
      subplot(1,length(Rulematrix(mindisplay,minindex)),NN), xlabel('A lot of this feature')
      disp(['Feature ',num2str(abs(Rulematrix(mindisplay,NN)))])
      disp(DESC)
      disp('A lot of this feature')
   else
      if abs(Rulematrix(mindisplay,NN))>250

subplot(1,length(Rulematrix(mindisplay,minindex)),NN),image(255),colormap(gray(256))
      else
         ImA(1,1,:)=hsv2rgb(COLORHSV);
         subplot(1,length(Rulematrix(mindisplay,minindex)),NN), image(ImA)
      end
      subplot(1,length(Rulematrix(mindisplay,minindex)),NN), title(['Feature
',num2str(abs(Rulematrix(mindisplay,NN)))])
      subplot(1,length(Rulematrix(mindisplay,minindex)),NN), ylabel(DESC)
      subplot(1,length(Rulematrix(mindisplay,minindex)),NN), xlabel('Little of this feature')
      disp(['Feature ',num2str(abs(Rulematrix(mindisplay,NN)))])
      disp(DESC)
      disp('Little of this feature')
   end
end
```

applymin2.m

```
% classnum = 4;

% Numterms = the number of trees that were analyzed to make the rules
% Numterms = loopnumbertotal(length(loopnumbertotal));
%
% III = find(classtotal == classnum);
% classrules = rulebasetotal(III, :);
% loopclass = loopnumbertotal(III);
%
% %mask = ones(numrows*numcols, 1);
% [featrows, featcols] = size(features);
%mask = zeros(featrows, 1);

%for lcount = 1:Numterms
  % get rules from tree number lcount
%  I = find(loopclass == lcount);
%  looprules = classrules(I,:);
   looprules=savedminterms;

   [Decisionmat2, mask2] = applyimagerule2(features, looprules, 1, 1);

   mask2 = mask2 / sum(mask2);
   mask = mask2;
   %end

   figure('Position',[10 50 scrsz(3)*.985 scrsz(4)*2/6]), imagesc( reshape(mask, numrows,
numcols) ), colormap(gray(256))
   title('Image of Saved Minterms applied to selected image ')
% figure, imagesc( reshape(histeq(mask2), numrows, numcols) ), colormap(gray(256))
% title('Histogram equilized Image of Saved Minterms applied to selected image ')
```

discriminateratio.m

```
function discrimratio=discriminateratio(sumvectdata,classnum,lengthclassname)

sumvectcorrect=sumvectdata(classnum,:);
sumvectwrong=sum([ sumvectdata(1:classnum-1,:);
sumvectdata(classnum+1:lengthclassname,:)]);
discrimratio=sumvectcorrect./sumvectwrong;
```

describefeature.m

```
function [description,colorhsv] = describefeature(featnum)

hues = 0:(1/12):1;
sats = 0:(1/4):1;
vals = 0:(1/4):1;

hues = [0 .05 .14 .22 .28 .45 .54 .75 .81 .92 1];
sats = 0:(1/5):1;
vals = 0:(1/5):1;

colorhist = [];
for h = 1:length(hues)-1
  for s = 1:length(sats)-1
    for v = 1:length(vals)-1
      colorhist = [colorhist
        mean([hues(h) hues(h+1)]) mean([sats(s) sats(s+1)]) mean([vals(v) vals(v+1)]) ];
    end
  end
end

if (featnum <= 250)
   colorhsv = colorhist(featnum, :);
   description = sprintf('hsv color parameters hue=%g saturation=%g value=%g',colorhsv(1),
colorhsv(2), colorhsv(3) );
end

if featnum == 251
   description = 'Texture measure based on Laplacian';
   colorhsv=[0 0 0];
end

if featnum == 252
   description = 'Motion detection measure';
   colorhsv=[0 0 0];
end

if featnum == 253
   description = 'Symmetry measure';
   colorhsv=[0 0 0];
end

if featnum > 253
   angle = featnum - 253 - 90;
```

```
    description = sprintf('straight line measure at angle of %g degrees',angle);
    colorhsv=[0 0 0];
end
```

getpixelminterms.m

```
function
[foundarray,foundarrayperc]=getpixelminterms(mask,Decisionmatim,Rulematrixim,numrows,nu
mcols)

[lookcol lookrow] = ginput(1);
searchpix = zeros(numrows,numcols);
searchpix(round(lookrow),round(lookcol)) = 1;
searchpixarray=reshape(searchpix, numrows*numcols,1);
searchindex=find(searchpixarray == 1);
foundarray=Decisionmatim(searchindex,:);
foundarrayperc=foundarray/sum(foundarray)*100;
figure,plot(foundarrayperc)
title('Percentage of the pixel intensity each minterm contributed')
```

imagefeatures3.m

```
y = rgb2hsv(x);
yh = y(:,:,1);
ys = y(:,:,2);
yv = y(:,:,3);

yg = rgb2gray(x);

[nrows, ncols] = size(yh);

hues = [0 .05 .14 .22 .28 .45 .54 .75 .81 .92 1];
sats = 0:(1/5):1;
vals = 0:(1/5):1;

Nfilt = 15;
Ndec = 10;

havg = ones(Nfilt, Nfilt)/(Nfilt^2);

features = [];

rad = (Nfilt+1)/2;

rowsamp = rad:Ndec:(nrows - rad);
colsamp = rad:Ndec:(ncols - rad);
numrows = length(rowsamp);
numcols = length(colsamp);


disp('Adding color features');

yhq = zeros(size(yh));
for i=2:10
  yhq = yhq + ((yh > hues(i))&(yh <= hues(i+1)))*i;
end
yhq = yhq + ((yh >= hues(1))&(yh <= hues(2)))*1;

ysq = zeros(size(yh));
for i=2:5
  ysq = ysq + ((ys > sats(i))&(ys <= sats(i+1)))*i;
end
ysq = ysq + ((ys >= sats(1))&(ys <= sats(2)))*1;

yvq = zeros(size(yh));
for i=2:5
```

```
  yvq = yvq + ((yv > vals(i))&(yv <= vals(i+1)))*i;
end
yvq = yvq + ((yv >= vals(1))&(yv <= vals(2)))*1;

yq = (yhq-1)*25 + (ysq-1)*5 + (yvq-1);

        rowindx = 0;
        colorfilt = [];
        for jcol = colsamp
           for irow = rowsamp
              tmp = yq( (irow-((Nfilt-1)/2)):(irow+((Nfilt-1)/2)) , (jcol-((Nfilt-1)/2)):(jcol+((Nfilt-
1)/2)) );
              ctmp = hist(tmp(:), 0:249);
              colorfilt = [colorfilt ; ctmp];
           end
        end



features = colorfilt/(Nfilt^2);

[reducedrows, reducedcols] = size(colorfilt);

disp('Adding texture feature');
%Laplacian Texture Feature
hlap = [-1 -1 -1
   -1 8 -1
   -1 -1 -1]/8;
texture1 = filter2(havg, abs(filter2(hlap, yg)/256));
texture1 = texture1(rowsamp, colsamp );
features = [features texture1(:)];

disp('Adding canny edge feature');
%Canny Edge Feature
texture2 = filter2(havg, edge(yg, 'canny'));
texture2 = texture2(rowsamp, colsamp);
features = [features texture2(:)];

disp('Adding phase symmetry feature');
%Phase Symmetry Feature - removed - just add zeros here
features = [features zeros(numrows*numcols,1)];

disp('Adding line feature');
% features2 = [];
% eg = edge(yg, 'canny');
% for j = colsamp
```

```
%     for i = rowsamp
%         xwin = eg((i-rad+1):(i-rad+Nfilt),(j-rad+1):(j-rad+Nfilt));
%         [xwinr,xp]=radon( xwin ,0:179);
%         line = max(xwinr)/(Nfilt^2);
%         features2 = [features2
%                     line];
%     end
% end
% features = [features features2];
%
% [numrows,numcols]=size(texture1);

features(:, 254:433) = 0;
```

savemintermsauto.m

```
function savedminterms=savemintermsauto(Rulematrix,discrimratio,savedminterms)

if discrimratio == -1
   choice=menu('Minterm options','Enter minterms manually');
else
   choice=menu('Minterm options','Enter minterms manually','Create a threshold from plot','Save
the top (number) minterms');
end
if choice == 2
   disp('If you want a min and max, enter in two numbers (in min max order) in array form');
   disp('Otherwise the number will collect all of the minterms above entered value');
   decidethresh=input('What is the deciding threshhold?  ');
   if length(decidethresh) == 1
      index=find(discrimratio >= decidethresh);
      savedminterms=[savedminterms; Rulematrix(index,:)];
   else
      index=find(discrimratio >= decidethresh(1) & discrimratio <= decidethresh(2))
      savedminterms=[savedminterms; Rulematrix(index,:)];
   end
elseif choice == 1
   index=input(' What are the minterms to be saved (in form of array if more than 1)'  );
   savedminterms=[savedminterms; Rulematrix(index,:)];
elseif choice == 3
   topnum=input(' How many of the top minterms do you want to use  '  );
   [Y,I] = sort(discrimratio);
   I=fliplr(I);
   savedminterms=[savedminterms; Rulematrix(I(1:topnum),:)];
end
```

savemintermsman.m

```
function savedminterms=savemintermsman(Rulematrix,savedminterms)

index=input(' What are the minterms to be saved (in form of array if more than 1)'  );
savedminterms=[savedminterms; Rulematrix(index,:)];
```

BIBLIOGRAPHY

[1] J. Chamorro-Martinez, D. Sanchez, and B. Prados-Suarez, *A Fuzzy Colour Image Segmentation Applied to Robot Vision.* In Advances in Soft Computing – Engineering, Design and Manufacturing. Springer, 2002. In Press.

[2] D. Pham and J. Prince, "An Adaptive Fuzzy C-Means Algorithm for Image Segmentation in the Presence of Intensity Inhomogeneities" Pattern Recognition Letters, vol. 20, no. 1, pp. 57-68, 15-January,1999.

[3] Y. Fang and T. Tan, *A Novel Adaptive Colour Segmentation Algorithm and its Application to Skin Detection.* In Proc. BMVC 2000, pp. 23-31. BMVA, 2000

[4] J. Chamorro, D. Sanchez, B. Prados-Suarez, E. Galan-Perales, and M. Vila, "A hierarchical approach to fuzzy segmentation of colour images" En: Actas del FUZZ-IEEE 2003. ST. Louis(USA): 2003.

[5] K. Woods, D. Cook, L. Hall, K. Bowyer, and L. Stark: "Learning Membership Functions in a Function-Based Object Recognition System" J. Artif. Intell. Res. (JAIR) 3: 187-222 (1995)

[6] S. Lemon, J. Roy, M. Clark, P. Friedmann, and W. Rakowski, "Classification and regression tree analysis in public health: methodological review and comparison with logistic regression" Ann Behav Med 2003; 26:172-81.

[7] N. Ito, Y. Shimazu, T. Yokoyama, and Y. Matushita, "Fuzzy Logic Based Non-Parametric Color Image Segmentation with Optional Block Processing", pp 119-126, Association of Computing Machinery, 1995 ACM 0-89791-737-5

[8] A. Malaviya and P. Malaviya, "Object recognition using fuzzy set theoretic techniques" SPIE Proceedings Vol 1962, Paper #6, 12-14 April, 1993, Orlando, FL

[9] http://en.wikipedia.org/wiki/RGB_color_model, Copyright © 2000,2001,2002 Free Software Foundation, Inc., Boston, MA

[10] http://en.wikipedia.org/wiki/HSV_color_space, Copyright © 2000,2001,2002 Free Software Foundation, Inc., Boston, MA

[11] http://en.wikipedia.org/wiki/Probability_axioms, Copyright © 2000,2001,2002 Free Software Foundation, Inc., Boston, MA

[12] J. Bezdek and S. Pal, eds. (1992), "Fuzzy Models for Pattern Recognition", New York: IEEE Press.

[13] H. Stark and J. Woods, "Probability and Random Processes with Applications to Signal Processing", Prentice-Hall, Upper Saddle River, NJ (2002).

[14] R. Larsen and M. Marx, "An Introduction to Mathematical Statistics and Its Applications".Third Edition, Prentice-Hall, Upper Saddle River, NJ (2001)

[15] http://en.wikipedia.org/wiki/Fuzzy_logic, Copyright © 2000,2001,2002 Free Software Foundation, Inc., Boston, MA

[16] R. Haralick and L. Shapiro, "Computer and Robot Vision: Vol. 2," Reading, MA: Addison-Wesley, 1992.

[17] J. Holland, “Adaptation in Natural and Artificial Systems”, MIT Press, Cambridge, MA, 1975

[18] L. A. Zadeh, *Fuzzy Sets and Systems*. In: Fox, J. (ed.): Proceedings Symposium on System Theory, Polytechnic Institute of Brooklyn, April 1965, pp. 29-37**.**