(4)

Applied Research Laboratory
The Pennsylvania State University

USING COMPUTER VISION TECHNIQUES TO LOCATE OBJECTS IN AN IMAGE

by

Sujata Kakarla
J. Wakeley
A. S. Maida

ARL-PSU TECHNICAL REPORT

4

The Pennsylvania State University
APPLIED RESEARCH LABORATORY
P. O. Box 30
State College, PA 16804

USING COMPUTER VISION TECHNIQUES TO LOCATE OBJECTS IN AN IMAGE

by

Sujata Kakarla
J. Wakeley
A. S. Maida

Technical Report No. TR 88-013

September 1988

Supported by:
Naval Sea Systems Command

L. R. Hettche, Director
Applied Research Laboratory

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

ADA199835

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Unlimited
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Applied Research Laboratory The Pennsylvania State University — 6b OFFICE SYMBOL (If applicable) ARL	7a. NAME OF MONITORING ORGANIZATION Naval Sea Systems Command Department of the Navy
6c ADDRESS (City, State, and ZIP Code) P. O. Box 30 State College, PA 16804	7b. ADDRESS (City, State, and ZIP Code) Washington, DC 20362
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Naval Sea Systems Command — 8b OFFICE SYMBOL (If applicable) NAVSEA	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code) Department of the Navy Washington, DC 20362	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
Using Computer Vision Techniques to Locate Objects in an Image

12 PERSONAL AUTHOR(S)
Sujata Kakarla, J. Wakeley, A. S. Maida

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
M.S. Thesis	FROM ______ TO ______	September 1988	45

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	artificial intelligence, computer imaging, digitized images image filters, signal images

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The purpose of this research was to study and implement a computer vision technique. Our goal was to have the ability to input a digitized image containing objects in a noise background, and output an image that contains only the objects. The results of applying the subsequent algorithms have been encouraging. Objects have been isolated on several images. At this point we have the capability to analyze the image at a higher level. Instead of attempting to look at every cell in the image, we can look at groups of cells or regions. This will allow us to give properties to each region that will distinguish it from the other regions.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

This work is the result of a joint effort by the Applied Research Laboratory and the Computer Science Department of the Pennsylvania State University. Through an Exploratory/Foundational Program, the University provides graduate assistantships to University students.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

LIST OF FIGURES

ACKNOWLEDGEMENTS

There are a number of people that I wish to thank for their help in completing this paper. First of all, I would like to thank Mr. Joseph Wakeley for his quidance and patience throughout these past two years. I also want to thank him for giving me the opportunity to work with him on this project. I would like to thank Dr. Paolo Roberti for the use of his computer facilities, and for his help in obtaining hardcopy plots of the figures in Chapter 3.

Dr. Anthony Maida has been both my advisor and my friend, and I cannot thank him enough for all that he has done for me. Without his help I would not have had the opportunity to do this project or to continue my education and obtain my Master s degree.

I would like to thank my friends, family, and fiance for their support and encouragement. I could not have completed either the research or this paper without them.

Chapter 1

INTRODUCTION

A picture is said to be worth a thousand words. It seems an appropriate saying when you contemplate trying to describe a scene to someone. There is a vast amount of information that can be extracted from just a single image, and to obtain a symbolic description of all this information is virtually impossible. One possible way around this is to narrow our goals, and attempt to locate and identify only particular features contained in the image. For instance we may want to distinguish whether or not noise is present in the image. If the image contains objects, we may wish to know where they are located, and their general configuration.

The purpose of this research was to study and implement a computer vision technique. Our goal was to have the ability to input a digitized image containing objects in a noise background, and output an image that contains only the objects. This goal was reached. How we have reached this goal is the subject of this paper.

The following chapters discuss the methods used to isolate the objects in the image. Chapter 2 gives an overview of the ideas underlying the techniques that have been developed to display the image, and to segment the image into regions. Instead of referring to the image in terms of the intensity levels of the individual cells, we group cells together into regions, and then refer to the regions in the image.

Chapter 3 focuses on the implementation of the techniques described in Chapter 2. A detailed description is given of the stages of processing involved in obtaining an image that contains only the objects that were in the original image. The processing involves filtering the image at different spatial scales, locating and representing the intensity changes, known as zero-crossings, in the filtered image, and then forming regions from the zero-crossings. Smaller regions are thresholded out. The remaining regions are compared across the different spatial scales. If regions are found to be in the same location, then they are saved. These regions that have been saved are most probably due to an object in the image.

The final chapter, Chapter 4, discusses the results of implementing the algorithms presented in Chapter 3, and offers recommendations for further study.

Chapter 2

IMAGE REPRESENTATION

This chapter focuses on the methods used to represent an image. Initially, the image is represented by a two-dimensional matrix of intensity levels. It is very difficult to look at this raw data and understand what the image is and what it represents. Therefore, we need to obtain a symbolic representation.

The first method discussed does not change the format of the image or encode information symbolically, but it does allow a graphic representation of the image. The intensity levels are mapped into a greyscale, and then the greyscale corresponding to the intensity level is plotted. The second method processes the image and gives a symbolic representation of the image. Instead of "seeing" the image in terms of individual cells (or pixels), as in the first method, we can group the pixels together and look at the image in terms of regions.

2.1 Greyscale

The first method used to give a clearer representation of the image is a greyscale. Each intensity level in the original image matrix is given a grey level. The grey levels start at 0, indicating white, and go to however many levels are desired, or obtainable. The highest level indicates black. Each level in between will be an increasingly darker shade of grey. The intensity levels are scaled to fit the number of grey levels available. If any objects are located in the image, they should be relatively easy to see, using the greyscale, unless the noise level is of such high intensity that the object intensity gets lost.

2.2 Constructing Regions

The greyscale representation is helpful in understanding what the image looks like, but it does not help to output a description of the image. The description of an image, that is a description of the objects contained within the image, consists of three phases. The first phase is to construct the **Primal Sketch**. The primal sketch consists of a symbolic representation of the intensity changes in the image, and their location. The second phase is to develop the $2\frac{1}{2}$ - **Dimensional Sketch**. This phase consists of a representation of the properties of the visible surfaces in the image from the perspective of the viewer. The last phase, the **3-Dimensional Model Representation**, gives a description of the shapes in the image from the perspective of the object. This framework was derived by David Marr [3]

We will concentrate our efforts on the first of these phases, developing a method to segment the image using the locations of the intensity changes as the partitioning agents. There are three stages to the construction of the **Primal Sketch**. There is the detection of the intensity changes within the image, the formation of the Raw Primal Sketch, and the creation of the Full Primal Sketch.

2.2.1 Detection of Intensity Changes

One method to locate the intensity changes in the image, is based on the following two principles. First, regularities in intensity changes *occur at various spatial scales throughout the image*. Second, an intensity change will cause a peak or dip in the first derivative or, a zero-crossing in the second derivative. A zero-crossing is a *place where the value of the function changes sign*. According to Marr and Hildreth (1980) [2], the operator that fits both these criteria best is the filter $\nabla^2 G$, where ∇^2 is the Laplacian operator

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

and G is the two-dimensional Gaussian distribution

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

$\nabla^2 G$ is a circularly symmetric Mexican-hat shaped function whose distribution can be expressed as follows:

$$\nabla^2 G(r) = -\frac{1}{\pi\sigma^4} \left[1 - \frac{r^2}{2\sigma^2}\right] e^{-\frac{r^2}{2\sigma^2}}$$

There are two reasons why the $\nabla^2 G$ filter is attractive. The Gaussian blurs, or smooths, the image without introducing changes that were not present in the original image. The second consideration is that ∇^2 saves in computation time. First-order or second-order directional derivatives can be used rather than ∇^2. For first-order we must search for the peaks or dips indicating intensity changes, and for second-order we look for zero-crossings. The disadvantage to using these methods is that they are directional. For first-order, there is a great deal of computation that must be performed, and the computations needed for second-order are worse. Using an orientation-independent operator will allow us to avoid performing all these computations. For reasons of parsimony, we choose the lowest-order operator we can find. We also would like a circularly symmetric operator, making it directionally independent. The Laplacian, ∇^2 , fits these requirements, and it can be used to detect intensity changes.

We will first smooth the image using the Gaussian,

$$G * I$$

where, *, represents the convolution operation, and I, represents the image. We then apply the Laplacian to the smoothed image,

$$\nabla^2(G * I) \equiv (\nabla^2 G) * I.$$

A mathematical identity allows us to move the ∇^2 inside the convolution allowing us to convolve the image with $\nabla^2 G$.

2.2.2 The Raw Primal Sketch

Now that we have a method to detect the intensity changes in the image, we can proceed to the next stage in developing the Primal Sketch. In this second stage, we need to combine the information obtained by convolving the image using various-sized filters. According to Marr, there is a physical reason why the zero-crossing detected by using one filter is related to the zero-crossing obtained when using a different sized filter. It is due to the spatial coincidence assumption [3]:

> If a zero-crossing segment is present in a set of independent $\nabla^2 G$ channels over a contiguous range of sizes, and the segment has the same position and orientation in each channel, then the set of such zero-crossing segments indicates the presence of an intensity change in the image that is due to a single physical phenomenon (a change in reflectance, illumination, depth, or surface orientation)

According to this assumption, we can state that if a zero-crossing is present across several filtered images, then it is probably due to an object located in the image and not due to noise. If it was due to noise, then there is high probability that it will not appear in several filtered images.

We will modify this assumption to include groups of zero-crossing segments. If a group of zero-crossing segments appear on several filtered images, then we can assume that they are due to an object in the image and not to noise. We want to do this because we would like to group zero-crossings together to form regions, and then compare regions across several filtered images to see if a particular region appears inside of a region from a filtered image that is tuned to a larger scale. If we find that a region does have a corresponding region in a filtered image of a different scale, then we can presume that the region is not due to noise and that it is due to an object located in the image.

We would like to form these regions, because it is easier to deal with the image at the level of regions, or groups of pixels, rather than at the pixel level. The description of the image using the primitives **edges**, **bars**, and **blobs**, and **terminations** having attributes of length, width, and position, is called the **Raw Primal Sketch** [1]. Since our representation of the image using regions does not use those primitives, we cannot truly call it a **Raw Primal Sketch**. We can however give the regions attributes of length, width, and position, if we desire. Each region is either made up of edges, bars, or blobs, or combinations of these. There are, however, no terminations. So, in some sense, we can consider that after we compare regions from several filtered images, and choose the regions that have been present in all, or most of, the filtered images we have obtained the ***Raw Primal Sketch***.

2.2.3 The Full Primal Sketch

The last step according to Marr was to make tokens and find boundaries to obtain the **Full Primal Sketch**. Since we have adapted the **Raw Primal Sketch** for our purposes, we have obtained the **Full Primal Sketch** already. The regions formed from the zero-crossings can be thought of as tokens, and the borders of these regions can be thought of as their boundaries.

The next chapter gives a detailed description of the implementation of the filtering, detection of the zero-crossings, and formation of regions.

Chapter 3

IMPLEMENTATION

The previous chapter discussed the various methods of representing the image, and modifying it so that it can be easily analyzed. This chapter focuses on the implementation of the techniques discussed. The techniques can be broken down into five main parts. First, we discuss a way to represent the image in a different format. Then a greyscale representation of the image is presented. The third part involves the construction and application of filters for the image. Once the image is filtered, we can detect and represent the intensity changes within it. These intensity changes, known as **zero-crossings**, can then be grouped together to form regions. Lastly, regions from a filtered image of a particular σ value are compared with regions from a a filtered image of a smaller σ value. Regions that correspond to each other are then placed in a resultant image. These regions can then be compared with another filtered image of an even smaller σ value. The final image obtained after all comparisons have been done should contain only the objects of interest within the image.

3.1 Representation of Image

Before we modify the image, we must represent it so that the various techniques can be implemented conveniently. The initial representation of the image is as a two-dimensional matrix of intensity levels. Specifically, it is a 64 by 64 matrix. Each cell in the image contains a value that represents the intensity at that position in the image. Another way the image can be represented is to give each of the cells a number to distinguish it from the others, and to allow each cell to be given properties. The only property at the moment is the intensity level that it contains. The numbers for the cells are shown in Figure 1.

0	1	2	3	4	5	...	58	59	60	61	62	63
64	65					...					126	127
128	129	130				...				189	190	191
192	193	194	195			...			252	253	254	255
256	257	258	259	260		...		315	316	317	318	319
320	321	322	323	324	325	...	378	379	380	381	382	383
.	.	.	.	.	.	.	.	.	.	.	.	
.	.	.	.	.	.	.	.	.	.	.	.	
.	.	.	.	.	.	.	.	.	.	.	.	
3712	3713	3714	3715	3716	3717	...	3770	3771	3772	3773	3774	3775
3776	3777	3778	3779	3780		...		3835	3836	3837	3838	3839
3840	3841	3842	3843			...			3900	3901	3902	3903
3904	3905	3906				...				3965	3966	3967
3968	3969					...					4030	4031
4032	4033	4034	4035	4036	4037	...	4090	4091	4092	4093	4094	4095

Figure 1: Cell numbers for 64 by 64 matrix

3.2 Greyscale

The next stage in our representation is to plot the image using grey levels in place of the intensity level. This representation is only useful in giving us a visual image of the intensity levels. Each cell is represented by an 8 by 8 matrix of pixels. These 64 pixels are used to give 65 grey levels. For each level, the corresponding number of pixels is turned on. For instance, if the grey level is 0, no pixels are turned on. If the grey level is 4, 4 of the 64 pixels are turned on, and so on. The highest level is 64, where all the pixels in the 8 by 8 matrix are turned on.

To make a grey scale plot of the entire image, the maximum and minimum values of the image are found, and the intensity values are then scaled from 0 to 64 to plot the correct grey scale for that intensity.

To make the plotting of these grey levels easier, the greyscale was made into a font. Each character of the font represents a particular grey level. So, when a grey level needs to be plotted, each individual pixel does not need to be turned on separately. The character in the font that represents a particular grey level is plotted, and so all the pixels for that character are turned on at once.

3.3 Filtering

There are two stages to the process of filtering an image. First we need to construct filtering matrices, and then we convolve them with the image matrix.

3.3.1 Filter Construction

To construct the filters, we will use the ∇^2 introduced in the previous chapter. We will construct five different filters, each representing the σ values of .5, .75, 1.0, 1.5, and 2.0. Using these five different values will allow us to filter the image at five different spatial scales.

The process used to construct the filters can be thought of as a double summation within a double loop. The outer loops go through every cell in the filtering matrix and calculate the value for each cell using a double summation. The double summation computes 121 values within each cell in increments of $\frac{1}{11}$ in both the x direction and the y direction giving 121 values for the subcells contained within the cell. The sum of these values is divided by 121, and that number is used as the value for the cell.

Using this method of calculating the value for the cell will yield a better approximation of the actual value for each cell. The value 121 was obtained through experimentation. Using the average of 121 values gives a very accurate value for the value of the cell. The number of values used for the average depends on how accurate the value for the cell needs to be. If a more accurate value is needed, then more than 121 values can be used. Using less than 121 values will yield a less accurate value for the cell. The increment $\frac{1}{11}$ is the reciprocal of the square root 121.

Once the values have been calculated for each cell, an appropriate scale value is determined, and each value in the matrix is multiplied by the scale value. The scale value is determined by evaluating what size matrix is needed that will represent the shape of the curve accurately enough, with only integer values, for a particular σ. Integer values are used rather than real values to increase speed of computation. Once the matrix is constructed, it can be reduced in size if all the values located in the outer rings are zero's. An illustration of this is given in Figure 2.

$$\begin{matrix}0&0&0&0&0\\0&1&1&1&0\\0&1&-8&1&0\\0&1&1&1&0\\0&0&0&0&0\end{matrix} \longrightarrow \begin{matrix}1&1&1\\1&-8&1\\1&1&1\end{matrix}$$

Figure 2: Reduction of filter size

3.3.2 Convolution

After the filtering matrices are constructed, the next step is to convolve the image with each of the them. There are two ways that the image matrix and the filtering matrix can be convolved. The first method is to use **Fourier Transforms**. The method involves taking the Fourier Transforms of the image matrix and filtering martix, and then doing an inverse Fourier Transform of the product of the two transforms. This method is probably the most efficient and time saving computationally, but we chose to use the method of **Digital Convolutions** rather than the Fourier Transform method because of its simplicity and because our primary interest lies in locating the intensity changes in the image and not in how efficiently we can do convolutions.

Digital Convolution of the image matrix and the filtering matrix can be represented by the following equation:

$$F * I = FI(x,y) = \sum_{i=-u}^{u} \sum_{j=-u}^{u} F(m-u+i, m-u+j)\, I(x+i, y+j)$$

$$u = \frac{m-1}{2}$$

where m is the size of the filtering matrix. Let F be the 3 by 3 filtering matrix,

$$\begin{matrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{matrix}$$

and let I be the image matrix

$$\begin{matrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{matrix}$$

m = 3, the size of the filtering matrix, and $u = \frac{m-1}{2} = 1$. The filtered image matrix, FI, will be as follows:

$$\begin{matrix} X & X & X & X & X & X \\ X & 0 & 3 & -3 & 0 & X \\ X & 0 & 3 & -3 & 0 & X \\ X & 0 & 3 & -3 & 0 & X \\ X & 0 & 3 & -3 & 0 & X \\ X & X & X & X & X & X \end{matrix}$$

The "X" is used to indicate that FI, the filtered matrix, does not have values for those positions. This problem is known as **Start-up Artifact** and **Ending Artifact**. There are four ways that we can handle this. The first way is to just ignore it and accept the fact that we will lose an outer border proportional to the size of the filtering matrix every time we convolve an image.

The second and third methods give values for the locations in the image matrix that are needed for the calculation of the filtered image. For instance, to calculate FI(1,1), we need to know the values of I(0,0), I(0,1), and I(1,0). The second method sets these values to zero, along with all the other unknown values needed for the computation of the filtered matrix. The third method sets the values equal to the closest known value. I(0,0) , I(0,1), and I(1,0) would all be set equal to I(1,1).

There are problems that we need to be aware of for both of these methods. For the second method, giving a value of zero to those positions would cause a zero-crossing to be detected along the border if the background noise is of sufficient intensity. This may not be desirable. Using the closest value, as in the third method, may not lead to a zero-crossing being detected, but if the values in the cells being copied are due to some particular characteristic of the image that is present at that location, then that characteristic will be evident in the outer cells as well. This also may not be desirable. Another consideration for both of these methods is that setting the unknown values to the appropriate value will take time, especially if the filtering matrices are large.

The fourth method is to set those cells equal to a value based on the previous two points. For example, the value for I(1,0) is equal to I(1,1) + (I(1,1) - I (1,2)), or equivalently, 2 I(1,1) - I(1,2). This computation is done to get all the needed values. Although this method seems to solve the problems associated with the previous two methods, the computation involved is very expensive, especially when large filters are used. It does not seem worthwhile to use it, unless it is crucial to preserve the size of the image matrix.

It is important to realize that something can be done about the Start-up and Ending Artifact. but for our purposes, it is not necessary to preserve the image size. Therefore, we will choose the first option, and ignore the fact that the outer border is lost.

3.4 Zero-Crossings

The next phase in our implementation is to detect and represent zero-crossings. The zero-crossings will be used to detect where the intensity changes are in an image, and then will be grouped together to form regions. There are a number of ways to detect and represent zero-crossings. The following sections outline the methods we used.

3.4.1 Detection

We know that a zero-crossing occurs when there is a change in sign between the numbers contained in adjacent cells in the filtered image. To detect where these intensity changes occur, all we need to do is go across and down each filtered image matrix and check to see if there is a sign change between the values contained in adjacent cells. To make it easy to determine whether or not there is a sign change, the values in the matrix are first converted to positive one's for all positive numbers, and negative one's for all negative numbers. Then, if the sum of two adjacent cells adds up to zero, then we can state that a zero-crossing exists between those two cells.

One problem with this method is if the value in a cell is zero. In our implementation, we assume that zero is positive, and so, it becomes a positive one. Doing this causes the zero-crossing to be shifted slightly, or an extra zero-crossing to appear. Both these two cases are illustrated in Figure 3.

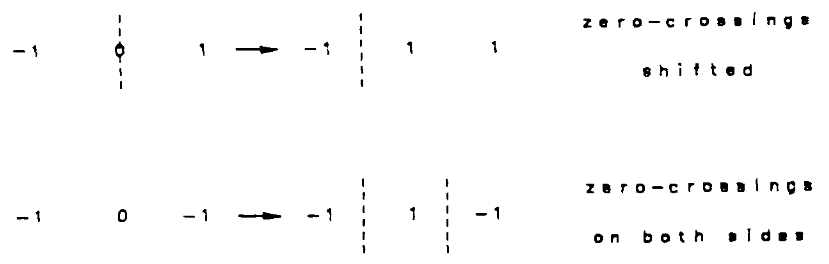


Figure 3: The Case of Zero

In the first case, there should only be one zero-crossing which appears through the middle of the cell containing the value zero. However, the zero-crossing is shifted to the left so that it lies between the first cell and the second cell. In the second case, two zero-crossings appear where there should not have been a zero-crossing, one between the first cell and the second cell, and the other is between the second cell and the third.

We are allowing this error to be introduced into our zero-crossing representation of the image for two reasons. The first is because there will be very few cells whose values are zero. The second is because the segmenting techniques used to group the zero-crossings together to form regions rely on the zero-crossings being between two cells, and not through cells. If a cell has the value zero and its zero-crossing goes through the middle of the cell, and the cell adjacent to it has its zero-crossing at the edge of the cell, there is no way to connect them. The segmenting techniques also rely on the zero-crossings being connected on both ends. Having a zero-crossing that did not connect to anything will cause an error to be reported, and the segmenting program will not continue to segment the image.

Once the zero-crossings are detected, we need to select a representation that will store what zero-crossings are located around a particular cell. If we can store this information, we can access it to traverse the zero-crossings to group them together to form regions.

3.4.2 Representation

The first way we will represent the zero-crossings is graphically. Its purpose is to conveniently display zero-crossings. Recall that the image is represented as a 64 by 64 matrix of cells. Each of these cells was then represented by an 8 by 8 matrix of pixels to plot the greyscale of the image. We will again use the 8 by 8 matrix of pixels, but this time, we will plot lines to indicate the presence of zero-crossings between two cells.

If there is a sign change between two adjacent cells, then a line is plotted down the last column of pixels in the 8 by 8 matrix of pixels of the cell to the left, and a line is plotted down the first column of the cell to the right. If the sign change occurs between cells sitting one on top of the other, then the lines plotted are in the last row of the top cell and the first row of the bottom cell. The two types of zero-crossings are illustrated in Figure 4.

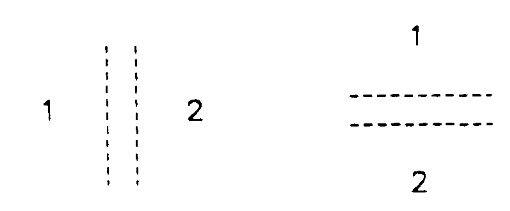


Figure 4: Two types of zero-crossings

Along with plotting the zero-crossings, we need to save them in some way. One method is to make an array of 4096 elements, one for each cell. Each element of the array will contain 2 bits, one bit for each zero-crossing. A bit is set if a cell contains a particular zero-crossing. The first bit corresponds to the **Right** zero-crossing, and the second bit corresponds to the **Bottom**. Using bits will allow us to access the information very quickly, because bit operations can be performed to determine whether or not the cell contains a particular zero-crossing. An example of this representation is illustrated in Figure 5 and Figure 6.

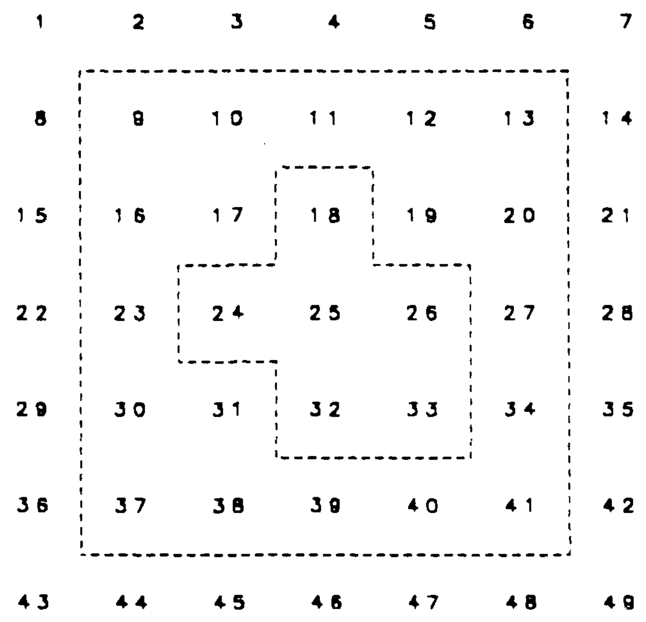


Figure 5: Cells with zero-crossings

1	:	00	8	:	10	15	:	10	22	:	10	29	:	10	36	:	10	43	:	00
2	:	01	9	:	00	16	:	00	23	:	10	30	:	00	37	:	01	44	:	00
3	:	01	10	:	00	17	:	11	24	:	01	31	:	10	38	:	01	45	:	00
4	:	01	11	:	01	18	:	10	25	:	00	32	:	01	39	:	01	46	:	00
5	:	01	12	:	00	19	:	01	26	:	10	33	:	11	40	:	01	47	:	00
6	:	01	13	:	10	20	:	10	27	:	10	34	:	10	41	:	11	48	:	00
7	:	00	14	:	00	21	:	00	28	:	00	35	:	00	42	:	00	49	:	00

Figure 6: Bits set for each cell of previous figure

Even though each cell only has 2 zero-crossings that are actually stored, to understand the segmentation algorithm, we will think of each cell as having 4 possible zero-crossings. A cell could also have zero-crossing above it, referred to as the **Top** zero-crossing, and a **Left** zero-crossing, along with the **Right** and **Bottom**. To see if a cell has a **Top** zero-crossing, we check if the cell above it has a **Bottom** zero-crossing. Similarly, to check if a cell has a **Left** zero-crossing, we check if the previous cell has a **Right** zero-crossing.

3.5 Regions

The final stage is to form the regions. This involves two major steps. We first find the borders of the regions, and then we "fill in the borders" to form the regions. Before we can begin to do this, however, we need to take care of one detail. The outside border of the filtered image does not contain zero-crossings. If we attempt to implement a segmentation algorithm to group zero-crossings together to form regions, many potential regions will not be considered because they are not closed contours.

The first section shows a way to handle this problem and the following sections discuss the method used to segment the image into regions.

3.5.1 Outside Border

To insure that potential regions are not lost, we will place zero-crossings around the outside of the filtered image. The cells on the corners of the border can be determined ahead of time, based on the σ value of the filtered image, and zero-crossings can be placed at the appropriate places to create a border surrounding the filtered image. The point to remember here is that the zero-crossings placed around the border are not due to intensity changes in the image. They have been placed there to avoid losing critical information about the image.

3.5.2 Traversing Zero-Crossings to Create Border Paths

This section explains the decision-making process that occurs when we follow the lines made by zero-crossings to form regions. We first need to analyze the steps we go through when we traverse the zero-crossing lines on the plots. We will formulate rules that can be applied to the representation of the zero-crossings that we developed in the previous section.

Before we can analyze what decisions we make when we traverse the lines to form zero-crossings, we must define what a region is. A **region** is an area which is completely bordered by zero-crossings. If there are zero-crossings within the region, then no part of the those zero-crossings can be a part of the border of zero-crossings of the outer region. Two or more regions can be adjacent to each other, in which case, the outer borders of the regions are touching. In Figure 7, cells 11, 17, 18, 19, 25, and 26 form a region, while cells 24, 31, 32, 33, and 39 form a second region adjacent to the first.

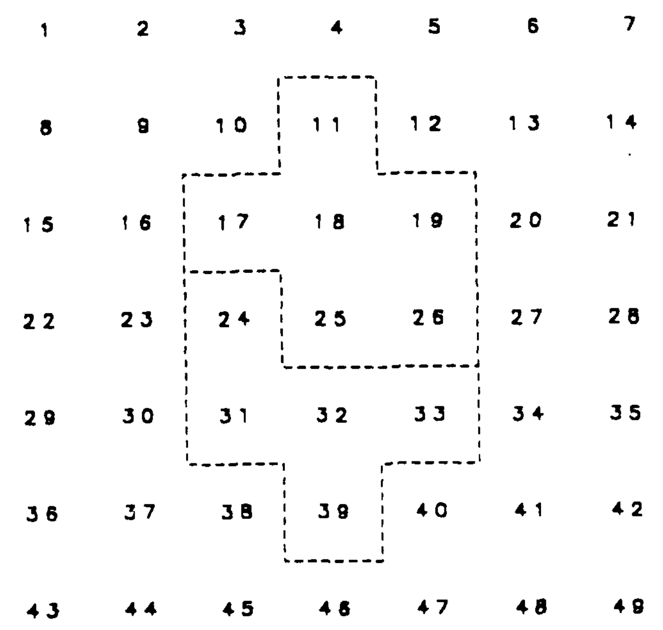


Figure 7: Example of adjacent regions

In Figure 8, cells 10, 11, 12, 17, 18, 19, 24, 25, 26, 31, 32, 33, 38, and 39 form a region, while cells 18 and 25 form a second region.

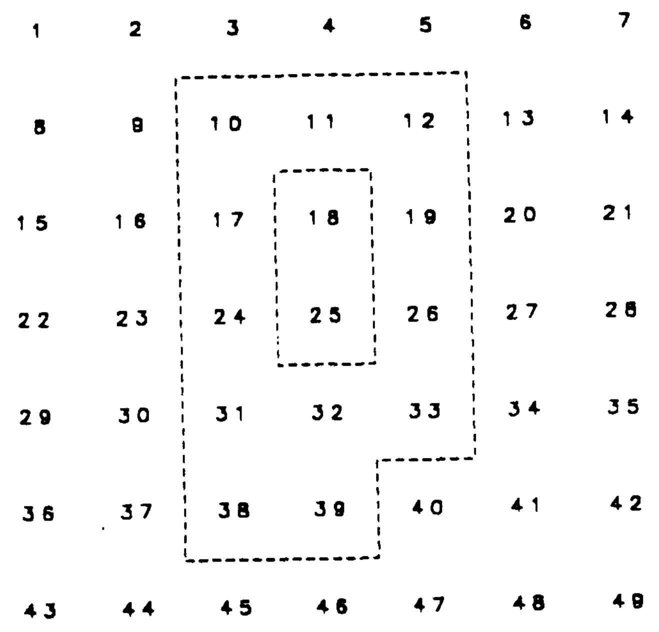


Figure 8: Example of nested regions

Notice that cells 18 and 25 are located in both regions. It is possible to stipulate that these cells are only in the second region and not the first, but for our purposes, we will allow them to exist in both regions.

Now that we have a definition for what we consider a region to be, we need to make up a set of rules that will tell us how to traverse the zero-crossings to form regions to fit that criterion. We first need to think about how we mentally form the regions when we look at a map of zero-crossings. Looking at Figure 7 on page 20, we could form a region by starting at the cell numbered 11 and going in a clockwise direction along the lines until we reach cell 11 again. We notice that at cell 26, we have to decide whether to make a right turn, or continue going past cell 33 and then make the right turn. To form the smallest region possible, we turn at cell 26. Then, between cells 17 and 24, we have to make another decision. We need to decide whether to go right or left. Again, to make the smallest region, we choose to turn right.

Before we can form a set of rules to traverse the zero-crossings, we need to realize that it was not just at the intersections of cells 26 and 33 and cells 17 and 24 that we had to make decisions about what direction to go. We were making decisions from the moment we began to follow the zero-crossings.

To make a set of rules about the method of traversal, we need to first specify three things. We need to specify a starting place, a direction in which we will traverse, and a stopping place. Once these things are set, we can form a set of rules to do the traversing.

We will always start at the top-left cell of a region. This will be the starting cell. The starting cell must have a **Top** zero-crossing, a **Left** zero-crossing, and must not be a part of any other region. If it does not have **Top** or **Left** zero-crossings, or is part of a region, then we attempt to form a region using the next cell. If that cell does satisfy the starting conditions, then we begin traversing with the **Top** zero-crossing, and travel in a clockwise direction. We know that we have finished when we have reached a cell which has a **Left** zero-crossing, and it is the starting cell. This will be our stopping condition. Then we attempt to form another region starting at the next cell, and we start the whole process again until we have processed the last cell in the filtered image.

The thing to keep in mind when we are forming the rules is that we always want to move in a clockwise direction, and we want to form the smallest region possible. Figure 9 presents an algorithm which shows the steps taken to form a region from zero-crossing segments.

```
start = first cell on map
cell = first cell on map
end = last cell on map
region = 1
while start ¬= end do
  if Top(cell) and Left(cell) and
    cell not in a region then
    goto 1
    if border-path (region) then
      region = region + 1
    endif
  endif
  start = start + 1
  cell = start
endwhile
```

```
     condition                    operation(s)

1  start > cell               border-path(region) = nil;RETURN
   Right(cell)                goto 2
   Top(cell + 1)              cell = cell + 1;goto 1
   Left(cell - 63)            cell = cell - 63;goto 4

2  start > cell               border-path(region) = nil;RETURN
   Bottom(cell)               goto 3
   Right(cell + 64)           cell = cell + 64;goto 2
   Top(cell + 65)             cell = cell + 65;goto 1

3  start > cell               border-path(region) = nil;RETURN
   Left(cell)                 goto 4
   Bottom(cell - 1)           cell = cell - 1;goto 3
   Right(cell + 63)           cell = cell + 63;goto 2

4  start > cell               border-path(region) = nil;RETURN
   start = cell               Done With Region ; RETURN
   Top(cell)                  goto 1
   Left(cell - 64)            cell = cell - 64;goto 4
   Bottom(cell - 65)          cell = cell - 65;goto 3
```

Figure 9: Region Making Algorithm

In the algorithm, the notation "**Top**(cell)" is a check to see if that particular cell has the zero-crossing **Top**. The other zero-crossings are checked in the same manner.

The first part of the algorithm is an initialization routine that sets up the start, end, cell, and region variables accordingly. Then it calls the first function if all the conditions are satisfied. The start cell must have both the **Top** and **Left** zero-crossings and must not be part of a region. A goto statement is used in the algorithm to represent that a function is being called. At the label, 1, we begin traversing with the **Top** zero-crossing of cell. Here, we first check to make sure that we have not somehow started to traverse backwards, and reached a cell that is <u>smaller</u> than the starting cell. If this condition is true, then we do not consider this as a possible region, and set it to nil. We do this so that we can use the region number for a legitimate region. Before we increment the region number to the next region, we check to make sure that it has not been set to nil, indicating that there was a problem with the traversal.

If the cell is <u>greater</u> than the start cell, then we proceed to the other choices. We select the first condition that is satisfied in the list, and perform the indicated operation or operations. If none of the conditions are satisfied, then we have an open contour, and will have to signal an error. Since we should not have open contours, we should never have the problem that none of the conditions are satisfied.

The other sections are basically the same as this first one, except the last section, Section 4, which has an extra condition. If the cell is the same as the start-cell, then we have finished forming a region, and can return.

The algorithm looks more complicated than it is. The easiest way to understand what the algorithm is doing, and if it is constructing regions, is to go through the algorithm using one of the zero-crossing plots in the previous figures. Another way it can be viewed is to pretend that the zero-crossings represent paths in a maze. Each zero-crossing will be opposite sides of a wall in the maze. The object is to place your finger on a side of the wall, and follow it around until you come back to where you started. The direction you move is unimportant. It can be either clockwise or counter-clockwise. The algorithm arbitrarily uses a clockwise direction, and all the "turns" are made accordingly.

The algorithm will find all regions located within one map of zero-crossings. We first set the variables starting-cell and cell equal to the first cell in the map. The value of this depends on the value of σ. If σ is equal to .5, the value of the first cell would be 65. It is 130, 195, 260, and 325 for σ values of .75, 1.0, 1.5, and 2.0, respectively. Similarly, the last cell on the map also depends on the value of σ. The last cells are 4030, 3965, 3900, 3835, and 3770 for σ values of .5, .75, 1.0, 1.5, and 2.0, respectively.

3.5.3 Marking Cells Within Borders

The algorithm assigns region numbers only to the cells located along the border of the region. The cells that are contained within the region will not be given a region number. The test used to start making a region is that a cell must have **Top** and **Left** zero-crossings, and that it cannot be part of a region. Assume that a cell <u>is</u> part of a region and has **Top** and **Left** zero-crossings. Since we have not given region numbers to cells located in the interior of regions, we can assume that the cell is part of the border of a region, and cannot be allowed to be part of another region. If the cell has **Top** and **Left** zero-crossings, and <u>has not been</u> assigned a region number, then we can start a new region using this cell as the starting cell. We can do this because this new region will either be adjacent to another region or be in the interior of one. Both of these possibilities are acceptable according to our definition of a region.

Once all the regions are found on a map, we will know what cells are located along the borders of these regions. We need to figure out a way to identify the cells contained within the borders and mark them with the appropriate region numbers. We must analyze exactly what information we use to decide what cells are contained within a region and what cells are not.

Looking at Figure 10, we notice that there is one region that takes up the entire 12 by 12 matrix, and there is a second region that is contained completely within the first, according to our definition of region. Using our algorithm for traversing the lines allows us to mark the cells around the border as we go along, but to mark the region number for the cells in the interior will require information about the path of the traversal.

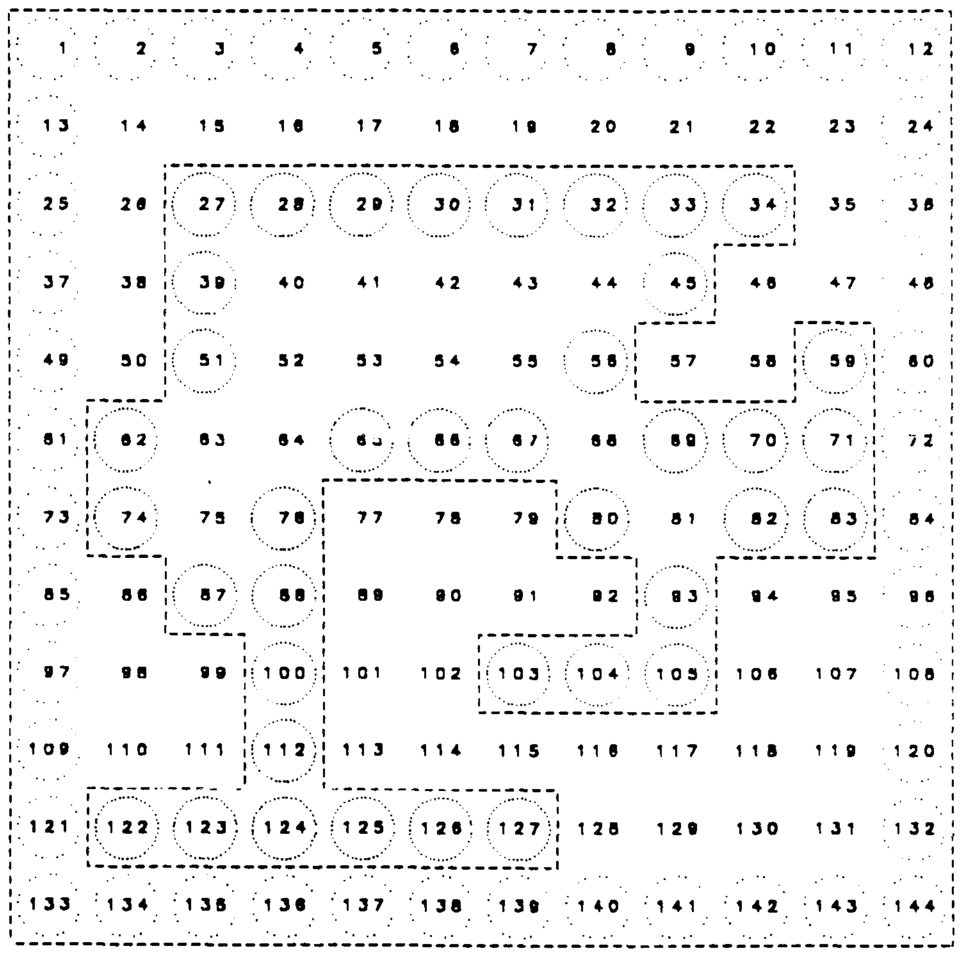


Figure 10: Marking Cells

We need to know what cells we went through and marked and which zero-crossing in the cell we were following. One way to keep track of the traversal path is to create a border-path as we traverse the zero-crossings to form a region. The border-path is a list of cell numbers and zero-crossings encountered while traversing the zero-crossings to form a region. It is represented as a list of lists. The lists contained within the outer list will have two elements. The first element is the cell number that we have come to in our traversal. The second element is the position of the zero-crossing. For example, the very first list in the border-path list will be "(first-cell Top)" indicating the first cell in the region and the first position of the zero-crossing we start the traversal. We continue adding these lists to the outer list as we traverse along the zero-crossings to form the region. Eventually, we reach the last element of the list. This will be "(first-cell Left)". Looking at our example, region 1 (the region that is the entire 12 by 12 matrix) has the following border-path:

```
( (1 Top) (2 Top) ... (12 Top)
  (12 Right) (24 Right) ... (144 Right)
  (144 Bottom) (143 Bottom) ...(133 Bottom)
  (133 Left) (121 Left) ... (1 Left) )
```

The cells contained in the border-path for region 1 have been circled with the fainter circles, and the cells contained in the border-path for region 2 (contained within region 1) have been circled with the more solid circle.

Now that this is set up, we can figure out what cells are within this border. The following algorithm will mark the cells contained in the region.

```
border_path_copy = border_path
cell_list = (pop border_path_copy)
while cell_list do
   if (cadr cell-list) = Right then
      cell = (car cell_list)
      while (list cell Left)
                  not member of border_path
                  and cell-1 not in region do
         mark cell-1 with region
         cell = cell - 1
      endwhile
   endif
   cell_list = (pop border_path_copy)
endwhile
```

Figure 11: Algorithm to mark all cells in a region

The algorithm goes through the border-path of a region, locates the cell that has a **Right** zero-crossing and marks the cells located to its left until it reaches a cell that has alreadv been marked, or one that contains a **Left** zero-crossing.

3.6 Comparison of Regions

Now that we have methods to locate and mark regions within a filtered image, we will apply them to various filtered images. In Figure 12, we can see the filtered images of an image for three

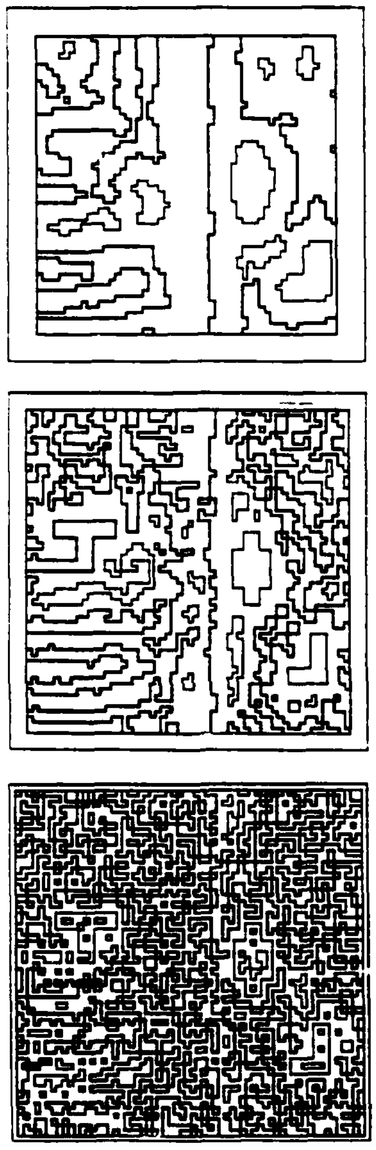
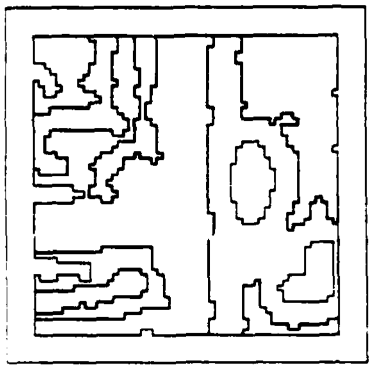


Figure 12: Filtered Images of Image

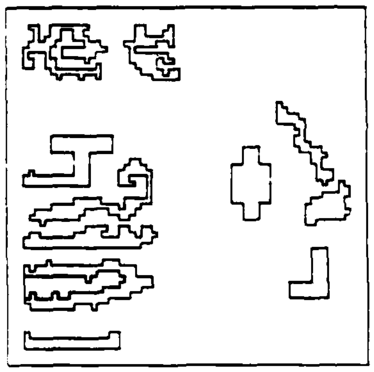
different values of σ. The σ values from top to bottom are 2.0, 1.0, and 0.5. The image that the filtered images were obtained from contains three objects that were placed in a noise background. There is a considerable amount of noise through the center of the image giving the effect of a mountain range. We would like to be able to isolate the three objects contained within the image.

To do this we will first threshold the images using the area of the regions as the thresholding factor. This will allow us to disregard many of the smaller regions formed when connecting the zero-crossings. Figure 13 shows the images after they have been thresholded.

35 < Threshold



35 < Threshold < 100



35 < Threshold < 100

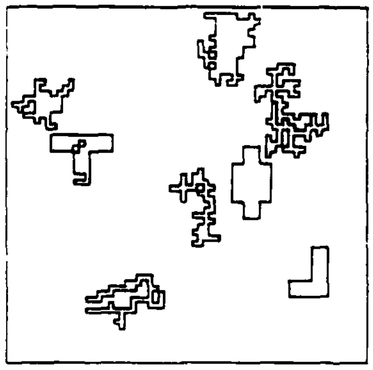
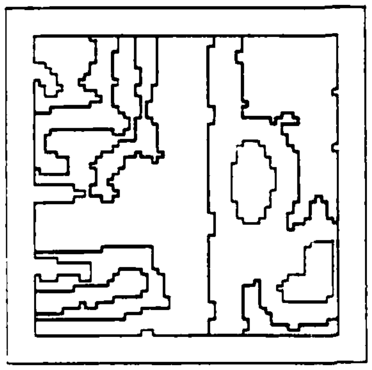


Figure 13: Thresholded Filtered Images

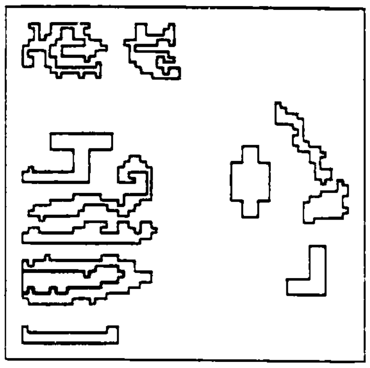
Once the images are thresholded, regions on two different images are compared to see if they match up. The comparison process involves placing a region contained in the image of the smaller σ on top of a region contained in the image of the larger σ, and matching up the cell numbers that make up the region. If there are more than a predetermined number of cells in the region of the smaller σ that do not match up with the cell numbers of the larger σ, then the regions do not correspond.

If, however, there are only a limited number of cells that do not match, then the regions do correspond, and the region in the smaller σ image is placed in a third image. This third image will contain all the regions that have matched up when the first two images were compared. This third image can then be compared with an image of an even smaller σ value, and so on, until we decide to stop.

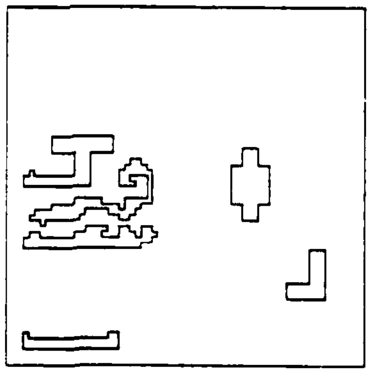
The next two figures, Figure 14 and Figure 15, illustrate the comparison process.



$\sigma=2.0$



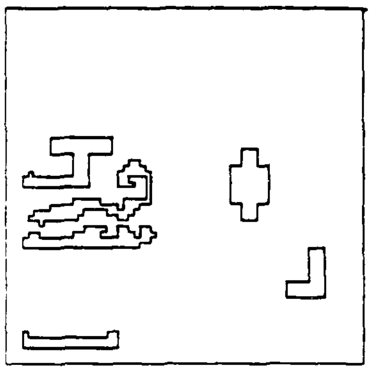
$\sigma=1.0$



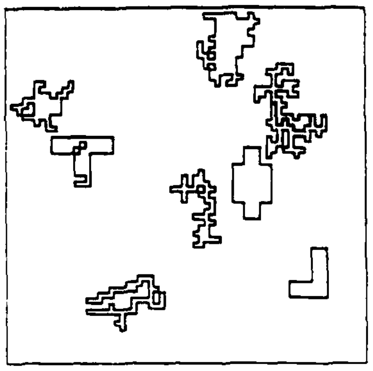
Result 1

Figure 14: Comparison Number 1

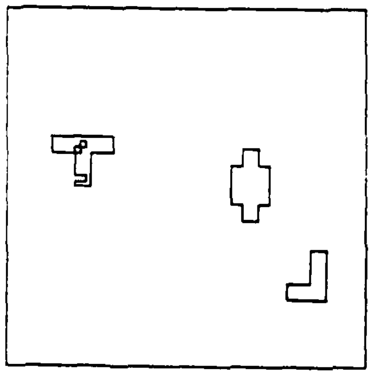
Figure 14 shows the result of comparing the thresholded images with σ values of 2.0 and 1.0.



Result 1



$\sigma = 0.5$



Result 2

Figure 15: Comparison Number 2

Then, Figure 15 shows the results of comparing the result of the previous comparison with the thresholded image of σ value of 0.5. The three objects placed in the noise background in the image have been isolated.

3.7 Summary

This chapter has described the implementation of the techniques discussed in Chapter 2. We now have a way to interpret a greyscale representation. We can detect and represent the intensity changes in the image by finding and storing zero-crossings. And, lastly, we have the capability to group these zero-crossings into regions, and comparing them across several spatial scales.

Chapter 4

RESULTS AND CONCLUSIONS

The application of the algorithms in the preceding chapter resulted in segmenting the initial digitized image into regions, and then regions were compared across several spatial scales to select particular regions that could represent objects contained in the image.

The results of applying the algorithms to images have been encouraging. The objects have been isolated on several images. The regions obtained using an image of a greater amount of noise tend to be distorted, and so make it difficult to isolate the objects completely. More testing needs to be done to determine at what point the algorithm fails to locate the objects as we increase the amount of noise.

Now that we have these regions that exist across several spatial scales, we can look at what we should do next. At this point we have the capability to analyze the image at a higher level. Instead of attempting to look at every cell in the image, we can look at groups of cells. This will allow us to give properties to each region that will distinguish it from the other regions. The purpose of doing this is to use the properties to identify the object that caused the region. The properties that can be given to the regions include such things as area, length, width, center of mass, and so on. A combination of some of the properties may be enough to identify the object, especially if we have an idea of what it could be.

This is only one of the possible ways to continue this project. Another possibility is to automate the decision-making process involved throughout the application of the algorithms. Decisions are made when selecting the threshold values for the region size, determining when to halt the comparison of regions, which values of σ will be more effective for a particular image, and so on. Rules can be developed to make these types of decisions, and an Expert System can be built.

Finally, a third possibility is to apply the algorithms to real images, and test whether or not the algorithms are as effective on them as on the generated ones. Real images will introduce several problems that need to be addressed if our vision system is to work properly. There are a number of factors that affect the intensity levels, and each factor must be considered for the results of the processing to be correct.

BIBLIOGRAPHY

1. Charniak, Eugene, and Drew McDermott. Introduction to Artificial Intelligence. Massachusetts: Addison-Wesley Publishing Company, Inc., 1985.

2. Grimson, William Eric Leifur. From Images to Surfaces. Massachusetts: The MIT Press, 1981.

3. Marr, David. Vision. San Francisco: W. H. Freeman and Company, 1982.