# COSC 602: Team Project 1. Binary Image Analysis, Feature Calculation and Object Classification Due 11/05/2018

**Austin Dase and Justine Caylor**

**Part A.**

The code with the functions to calculate the testimages mark8, chain8, area, diameter, perimeter, and circularity is submitted separately and can be seen in the final code described in Part C. Below is the results that are printed in a logfile.

Object 1

Area: 661

Diameter: 29.017895

Perimeter: 101.280000

Circularity: 0.809364

Object 2

Area: 419

Diameter: 23.103202

Perimeter: 77.050000

Circularity: 0.886458

Object 3

Area: 252

Diameter: 17.917006

Perimeter: 57.640000

Circularity: 0.952669

Object 4

Area: 110

Diameter: 11.837541

Perimeter: 37.050000

Circularity: 1.006482

Object 5

Area: 39

Diameter: 7.048513

Perimeter: 21.050000

Circularity: 1.105478

Object 6

Area: 46

Diameter: 7.654980

Perimeter: 23.640000

Circularity: 1.033838

Object 7

Area: 29

Diameter: 6.078049

Perimeter: 17.050000

Circularity: 1.252965

Object 8

Area: 4

Diameter: 2.257331

Perimeter: 5.410000

Circularity: 1.716545

Object 9

Area: 13

Diameter: 4.069461

Perimeter: 24.000000

Circularity: 0.283472

Object 10

Area: 15

Diameter: 4.371302

Perimeter: 28.000000

Circularity: 0.240306

Object 11

Area: 5

Diameter: 2.523772

Perimeter: 8.000000

Circularity: 0.981250

Object 12

Area: 6

Diameter: 2.764654

Perimeter: 6.000000

Circularity: 2.093333

Object 13

Area: 62

Diameter: 8.887119

Perimeter: 92.230000

Circularity: 0.091545

Object 14

Area: 63

Diameter: 8.958503

Perimeter: 43.740000

Circularity: 0.413592

Object 15

Area: 56

Diameter: 8.446158

Perimeter: 27.280000

Circularity: 0.945124

Object 16

Area: 70

Diameter: 9.443091

Perimeter: 29.280000

Circularity: 1.025523

Object 17

Area: 133

Diameter: 13.016403

Perimeter: 44.560000

Circularity: 0.841300

Object 18

Area: 13

Diameter: 4.069461

Perimeter: 10.230000

Circularity: 1.560205

Object 19

Area: 17

Diameter: 4.653606

Perimeter: 18.460000

Circularity: 0.626578

Object 20

Area: 27

Diameter: 5.864717

Perimeter: 18.460000

Circularity: 0.995153

Object 21

Area: 80

Diameter: 10.095089

Perimeter: 35.150000

Circularity: 0.813259

Object 22

Area: 16

Diameter: 4.514661

Perimeter: 13.640000

Circularity: 1.080142

Object 23

Area: 53

Diameter: 8.216807

Perimeter: 37.050000

Circularity: 0.484941

Object 24

Area: 34

Diameter: 6.581193

Perimeter: 20.690000

Circularity: 0.997580

Object 25

Area: 159

Diameter: 14.231928

Perimeter: 64.330000

Circularity: 0.482569

Object 26

Area: 95

Diameter: 11.000869

Perimeter: 55.610000

Circularity: 0.385840

Object 27

Area: 345

Diameter: 20.964027

Perimeter: 71.250000

Circularity: 0.853570

Object 28

Area: 73

Diameter: 9.643321

Perimeter: 39.380000

Circularity: 0.591236

Object 29

Area: 91

Diameter: 10.766781

Perimeter: 43.150000

Circularity: 0.613860

Object 30

Area: 124

Diameter: 12.568285

Perimeter: 51.150000

Circularity: 0.595278

Object 31

Area: 146

Diameter: 13.637715

Perimeter: 49.970000

Circularity: 0.734385

Object 32

Area: 224

Diameter: 16.892315

Perimeter: 71.380000

Circularity: 0.552185

Object 33

Area: 154

Diameter: 14.006368

Perimeter: 47.380000

Circularity: 0.861629

Object 34

Area: 128

Diameter: 12.769390

Perimeter: 50.920000

Circularity: 0.620044

Object 35

Area: 739

Diameter: 30.682263

Perimeter: 113.810000

Circularity: 0.716594

Object 36

Area: 116

Diameter: 12.156097

Perimeter: 48.660000

Circularity: 0.615323

Object 37

Area: 185

Diameter: 15.351508

Perimeter: 59.380000

Circularity: 0.658993

Object 38

Area: 380

Diameter: 22.001737

Perimeter: 82.300000

Circularity: 0.704650

Object 39

Area: 211

Diameter: 16.394810

Perimeter: 65.020000

Circularity: 0.626871

Object 40

Area: 481

Diameter: 24.753562

Perimeter: 85.710000

Circularity: 0.822378

Object 41

Area: 566

Diameter: 26.851799

Perimeter: 102.760000

Circularity: 0.673221

Object 42

Area: 354

Diameter: 21.235711

Perimeter: 110.170000

Circularity: 0.366325

Object 43

Area: 1332

Diameter: 41.192418

Perimeter: 213.980000

Circularity: 0.365382

Object 44

Area: 933

Diameter: 34.475136

Perimeter: 133.810000

Circularity: 0.654477

Object 45

Area: 578

Diameter: 27.134954

Perimeter: 100.990000

Circularity: 0.711805

Object 46

Area: 504

Diameter: 25.338473

Perimeter: 84.530000

Circularity: 0.885928

Object 47

Area: 819

Diameter: 32.300342

Perimeter: 161.190000

Circularity: 0.395911

**Part B**

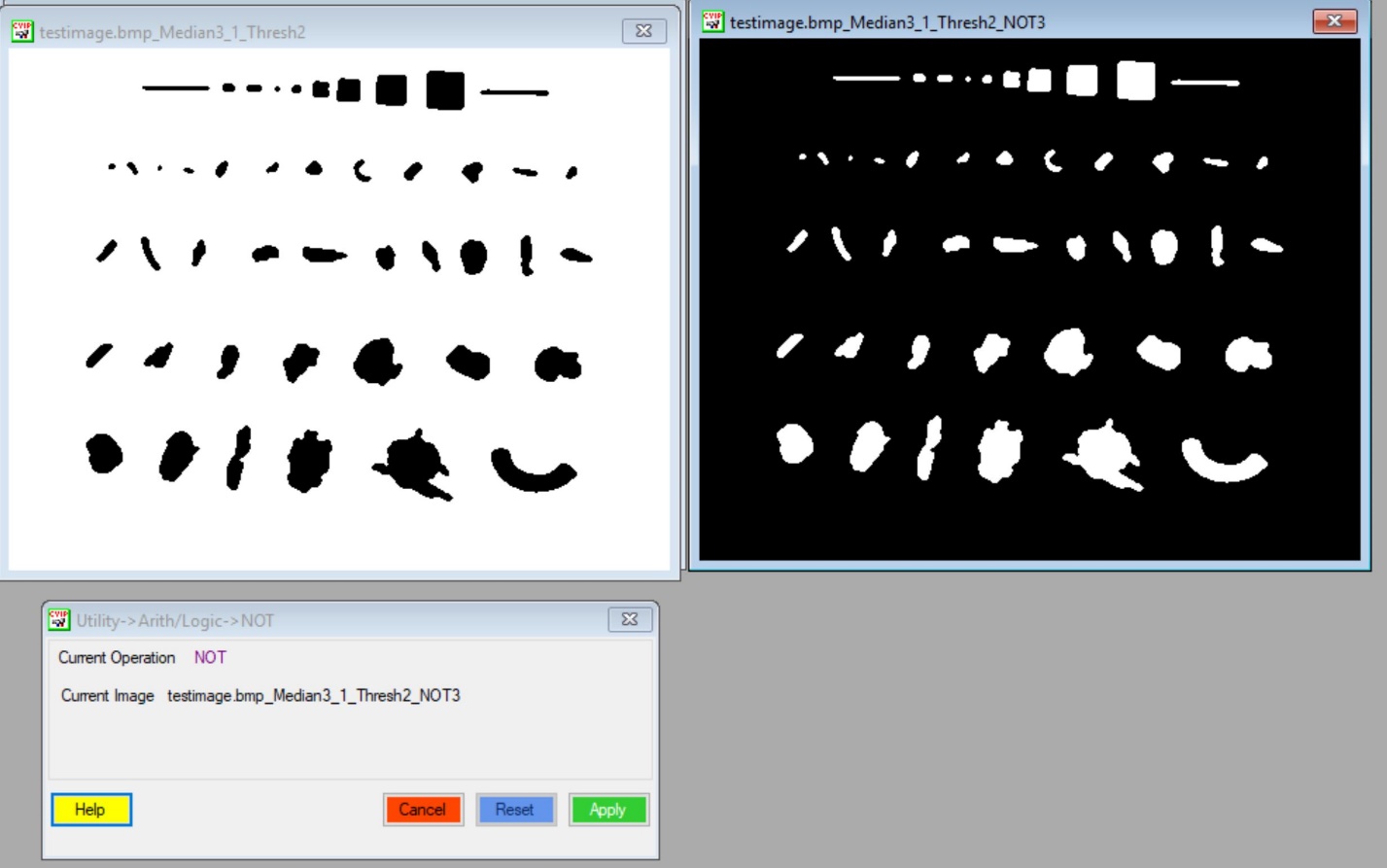
To start Part B, we open CVIPtools and then open the image to use for classification (testimage.bmp). Because the image has some noise in the background, it is important to filter it before classification to ensure that accurate calculations and classifications are made. In the CVIPtools menu, click on “Utilities”, and within that menu click on “Filter”. The filtering method used is the Median Filter.



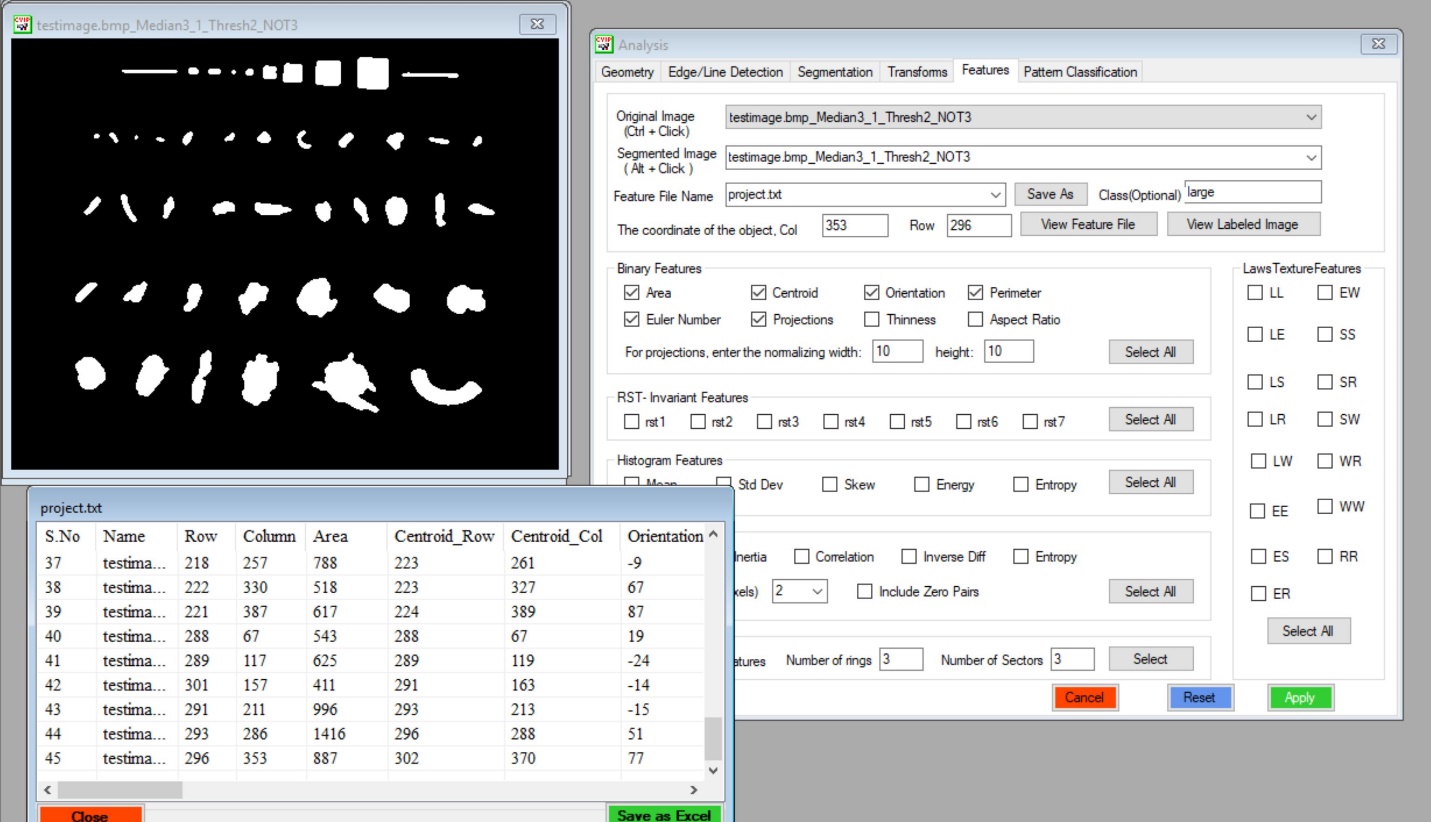
After the image is filtered, the next step is to perform a binary threshold on the filtered image. In the CVIPtools menu, click on “Utilities”, and within that menu click on “Convert”. The Binary Threshold was performed using a threshold value of 175.



Once the image is now a binary image, the next step is to get the inverse of new image. This is to give each of the objects a value of ‘1’ and so that accurate calculations can be made about each individual object. In the CVIPtools menu, click on “Utilities”, and within that menu click on “Arith/Logic”. To get the inverse of the image, the ‘NOT’ operation is selected.



In the main CVIPtools window, Analysis menu is selected, and then the option Features is selected. This will determine and calculate all of the binary features of each object in the binary image using the coordinates of the object. The features determined and calculated is area, centroid, orientation (axis of least second moment), perimeter, Euler number, and projections. The objects are classified by three types in relation to the objects size: small, medium, or large.



The separate excel sheet containing the results will be attached.

**Part C**

The objects that were identified in Part A were classified into three different types of objects. The different object classes are *Small*, *Medium*, or *Large*. The classification is based on the size (area) of the object: objects with an area less than 150 is classified as Small, objects with an area between greater than or equal to 150 and less than 500 is classified as Medium, and objects with an area greater than or equal to 500 is classified as Large. Below is the final source code and the logfile results after the classification. The source code and logfile are also attached separately.

Image \*teamproject1(Image \*inputImage){

byte \*\*image; /\* 2-d matrix data pointer \*/

FILE \*logFile; /\* Log file \*/

int r, /\* row index \*/

c, /\* column index \*/

bands, /\* band index \*/

no\_of\_rows, /\* number of rows in image \*/

no\_of\_cols, /\* number of columns in image \*/

no\_of\_bands, /\* number of image bands \*/

nextr, /\* next row index \*/

nextc; /\* next column index \*/

int threshval = 128;

int frequency[256] = {0};

int temp;

int minVal;

int outMin = 1000;

int outMax = 0;

int count = 0;

int sum = 0;

int area = 0;

int tableKey[3000] = {0};

int one,two,three,four;

int neighbors[4];

int object\_count = 0;

int object = 0;

int true\_object\_count = 0;

int compass[8][2] = {{0,1},{-1,1},{-1,0},{-1,-1},{0,-1},{1,-1},{1,0},{1,1}}; // Values to add to r,c when searcing neighbors for the chain8 code

int d = 0;

double even = 1.0; // Value to add to p for even movement

double odd = 1.41; // Value to add to p for odd movement

double diameter = 0.0;

double circularity = 0.0;

double p = 0; // Perimeter

int start[2]; // Hold the starting r,c for the object

int areas[1000] = {0}; // Hold the areas of all the objecs at the index of the object #

double perims[1000] = {0}; // Hold the perimeters of all the objecs at the index of the object #

double diameters[1000] = {0}; // Hold the diameters of all the objecs at the index of the object #

double circularities[1000] = {0}; // Hold the circularities of all the objecs at the index of the object #

int mean1 = 0;

int mean2 = 0;

int average;

int thresh2;

int diff;

int limit = 5;

int val;

//char classifier1[10]="Small";

//string classifier1 = "Small";

//string classifier2 = "Medium";

//string classifier3 = "Large";

//char classifier2[10]="Medium";

//char classifier3[10]="Large";

//string classified[100]={""};

char \*classified[1000] = {""};

char \*classes[3] = {"Small","Medium","Large"};

odd = sqrt(2);

logFile = fopen("objectClassifiaction.txt","w");

//Gets the number of image bands (planes)

no\_of\_bands = getNoOfBands\_Image(inputImage);

//Gets the number of rows in the input image

no\_of\_rows = getNoOfRows\_Image(inputImage);

//Gets the number of columns in the input image

no\_of\_cols = getNoOfCols\_Image(inputImage);

/\* Step 1 - Automatic Threshold the image \*/

/\* Calculate the average pixel value \*/

for(bands=0; bands < no\_of\_bands; bands++) {

image = (byte \*\*)getData\_Image(inputImage, bands);

for(r=0; r < no\_of\_rows; r++) {

for(c=0; c < no\_of\_cols; c++) {

sum = sum + image[r][c];

count = count + 1;

}

}

}

average = sum / count;

sum = 0; // Clear out the sum

threshval = average; // Book suggested that the starting threshold is the average pixel value

/\* Iterate through the automatic thresholding algorithm until limit is reached \*/

printf("\nUsing a limit of %d ",limit);

while(1 == 1){

printf("\nCurrent threshold is: %d",threshval);

mean1 = 0;

mean2 = 0;

frequency[BLACK\_LAB] = 0;

frequency[WHITE\_LAB] = 0;

for(bands=0; bands < no\_of\_bands; bands++) {

image = (byte \*\*)getData\_Image(inputImage, bands);

for(r=0; r < no\_of\_rows; r++) {

for(c=0; c < no\_of\_cols; c++) {

if(image[r][c] > (byte) threshval){

//image[r][c] = BLACK\_LAB;

frequency[BLACK\_LAB] = frequency[BLACK\_LAB] + 1; // Count

mean1 = mean1 + image[r][c]; // Sum

}

else {

//image[r][c] = WHITE\_LAB;

frequency[WHITE\_LAB] = frequency[WHITE\_LAB] + 1; // Count

mean2 = mean2 + image[r][c]; // Sum

}

}

}

}

//printf("\nsum1: %d sum2: %d mean1: %d mean2: %d ",frequency[BLACK\_LAB],frequency[WHITE\_LAB] , mean1, mean2);

/\* Mean of each side \*/

mean1 = mean1 / frequency[BLACK\_LAB] ;

mean2 = mean2 / frequency[WHITE\_LAB] ;

thresh2 = (mean1 + mean2) / 2;

diff = abs(thresh2 - threshval);

threshval = thresh2;

//printf("\nmean1: %d mean2: %d thresh2: %d diff: %d ", mean1, mean2, thresh2, diff);

if(diff < limit){

break;

}

}

/\* Actually Apply the Threshold \*/

printf("\nUsing threshold value of: %d",threshval);

for(bands=0; bands < no\_of\_bands; bands++) {

image = (byte \*\*)getData\_Image(inputImage, bands);

for(r=0; r < no\_of\_rows; r++) {

for(c=0; c < no\_of\_cols; c++) {

if(image[r][c] > (byte) threshval){

image[r][c] = BLACK\_LAB;

}

else {

image[r][c] = WHITE\_LAB;

}

}

}

}

// Show the binary image

view\_Image(inputImage,"binaryImage");

/\* End of Automatic Threshold \*/

/\* Step 2 - Set edges to 0 \*/

for(bands=0; bands < no\_of\_bands; bands++) {

image = (byte \*\*)getData\_Image(inputImage, bands);

for(r=0; r < no\_of\_rows; r++) {

if(r == 0 || r == no\_of\_rows -1){

for(c=0;c < no\_of\_cols;c++){

image[r][c] = BLACK\_LAB;

}

}else{

image[r][0] = BLACK\_LAB;

image[r][no\_of\_cols - 1] = BLACK\_LAB;

}

}

}

/\* Step 3 - First Pass mark8 \*/

for(bands=0; bands < no\_of\_bands; bands++) {

image = (byte \*\*)getData\_Image(inputImage, bands);

for(r=1; r < no\_of\_rows - 1; r++) {

for(c=1; c < no\_of\_cols - 1; c++) {

if(image[r][c] == WHITE\_LAB){

// Add neighbors to neighbors array

one = image[r-1][c+1];

two = image[r-1][c];

three = image[r-1][c-1];

four = image[r][c+-1];

neighbors[0] = one;

neighbors[1] = two;

neighbors[2] = three;

neighbors[3] = four;

// If all neighbors are background its a new object

if(one == two && two == three && three == four && four == BLACK\_LAB){

object\_count = object\_count + 1; // Start at 1 because background is zero

image[r][c] = object\_count;

}else{

//Otherwise find the min of the neighbors that arn't zero

minVal = 1000;

for(count = 0; count < 4; count++){

if(neighbors[count] > 0 && neighbors[count] < minVal){

minVal = neighbors[count];

}

}

// Now we have the min - assign that value to the image

image[r][c] = minVal;

// Now we have to check if neighbors have more than one value i.e. a conflict

//if all surrounding are from the same object , its part of that one

if(one == two && two == three && three == four && four == minVal){

continue;

}else{

//There is a conflict - record the value and then record the conflict

for(count = 0; count < 4; count++){

if(neighbors[count] == 0 || neighbors[count] == minVal){

// Dont worry if it's a zero value or the same as the min

continue;

}else{

// If there isnt already a value for that key set ti

if(tableKey[neighbors[count]] == 0){

tableKey[neighbors[count]] = minVal;

}else{

// Otherwise set the value to the min of the minVal and the existing key

temp = tableKey[neighbors[count]];

tableKey[neighbors[count]] = min(temp,minVal);

}

//printf("\ntableKey %d is %d", neighbors[count], tableKey[neighbors[count]]);

}

}

}

}

}

}

}

}

/\* Step 4 - Second Pass mark8 \*/

for(temp = 3000; temp >=0; temp--){

if(tableKey[temp] > 0){

//printf("\n%d , %d", temp, tableKey[temp]);

for(bands=0; bands < no\_of\_bands; bands++) {

image = (byte \*\*)getData\_Image(inputImage, bands);

for(r=1; r < no\_of\_rows -1; r++) {

for(c=1; c < no\_of\_cols -1; c++) {

if(image[r][c] == temp ){

image[r][c] = tableKey[temp];

}

}

}

}

}

}

/\* Step 5 - Get new object count \*/

for(bands=0; bands < no\_of\_bands; bands++) {

image = (byte \*\*)getData\_Image(inputImage, bands);

for(r=1; r < no\_of\_rows -1 ; r++) {

for(c=1; c < no\_of\_cols -1 ; c++) {

if(image[r][c] > outMax){

outMax = image[r][c];

}

if(image[r][c] < outMin){

outMin = image[r][c];

}

}

}

}

//printf("\noutMax: %d outMin: %d",outMax,outMin);

//printf("\nOriginal Object Count: %d", object\_count);

//printf("\nSecond Pass Object Count: %d", outMax);

printf("\n");

object\_count = outMax;

/\* Step 5.5 - Get areas & diameters \*/

for(bands=0; bands < no\_of\_bands; bands++) {

image = (byte \*\*)getData\_Image(inputImage, bands);

for(object = 1; object < object\_count + 1; object++){

for(r=1; r < no\_of\_rows -1 ; r++) {

for(c=1; c < no\_of\_cols -1 ; c++) {

if(image[r][c] == object){

sum = sum + object;

}

}

}

area = sum / object;

diameter = sqrt(area/PIE) \* 2;

if(area > 0){

//printf("\nArea for object %d is %d",object, area);

//printf("\nDiameter for object %d is %f",object, diameter);

}

// Add the area to the areas array for use calculating the circularity

areas[object] = area;

diameters[object] = diameter;

sum = 0;

//classify based on the area

if(area < 150){

classified[object]=classes[0];

printf("\nClassification for object %d is %s",object, classes[0]);

//printf("\nClassification for object %d is SMALL",object);

}

else if(area>=150 && area < 500){

classified[object]=classes[1];

printf("\nClassification for object %d is %s",object, classes[1]);

//printf("\nClassification for object %d is MEDIUM",object);

}

else{

classified[object]=classes[2];

printf("\nClassification for object %d is %s",object, classes[2]);

//printf("\nClassification for object %d is LARGE",object);

}

}

}

/\* End of mark8 \*/

/\* Start of chain8 \*/

/\* Step 6 - chain8 \*/

for(bands=0; bands < no\_of\_bands; bands++) {

image = (byte \*\*)getData\_Image(inputImage, bands);

//image\_out = (byte \*\*)getData\_Image(outputImage, bands);

for(object = 1; object < object\_count + 1; object++){

for(r=0;r < no\_of\_rows;r++){

for(c=0;c < no\_of\_cols;c++){

//If its equal to the object number its part of an object - start the chain

if(image[r][c] == object){

//printf("\nStart of object %d ",object);

true\_object\_count = true\_object\_count + 1;

start[0] = r;

start[1] = c; // Store the starting pixel location in start array

//image[r][c] = WHITE\_LAB; // Optional - turn the perimeter white

d = (d + 5) % 8; // Next direction to look

do{

nextr = r + compass[d][0];

nextc = c + compass[d][1]; // Possible next pixle coordinates

//Increase the count

count = count + 1;

// Now make sure next r and next c are inbounds

if(nextr >= 0 && nextc >= 0 && nextr < no\_of\_rows && nextc < no\_of\_cols){

//Check to see if image[nextr][nextc] is still part of an object

if(image[nextr][nextc] == object || image[nextr][nextc] == WHITE\_LAB){

// If so - move there and increase p

r = nextr;

c = nextc;

//image[r][c] = WHITE\_LAB; // Optional - turn the perimeter white

// Reset the neighbor conut

count = 0;

// Check if d is even or odd to increase p by the right amount

if((d % 2) == 0){

//Even

p = p + even;

}else{

// Odd

p = p + odd;

}

d = (d + 5) % 8; // Next direction to look

}else{

// pixel at neighbor is not part of the object - increase d by 1 and try again

d = (d + 1) % 8;

}

}else{

// next r or next c is out of bounds - increase d by one and try again

d = (d + 1) % 8;

}

// If we've checked 8 neighbors and nothing it must be a one pixel object

if(count > 8){

printf("\n\nBREAKING! %d\n", object);

break;

}

// Check to see if we're at the starting pixel - if so break the loop

}while(nextr != start[0] || nextc != start[1]);

// Finished with that object - print its p

//printf("\n\tObject %d perimeter is: %f",object,p);

circularity = ( ((4\*PIE) \* areas[object]) / (p \* p) );

//circularity = 2 \* sqrt(areas[object]) / (p \* p);

//printf("\n\tObject %d circularity is: %f",object,circularity);

circularities[object] = circularity;

perims[object] = p;

// Reset d and p

d = 0;

p = 0;

// No need to scan the rest

c = no\_of\_cols;

r = no\_of\_rows;

}

}

}

}

}

printf("\n\nTrue Object Count: %d", true\_object\_count);

count = 0;

for(bands=0; bands < no\_of\_bands; bands++) {

image = (byte \*\*)getData\_Image(inputImage, bands);

for(object = 1; object < object\_count + 1; object++){

if(areas[object] > 0){

count = count + 1;

fprintf(logFile,"\nObject %d \n\tArea: %d\n\tDiameter: %f\n\tPerimeter: %f\n\tCircularity: %f\n\tClassification: %s\n",count,areas[object],diameters[object],perims[object],circularities[object],classified[object]);

}

}

}

fclose(logFile);

count = 0;

/\* Invert the object pixel values so they are easy to see \*/

for(bands=0; bands < no\_of\_bands; bands++) {

image = (byte \*\*)getData\_Image(inputImage, bands);

for(r=0;r < no\_of\_rows;r++){

for(c=0;c<no\_of\_cols;c++){

if(image[r][c] > 0){

image[r][c] = 255 - image[r][c];

}

}

}

}

return inputImage;

}

Object Classification:

Object 1

Area: 661

Diameter: 29.017895

Perimeter: 101.313708

Circularity: 0.808825

Classification: Large

Object 2

Area: 419

Diameter: 23.103202

Perimeter: 77.071068

Circularity: 0.885974

Classification: Medium

Object 3

Area: 252

Diameter: 17.917006

Perimeter: 57.656854

Circularity: 0.952113

Classification: Medium

Object 4

Area: 110

Diameter: 11.837541

Perimeter: 37.071068

Circularity: 1.005338

Classification: Small

Object 5

Area: 39

Diameter: 7.048513

Perimeter: 21.071068

Circularity: 1.103268

Classification: Small

Object 6

Area: 46

Diameter: 7.654980

Perimeter: 23.656854

Circularity: 1.032366

Classification: Small

Object 7

Area: 29

Diameter: 6.078049

Perimeter: 17.071068

Circularity: 1.249874

Classification: Small

Object 8

Area: 4

Diameter: 2.257331

Perimeter: 5.414214

Circularity: 1.713874

Classification: Small

Object 9

Area: 13

Diameter: 4.069461

Perimeter: 24.000000

Circularity: 0.283472

Classification: Small

Object 10

Area: 15

Diameter: 4.371302

Perimeter: 28.000000

Circularity: 0.240306

Classification: Small

Object 11

Area: 5

Diameter: 2.523772

Perimeter: 8.000000

Circularity: 0.981250

Classification: Small

Object 12

Area: 6

Diameter: 2.764654

Perimeter: 6.000000

Circularity: 2.093333

Classification: Small

Object 13

Area: 62

Diameter: 8.887119

Perimeter: 92.242641

Circularity: 0.091520

Classification: Small

Object 14

Area: 63

Diameter: 8.958503

Perimeter: 43.798990

Circularity: 0.412479

Classification: Small

Object 15

Area: 56

Diameter: 8.446158

Perimeter: 27.313708

Circularity: 0.942793

Classification: Small

Object 16

Area: 70

Diameter: 9.443091

Perimeter: 29.313708

Circularity: 1.023166

Classification: Small

Object 17

Area: 133

Diameter: 13.016403

Perimeter: 44.627417

Circularity: 0.838760

Classification: Small

Object 18

Area: 13

Diameter: 4.069461

Perimeter: 10.242641

Circularity: 1.556357

Classification: Small

Object 19

Area: 17

Diameter: 4.653606

Perimeter: 18.485281

Circularity: 0.624865

Classification: Small

Object 20

Area: 27

Diameter: 5.864717

Perimeter: 18.485281

Circularity: 0.992433

Classification: Small

Object 21

Area: 80

Diameter: 10.095089

Perimeter: 35.213203

Circularity: 0.810342

Classification: Small

Object 22

Area: 16

Diameter: 4.514661

Perimeter: 13.656854

Circularity: 1.077478

Classification: Small

Object 23

Area: 53

Diameter: 8.216807

Perimeter: 37.071068

Circularity: 0.484390

Classification: Small

Object 24

Area: 34

Diameter: 6.581193

Perimeter: 20.727922

Circularity: 0.993933

Classification: Small

Object 25

Area: 159

Diameter: 14.231928

Perimeter: 64.384776

Circularity: 0.481749

Classification: Medium

Object 26

Area: 95

Diameter: 11.000869

Perimeter: 55.698485

Circularity: 0.384615

Classification: Small

Object 27

Area: 345

Diameter: 20.964027

Perimeter: 71.355339

Circularity: 0.851051

Classification: Medium

Object 28

Area: 73

Diameter: 9.643321

Perimeter: 39.455844

Circularity: 0.588965

Classification: Small

Object 29

Area: 91

Diameter: 10.766781

Perimeter: 43.213203

Circularity: 0.612066

Classification: Small

Object 30

Area: 124

Diameter: 12.568285

Perimeter: 51.213203

Circularity: 0.593810

Classification: Small

Object 31

Area: 146

Diameter: 13.637715

Perimeter: 50.041631

Circularity: 0.732284

Classification: Small

Object 32

Area: 224

Diameter: 16.892315

Perimeter: 71.455844

Circularity: 0.551013

Classification: Medium

Object 33

Area: 154

Diameter: 14.006368

Perimeter: 47.455844

Circularity: 0.858877

Classification: Medium

Object 34

Area: 128

Diameter: 12.769390

Perimeter: 50.970563

Circularity: 0.618815

Classification: Small

Object 35

Area: 739

Diameter: 30.682263

Perimeter: 113.982756

Circularity: 0.714424

Classification: Large

Object 36

Area: 116

Diameter: 12.156097

Perimeter: 48.769553

Circularity: 0.612562

Classification: Small

Object 37

Area: 185

Diameter: 15.351508

Perimeter: 59.455844

Circularity: 0.657313

Classification: Medium

Object 38

Area: 380

Diameter: 22.001737

Perimeter: 82.426407

Circularity: 0.702491

Classification: Medium

Object 39

Area: 211

Diameter: 16.394810

Perimeter: 65.112698

Circularity: 0.625087

Classification: Medium

Object 40

Area: 481

Diameter: 24.753562

Perimeter: 85.840620

Circularity: 0.819878

Classification: Medium

Object 41

Area: 566

Diameter: 26.851799

Perimeter: 102.911688

Circularity: 0.671238

Classification: Large

Object 42

Area: 354

Diameter: 21.235711

Perimeter: 110.325902

Circularity: 0.365290

Classification: Medium

Object 43

Area: 1332

Diameter: 41.192418

Perimeter: 214.308658

Circularity: 0.364262

Classification: Large

Object 44

Area: 933

Diameter: 34.475136

Perimeter: 133.982756

Circularity: 0.652790

Classification: Large

Object 45

Area: 578

Diameter: 27.134954

Perimeter: 101.154329

Circularity: 0.709494

Classification: Large

Object 46

Area: 504

Diameter: 25.338473

Perimeter: 84.669048

Circularity: 0.883021

Classification: Large

Object 47

Area: 819

Diameter: 32.300342

Perimeter: 161.438600

Circularity: 0.394692

Classification: Large