AN ALGORITHM FOR IMAGE CLUSTERS DETECTION AND
IDENTIFICATION BASED ON COLOR FOR AN AUTONOMOUS MOBILE ROBOT*

Domingo L. Uy, Ph.D., P.E.

RECEIVED

JAN 23 1995

OSTI

SUMMARY

This research is about the development of an algorithm for image clusters or "blobs" detection and identification based on color information. This research is motivated by the need to find a way of extending the sensing capability of an autonomous mobile robot to include vision system via fuzzy logic approach. In the fuzzy logic domain, there is no need to know precisely the size and color of an object. All the robot needs to know are the approximate size and color of the object. Based on this information, the robot can navigate itself and at the same time not colliding with its environment. As a first step of this research, an algorithm that will detect and identify image clusters (or object) is developed. The algorithm works as follows. The input image data are first processed using a crisp fuzzyfier, a binary smoothing filter, and a median filter, in that order. The processed image data are then inputted to the image clusters detection and identification program. The program employed the concept of "elastic rectangle" that stretches in such a way that the whole blob is finally enclosed in a rectangle.

A C program is developed to test the algorithm. The program is tested only on image data of 8 x 8 sizes with different number of blobs in them. The algorithm works very well in detecting and identifying image clusters.

*This work is supported by NRC HBCU Summer Faculty Research Participation Program, the DOE and the ORNL's Robotics and Process Systems Division at Oak Ridge, Tennessee. The program is administered by ORAU/ORISE

9/14/1994

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DOE/OR/00033--T670

AN ALGORITHM FOR IMAGE CLUSTERS DETECTION AND
IDENTIFICATION BASED ON COLOR FOR AN AUTONOMOUS MOBILE ROBOT*

A

Report

Submitted to ORAU/ORISE

Domingo L. Uy, Ph.D., P.E.

Department of Engineering
Hampton University
Hampton, Virginia 23666

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

August 19, 1994

*This work is supported by NRC's HBCU Summer Faculty Research Participation program, the DOE, and the ORNL's Robotics and Process Systems Division at Oak Ridge, Tennessee. The program is administered by ORAU/ORISE.

ACKNOWLEDGEMENT

The idea of using color for object detection and identification is due to Dr. Francois G. Pin , my research advisor at ORNL Robotics and Process and System Division. I thank him for his guidance and assistance in this project.

I would like to thank ORISE NRC's HBCU Summer Faculty Research Participation Program for their financial assistance and for giving me the opportunity to work here at ORNL, Oak Ridge, Tennessee.

Finally, I would like also to thank Timothy Noell, David Thompson, Kristi Morgansen and the people at ORNL RPSD division for their assistance in many ways during my ten weeks stay in the Lab.

ABSTRACT

An algorithm for detection and identification of image clusters or "blobs" based on color information for an autonomous mobile robot is developed. The input image data are first processed using a crisp color fuzzyfier, a binary smoothing filter, and a median filter. The processed image data is then inputed to the image clusters detection and identification program. The program employed the concept of "elastic rectangle", that stretches in such a way that the whole blob is finally enclosed in a rectangle.

A C-program is develop to test the algorithm. The algorithm is tested only on image data of 8x8 sizes with different number of blobs in them. The algorithm works very in detecting and identifying image clusters.

TABLE OF CONTENTS

	page
TITLE PAGE	i
ACKNOWLEDGEMENT	ii
ABSTRACT	iii
TABLE OF CONTENTS	iv
CHAPTER	
1. INTRODUCTION	1
2. PROJECT DESCRIPTION	3
3. HOW THE PROGRAM WORKS ?	4
3.1 Part I - Image Processing Algorithm	4
3.2 Part II - Image Clusters Detection and Identification Algorithm	4
4. FLOWCHART OF THE ALGORITHM	8
5. DATA AND RESULTS	11
6. DISCUSSION OF RESULTS	19
7. CONCLUSION	20
8. RECOMMENDATION FOR FUTURE WORK	21
REFERENCES	22
APPENDIX	
A - C-PROGRAM OF THE ALGORITHM	23

Chapter 1

INTRODUCTION

This project is about the development of an algorithm for image clusters or "blobs" detection and identification based on image color information.

The project is an outgrowth of finding methods or ways of extending the sensing capability of an autonomous mobile robot system (see [4], [5] & [6]). The given robot system in [4], [5], & [6] requires the (x,y) coordinates of the final destination, or target, be specified by the operator for the robot to automatically navigate itself to its destination, and at the same time avoid any collisions with its environment. There is no vision system in this robot.

One limitation of this robot is that it is not easy for the robot to arrive at its target destination, if the destination is not fixed, e. g., if the target is moving, like following a car. This is because it is difficult to specify all the possible paths of the target ahead of time. This is due to uncertainties of the direction of the target. The ability to track autonomously a moving target is an important problem in robot navigation. For if a mobile robot can do this task well, then it will open doors for what robot can accomplish, for example, in transportation system.

One way to solve this problem is to incorporate a vision system in the robot. But this approach introduces problems of its own. That is, it takes a lot of time to process frames of pixel image data. If the processing time is too long relative to the speed of the moving target, the robot will miss its target altogether. Thus, there is a need to find a way to reduce the image processing time. One possible way to solve this problem is to identify objects in the image based on its color property, for example, a red car. From this information, the approximate size of the red car can be identified.

In fuzzy logic domain, there is actually no need to know precisely the color of an object in the image. All we need to know are the shades of a color and the approximate sizes of the color blobs in the image. Thus, this problem boils down to a problem of detection and identification, or simply a problem of recognition of image clusters or blobs that has a certain color property.

For example, given a mobile robot that is following a red car on the road. If a blob of red color that is associated with the red car can be detected, identified, and sized in the image data, then the size of the object in the camera field of view (FOV) will give an indication of the distance of the red car from the robot. If the size of the blob is getting bigger, this will indicate that the red car is getting closer (relatively) to the robot, and if the size of the blob is getting smaller, this will indicate otherwise. And the robot will then execute the appropriate commands to avoid collision with the red car and with its environment.

This project attempts to solve this kind of a problem. As a first step, the goal is to develop an algorithm and a computer program to detect and identify blobs or image clusters. A C - program is written to implement the algorithm. Due to time limitations, the computer program are tested only on image data of 8x8 size matrices.

Chapter 2

PROJECT DESCRIPTION

This project is about the development of an algorithm for using color per se as a means of object detection and identification for an autonomous mobile robot system.

The approach is to use a crisp fuzzyfier to sift-out relevant color information and then processed the data in order to eliminate isolated points and noise. This is done by using a binary smoothing filter and a median filter. The processed image is then inputted to the object detection and identification program to search for image blobs. Each blob is then enclosed using an "elastic rectangle" that stretches in horizontal, vertical, and downward directions until the whole blob is enclosed in a rectangle. The process is then repeated for all blobs that are present in the image. The blob statistics for each blob, such as, the approximate location of the center gravity, the size of the rectangular enclosure, the actual size or pixel count of the blob, and volume bounded by the membership function values are then calculated.

The algorithm consists of two parts. Part I is about the processing of image data, and the output of Part I will serve as the input to Part II. Part II is about the image clusters or blobs detection and identification algorithm. The output of Part II are the number of blobs and blobs statistics.

3.1 Part I -- Image Processing Algorithm

Part I of the algorithm is an image processing algorithm and it works as follows:

- 1) Read an image pixel data. The data is passed through a crisp fuzzifier. If the value of the pixel, say p , is in the range of PL and PH ($PL \leq p \leq PH$), then the data is selected. Also, the data is assigned a membership function value of 1.0. Else, the data is set to zero. The process is repeated until all data are processed.

Note that PL and PH are the lower and upper limits of the pixel values of a color, say red.

- 2) The next step is to apply a binary smoothing filter to the image data obtained in step 1. The goals are to:
(a) remove isolated points; (b) fill in small holes in otherwise dark areas; and (c) fill in small notches in straight-edge segments.

- 3) The image data obtained in step 2 is then processed further using a median filter. This step will remove noise in the blobs and to force points with very distinct intensities to be more like their neighbors.

After these steps are done, the output will consist of color clusters or blobs with background filled with zeroes. This information will be fed to Part II - the image clusters detection and identification algorithm.

3.2 Part II - Image Clusters Detection and Identification Algorithm

The program operates based on the following assumptions:

- 1) The blobs do not overlap.
- 2) The rectangular enclosure for each blob does not intersect from each other.

The algorithm works based on a scanning search technique. The search starts from (0,0) of the processed image frame or data and then proceed from left to right, that is, from column 0 up to

the last column of the image data. Then, the search progresses to the next row, and continue up to the last row of the image. The image data is the output data of Part I. The details of the algorithm are as follows:

1) Read a data, say p . If the data is zero, read another data until a nonzero data is read.

2) If a nonzero data is obtained, then a blob is detected and the following decisions are made:

a) If it is the first blob, enclosed it in a rectangle. Calculate blob statistics - centroid measured with respect to, point $(0,0)$, area of rectangle, actual area, and volume. Actual area is the actual nonzero count of pixels in the rectangle, and the volume is the volume formed by the membership function values of the blob. The discussion for the enclosure method is given later in this section.

b) If not the first blob, the data point is checked whether or not it is part of an existing blob's rectangular enclosure (Note that only one rectangle will satisfy this condition. This is due to assumption (2)), If it is part of the rectangle, then move the search pointer to the upper right corner of the rectangle plus one (see Figure 1), and continue the search until a new blob is detected. If it is not part of an existing rectangle, then a new blob is detected. Once a new blob is detected, the blob is enclosed in a rectangle.

3) The process is repeated until all blobs are detected and enclosed in rectangles.

Figure 1.

3.3 Rectangular Enclosure Process

Every time a first nonzero point p in a blob is detected, the point is enclosed initially by a rectangle of the size 2×2

(or a 2x2 square). The point p is located at the upper-left corner of the rectangle. This is shown in Figure 2a. Figure 2b shows the starting points for left-dimension count, right-dimension count, and bottom-dimension count, respectively.

Figure 2a.

Figure 2b.

In short, the vertices of the square are the neighbors of point p . In here, a neighbor of point p is defined as the point immediately to the left, right or bottom of point p . Note that the bottom is actually the top, since the value of y coordinate increases toward south.

The search for expansion is done within the boundaries of the image frame. The search is conducted to the left, to the right and to the bottom of the square, in that order.

(1) For search-left, the data points immediately to the left of the rectangle are checked. If a nonzero pixel value is detected, the left-dimension of the rectangle is increased by one (left expansion). If not the left-dimension stays the same.

(2) For search -right, the data points immediately to the right of the rectangle are checked. If a nonzero pixel value is detected, the right-dimension of the rectangle is increased by one (right expansion). If not the right-dimension stays the same.

(3) For search bottom, the data points immediately below the bottom are checked. if a nonzero pixel value is detected, the bottom-dimension is increased by one. If not the following steps are done.

- (a) Search for lower-left, and
- (b) search for lower-right.

For the lower-left search, if a nonzero pixel is detected, the bottom-dimension is increased by one. And if the left-dimension did not change in (1), then increase the left-dimension by one. Else, the left-dimension remains the same. If the lower-left pixel is zero, then the bottom-dimension remains unchanged.

For the lower-right search, if a nonzero pixel is detected, the bottom-dimension is increased by one. And if the right-dimension did not change in (2), then increase the right-dimension

by one. Else, the right-dimension remains the same. If the lower-right pixel is zero, then the bottom-dimension remains unchanged.

(4) The process is repeated until all the dimensions (left-dimension, right-dimension, and bottom-dimension) remain unchanged. This indicates that a blob is fully enclosed in a rectangle. Note that the size of the rectangular enclosure stretches to the left, to the right, or to the bottom as the search procedures discussed above are successively executed.

Thus, one can see that the size of the rectangle keeps on expanding like a rubber band that stretches until the blob is enclosed. The flowchart of the algorithm is shown in Chapter 4.

Chapter 4

FLOWCHART OF THE ALGORITHM

Flowhart of part I - Image Processing Algorithm

Flowchart of Part II ---
Image Clusters Detection and Identification

FIND SEARCH DIRECTIONS -- to the left, to the right, and to the bottom (actually this is the top) -- for the rectangular enclosure. The search starts with a 2x2 rectangle (or square). The pixel p is at the upper-left corner of this rectangle (or square). The search is done until all the neighboring points of the blob are enclosed in a rectangle. This is done by successively increasing the size of the rectangle to the left, right, and bottom, whenever a neighboring point is found. In here, a neighborhood is defined as a point immediately next to a pixel p - to the left, right, and bottom only.

OBTAIN THE SIZE OF THE RECTANGULAR ENCLOSURE

COMPUTE BLOB STATISTICS:

- a) size of the rectangular enclosure;
- b) approximate distance of centroid of the blob with respect to the origin (0,0);
- c) volume of the surface bounded by membership function data

UPDATE BLOB COUNT

UPDATE SEARCH POINTERS

Chapter 5

DATA AND RESULTS

The algorithm was implemented in C program. The program consists of two part. The first part is the image processing part and the second part is the program for detecting and identifying blobs. Each part was written, debug, and tested separately. This is to make the program tracktable and manageable. Also image data of 8x8 sizes are used as test cases. Due to time limitations, only key test results are presented here.

Case 1. The 8x8 matrix below is used as sample image data.

1	0	3	0	0	0	0	0	(0,0)
0	0	0	0	15	0	0	0	
0	5	0	12	16	13	0	0	
0	7	0	15	17	14	0	11	
0	0	0	16	18	12	0	0	
0	0	7	0	19	0	0	16	
9	0	0	0	0	0	0	0	
10	0	0	2	5	0	20	0

This image data is processed by the first part I of the algorithm and yielded the following results.

The crips fuzzyfier is asssumed to be:

$$m(\text{pix}) = \begin{cases} 1, & 10 \leq \text{pix} \leq 20 \\ 0, & \text{otherwise.} \end{cases}$$

input image data:

1	0	3	0	0	0	0	0
0	0	0	0	15	0	0	0
0	5	0	12	16	13	0	0
0	7	0	15	17	14	0	11
0	0	0	16	18	12	0	0
0	0	7	0	19	0	0	16
9	0	0	0	0	0	0	0
10	0	0	2	5	0	20	0

output of crisp fuzzyfier:

0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	1	1	1	0	0
0	0	0	1	1	1	0	1
0	0	0	1	1	1	0	0
0	0	0	0	1	0	0	1
0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	0

The sifted pixels are given by:

0	0	0	0	0	0	0	0
0	0	0	0	15	0	0	0
0	0	0	12	16	13	0	0
0	0	0	15	17	14	0	11
0	0	0	16	18	12	0	0
0	0	0	0	19	0	0	16
0	0	0	0	0	0	0	0
10	0	0	0	0	0	20	0

The output of the binary smoothing filter is given below.

0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	1	1	1	0	0
0	0	0	1	1	1	0	0
0	0	0	1	1	1	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

The equivalent pre-processed data for input to the median filter:

0	0	0	0	0	0	0	0
0	0	0	0	15	0	0	0
0	0	0	12	16	13	0	0
0	0	0	15	17	14	0	0
0	0	0	16	18	12	0	0
0	0	0	0	19	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

output of median filter:

```

0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 12 14 13 0 0
0 0 0 15 15 13 0 0
0 0 0 15 15 12 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

```

This data is then passed to Part-II of the algorithm for blob detection and identification. The output results are as follows.

input processed image data:

```

0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 12 14 13 0 0
0 0 0 15 15 13 0 0
0 0 0 15 15 12 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

```

The color membership function is assumed to be:

pixel value, pix	membership function, m(pix)
------------------	-----------------------------

0	0.00
1	0.00
2	0.00
3	0.00
4	0.00
5	0.00
6	0.00
7	0.00
8	0.10
9	0.20
10	0.30
11	0.40
12	0.50
13	0.60
14	0.70
15	0.80
16	1.00
17	1.00
18	1.00
19	1.00
20	1.00

The blob identified is given by:

The number of blobs found is 1.

```

12 14 13
15 15 13
15 15 12

```

The blob statistic is given by

blob#	x-bar	y-bar	a-area	r-area	volume
1	5	4	9	9	6.10

The rectangular enclosure is given by:

blob#	corner_1	corner_2	corner_3	corner_4
1	(3 , 2)	(5 , 2)	(5 , 4)	(3 , 4)

The corners are shown below.

Case 2. In this case, Part - II of the algorithm is further tested for blobs detection and identification, using the test data below. This image data consists of 5 blobs.

Sample image data:

```

8  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0
0  0  0  15 15 15 0  0
9  0  0  15 15 15 0  0
9  0  0  0  0  0  0  0
9  0  0  0  0  0  0  0
0  0  0  0  0  0  18 18
0  0  0  20 20 0  0  0

```

The color membership function is the same as in Case 1.

The following blobs are identified:

blob #1:

8 0
0 0

blob #2:

15 15 15
15 15 15

blob #3:

9 0
9 0
9 0

blob #4:

18 18
0 0

blob #5:

20 20

The number of blobs found is 5.

The blob statistics are given by:

blob#	x-bar	y-bar	a-area	r-area	volume
1	1	1	1	0	0.10
2	5	3	6	6	4.80
3	1	5	3	6	0.60
4	7	7	2	6	2.00
5	4	8	2	6	2.00

The rectangular enclosures are given by:

blob#	corner _1	corner_2	corner_3	corner_4
1	(0 , 0)	(1 , 0)	(1 , 1)	(0 , 1)
2	(3 , 2)	(5 , 2)	(5 , 3)	(3 , 3)
3	(0 , 3)	(1 , 3)	(1 , 5)	(0 , 5)
4	(6 , 6)	(7 , 6)	(7 , 7)	(6 , 7)
5	(3 , 7)	(4 , 7)	(4 , 7)	(3 , 7)

Case 3. Here is an example in which the blob is not regular in shape. The image data is shown below.

```

8   9   10   0   0   0   0   0
11  12  13   0   0   0   0   0
14  15  16   0   0   0   0   0
9   0   0   0   0   0   0   0
9   0   0   0   0   0   0   0
9   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0

```

The number of blobs found is 1.

```

8   9   10   ;
11  12  13
14  15  16
9   0   0
9   0   0
9   0   0

```

The blob statistics are given below.

blob#	x-bar	y-bar	a-area	r-area	volume
1	2	3	12	18	5.20

The rectangular enclosure is given by:

blob#	corner _1	corner_2	corner_3	corner_4
1	(0 , 0)	(2 , 0)	(2 , 5)	(0 , 5)

Case 4, Another example is the case in which one blob size is not regular.

image data

```

0  0  0  0  0  0  0  0
0  0  0  0  10 0  0  0
0  0  0  8  8  0  0  0
0  0  7  8  8  9  0  0
0  0  0  8  8  0  0  0
0  0  0  0  0  0  0  0
0  0  0  0  0  0  9  9
0  0  0  0  0  0  9  9

```

The blobs detected are given below.

blob #1:

;

```

0  0  10  0
0  8  8  0
7  8  8  9
0  8  8  0

```

blob #2:

```

9  9
9  9

```

The number of blobs found is 2.

The blobs statistics:

blob#	x-bar	y-bar	a-area	r-area	volume
1	4	3	9	16	1.10
2	7	7	4	16	0.80

The rectangular enclosures:

blob#	corner_1	corner_2	corner_3	corner_4
1	(2 ,1)	(5 ,1)	(5 ,4)	(2 ,4)
2	(6 ,6)	(7 ,6)	(7 ,7)	(6 ,7)

;

Chapter 6

DISCUSSION OF RESULTS

The results indicated that the algorithm works very well in identifying image clusters. A crisp fuzzyfier is important in sifting out relevant color information. The binary smoothing filter is helpful in removing isolated points and the median filter performed as expected. The blob detection and identification works very well. It even detected a blob with one pixel in it.

Case 1 shows the complete results of the algorithm. and one blob is detected. Cases 2, 3, and 4 show more of the results of part II of the algorithm. This is because it is the most important part of the algorithm. Also, this is to show the capability of the algorithm in detecting and identifying more than one blobs .

Chpater 7

CONCLUSION

An image clusters or blobs detection and identification algorithm for autonomous mobile robot is developed. The crisp fuzzyfier and binary smoothing and median filters are useful in processing the image data for image clusters detection and identification. The computer program is tested for various sample image data and it shows that the algorithm works very well. It yielded very good results. The blobs in the test cases were detected and identified correctly.

Chapter 8

RECOMMENDATION FOR FUTURE WORK

I would recommend the following for further studies:

- 1) Test the algorithm for real image data.
- 2) Develop ways to speed-up the computer processing time.
- 3) If (1) and (2) are successful, implement the improved algorithm in the mobile robot discussed in [4], [5], or [6], and see how this algorithm works with the whole system.

REFERENCES

1. Fu, K.S., Gonzales, R.C., and Lee, C.S.G., "Robotics -- Control Sensing, Vision, and Intelligence," New York: McGraw-Hill, 1987, chapters 7 and 8.
2. Newman, W.M., and Sproull, R.F., "Principles of Interactive Graphics," 2nd ed., New York: McGraw-Hill Book Co., 1979.
3. Horn, B.K. Paul, "Robot Vision," Cambridge, MA: MIT Press, 1986, chapter 3.
4. Pin, F.G. & Watanabe, Y., Steps Towards Sensor-Based Vehicle Navigation in Outdoor Environment Using Fuzzy Behaviorist Approach, J. of Intelligent and Fuzzy Systems, Vol.1, pp. 95-107, Vol. 1, pp. 95-107, John Wiley and Sons, 1993.
5. Watanabe, Y. & Pin, F.G., Sensor-Based Navigation of a Mobile Robot Using Automatically constructed Fuzzy Rules, Proc. of International Conference on Advanced Robotics (ICAR), Tokyo, Japan, pp. 81 -87, Nov. 1-2, 1993.
6. Pin, F.G., Watanabe, H., and Symon, J, & Pattay, R.S., Autonomous Navigation of a Mobile Robot Using Custom-Designed Qualitativ Reasoning VLSI Chips and Boards, Proc. of 1992 IEEE International Conference on Robotics on Automation, Nice, France, pp. 123-128, May, 1992.
7. Zadeh, L., Fuzzy Logic, Information Control, IEEE Computer Society, pp. 83-93, 1988.
8. Zadeh, L., Outline of a New Approach to the Analysis of Complex Decision-Making Approach, IEEE Trans. of Systems, Man and Cybernetics SMC 3(1), pp. 28-45, 1973.
9. MATLAB, The Mathworks, Inc., 1993.
10. Waite, Michell, Martin, D.; and Prata, S., "Unix Primer Plus," Indianapolis: Sams & Co., Inc., 1983.
11. Chien, C.C., "Programmer's Guide to Turbo C2," New York: Bantam Books, 1989.
12. Kelly, Al & Pohl, Ira, "A Book on C", California: Benjamin/Cummings, 1984.

APPENDIX A

C - PROGRAM OF THE ALGORITHM

```

#include <stdio.h>
#include <stdlib.h>
#define LAST_ROW 7
#define LAST_COL 7
#define COLVAL_LO 10
#define COLVAL_HI 20
void myprint_out ();
int sort_temp ();

/*      program name:  juno.c                      */
/*      Dr. Domingo L. Uy, P.E.                    */
/*      Hampton University                          */
/*      Hampton, Virginia 23666                     */
/*      August 15, 1994                             */
/*      .                                             */
/*      Reserach Advisor:                           */
/*      Dr. Francois G. Pin                         */
/*      This work is supported by NRC's HBCU Faculty */
/*      Summer Research Participatio Program under  */
/*      the ORAU/ORISE administration.  Also supports */
/*      come from DOE and ORNL                      */
/*      This is a program to process image data based on : */
/*      color information .                          */
/*      The program sift the data based on          */
/*      range or ranges of color memebership funtion(s), */
/*      for example, ranges of blue and red colors.  */
/*      A simple crips-type fuzzifier is used for this */
/*      purpose.  The data obtained is then processes using */
/*      binary smoothing and median filters.         */
/*      The output of this program will consist of image blobs */
/*      or clusters.  A simple 8x8 matrix is used to test the */
/*      the program.                                */

main ( )
{

    int image[8][8], med_data[8][8], temp[9];
    int row, col;
    int new_data[8][8];
    int bin_data [8][8];
    int pix[8][8];
    int not_lastrow, not_lastcol;
    int p, i,j,r,c,b1,b2;
    int a,b,d,e,f,g,h;
    int cy, med1;

/*      Read in test data from pick2.dat file      */
/*      pick2.dat file an 8x8 matrix                */

    FILE *datafile;

```

```

/*      char *filespec = (char *) malloc (sizeof (char) * 13); */
/*      char *filespec2 = (char *) malloc (sizeof (char) * 13); */
/*      printf ("Enter input_file:"); */
/*      gets (filespec); */
      datafile = fopen ("pick2.dat", "r");
      if (datafile == NULL) {
          puts ("cannot open file.");
      }
      else {
          for (i = 0; i <= LAST_ROW; i++) {
              for (j = 0; j <= LAST_COL; j++) {
                  fscanf (datafile, "%d", &image[i][j]);
              } /* for j */
          } /* for else */
      }
      myprint_out (image);
      fclose (datafile);

/*
/*      color masking
/*      In this test the color filter is in the range of
/*      COLVAL_LO to COLVAL_HI inclusive
/*      Addition if-else statement can be inserted to sift
/*      other color spectrum
*/

for (row = 0 ; row <= LAST_ROW; row++) {
    for (col = 0; col <= LAST_COL; col++) {
        p = image[row][col];
        if (p >= COLVAL_LO && p <= COLVAL_HI) {
            pix[row][col] = p;
            bin_data[row][col] = 1;
        }
        else {
            pix[row][col] = 0;
            bin_data[row][col] = 0;
        }
    } /* for col */
} /* for row */

myprint_out (bin_data);

myprint_out (pix);
/* procedure for pre-processing */

printf ("\n pre-processing -- binary smoothing \n");

/* initialize newdata array */

not_lastrow = LAST_ROW - 1;
not_lastcol = LAST_COL - 1;

for ( i = 0; i <= LAST_ROW; i++) {
    for (j = 0; j <= LAST_COL; j++){
        new_data[i][j] = 0; med_data [i][j] = 0;
    } /* for i */

a = LAST_ROW;
b = LAST_COL;

/*      fill-in the boundaries with zeroes
*/

```

```

/*      Note that filling the boundaries with zero will not effect the      */
/*      the objective of the program.  Thus, this approach is used        */
/*      for simplicity.  In short, binary smoothing and median filters    */

for ( j = 1; j <= not_lastcol; ) {
    new_data[0][j] = 0;
    new_data[a][j] = 0;
    j++;}
for (j = 0; j <= LAST_ROW;) {
    new_data[j][0] = 0;
    new_data[j][b] = 0;
    j++;}

/*      binary smoothing      */

for(r=1; r <= not_lastrow; r++) {
    for (c=1; c <= not_lastcol; c++){

        a = bin_data[r-1][c-1];
        b = bin_data[r-1][c];
        cy = bin_data[r-1][c+1];
        d = bin_data[r][c-1];
        e = bin_data[r][c+1];
        f = bin_data[r+1][c-1];
        g = bin_data[r+1][c];
        h = bin_data[r+1][c+1];
        p = bin_data[r][c];

        /* b1 == p+bg(d+e) + de(b+g) */
        b1 == p || ( b && g && (d || e)) || ( d && e && (b || g));
        p == b1;

        new_data[r][c] = p;} /* for col */
    myprint_out (new_data);
}/* for row */

myprint_out (new_data);
*/
for ( row =0; row <= LAST_ROW; row++){
    for (col =0; col <= LAST_COL; col++) {
        bin_data[row][col] = new_data[row][col];}
    } /* for row */

for (r=1; r <= not_lastrow; r++) {
    for (c=1; c <= not_lastcol; c++){

        a = bin_data[r-1][c-1];
        b = bin_data[r-1][c];
        cy = bin_data[r-1][c+1];
        d = bin_data[r][c-1];
        e = bin_data[r][c+1];
        f = bin_data[r+1][c-1];

```

```

g = bin_data[r+1][c];
h = bin_data[r+1][c+1];
p = bin_data[r][c];

```

```

/* b2 == p [(a+b+cy). (e+g+h) + (b+cy+e). (d+f+g)] */

```

```

b2 == p && ((a || b || cy) && (e || g || h) || (b || cy || e) && (d ||
f || g));

```

```

p == b2;

```

```

new_data[r][c] = p;} /* for col */

```

```

myprint_out (new_data);

```

```

} /* for r */

```

```

myprint_out (new_data);

```

```

/* decoding of data */

```

```

for (row =0; row <= LAST_ROW; row++ ) {
    for ( col = 0; col <= LAST_COL; col++) {
        new_data[row][col] = ((new_data[row][col]) * (pix [row][col]));}
    } /* for row */

```

```

printf(" %s print pre-processed data -- data to median filter" );

```

```

myprint_out (new_data);

```

```

printf ("\n");

```

```

/*
/* median filtering */

```

```

a = LAST_ROW;
b = LAST_COL;
for (j=1; j <= not_lastcol; j++) {
    med_data[0][j] = new_data[0][j];
    med_data[a][j] = new_data[a][j];}

```

```

for (j = 0; j <= LAST_ROW; j++) {
    med_data[j][0] = new_data[j][0];
    med_data[j][b] = new_data[j][b];}

```

```

for (row =1; row <= not_lastrow; row++){
    for (col =1; col <= not_lastcol; col++) {
        r = row;
        c = col;

```

```

temp[0] = new_data[r-1][c-1];
temp[1] = new_data[r-1][c];
temp[2] = new_data[r-1][c+1];
temp[3] = new_data[r][c-1];
temp[4] = new_data[r][c];
temp[5] = new_data[r][c+1];

```



```

temp[6] = new_data[r+1][c-1];
temp[7] = new_data[r+1][c];
temp[8] = new_data[r+1][c+1];

med1 = sort_temp (temp);

med_data[row][col] = med1;
printf (" med1 %d = ", med1);
printf ("\n");
} /* for row */
printf ("\n, output of median filter");

myprint_out (med_data);

/* write to output file */

/*      printf ("Enter filespec2:"); */
/*      gets (filespec2); */
datafile = fopen ("pout.dat", "w");

    if (datafile == NULL ) {
        puts ("cannot open file2." );}
    else {
        for (i = 0; i <= LAST_ROW; i++) {
            for (j = 0; j <= LAST_COL; j++){
                fprintf (datafile, "%d ", med_data. [i][j]);}
            fprintf (datafile, "\n");
        } /* for j */
        } /* for else */
        fclose (datafile);

} /* main */
void myprint_out (image)

    int image [8][8];

{
    int row, col;
    printf ("\n print output\n");
    for (row =0; row <= LAST_ROW; row++) {
        for (col =0; col<=LAST_COL; col++) {
            printf("%d ", image[row][col]);}
        printf("\n"); /* new line */
    } /* for row */
}/* myprint_out */
int sort_temp( temp)
    int temp[9];
{
    int k,i, t, med, km;

    for (i = 0; i <= 8; i++) {
        printf ("%d ", temp [i]);}
        printf ("\n");

    for ( i =0; i <= 7; i++) {
        t = temp[i];
        for (k = i+1; k <= 8; k++ ) {
            if (t <= temp[k]) {t =t.}

```

```

        else{
            temp[i] = temp[k];
            temp[k] = t;
            t = temp[i];}
        }/* for k */
        }/* for i */
    for (i = 0; i <= LAST_ROW; i++){
        printf ("%d ", temp[i]);

        printf ("\n");

/* get the median value */

        km = (LAST_ROW/2) + 1;
        med = temp[km];
        return ( med );
    } /* sort_temp */

```

```

#include <stdio.h>
#define LAST_ROW 7
#define LAST_COL 7
void myprint_out ();
/*
/*
/*      program to identify image blobs or clusters
/*
/*      Dr. Domingo L. UY, P.E.
/*      Hampton University
/*      August 15, 1994
/*
/*      Research Advisor:
/*      Dr. Francois G. Pin
/*
/*      This work is supported by NRC's HBCU Faculty
/*      Summer Research Participation Program under
/*      ORAU/ORISE administration. Also supports come
/*      from DOE and ORNL
/*
/*      Program Limitations:
/*      1) The object enclosed in a rectangle
/*          must not overlap.
/*      2) This program cannot detect a blob
/*          enclosed within a blob.
/*
/*
*/

```

```

main ()
{
    int p = 0;
    int xi = 0;
    int yi = 0;
    int i = 0;
    int j = 0;
    int ix = 0;
    int jx = 0;
    int count = 0;
    int iblob = 0;
    int l=0;
    int count_left = 0;
    int count_right = 0;
    int count_vert = 0;
    int xhigh = 0;
    int xlow = 0;
    int yhigh = 0;
    int nsum = 0;
    int NHR = 0;
    int NHL = 0;
    int NHV = 0;
    int q1 = 0;
    int q2 = 0;
    int q3 = 0;
    int ncount;
    int LH = 0;
    int LL = 0;
    int Tlength = 0;
    int Tarea = 0;
    int Theight = 0;

```

```

int VL = 0;
int VH = 0;
int px= 0;
int xmid= 0;
int ymid =0;
int xu[10], xl[10], yu[10], yl[10];
int length[10], TXL[10], TYL[10], area[10];
int height [10], nncount [10], actual_area[10];
int fill[10], x_centroid[10], y_centroid[10];
int flag = 0, xarea = 0;
float volume[10];
int med_data [8][8];
float mf, sum_mem;
float color_mem[21] = {0,0,0,0,0,0,0,0,0,
                      0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,
                      1.0,1.0,1.0,1.0,1.0};

```

```

/* test data */

```

```

FILE *datafile;
datafile = fopen ("vick.dat", "r");
if (datafile == NULL ) {
    puts ("Cannot open file."); }
else {
    for (ix = 0; ix <= LAST_ROW; ix++) {
        for (jx = 0; jx <= LAST_COL; jx++) {
            fscanf (datafile, "%d", &med_data[ix][jx]);}
        } /* for jx */
    } /* for else */
myprint_out (med_data);
fclose (datafile);

```

```

/* sample test data form : */
/* int med_data [8][8]= {8,0,0,0,0,0,0,0, */
/*                        0,0,0,0,0,0,0,0, */
/*                        0,0,0,15,15,15,0,0, */
/*                        9,0,0,15,15,15,0,0, */
/*                        9,0,0,0,0,0,0,0, */
/*                        9,0,0,0,0,0,0,0, */
/*                        0,0,0,0,0,0,18,18, */
/*                        0,0,0,20,20,0,0,0}; */

```

```

/* initialization of all arrays */

```

```

ix =0; jx = 0;

for (ix =0; ix <= 9; ix++)
{
    xu[ix] =0;      xl[ix]=0;
    yu[ix] =0;      yl[ix] =0;
    TXL[ix]=0;      TYL[ix]=0;
    length[ix]=0;    height[ix]=0;
    nncount[ix]=0;   volume[ix]=0.0;
    fill[ix]=0;      actual_area[ix]=0;
    x_centroid[ix]=0; y_centroid[ix]=0;
    area [ix] = 0;}

```

```

for (ix =0; ix <= 20; ix++)
printf ("color_mem = %f\n", color_mem[ix]);

myprint_out (med_data);

/*
/*
/* Start the process...
/*
/* Read data
*/

i =0;
j =0;

LOOP:  if ( i <= LAST_ROW)      /* tif #1 */
      {
LOOP5:  if (j <= LAST_COL)      /* tif #2 */
      {
          p = med_data[i][j];

          if (p == 0 ) /* tif #3 */
          {
              /* update column */

              j = j + 1;
              xi = j ;

              goto LOOP5;} /* tif #3 */

          else { /* telse #3 */
              goto LOOP7; } /* telse #3 */
      } /* tif #2 */

      else /* telse #2 */
      { /* update row and reset column */

          i = i +1;      ;
          yi = i;

          xi =0;
          j = xi; /* start at column 0 */

          goto LOOP; } /* telse */
      } /* tif #1 */
else /* telse #1 */

{ /* telse #1 -- super super else */

/* print final output results */

printf ("Print output results\n");
printf ("The number of blobs found is %d ", iblob);
/* more print statement here */

goto ZPRINT; }

/* Identify where the nonzero point (or pixel) belongs */

```

```

/* to a blob that was previously identified. If so, */
/* skip this blob in an horizontal scan manner, and search */
/* for another blob on the same line. If no other existing */
/* blob is found, then a new blob is identified. Proceed the */
/* process of enclosing it in a rectangle. */

```

```

/* store coordinates */

```

```

LOOP7: { TXL[iblob] = xi;
        TYL[iblob] = yi;
        printf ("TXL = %d , and TYL = %d \n", xi, yi);
        printf ("yi, xi,p, and i = %d %d %d %d \n", yi,xi,p,i);
        flag =0;

        if (iblob == 0) { goto ZM;} /* get the first blob */
        else { l = 1; goto LOOP3;}

LOOP3: for (l =1; l <= iblob; ) /* for l */
        {
            if ((xi >= xl[l]) && (xi <= xu[l]) ) /* if #2 */
            {
                if ((yi >= yl[l]) && (yi <= yu[l])) /* if #3 */
                {
                    count = 1;
                    printf ("It is inside the blob\n");
                    printf ("count and l = %d %d \n", count, l);
                    goto GH; } /* if #3 */

                else /* else #3 */
                {
                    count =0;
                    printf ("It is outside the blob\n");
                    l++;} /* else #3 */

            } /* if #2 */

            else /* else #2 */
            {
                count = 0;
                printf ("It is outside the blob\n");
                l++;}

        } /* for l */
/* ***** */

```

```

GH:      if (count > 0) /* if count */
        {
            /* identified old or existing blob */
            /* old blob identified */
            /* get another data */

            xi = xu[count] + 1;
            j = xi;
            yi = yi;

```

```
i = yi;
```

```
printf ("xi,j,yi,i = %d %d %d %d \n", xi,j,yi,i);  
printf ("count =%d\n", count);
```

```
if (xi <= LAST_COL) { goto LOOP;} /* if count */  
else /* else #4 */  
{  
    /* set the following values */  
    j =0;  
    xi =0;  
    yi =yi + 1;  
    i = yi;
```

```
    /* get another data */  
    printf ("xi,j,yi,i = %d %d %d %d \n", xi,j,yi,i);  
    printf ("count =%d -- the blob number\n", count);
```

```
    printf ("No new blob Identified\n");  
    printf ("iblob = %d\n", iblob);
```

```
        goto LOOP;} /* else #4 */  
    } /* if of count */
```

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

```
else /* else of count */  
{ goto ZM; } /* a new blob is identified */
```

```
/* Look for new blob */  
/* */
```

```
ZM:      iblob = iblob + 1;      ;
```

```
printf ("iblob = %d \n",iblob);  
/* perform search for the direction of a pixel rectangular */  
/* expansion - to the Left, to the Right, or to the */  
/* Bottom */
```

```
if (xi == LAST_COL) { count_right = 0; goto AA;}  
else { count_right = 1; goto AA;}
```

```
AA:      if (yi == LAST_ROW) { count_vert =0; goto BB;}
```

```
else {count_vert = 1; goto BB;}
```

```
BB:      {count_left = 0 ; goto ZR;}
```

```
ZR: {      NHR = 0; /* ZR */  
        NHL = 0;  
        NHV = 0;
```

```
/* start with a small pixel rectangle of the size 2x2, and expand */  
/* from there */
```

```

        xlow = xi - count_left;
        xhigh = xi + count_right;
        yhigh = yi + count_vert;

printf ("xi and yi = %d %d \n", xi, yi);
printf ("ylow and yhigh = %d %d \n", yi, yhigh);

printf ("xlow, xhigh, and yhigh = %d %d %d \n", xlow, xhigh, yhigh);
printf ("\n");

        for (ix = yi; ix <= yhigh; ix++)
        { for (jx = xlow; jx <= xhigh; jx++)

                printf (" %d ", med_data[ix][jx]);
                printf ("\n"); /* new line */
        } /* for ix */
        printf ("\n");

if (xlow > 0)                                /* if #2 */
{
    if (xhigh > LAST_COL)                    /* if #3 */
    {
        NHR = 0;
        xhigh = LAST_COL;

        if (yhigh > LAST_ROW)                /* if #4 */
        {
            yhigh = LAST_ROW;
            NHV = 0;
            /* do search left */

            /* Number 1 */

printf ("Number 1 in search direction\n");
printf ("NHR, NHL, NHV %d %d %d \n", NHR, NHL, NHV);
            if ((xlow-1) >= 0) {goto KR1;}
            else {NHL =0; xlow=0; goto Z1;}

KR1:
            for (jx = yi; jx <= yhigh;) {
                q1 = med_data [jx][xlow-1];
                if (q1 > 0 ) {NHL = NHL +1; jx++;}
                else {jx++;}}

                goto Z1;
            }/* if #4 */

            else /* else #4*/
            {
                /* do search left and search Bottom */

                /* Number 2 */

printf ("Number 2 in search direction\n");
printf ("NHR, NHL, NHV %d %d %d \n", NHR, NHL, NHV);

                if ((xlow-1) >= 0) {goto KR2;}
                else {NHL =0; xlow =0; goto KJ;}

```



```

        q3 = med_data[yhigh+1][xlow-1];
        if ( q3 == 0 ) /* bif #4 */
        {
            NHL=NHL; NHV = NHV;

            q3 = med_data[yhigh+1][xhigh+1];

            if (q3 == 0 ) {NHR=NHR;
                NHV=NHV; goto Z1;} /* path e */

            else
            {
                if (NHR == 0 ) {NHR = NHR +1;
                    NHV=NHV+1; goto Z1;} /* path f */

                else {NHR = NHR;

                    NHV =NHV+1; goto Z1;} /* path g */
                }
            } /* bif #4 */

        else /* belse #4 */
        {
            if (NHL == 0 ) {NHL = NHL +1;
                NHV=NHV +1;
                    goto KK1;} /* path h */
            else {NHL = NHL; NHV= NHV +1;
                goto KK1;} /* path i */
        }

KK1:
        q3 = med_data[yhigh+1][xhigh + 1];
        if (q3 == 0 ) { NHR = NHR; goto Z1;} /* path l */
        else { if (NHR == 0 ) { NHR = NHR +1;
            goto Z1; } /* path k */
            else {NHR = NHR; goto Z1; } /* path j */
        } /* for else */

        } /* belse #4 */
        } /* belese #3 */
    } /* belese #2 */

    } /* bif #1 */

else { /* belse #1 */
    goto Z1; }

/* .....search bottem (tail)..... */

    } /* else #4 */

} /* if # 3 */

else /* else #3 */

```

```

{
    if(yhigh >= LAST_ROW)    /* if #4 */
    {
        NHV =0;
        yhigh = LAST_ROW;
        /* do search left and right */
    }
    printf ("Number 5 in search direction\n");
    printf ("NHR, NHL, NHV %d %d %d \n", NHR, NHL, NHV);

```

```

        /* Number 5 */

```

```

    /* do search left */

```

```

    if ((xlow -1) >= 0 ) {goto KR3;}
    else {NHL =0; xlow = 0; goto KM;}

```

```

KR3:        for (jx = yi; jx <= yhigh;){
                q1 = med_data[jx][xlow-1];
                if(q1 > 0 ) {NHL = NHL +1; jx++;}
                else {jx++;}}

```

```

    /* do search right */

```

```

KM:        if ((xhigh +1) <= LAST_COL) { goto KQ1; }
            else { NHR = 0; xhigh =LAST_COL;goto Z1;}

```

```

KQ1:        for (jx = yi; jx <= yhigh;){
                q2 = med_data[jx][xhigh+1];
                if (q2 > 0) {NHR = NHR + 1 ;jx++;}
                else {jx++;}}
            goto Z1; }

```

```

    else /* else #4 */
    {

```

```

        /* do search left, right and bottom */

```

```

        /* Number 6 */

```

```

    printf ("Number 6 in search direction\n");
    printf ("NHR, NHL, NHV %d %d %d \n", NHR, NHL, NHV);

```

```

    /* search Left */

```

```

    if ((xlow -1) >= 0 ) {goto KR4;}
    else {NHL =0; xlow =0; goto KP1;}

```

```

KR4:        for (jx = yi; jx <= yhigh;){
                q1 = med_data[jx][xlow-1];
                if(q1 > 0 ) {NHL = NHL +1; jx++;}
                else { jx++;}}

```

```

    /* do search right */

```

```

KP1:        if ((xhigh +1) <= LAST_COL) {goto KQ;}
            else {NHR = 0; xhigh = LAST_COL; goto KP2;}

```

```

KQ:        for (jx = yi; jx <= yhigh;){
                q2 = med_data[jx][xhigh +1];

```

```

        if (q2 > 0) {NHR = NHR + 1; jx++;
printf ("yi, yhigh, and q2 %d %d %d \n", yi, yhigh, q2);}
        else { jx++;}}

```

```

printf ("xhigh = %d \n", xhigh);
printf ("NHR = %d in search NUMBER 6 -right\n", NHR);

```

```

/* .....search Bottom ( head) ..... */
/* .....Note that the bottom is actually the top ..... */

```

```

KP2:      if (yhigh <= (LAST_ROW -1)) { goto KF2;}
          else { NHV = 0; yhigh = LAST_ROW; goto Z1;}
KF2:      if ((xhigh +1) > LAST_COL) {NHR = 0;
          xhigh = LAST_COL -1;} /* reset value */

```

```

for (jx = xlow; jx <= xhigh;) {
    q3 = med_data[yhigh+1][jx];
    if (q3 > 0) {NHV = NHV +1; jx++;}
    else {jx++;}}

```

```

if (NHV == 0) /* bif #1 */
{

```

```

    if (NHL == 0) /* bif #2 */

```

```

/* search upper right diagonal direction */

```

```

{

```

```

    /* search upper right diagonal direction */
    q3 = med_data[yhigh+1][xhigh+1];
    if(q3 == 0) {NHR =NHR;
        NHV = NHV; goto Z1;} /* path a */
    else {NHR = NHR; NHV = NHV + 1;
        goto Z1;} /* path b */
} /* bif #2 */

```

```

else /* belese #2 */
{

```

```

    if(NHR == 0) /* bif #3 */
    { /* do search upper-left corner */
        q3 = med_data[yhigh+1][xlow-1];
        if (q3 == 0 )
        {
            NHL = NHL; NHV = NHV;
            goto Z1;} /* path c */
        else {NHL = NHL +1;
            NHV = NHV +1; goto Z1;} /* path d */
        } /* bif #3 */
    else /* belese #3 */
    {

```

```

/* search both upper left and right */
/* corner --- JJ -- */

```

```

    q3 = med_data[yhigh+1][xlow-1];

```

```

        if (q3 == 0 ) /* bif #4 */
        {
            NHL = NHL;
            NHV = NHV;

            q3 = med_data[yhigh+1][xhigh+1];

            if (q3 == 0 ) {NHR = NHR;
                NHV = NHV; goto Z1;} /* path e */

            else
            { if (NHR == 0 ) {NHR = NHR +1;
                NHV = NHV +1;
                goto Z1;} /* path f */

                else {NHR = NHR;
                NHV = NHV +1; goto Z1;} /* path g */
            } /* bif #4 */

            else /* belese #4 */
            {
                if (NHL == 0 ) {NHL = NHL +1; /* path h */
                    NHV = NHV +1; goto KK2;}
                else {NHL = NHL;
                    NHV = NHV +1; goto KK2;} /* path i */
            }

KK2:
        q3 = med_data [yhigh+1][xhigh+1];
        if (q3 == 0 ) { NHR = NHR; goto Z1; } /*path l */
        else { if (NHR == 0) { NHR = NHR +1; goto Z1;}
            /* path k */
            else {NHR = NHR;
                goto Z1; }
        } /* else */

            } /* belese #4 */
        } /* belese #3 */
    } /* belese #2 */
} /* bif #1 */

else { /* belese #1 */
goto Z1;}

/* .....search bottom (tail) ..... */

        } /* else #4 */
    } /* else #3 */
} /* if #2 */

/* ***** */

else /* else # 2*/

```

```

{
    NHL = 0;
    xlow = 0;
    if (yhigh >= LAST_ROW) /* if #3 */
    {
        NHV = 0;
        yhigh = LAST_ROW;
        /* do search Right */
printf ("Number 3 in search direction\n");
printf ("NHR, NHL, NHV %d %d %d \n", NHR, NHL, NHV);

        /* Number 3 */

        /* do search right */

        if ((xhigh +1) <= LAST_COL) { goto KQ2;}
        else { NHR = 0; xhigh = LAST_COL; goto Z1;}

KQ2:
        for (jx = yi; jx <= yhigh;){
            q2 = med_data[jx][xhigh+1];
            if (q2 > 0) {NHR =NHR + 1; jx++;}
            else {jx++;}}
            goto Z1; } /* if #3 */

        else /* else #3 */
        {
            /* do search right */
            /* do search Bottom */

            /* do search right */

            /* Number 4 */

printf ("Number 4 in search direction\n");
printf ("NHR, NHL, NHV %d %d %d \n", NHR, NHL, NHV);

            if ((xhigh +1) <= LAST_COL) {goto KQ3;}
            else {NHR = 0; xhigh= LAST_COL; goto KW;}

KQ3:
            for (jx = yi; jx <= yhigh;){
                q2 = med_data[jx][xhigh+1];
                if (q2 > 0) {NHR =NHR + 1; jx++;}
                else {jx++;}}

/* .....search Bottom (head) ..... */
/* .....Note that the bottom is actually the top ... */

KW:
            if (yhigh <= LAST_ROW - 1) { goto KF3;}
            else {NHV =0; yhigh = LAST_ROW; goto Z1;}

KF3:
            if( (xhigh +1 ) > LAST_COL) { NHR=0;
                xhigh = LAST_COL -1;} /* reset value */

            for (jx = xlow; jx <= xhigh;){
                q3 = med_data[yhigh+1][jx];
                if (q3 > 0) {NHV = NHV +1; jx++;}
                else {jx++;}}

            if (NHV == 0) /* bif #1 */
            {
                if (NHL == 0) /* bif #2 */

```

```

{
    /* search upper right diagonal direction */
    q3 = med_data[yhigh+1][xhigh+1];
    if(q3 == 0) {NHR =NHR;
        NHV = NHV; goto Z1;} /* path a */
    else {NHR = NHR;
        NHV = NHV +1; goto Z1;} /* path b */
    } /* bif #2 */

    else /* belse #2 */
    {
        if(NHR == 0) /* bif #3 */
        { /* do search upper-left corner */
            q3 = med_data[yhigh+1][xlow-1];
            if (q3 == 0 )
            {
                NHL = NHL; NHV = NHV;
                goto Z1;} /* path c */
            else {NHL = NHL +1;
                NHV = NHV +1; goto Z1;} /* path d */

            } /* bif #3 */
            else /* belse #3 */
            {
                /* search both upper left and right */
                /* corner --- JJ -- */

                q3 = med_data[yhigh+1][xlow-1];

                if (q3 == 0 ) /* bif #4 */
                {
                    NHL = NHL;
                    NHV = NHV;

                    q3 = med_data[yhigh+1][xhigh+1];
                    if (q3 == 0 ) {NHV = NHR;
                        NHV = NHV; goto Z1;} /* path e */

                    else
                    { if (NHR == 0 ) {NHR = NHR +1;
                        NHV = NHV +1;
                        goto Z1;} /* path f */

                        else {NHR = NHR;
                            NHV = NHV +1; goto Z1;} /* path g */
                        }
                    } /* bif #4 */

                    else /* belse #4 */
                    {
                        if (NHL == 0 ) {NHL = NHL +1;
                            NHV = NHV + 1;
                            goto KK3;} /* path h */
                        else {NHL = NHL;
                            NHV = NHV +1; goto KK3;} /* path i */
                    }
                }
            }
        }
    }

```

```

KK3:      q3 = med_data [yhigh +1][xhigh+1];
          if (q3 == 0 ) { NHR = NHR; goto Z1; } /* pathj 1 */
          else { if (NHR == 0) { NHR = NHR +1;
                                goto Z1; } /* path k */
                else { NHR = NHR ;
                        goto Z1; }
                } /* else */

```

```

                } /* belse #4 */
                } /* belse #3 */
                } /* belse #2 */
                } /* bif #1 */

```

```

          else { /* belse #1 */
                .goto Z1; }

```

```

/* .....Search Bottom (tail) ..... */

```

```

        } /* else #3 */
    } /* else #2 */

```

```

/* Direction of expansion for rectangle is found */
/* Now, get the size of the rectangle enclosure. Repeat */
/* if necessary until all or almost the points in the blob */
/* is enclosed */

```

```

Z1: { printf ("output results: NHR=%d, NHV= %d, and NHL= %d\n",NHR,NHV, NHL);
      goto LOOPZ;}

```

```

LOOPZ: { if (NHR == 0 ) /* LOOPZ */

```

```

    {
        /* zif #1 */
        if (NHL == 0 ) /* zif #2 */
        {
            if (NHV == 0 ) /* zif #3 */
            {
                printf ("NHR NHV NHL %d %d %d in LOOPZ \n", NHR, NHV, NHL);
                goto ZY;}

```

```

                /* xnumber 1 */

```

```

            else /* zelse #3 */
            { count_vert = count_vert +1;

```

```

                printf ("count_vert = %d \n", count_vert);

```

```

                printf ("NHR NHV NHL %d %d %d in LOOPZ \n", NHR, NHV, NHL);

```

```

                    goto ZZ; /* xNumber 2 */
                } /* zelse #3 */
            } /* zif #2 */
        else /* zelse # 2 */

```



```

{
    count_left = count_left +1;
    printf ("count_left = %d \n", count_left);
    if (NHV == 0) /* zif #3 */{
printf ("NHR NHV NHL %d %d %d  in LOOPZ \n", NHR, NHV, NHL);

    printf ("count_vert = % d \n", count_vert);
        goto ZZ;}          /* xNumber 3 */
        else /* zelse #3 */
        { count_vert = count_vert +1;
printf ("NHR NHV NHL %d %d %d  in LOOPZ \n", NHR, NHV, NHL);
    printf ("count_vert = % d \n", count_vert);
        goto ZZ;          /* xNumber 4 */
    } /* zelse #2 */
    } /* zif #1 */
else /* zelse #1 */
{
    count_right = count_right + 1;
    printf ("count_right = %d \n", count_right);
    if (NHL == 0) /* zif #2 */
    {
        if (NHV == 0 ) /* zif #3 */{
printf ("NHR NHV NHL %d %d %d  in LOOPZ \n", NHR, NHV, NHL);

        goto ZZ;}          /* xNumber 5 */
        else /* zelse #3 */
        { count_vert = count_vert +1;
printf ("count_vert = %d \n", count_vert);
printf ("NHR NHV NHL %d %d %d  in LOOPZ \n", NHR, NHV, NHL);

        goto ZZ;}          /* xNumber 6 */
    } /* zif #2 */
else
{
    /* zelse #2 */
    count_left = count_left +1;
    printf ("coubt_left = %d \n", count_left);

    if (NHV == 0) { /* zif #3 */
printf ("NHR NHV NHL %d %d %d  in LOOPZ \n", NHR, NHV, NHL);

```

```

        goto ZZ;}
    else /* zelse #3 */ /* xNumber 7 */
    {
printf ("NHR NHV NHL %d %d %d in LOOPZ \n", NHR, NHV, NHL);

        count_vert = count_vert + 1;

        printf ("count_vert = %d \n", count_vert);

        goto ZZ; /* xNumber 8 */
    } /* zelse # 3 */
} /* zelse #2 */
} /* zelse #1 */
} /* LOOPZ */

} /* ZR */
/* Get the intermediate pixel rectangular size and area */

ZZ:  { LH = xi + count_right;
      LL = xi - count_left;
      Tlength = LH - LL +1;
      VL = yi;
      VH = yi + count_vert;
      Theight = VH - VL +1;
      Tarea = (LH - LL +1) * (VH - VL +1);
      xarea = Tlength * Theight;

      printf ("xl and xu %d %d \n", LL, LH);
      printf ("yl and yu %d %d \n", VL, VH);

      printf ("Length = %d \n", Tlength);
      printf ("Height = %d \n", Theight);
      printf ("Area = %d \n", Tarea);
      printf ("xarea = %d \n", xarea);

      KY:  goto ZR; } /* ZZ */

/* repeat the process and search for bigger rectangular enclosure */
/* ***** */

/*
/*
/* calculate final blob parameters */
/*
ZY:  { printf ("..... enter ZY section\n");
      LH = xi + count_right;
      LL = xi - count_left;
      Tlength = LH -LL +1;
      VL = yi;
      VH = yi + count_vert;
      Theight = VH - VL +1;
      Tarea = Tlength * Theight;
      Tarea = Tarea;
      area[iblob] = (LH - LL +1) * (VH - VL +1);
      length[iblob] = LH - LL +1;

```

```

    height[iblob] = VH - VL + 1;
/* store the coordinates of rectangle */

    xl[iblob] = LL;
    xu[iblob] = LH;
    yl[iblob] = VL;
    yu[iblob] = VH;

    printf ("Tarea in ZY section \n");

    printf ("xl, xu %d %d\n", xl[iblob], xu[iblob]);
    printf ("yl, yu %d %d \n", yl[iblob], yu[iblob]);

/* calculate approximate centroid and information about the blob */

    ncount = 0;
    sum_mem = 0;
    nsum = 0;
    jx = 0;
    ix = 0;

    printf ("LL, LH, VL, and VH = %d %d %d %d \n", LL, LH, VL, VH);
    for (jx = LL; jx <= LH; jx++) /* column */
    {
        for (ix = VL; ix <= VH;) /* row */
        {
            px = med_data[ix][jx];

            if (px > 0 ) {

                ncount = ncount + 1;
                nsum = nsum + px;
                mf = color_mem[px];
                sum_mem = sum_mem + mf; ix++;}

            else {
                ncount = ncount;
                nsum = nsum;
                sum_mem = sum_mem; ix++;}
        }
    }

    printf ("nsum in ZY section\n");
    area[iblob] = xarea;
    actual_area[iblob] = ncount;
    volume[iblob] = sum_mem;

    printf ("rectangular area = %d \n", area[iblob]);
    printf ("actual area = %d \n", ncount);
    printf ("volume = %f\n", volume[iblob]);

/*
/*
/*
    calculate approximate centroid */
    xmid = (LH - LL)/2 + 1 + xl[iblob];
    ymid = (VH - VL)/2 + 1 + yl[iblob];
    x_centroid [iblob] = xmid;
    y_centroid[iblob] = ymid;

    printf (" A blob is found --xmid and ymid = %d %d \n", xmid, ymid);

```

```

    } /* ZY */

/*    A blob is found    */
/*    return to the earliest loop (LOOP) and search for next blob */
/*    set values    */
yi = VL;
i = yi;
xi = LH + 1;
j = xi;

goto LOOP;

} /* super super else */
ZPRINT: printf ("final rectangular enclosure \n");
        printf ("\n");

        for (ix = yl[iblob] ; ix <= yu[iblob]; ix++)
            { for (jx = xl[iblob]; jx <= xu[iblob]; jx++)
                printf ("%d ", med_data[ix][jx]);
                printf ("\n"); /* new line */
            } /* for ix */

        printf ("_____ \n");
        printf ("blob# x-bar y-bar a-area r-area volume \n\n");
        printf ("_____ \n");

        for (ix = 1; ix <= iblob; ix++)
        {
            ;

printf (" %d %d %d %d %d %f \n", ix,
        x_centroid[ix], y_centroid[ix], actual_area[ix], area[ix],
        volume[ix]);

printf ("_____ \n\n");
        }

printf ("_____ \n");

printf ("\n\n");

printf ("blob# corner_1 corner_2 corner_3 corner_4 \n\n");
printf ("_____ \n");

```

```

for (ix =1; ix <= iblob; ix++)
{
printf ("   %d           (%d ,%d )           (%d ,%d )           (%d ,%d )           (%d ,

```

```

printf ("_____ \n\n")
}

```

```

printf ("\n\n");

```

```

printf ("stop\n");
} /* main */

```

```

void myprint_out (image)
int image [8][8];
{
    int row, col;

    printf ("\n print output \n");
    for (row = 0; row <= LAST_ROW; row++)
    {
        for (col = 0; col <= LAST_COL; col++)
            printf ("%d \t ", image[row][col]);
        printf ("\n"); /* new line */
    } /* for row */
} /* myprint_out */

```

```

;

```