University of the West of England
BRISTOL

UFMFC9-15-3 – MACHINE VISION

Assignment

Author: Túlio Dapper e Silva

Student Number: 14004454

Bristol, 11 December 2014

Contents

1. Grey Level Image Display and Thresholding

1.1. Objective

Firstly, the purpose of this tutorial is getting started with the LabWindows CVI development application. For that, basic commands such as loading and displaying images are required.

After that, the objective becomes more applied to image processing. The exercise requires to display a binary image modified based on a threshold value. That value, specified by the user and situated between in an integer range from 0 to 255, can highlight features in pictures or merely simplify them. Some pictures were supplied in order to try that technique. The pictures given are shown below.

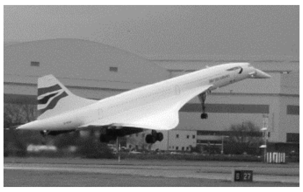


Figure 1.1

Concord.bmp

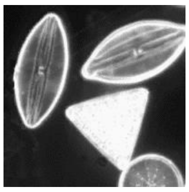


Figure 1.2

Diatoms.bmp

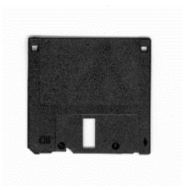


Figure 1.3

Disk1.bmp

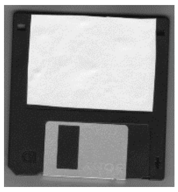


Figure 1.4

Disk2.bmp

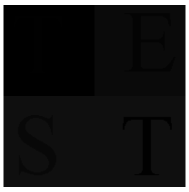


Figure 1.5

Test.bmp

The task basically is to highlight features of each picture simply using the threshold method. The extraction of desired features in some pictures are easier than others. In this work, the difficulties and some possible solutions for the problem will be discussed.

1.2. Methods

The analysis of a binary images is much simpler than using a grey-scale image (Awcock & Thomas, 1995). In order to have a binary image, the image need to be pre-processed. In this process, grey-level pixels within a certain interval will be considered white and the grey-level pixels outside that interval will be considered black.

A program was written in order to perform that task. Firstly, there is only one threshold value. If the grey-level value is less than that threshold value, then it becomes 0, otherwise, it becomes 255. The program shown in Figure 1.6 open an image file and display the resultant image. However, this program was not practical because the users need to locate the file every time that they set a new threshold value. Thus, the Figure 1.7 is a second version that makes possible to load the image and easily try different threshold values.

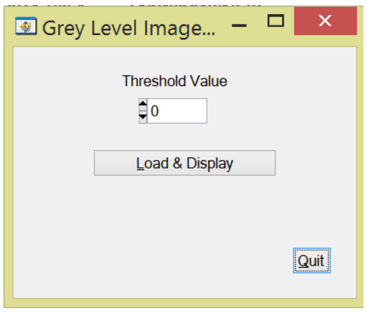


Figure 1.6

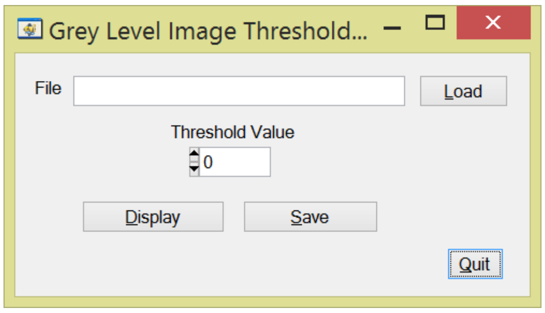


Figure 1.7

The command button labelled “Load” is responsible for filling the string control (PANEL_STRING) with the file directory of the image. The call-back function of “Load” is shown below. The function *FileSelectPopup* requires a directory file from the user.

```
GetProjectDir(dirname);
status = FileSelectPopup("","","","Select Image",VAL_OK_BUTTON,0,1,1,0,pathname);
if(status==0){break;}
SetCtrlVal(panelHandle, PANEL_STRING, pathname);
```

The command button labelled *“Display”* will load the picture from the string control (PANEL_STRING) and generate a binary thresholded image using the threshold value given by the user in the numeric control (PANEL_NUMERIC). The values of both controls are collect using the function *GetCtrlVal*. The image is converted in array using the function *imaqImageToArray*. Then, a for-loop goes through every element of the array and change it to 255 if the value is greater than the threshold value or change it to 0 otherwise. The call-back function is shown below.

```
myImage1 = imaqCreateImage(IMAQ_IMAGE_U8,0);
GetCtrlVal(panelHandle, PANEL_STRING, pathname);
imaqReadFile(myImage1,pathname, NULL, NULL);
imageArray = imaqImageToArray (myImage1, IMAQ_NO_RECT, &cols, &rows);
GetCtrlVal(panelHandle, PANEL_NUMERIC, &thresh);
for (rr=0; rr<(rows*cols); rr+=1) {
        if (imageArray[rr] > thresh) {imageArray[rr] = 255;}
        if (imageArray[rr] < thresh) {imageArray[rr] = 0;}
}
imaqArrayToImage(myImage1,imageArray, cols, rows);
imaqDisplayImage(myImage1,0,1);
imaqDispose(imageArray);
imaqDispose(myImage1);
```

The command button labelled *“Save”* saves the picture in a fixed directory. The IMAQ function *imaqWriteBMPFile* converts an image array to a BMP file.

Using that program to isolate features of the given pictures, there was one of the pictures that the program was not performing properly. The purpose was to isolate only the metal shutter of the picture depicted in Figure 1.4. The isolation was impossible to be performed due to the medium grey-level of this part. Selecting a threshold value greater or less than the medium grey-level, it will not isolate only the metal shutter.

Therefore, a new program was developed. This program can highlight pixels which the value is greater than a minimum threshold value and less than a maximum value. The function that display the image was changed.

The function *ProcessImage* depends on parameters to generate the image. The parameter *DisplayOrSave* indicates if the function should display the image (when equal to 0) or save the image on the hard disk (when equal to 1). The parameter *SingleOrMinMax* indicates whether the function should use the standard method (when equal to 0) or use the minimum and maximum threshold values (when equal to 0).

```
void ProcessImage(int DisplayorSave, int SingleOrMinMax)
// ################
// -> DisplayorSave:
// 0 - Display
// 1 - Save
// ################
// -> SingleOrMinMax
// 0 - Threshold is a single value
// 1 - There are minimum and maximum threshold values
// ################
{
        char pathname[MAX_PATHNAME_LEN];
        unsigned char *imageArray;
        int rows, cols, rr;
        int thresh = 0;
        int threshMin = 0;
        int threshMax = 0;

        myImage1 = imaqCreateImage(IMAQ_IMAGE_U8,0);

        GetCtrlVal(panelHandle, PANEL_FILENAME, pathname);
        GetCtrlVal(panelHandle, PANEL_THRESHOLD, &thresh);
        GetCtrlVal(panelHandle, PANEL_THRESHOLD_MIN, &threshMin);
        GetCtrlVal(panelHandle, PANEL_THRESHOLD_MAX, &threshMax);

        imaqReadFile(myImage1,pathname, NULL, NULL);
        imageArray = imaqImageToArray (myImage1, IMAQ_NO_RECT, &cols, &rows);

        switch (SingleOrMinMax)
        {
                case 0:

                        for (rr=0; rr<(rows*cols); rr+=1)
                        {
                                if (imageArray[rr] > thresh) {imageArray[rr] = 255;}
                                if (imageArray[rr] < thresh) {imageArray[rr] = 0;}
```

```
                    }
                    break;

          case 1:

                    for (rr=0; rr<(rows*cols); rr+=1)
                    {
                              if ((imageArray[rr] > threshMin) && (imageArray[rr] < threshMax))
{imageArray[rr] = 255;}
                              else { imageArray[rr]=0; }
                    }
                    break;
     }

     imaqArrayToImage(myImage1,imageArray, cols, rows);

     switch (DisplayorSave)
     {
          case 0: imaqDisplayImage(myImage1,0,1);
                    break;
          case 1: imaqWriteBMPFile(myImage1, "C:\\IMAGEM.bmp", FALSE, NULL);
                    break;
     }

     imaqDispose(imageArray);

     imaqDispose(myImage1);
}
```

This modified program is shown in the Figure 1.8 below.

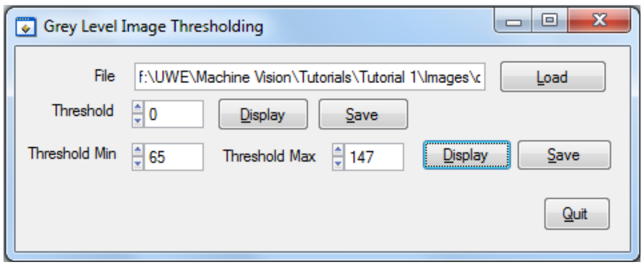

Grey Level Image Thresholding
File
f:\UWE\Machine Vision\Tutorials\Tutorial 1\Images\c
Load
Threshold
0
Display
Save
Threshold Min
65
Threshold Max
147
Display
Save
Quit

Figure 1.8

1.3. Results and Conclusion

The pictures shown in the Figure 1.1, Figure 1.2, Figure 1.3, Figure 1.4 and Figure 1.5 were uploaded and different threshold values were applied in these pictures. The objective was to highlight the main features of each figure.

The first picture opened is shown in Figure 1.1. Different values was applied in this picture in order to analyse different results. The Figure 1.9 shows the result when the threshold value is equal to

195. In this figure, the plane is isolated from the rest of the scenario. In this way, it is possible to analyse the position of the airplane in the air. There are two white points on the right-bottom of the image that might be eliminated using another technique. In addition, some parts of the airplane was removed due to their low value on the grey-scale.

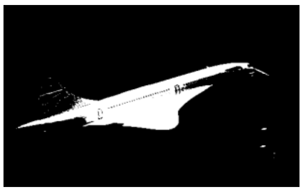


Figure 1.9

The next picture opened is shown in Figure 1.2. Firstly, the purpose is to highlight the area of every object. However, the task is not straightforward. The best threshold value found was 110. The resultant picture is shown in Figure 1.10. In this picture, we see the borders completely defined, however the area of some objects is not full filled. After that, the main idea was to highlight only the borders of every object. The Figure 1.11 and Figure 1.12 are the resultant attempts. The Figure 1.11 shows the whole border of three out four objects. The Figure 1.12 is another attempt trying to remove this remaining points. However, the border of the object in the bottom of the page does not appear clearly.

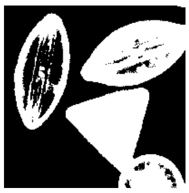


Figure 1.10

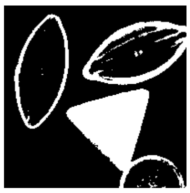


Figure 1.11

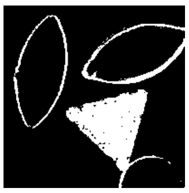


Figure 1.12

The Figure 1.3 was loaded in the program. The picture contains a disk that is highlighted easily. That because there is a huge gap between the grey level value of the disk and the background. The Figure 1.13 shows the disk isolated from the background. In this case, the threshold value applied in the image is equal to 102.

Figure 1.13

The Figure 1.4 shows a different disk. For that picture, the exercise requires to isolate different parts of the object. Each item below is related to a different part of the object.

a) The paper label only

In this case, the task was not complicated. The grey level value of the paper label is notably close to 255. The Figure 1.14 shows the resultant image when the threshold value is equal to 210.

At the bottom of the picture, a line remains from the original image. This part is related to a white level part of the metal shutter. This line should be removed using another image processing.

Figure 1.14

b) The paper label and the metal shutter together

The isolation of these parts can be performed because they are whiter than the rest of the image. If the threshold value is slightly less than the grey-level of the metal shutter, the result will show the desired parts. A threshold value which provides a proper result is equal to 147. The Figure 1.15 shows the resultant image.

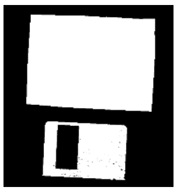


Figure 1.15

c) The metal shutter only

The isolation of the metal shutter is not possible only using a single threshold value. That is because the metal shutter is darker than the paper label. There is no threshold value that isolates only the metal shutter. For that reason, another program was necessary to be developed (Figure 1.8). In this new application, the figure was loaded and two threshold values were applied. One value is the low-limit and other value is the high-limit. A proper result is given when the minimum value is equal to 146 and the maximum value is equal to 176. The Figure 1.16 shows the resultant picture. There is a line remaining from the paper label, which should be removed using other methods.

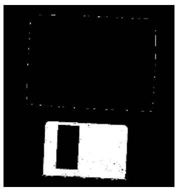


Figure 1.16

d) The plastic body only

This task also requires the program shown in Figure 1.8Figure 1.7. The background has a grey-level value in the middle of the range 0-255. In the same image, there are parts darker than the background (the plastic body) and there are parts whiter than the background (the paper label and the metal shutter). The Figure 1.17 shows the resultant image when the minimum value is equal to 65 and the maximum value is equal to 147. The picture contains some white lines that should be removed applying other techniques.

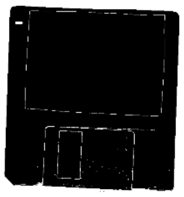


Figure 1.17

The last picture analysed is shown in the Figure 1.5. This picture, at first glance, seems to be just a dark picture. However, applying different threshold values in that image, characters became to be visible. Using the standard program shown by the Figure 1.7, which applies a single threshold value, some results were collected. The Figure 1.18 shows the resultant image when the threshold value is equal to 0; and Figure 1.19 when the threshold value is equal to 9. From the Figure 1.18, the grey-level value of the background of T is equal to 0. From the Figure 1.19, all the characters have grey-level value less than 8 and the background of S, E and T are greater than 9. Therefore, if the second program developed is used for the Figure 1.5, it is possible to isolate each character from its background. The Figure 1.20 shows the resultant image when the minimum threshold value is equal to 0 and the maximum threshold value is equal to 9.

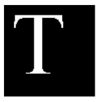


Figure 1.18

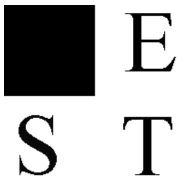


Figure 1.19

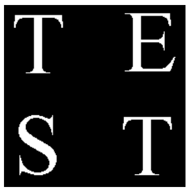


Figure 1.20

2. Histogram Display

2.1. Objective

The objective of this tutorial is to generate a frequency histogram as a graph of different grey scale images. The mean and median grey level values will be calculated and displayed to the user. These results are important to process and select features of a picture. In this tutorial, the following images will be analysed and the results will be discussed.

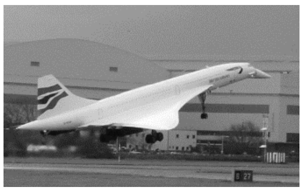


Figure 2.1

Concord.bmp

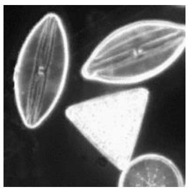


Figure 2.2

Diatoms.bmp

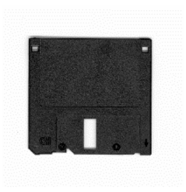


Figure 2.3

Disk1.bmp

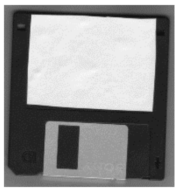


Figure 2.4

Disk2.bmp

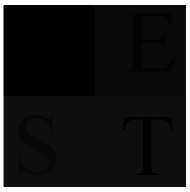


Figure 2.5

Test.bmp

2.2. Methods

A program was developed in order to load an image, display it, display its frequency histogram, and calculate its mean and median grey level. The Figure 2.6 depicted below is the print screen of the program.

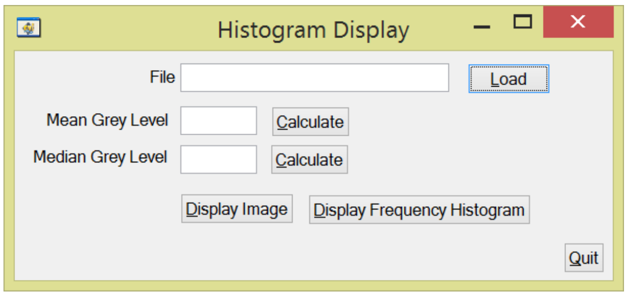


Figure 2.6

In this program, a single function is responsible to manage the main tasks. At the beginning of this function, the picture is loaded and the image converted to grey-level values into an array. Depending on the input parameter, the function will execute a specific task. The possible input values are shown on the following table.

Table 2.1

inOperation	Task
1	Display Image
2	Display Frequency Histogram
3	Calculate Mean Grey Level
4	Calculate Median Grey Level

When *inOperation* is equal to 1, the user desires to show the original image. Hence, that procedure is basically executed by the function “imaqDisplayImage”. The code is shown below.

```
case 1: // Display Image
                imaqDisplayImage(myImage1,0,1);
                break;
```

When *inOperation* is equal to 2, the user desired to display the frequency histogram of the original image. In order to perform the desired task, the function stores the amount of pixels for each grey-level value into an array (histo). After, the function plots a Y Graph using the function *YGraphPopup*. The code is shown below.

```
case 2: // Display Frequency Histogram
                // Processing Histogram Values
                for (i=0; i<(rows*cols); i++) {
                        k = imageArray[i];
                        histo[k] = histo[k] + 1;
                }
                // Ploting Graph
YGraphPopup("Frequency Histogram", &histo, 256, VAL_UNSIGNED_INTEGER);
                break;
```

When inOperation is equal to 3, the user desires to calculate and display the mean grey level value. For that, there is a for-loop that adds every grey level value of the image and the resulting value is divided by the total number of pixels. The resulting number is displayed in a string control.

```
case 3: // Calculate Mean Grey Level
                // Summing all grey level values
                for (i=0; i<(rows*cols); i++) {
                        sum = sum + imageArray[i];
                }
                // Mean Grey Level = (Sum of grey values) / (Number of pixels)
                // Convert Float to String
                snprintf(output,50,"%f",(float) sum/(rows*cols));
                // Set textbox strMeanGreyLevel with Mean Grey Level (output)
                SetCtrlVal(panelHandle, PANEL_strMEANGREYLEVEL, output);
                break;
```

When *inOperation* is equal to 4, the user desires to calculate and display the median grey level value. Firstly, the program needs organizing every grey level value into an array in a crescent order. After that, the desired value is located in the middle position of the array. The code is shown below.

```
case 4: // Calculate Median Grey Level
// sum = (Number of pixels)
```

```
sum = (rows*cols);
                    // Processing Histogram Values
                    for (i=0; i<(sum); i++) {
                              k = imageArray[i];
                              histo[k] = histo[k] + 1;
                    }
l = 0;
                    // Arranging the values in a crescent order
                    // Loop through the histogram diagram
                    for (i=0; i<256; i++) {
                    // Placing the value 'i' for 'histo[i]' times
                              for (k=0; k<histo[i]; k++) {
                                        imageArray[l] = i;
                                        l++;
                              }
                    }
/* If the Number of Pixels is even, so the Median value is equal to (imageArray[(sum/2) - 1] +
imageArray[(sum/2)])/2" else if the Number of Pixels is odd, so the Median value is equal to
imageArray[(sum/2)
*/
if(((sum%2) == 0) && sum!=0){   // Number of pixels is Even
                              // Median Grey Level = (imageArray[(sum/2) - 1] + imageArray[(sum/2)])/2
                    // Convert Float to String
                    snprintf(output,50,"%f",(float) (imageArray[(sum/2) - 1] + imageArray[(sum/2)])/2);
                    } else {  // Number of pixels is Odd
                              // Median Grey Level = imageArray[(sum/2)
                              // Convert Float to String
                              snprintf(output,50,"%f",(float) imageArray[(sum/2)]);
}
                    // Set textbox strMedianGreyLevel with Median Grey Level (output)
                    SetCtrlVal(panelHandle, PANEL_strMEDIANGREYLEVEL, output);
                    break;
```

2.3. Results and Discussion

The images shown in Figure 2.1, Figure 2.2, Figure 2.3, Figure 2.4 and Figure 2.5 will be loaded and analysed by the program described previously. The results will be discussed in order to understand the advantages of knowing each value.

The picture depicted in the Figure 2.1 was loaded and the histogram diagram was generated. The Figure 2.7 shows the diagram. The image contains at least one pixel in each grey level value from 10 to 255. It means that is very difficult to define a threshold value to highlight desired features.

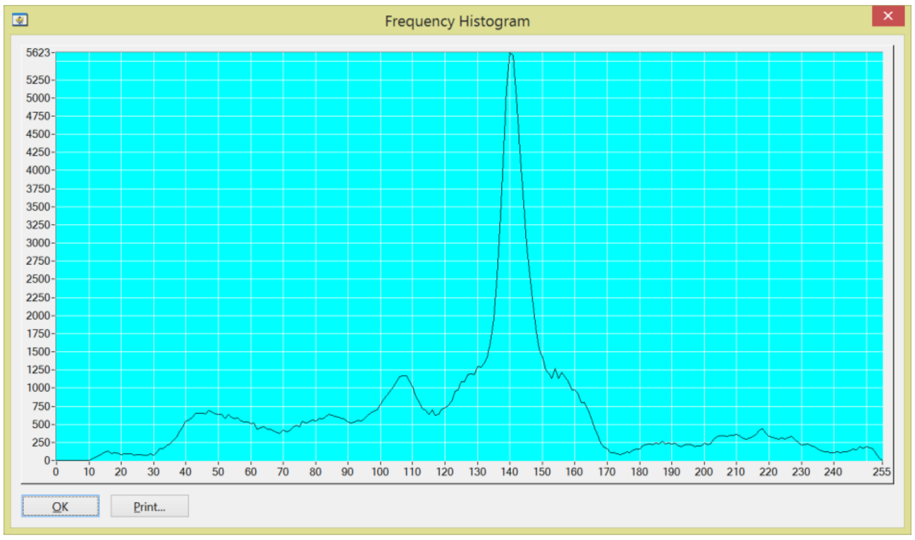
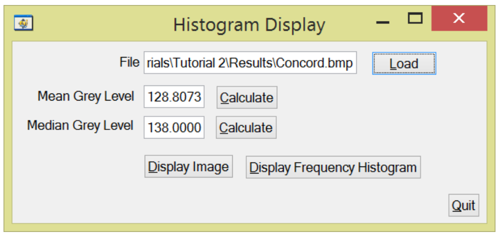


Figure 2.7

Using the program, the mean and the median values were calculated. The following figure shows the resultant values. The mean grey level value is equal to 128.8 and the median grey value is equal to 138.



File rials\Tutorial 2\Results\Concord.bmp
Mean Grey Level 128.8073
Median Grey Level 138.0000

Figure 2.8

The next picture loaded is shown in Figure 2.2. The Figure 2.9 illustrates the histogram diagram generated. The histogram indicates that the image contains all range of grey level values spaced by little gaps. The Figure 2.10 shows the picture when the values are calculated. The mean grey level value is equal to 101.69 and the median grey level value is equal to 69.

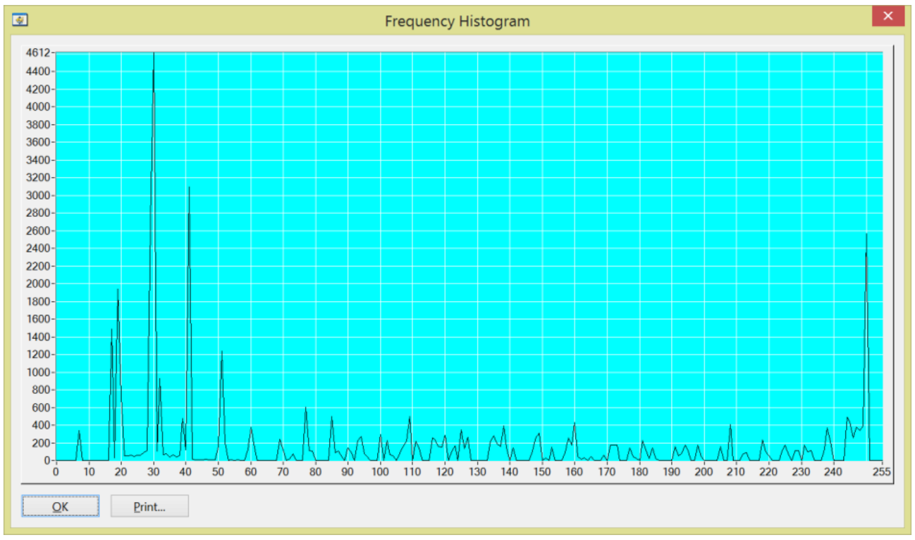
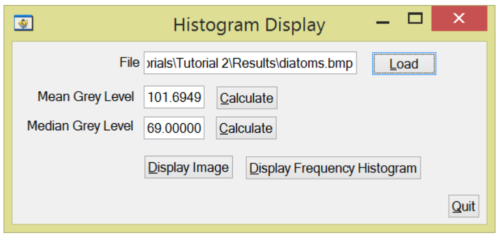


Figure 2.9



File ›rials\Tutorial 2\Results\diatoms.bmp
Mean Grey Level 101.6949
Median Grey Level 69.00000

Figure 2.10

The picture depicted in Figure 2.3 is the next image to be analysed. The histogram diagram is depicted in Figure 2.11. It is important to realise that the origin of the vertical axis is equal to 5. Hence, even it does not seem, the picture contains all grey level values from 0 to 255. However, it is notable that the disk body is mainly based in grey level value from 220 to 255. In addition, the background is mostly constituted by grey level values from 0 to 90.

The mean and median grey level values were also calculated. The Figure 2.11 shows the program screen displaying the resultant values. The mean grey level value is equal to 165.65 and the median grey level value is equal to 253.

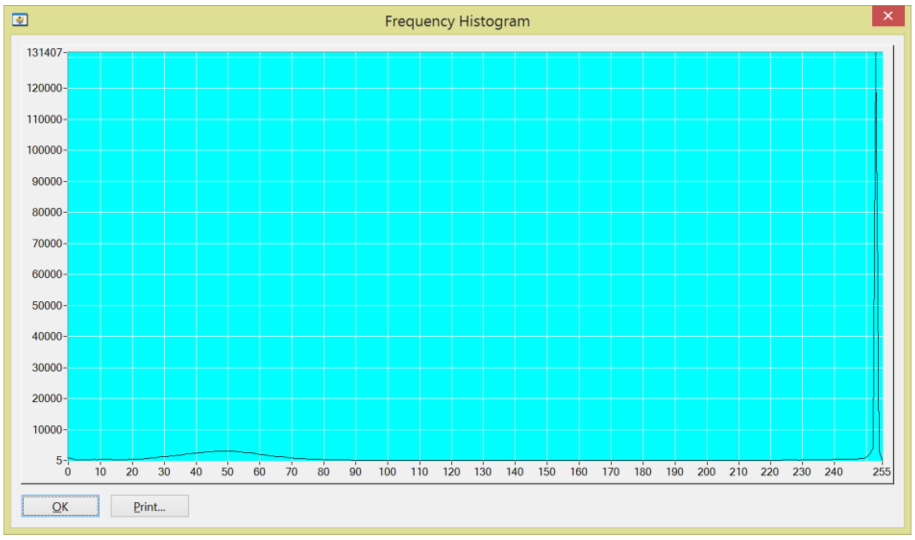
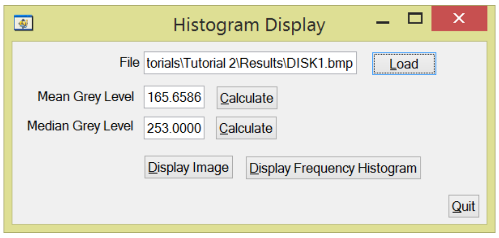


Figure 2.11



File torials\Tutorial 2\Results\DISK1.bmp Load

Mean Grey Level 165.6586 Calculate

Median Grey Level 253.0000 Calculate

Figure 2.12

The next image is depicted in Figure 2.4. In this case, the frequency diagram is shown in Figure 2.13. In order to separate each component of the disk, a proper threshold value it is easy to be determined. For example, when the range from 60 to 130 is isolated from the rest of the grey scale, the resultant image is depicted in Figure 2.14. The disk body was isolated. The Figure 2.15 shows the resultant image when highlighted from 130 to 180. As can be seen, the picture isolates the metal shutter from the disk. Finally, the following image (Figure 2.16) shows the picture when the grey level values from 200 to 255 are isolated. In this case, the white label paper stuck on the disk is notable isolated from the rest of the picture. It happens because the curve from 200 to 255 is basically the white label paper.

The mean and the median grey level values are calculated. The Figure 2.17 shows the program screen displaying the desired values. The mean grey level value is equal to 137.65 and the median grey level value is equal to 130.

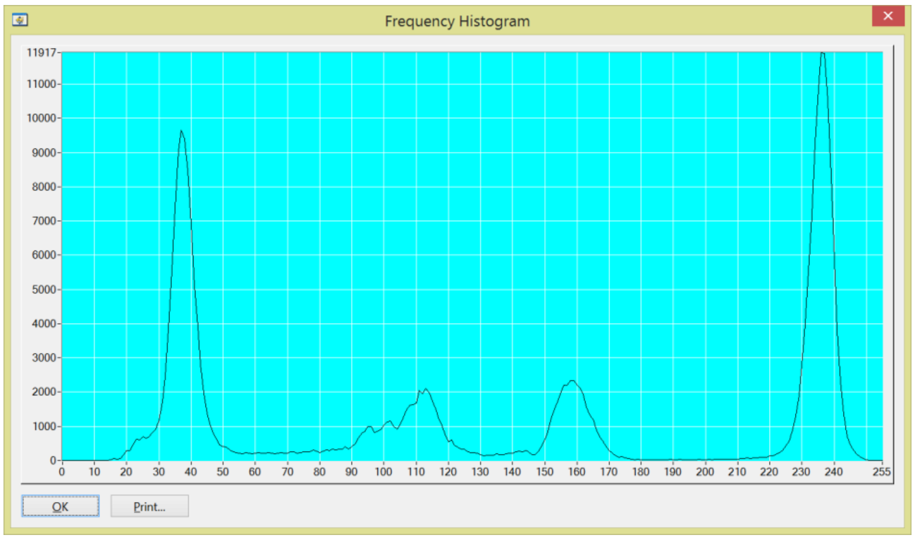


Figure 2.13

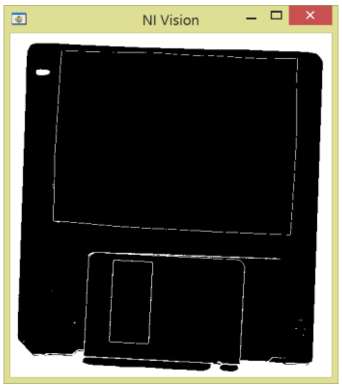


Figure 2.14

60 – 130

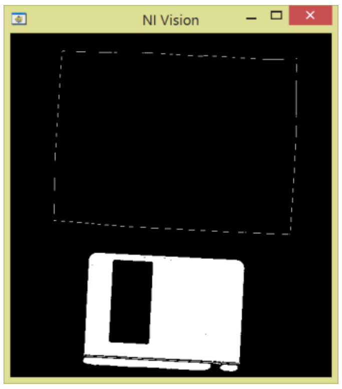


Figure 2.15

130 – 180

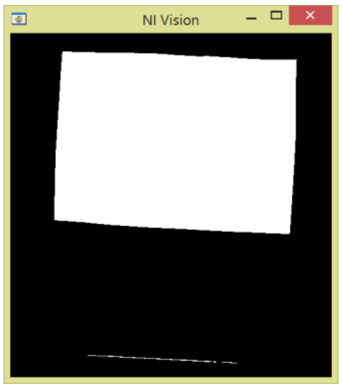
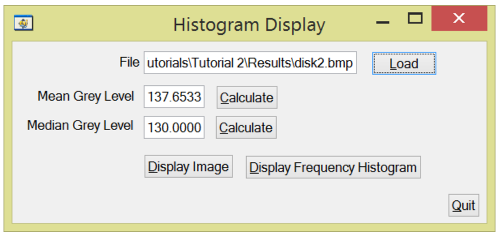


Figure 2.16

200 – 255



File utorials\Tutorial 2\Results\disk2.bmp
Mean Grey Level 137.6533
Median Grey Level 130.0000

Figure 2.17

The last picture is the image depicted in Figure 2.5. The frequency histogram is shown below (Figure 2.18). Using the histogram diagram, it becomes easier to isolate each characters. The figures from Figure 2.19 to Figure 2.23 show each peak of the graph isolated. It is possible to see that each image is one or more characters, or one part of the background highlighted.

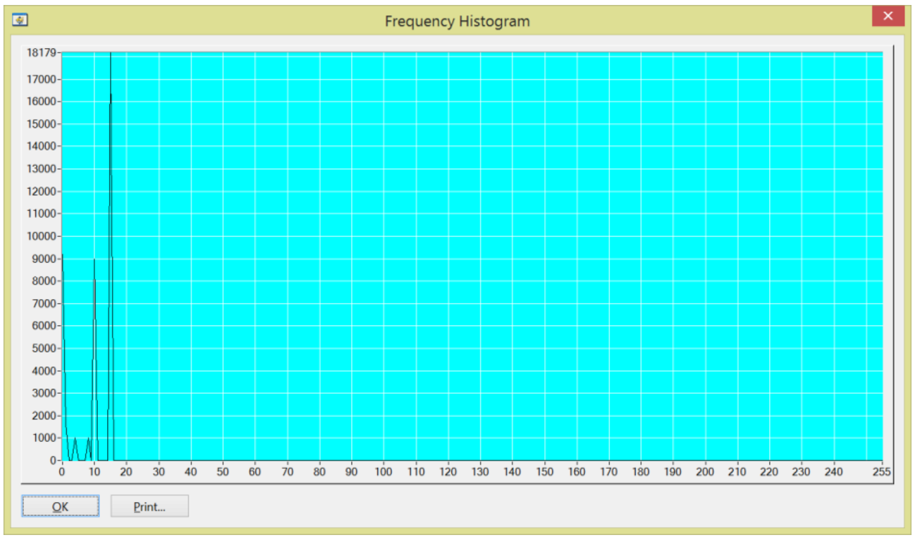


Figure 2.18

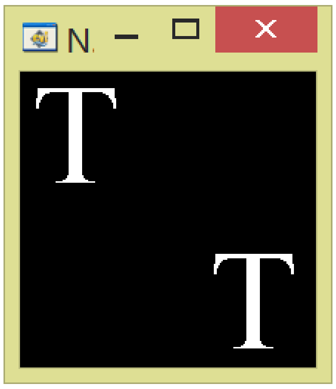


Figure 2.19

0 - 3

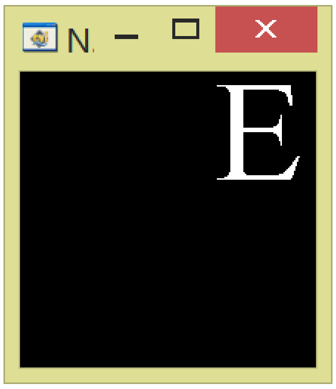


Figure 2.20

3 - 5

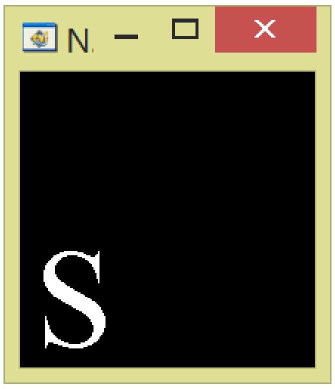


Figure 2.21

5 - 9

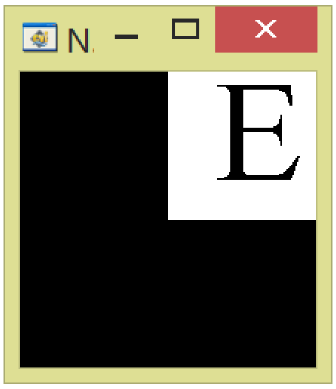


Figure 2.22

9 - 12

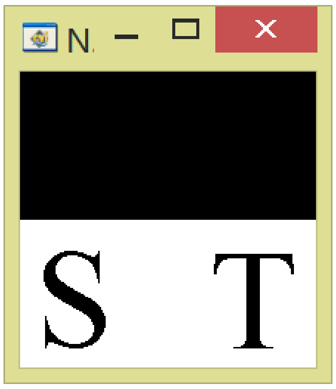
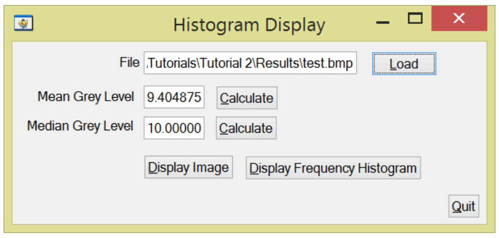


Figure 2.23

12 - 20

The mean and the median value of the original image can be calculated. The following figure (Figure 2.24) displays both values. The mean grey level value is equal to 9.40 and the median grey level value is equal to 10.


Histogram Display

File .Tutorials\Tutorial 2\Results\test.bmp

Mean Grey Level 9.404875

Median Grey Level 10.00000

Figure 2.24

Histogram distribution can be used to determine a threshold value automatically. For that, the program calculate the grey-level value that is able to separate the desirable features. In addition, pixels into a specific range can be enhanced when supressed. It can be done using histogram stretching or histogram compression.

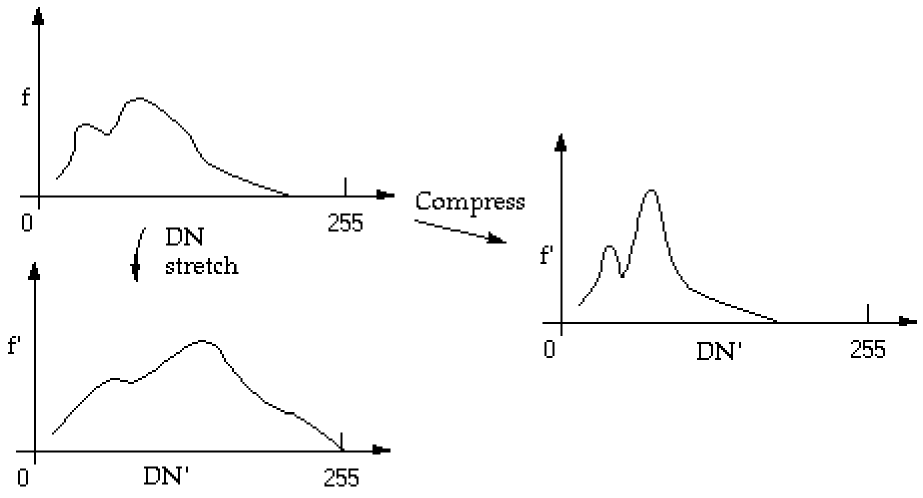


Figure 2.25: Histogram stretching and compression

3. Automatic Threshold

3.1. Objective

The objective of this work is to create a program that calculates automatically a threshold value and displays the resultant image. The mean and the median grey level values should also be calculated and displayed by the program.

3.2. Methods

The threshold value is calculated from iterative attempts to find the grey level value which divide the histogram between two main parts. For example, if the picture contains a dark object and a white background, the threshold value should be equal to the middle value between the mean grey levels of both intensities.

The Figure 3.1 and Figure 3.2 show hypothetical frequency histograms of one object in a well-defined background. In Figure 3.1, the red line indicates the mean grey level value of the data. If that value is used as threshold, the objects will not be separated in a proper way. However, the Figure 3.2 indicates a red line calculated based on the mean value of the mean values of both sides (left and right hand). In this case, it is clearly t is a proper threshold in order to separate the object and the background.

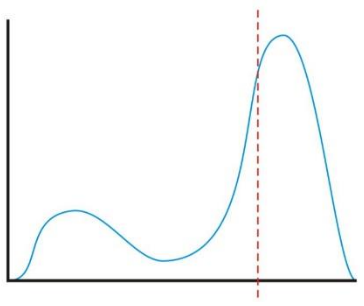


Figure 3.1

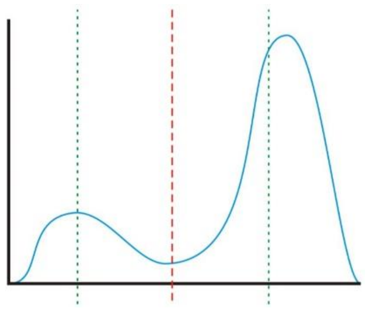


Figure 3.2

To sum up, in order to have the desired result, the object and the background should be well defined. For example, a mean grey level value might represent the object and another mean grey level value might represent the background. The Figure 3.3 and Figure 3.4 show two examples. The frequency histogram of these images are depicted, respectively, by the Figure 3.5 and the Figure 3.6.

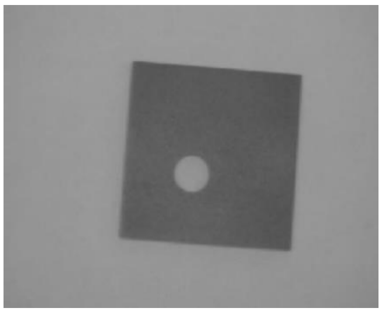


Figure 3.3

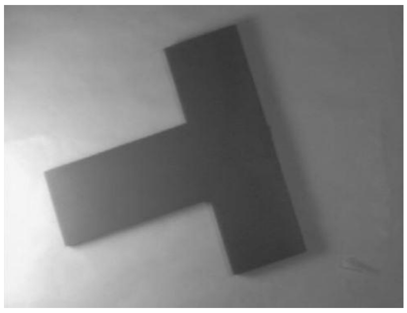


Figure 3.4

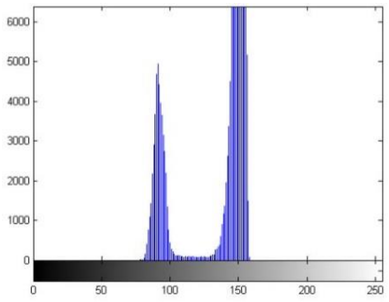


Figure 3.5

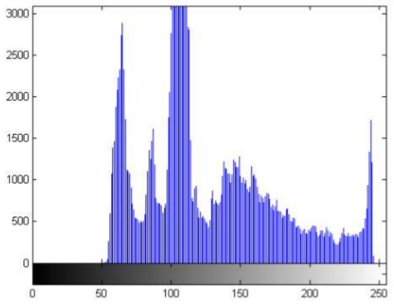


Figure 3.6

In Figure 3.3 and Figure 3.4, an ideal threshold might be found when the background and the object belongs to well distinct grey level values. A threshold value is relatively easy to be found in the image depicted in Figure 3.3, as can be seen in the frequency histogram (Figure 3.5). On the other hand, the Figure 3.4 shows an image that the object might not be easily separated from the background. In the frequency histogram (Figure 3.6), there are not two regions that could be separated.

The main purpose of this project is to create a program that executes an automatic threshold. This function will follow the steps indicated in the flow diagram below.

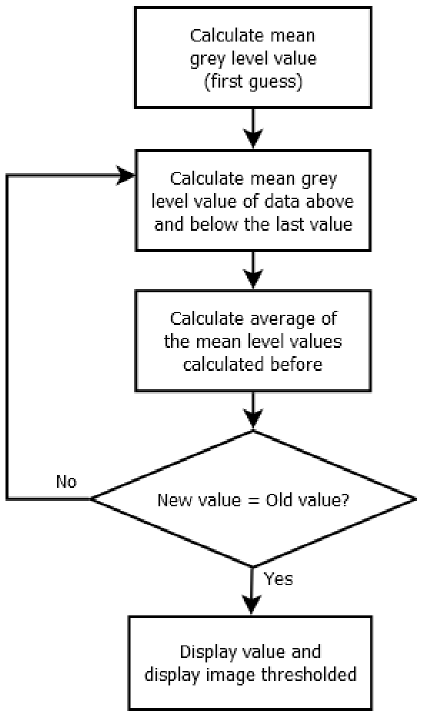

Calculate mean grey level value (first guess)
Calculate mean grey level value of data above and below the last value
Calculate average of the mean level values calculated before
New value = Old value?
No
Yes
Display value and display image thresholded

Figure 3.7

Using this main idea, the program was developed. The Figure 3.8 shows the main screen of the application.

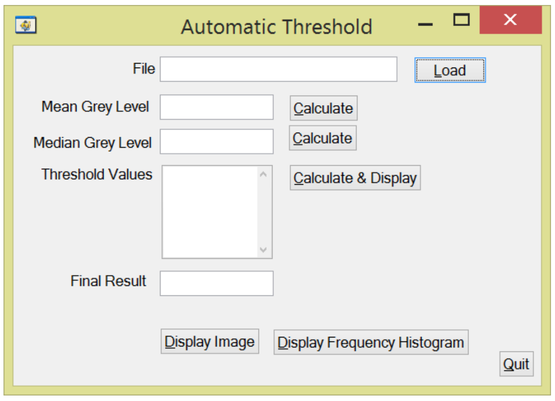


Figure 3.8

The button “Calculate & Display” automatically calculates the threshold value using the technique depicted in the flow diagram (Figure 3.7). Each threshold value is added in the list “Threshold Values” until the program finds the final value. The final value, resulted from undetermined interactions, is displayed in the textbox labelled “Final Result”. After that, the thresholded image is displayed to the user.

The following table shows the control structure of the program.

Table 3.1

Control	Constant Name	Callback Function	Label
String	strFILENAME		File

Command Button	cmdLOAD	Load	Load
String	strMEANGREYLEVEL		Mean Grey Level
Command Button	cmdCALCULATEMEAN	CalculateMean	Calculate
String	strMEDIANGREYLEVEL		Median Grey Level
Command Button	cmdCALCULATEMEDIAN	CalculateMedian	Calculate
Command Button	cmdDISPLAY	DisplayImage	Display Image
Command Button	cmdDISPLAYFREQHIST	DisplayFreqHist	Display Frequency Histogram
List Box	listTHRESHOLD		Threshold Values
String	strTHRESHOLD		Final Result
Command Button	cmdCALCULATETHRESHOLD	CalculateThreshold	Calculate & Display
Command Button	cmdQUIT	QuitProgram	

The code which calculates automatically the threshold value and display the resulted image can be seen in the following box.

```
// Clear the list which contains the calculated threshold values
ClearListCtrl(panelHandle, PANEL_listTHRESHOLD);

// Summing all grey level values
for (i=0; i<(rows*cols); i++) { sum = sum + imageArray[i]; }

// Calculate the initial mean grey level value (first guess)
mean = (float) sum / (rows*cols);

// Insert the initial mean grey level value in the list
snprintf(output,50,"%f",mean);
InsertListItem (panelHandle, PANEL_listTHRESHOLD, -1, output, -1);

// Try 200 times at maximum
result = 200;

while (result) {
        // Initialise variables
        sum_above = 0; sum_below = 0; num_above = 0.0; num_below = 0.0;

        // Loop through the imageArray
        for (i=0; i<(rows*cols); i++) {
                // If the grey level value of the pixel is above the mean grey level value, so...
                if ((float) (imageArray[i]) >= mean) {
                        sum_above = sum_above + (float)(imageArray[i]);
                        num_above++;
                } else {
                        sum_below = sum_below + (float)(imageArray[i]);
```

```
                                num_below++;
                        }
            }

            // Avoiding division by zero
            if (num_above == 0) num_above = 1;
            if (num_below == 0) num_below = 1;

            // Calculating the new mean grey level value
            new_mean = (sum_above / (float) num_above + sum_below / (float) num_below) / 2.0;

            // Finish the while if the mean grey level value is equal to the previous one.
            if (new_mean == mean) { result = 0; } else { result --; }

            // Define new mean grey level value and, consequently, the new threshold
            mean = new_mean;

            // Insert the new threshold value in the list
            snprintf(output,50,"%f",mean);
            InsertListItem (panelHandle, PANEL_listTHRESHOLD, -1, output, -1);
}

// Set textbox strTHRESHOLD with the final result
snprintf(output,50,"%f",mean);
SetCtrlVal(panelHandle, PANEL_strTHRESHOLD, output);

// Threshold the image using 'mean'
for (i=0; i<(rows*cols); i++) {
if (imageArray[i] > (int) mean) { imageArray[i] = 255; } else { imageArray[i] = 0; }
}

imaqArrayToImage (myImage1,imageArray, cols, rows); // convert array to image and display
imaqDisplayImage (myImage1, 0, 1);
```

3.3. Results and Conclusions

In this tutorial, some pictures will be loaded in order to determine automatically a threshold value. Firstly, the picture shown in Figure 3.9 is loaded. As can be seen in the original image, the airplane and the background are not well defined (both of them have pixels in similar grey level values). The frequency histogram diagram is shown in Figure 3.10. A similar analysis can be performed in the diagram; there is not a region of grey level values which well represents the airplane, neither the background. As a consequence, the program might not find a great value to separate both items.

The Figure 3.11 shows the values found by the program. The value calculated by the program does not have significant meaning. This value (115.13), looking in the diagram (Figure 3.10), will not properly separate features in the image. The resultant image, after the threshold operation, is shown in Figure 3.12.

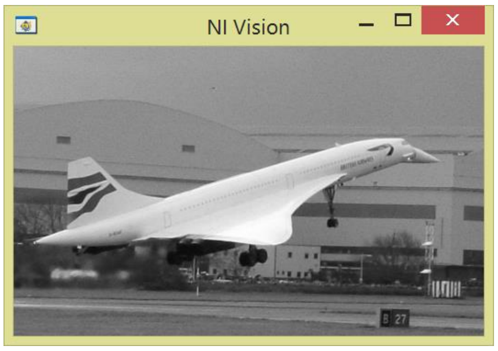


Figure 3.9

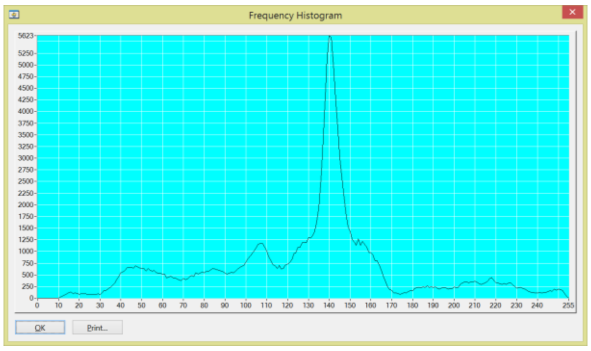
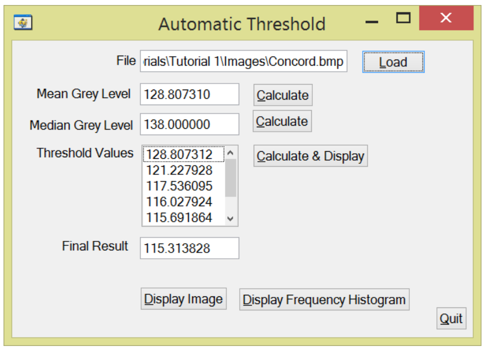


Figure 3.10



File rials\Tutorial 1\Images\Concord.bmp
Mean Grey Level 128.807310
Median Grey Level 138.000000
Threshold Values 128.807312
121.227928
117.536095
116.027924
115.691864
Final Result 115.313828

Figure 3.11

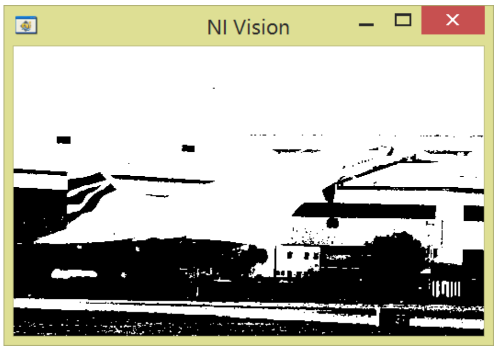


Figure 3.12

The next image for analysing is depicted in Figure 3.13. The figure contains a dark disk on a white background. Due to the contrast between them, a threshold value is not difficult to be determined. The frequency histogram is shown in Figure 3.14. The diagram shows a peak of white pixels (background) and black pixels (disk) concentrated between 20 and 90.

The program suggests that the threshold value should be equal to 149.59 (Figure 3.15). In contrast to the previous example, this value has clear significance. Analysing the frequency histogram (Figure 3.14), this value is located between the background and the disk. The resulted thresholded picture is shown in Figure 3.16. The threshold value properly separates the disk from the background.

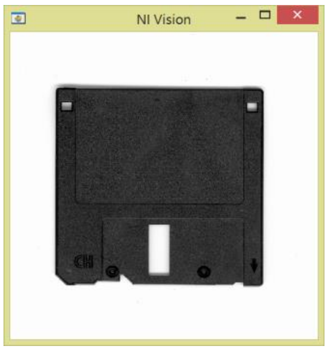


Figure 3.13

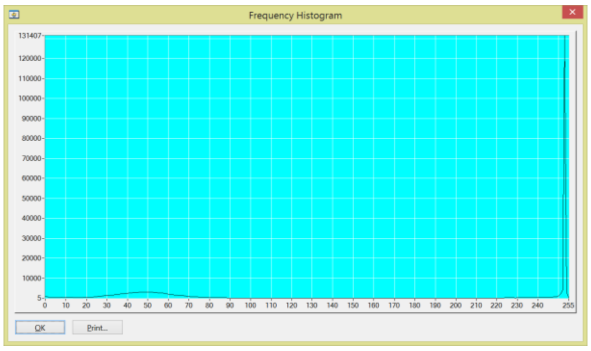
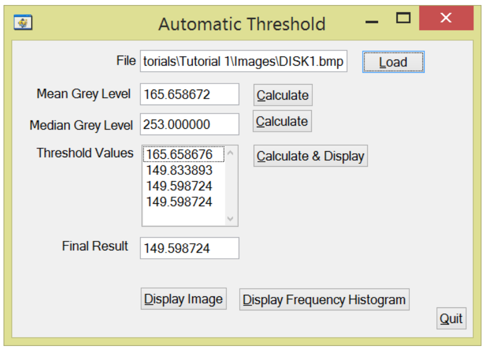


Figure 3.14



File torials\Tutorial 1\Images\DISK1.bmp
Mean Grey Level 165.658672
Median Grey Level 253.000000
Threshold Values 165.658676
149.833893
149.598724
149.598724
Final Result 149.598724

Figure 3.15

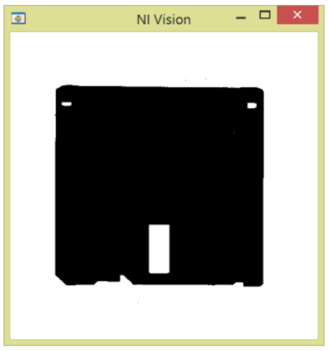


Figure 3.16

The next two examples are used to enhance the importance of the constrained environment in the acquisition of the image. For example, adjusting the lighting can facilitate the program to find a proper threshold value. The picture depicted in Figure 3.17 shows a hand on a wood table. The background is not well defined; there are also some shadows in the picture. The frequency histogram (Figure 3.18) shows three peaks of grey level values. It is not possible to clearly identify which value might separate the hand from the background. The Figure 3.19 indicates that the program suggests a threshold value equal to 98.37. The resultant image is depicted in Figure 3.20. The thresholded image is not isolating the hand from the table.

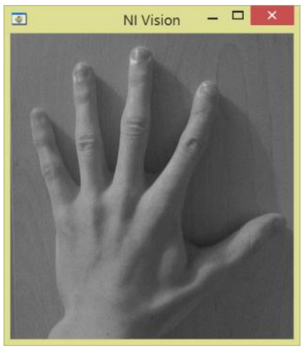


Figure 3.17

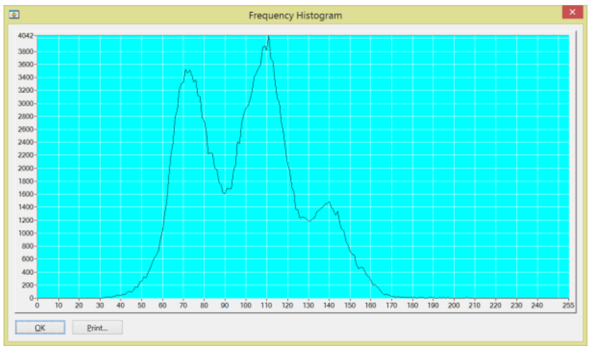
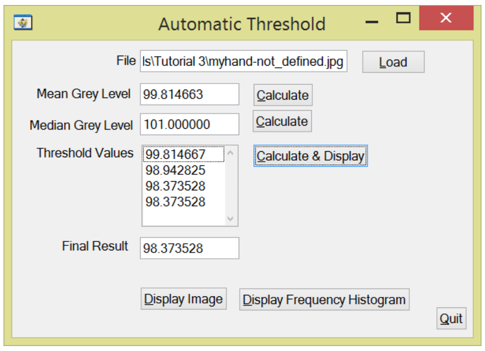


Figure 3.18



Automatic Threshold

File ls\Tutorial 3\myhand-not_defined.jpg Load

Mean Grey Level 99.814663 Calculate

Median Grey Level 101.000000 Calculate

Threshold Values 99.814667 98.942825 98.373528 98.373528 Calculate & Display

Final Result 98.373528

Display Image Display Frequency Histogram Quit

Figure 3.19

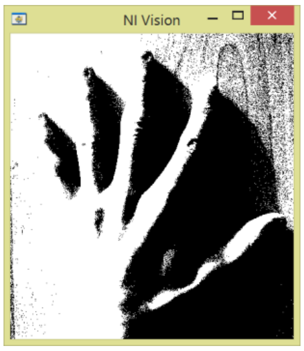


Figure 3.20

In order to have the hand clearly isolated from the table, a picture was taken in a constrained environment. The Figure 3.21 shows the original picture, where the hand is notably darker than the background. The frequency histogram clearly shows that pixels of the hand belong to a range from 20 to 140; on the other hand, pixels of the table belong to a range from 140 to 220. The program calculates the mean value of both sides and determines their average. The threshold value automatically calculated is equal to 139.5 (Figure 3.23). When this value is applied in the original image, the resultant image is shown in Figure 3.24. The hand is notably isolated from the background.

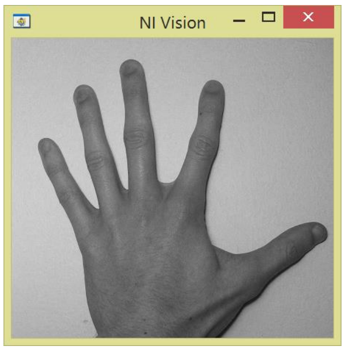


Figure 3.21

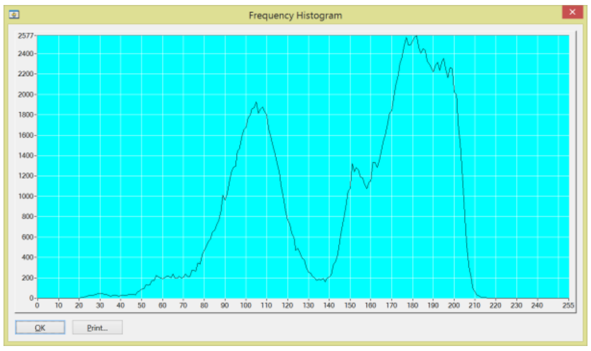
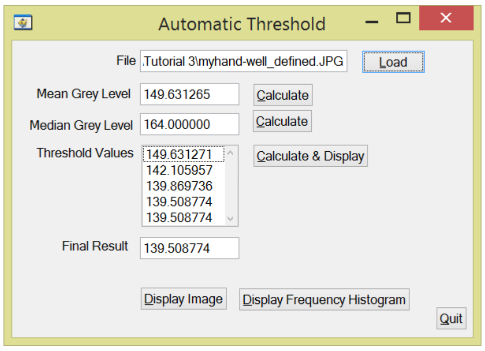


Figure 3.22



File Tutorial 3\myhand-well_defined.JPG Load

Mean Grey Level 149.631265 Calculate

Median Grey Level 164.000000 Calculate

Threshold Values
```
149.631271
142.105957
139.869736
139.508774
139.508774
```
Calculate & Display

Final Result 139.508774

Figure 3.23

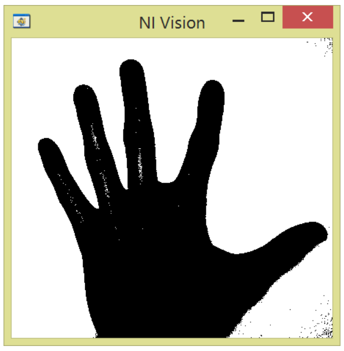


Figure 3.24

4. Image Blob Analysis / Binary Region Mensuration

4.1. Objective

The method of moments is regarded as a useful tool due to its mathematical simplicity and physical interpretation. This is a way to define figure into numbers, making easy to analyse, compare and compute a picture.

In this tutorial, different types of moment will be used to calculate a range of parameters from a blob. Every image used in this tutorial has a blob full filled with black colour. The following properties are desired: area, centroid position, angle of principal axis, shaper factor and blob sizes (width and height).

4.2. Methods

Generically, the two-dimensional moment of order (i + j) is defined as

$$M = \sum_{r=1}^{R}\sum_{c=1}^{C} r^i c^j I(r,c)$$

where R and C are constants (R is the number of rows and C is the number of columns), $I(r,c)$ is the image intensity at pixel (r,c), and i and j are non-negative integers.

The zeroth moment is characterized by $(i+j)$ equal to zero. There is only one case, corresponding to i and j equal to zero. When computed for a binary image, the zeroth moment represents the total area of the blob.

The two first order moments represent the centre of mass of the image. These are the points where all the mass of the image could be concentrated without changing the first moment of the image about any axis. The coordinates of the centre of mass can be found with the following equations.

$$\bar{x} = \frac{M_y}{A}, \qquad \bar{y} = \frac{M_x}{A}$$

where M_x is the first moment about x ($i = 1$ and $j = 0$), M_y is the first moment about y ($i = 0$ and $j = 1$) and A is the area of the binary image (zeroth moment).

The second order moments can be used to determine the orientation of the image. The orientation describes the inclination from a principal axis. The angle θ is the difference between the nearest principal axis of the blob and the x axis.

$$\theta = \frac{1}{2}\tan^{-1}\left(\frac{2N_{xyc}}{N_{xc} - N_{yc}}\right)$$

Equation 4.1

using

$$N_{xc} = N_x - A\bar{y}^2$$

$$N_{yc} = N_y - A\bar{x}^2$$

$$N_{xyc} = N_{xy} - A\bar{x}\bar{y}$$

where N_x is the second moment about x ($i = 2$ and $j = 0$), N_y is the second moment about y ($i = 0$ and $j = 2$), and N_{xy} is the second moment when $i = 1$ and $j = 1$.

In the Equation 4.1, θ is equal to half of a tangent; the tangent is valid from $-\frac{pi}{2}$ to $\frac{pi}{2}$. As a consequence, the angle θ belongs to the range $-\frac{pi}{4} < \theta < \frac{pi}{4}$.

In addition, the angle refers to the major or to the minor principal axis. The following two equations can be used in order to determine which principal axis is related to the angle. If N_θ is greater than $N_{\theta+\frac{\pi}{2}}$, then the angle is related to the major principal axis.

$$N_\theta = \frac{N_{xc} + N_{yc}}{2} + \frac{N_{xc} - N_{yc}}{2}\cos 2\theta - N_{xyc}\sin 2\theta$$

$$N_{\theta+\frac{\pi}{2}} = \frac{N_{xc} + N_{yc}}{2} + \frac{N_{xc} - N_{yc}}{2}\cos 2\left(\theta + \frac{\pi}{2}\right) - N_{xyc}\sin 2\left(\theta + \frac{\pi}{2}\right)$$

There are different types of shape factor. Each method might be adequate to a different application. In this tutorial, the shape factor is a measure about the elongation of the blob. In order to determine the shape factor, the smallest order moment N_2 and the largest second order moment N_1 are calculated.

$$N_1 = \frac{N_{xc} + N_{yc}}{2} + \sqrt{\left(\frac{N_{xc} - N_{yc}}{2}\right)^2 + {N_{xyc}}^2}$$

$$N_2 = \frac{N_{xc} + N_{yc}}{2} - \sqrt{\left(\frac{N_{xc} - N_{yc}}{2}\right)^2 + {N_{xyc}}^2}$$

The shape factor S is defined as:

$$S = \frac{N_1}{N_2}$$

The last values being determined are the width b and the height d of the blob. The ratio between them can be found using the following equation.

$$E = \sqrt{S} = \frac{d}{b}$$

Equation 4.2

The width b can also be calculated using the following expression.

$$b = \sqrt{\frac{A}{E}}$$

Equation 4.3

Using Equation 4.2 and Equation 4.3, b and d can be found.

A CVI program was developed in order to load an image containing a blob and calculate its properties. The code responsible for determining these values are displayed below. A single for-loop looks through the image array and fills the zeroth moment, first moments and second moments. Afterwards, some equations are used in order to determine the desired properties.

```
for (i=0; i<(rows); i++) {
for (j=0; j<(cols); j++) {
                temp=imageArray[c];
                if (temp == 0.0) {
                        zm = zm + 1; // Zeroth order moment (i=0 and j=0)
                        fmx = fmx + i; // First order moment about x (i=1 and j=0)
                        fmy = fmy + j; // First order moment about y (i=0 and j=1)
                        smx = smx + (i*i); // Second order moment about x (i=2 and j=0)
                        smy = smy + (j*j); // Second order moment about y (i=0 and j=2)
                        smxy = smxy + (i*j); // Second order moment when i=1 and j=1
                }
                c++;
        }
}

//Calculate position of centroid
cx = fmy/zm;
cy = fmx/zm;

//Express second order moments about centroid
smxc = smx - (zm*(cy*cy));
smyc = smy - (zm*(cx*cx));
smxcyc = smxy - (zm*(cy*cx));

//Calculate theta
if(smxc==smyc) theta=0.0; else theta=(atan(2.0*smxcyc/(smxc-smyc)))/2.0;

//Determine which of the two principal axis the angle is theta referring to
n_theta = ((smxc + smyc)/2)+(((smxc - smyc)/2)*cos(2*theta))-(smxcyc*sin(2*theta));
n_theta90 = ((smxc + smyc)/2)+(((smxc - smyc)/2)*cos(2*(theta+1.57079635)))-
(smxcyc*sin(2*(theta+1.57079635)));

//Calculate largest and smallest second order moments
smp1=((smxc+smyc)/2.0)+pow(pow((smxc-smyc)/2.0,2.0)+(smxcyc*smxcyc),0.5);
smp2=((smxc+smyc)/2.0)-pow(pow((smxc-smyc)/2.0,2.0)+(smxcyc*smxcyc),0.5);

//Calculate shape factor
if(smp2==0.0) shape=0.0; else shape=smp1/smp2;
```

4.3. Results and Discussion

The program was tested with sample images. Each image contains a single object in dark colour on a white background. The size, shape and inclination of the object vary in order to test every propriety. The program has the following screen.

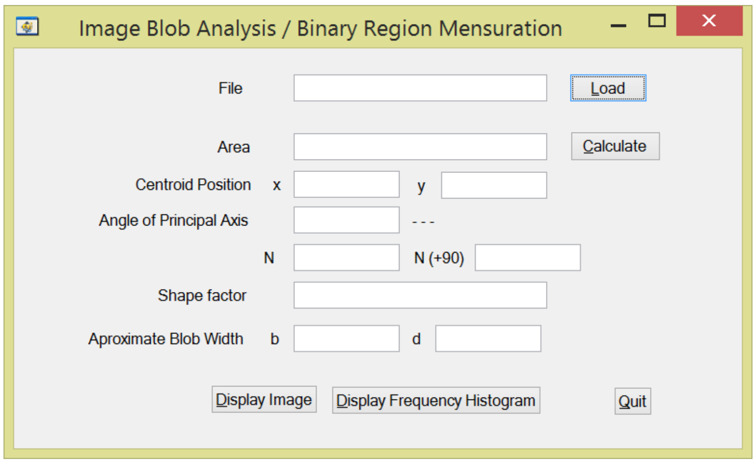


Figure 4.1

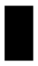
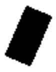
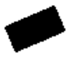
The Table 4.1 shows three images loaded in the program. The first and the second column are related to squares in different sizes and positions. The resultant area and the resultant centroid position can be checked in the second and third line. The last column shows a circle. The shape factor (sixth line) of the circle is similar to the square. It happens because the shape factor used in this report considers the elongation of the object. Both of shapes are symmetric and do not present elongation. For that reason, the method applied in this report might not be properly used to distinguish both shapes numerically.

Table 4.1

Image			
Area	1681	441	2802
Centroid Position X / Y	33 / 50	70 / 22	46.989292 / 49
Angle of Principal Axis	0	0	-0.010638
N_θ / $N_{\theta+\frac{\pi}{2}}$	235340 / 235340	16170 / 16170	624326 / 625226
Shape Factor	1	1	1.001442
App. Blob Width/Height	41 / 41	21 / 21	52.914860 / 52.952987

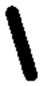
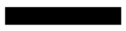
The Table 4.2 shows three different variation of inclination and position of a same rectangle. The rectangle is the same; as a consequence, the table provides the same area, sizes (width and height) and shape factor. The difference in the position of the centroid and in the angle of principal axis can be seen in the third and fourth lines. The object of the second column is inclined $-23.5°$ in reference to the minor principal axis (N_θ theta is less than $N_{\theta+\frac{\pi}{2}}$). The object of the last column is inclined $23.5°$, measured from the major principal axis (N_θ theta is greater than $N_{\theta+\frac{\pi}{2}}$).

Table 4.2

Image			
Area	861	861	861
Cent. Position X / Y	29 / 56	37.004646 / 39.030197	37.030197 / 39.004646
Angle of Princ. Axis	0	-23.504031	23.504030
$\boldsymbol{N_\theta}$ **/** $\boldsymbol{N_{\theta+\frac{\pi}{2}}}$	120540 / 31570	72305.453125 / 78375.726562	78375.718750 / 72305.453125
Shape Factor	3.818182	3.708117	3.708117
App. Blob Width/Height	20.991207 / 41.017172	21.145269 / 40.718327	21.145269 / 40.718325

The Table 4.3 shows two elongated rectangles. That feature can be directly analysed in the picture and it is numerically represented in the shape factor value. Comparing to the rectangles previously shown in Table 4.2 (the shape factor was approximately 3.8), the shape factor of the next rectangles are around ten times greater. There is one issue that was not resolved in this case. The angle θ of the first column is equal to $16.92°$. Analysing the image, that value could be correct if the reference axis was the minor principal axis. However, N_θ is greater than $N_{\theta+\frac{\pi}{2}}$, which indicates that this value is referring to the major principal axis.

Table 4.3

Image		
Area	566	1118
Centroid Position X / Y	24.491165 / 46.901058	48.5 / 42
Angle of Principal Axis	16.921455	0
$\boldsymbol{N_\theta}$ **/** $\boldsymbol{N_{\theta+\frac{\pi}{2}}}$	103165.14062 / 48958.902344	15652 / 688967.5
Shape Factor	31.513056	44.017857
App. Blob Width/Height	10.041200 / 56.367771	12.981165 / 86.124778

5. Image Segmentation

5.1. Objective

The purpose of this tutorial is to develop a program which can label every blob from a same picture. The program has to analyse each black pixel and its neighbour. There are two methods that might be applied: 4 or 8 connected pixels. Each one is adequate to different applications. At the end, the program should show the regions in different colours.

5.2. Methods

In order to organize the pixels into regions, the idea is to look for every black pixel. When the first dark pixel is found, that pixel is called *seed*. From the *seed*, the program only analyses the neighbour pixels. There are two kinds of neighbouring: one of them is called 4 adjacent pixels. In this case, every neighbour localised in position 1, 3, 5 and 7 can be considered as part of the same region. The second is called 8 adjacent pixels. In this case, all eight neighbour pixels can belong to the same region. When a neighbour pixel is detected as part of the same region, the neighbours of that pixel are also analysed. Each technique might react differently. Therefore, they can be applied for different applications. Generally, the 4-adjacent method do not consider every pixel around. For that reason, it might be a solution when it is necessary to separate objects which are touching each other or very close. The 8-adjacent method considers all eight neighbour pixels. Hence, this technique guarantees that all eight pixels will be related to the same region.

8 (i – 1 , j – 1)	1 (i – 1 , j - 1)	2 (i – 1 , j + 1)
7 (i , j – 1)	0 (i , j)	3 (i , j + 1)
6 (i + 1 , j - 1)	5 (i + 1 , j)	4 (i + 1 , j + 1)

The flow diagram of the main idea is shown below.

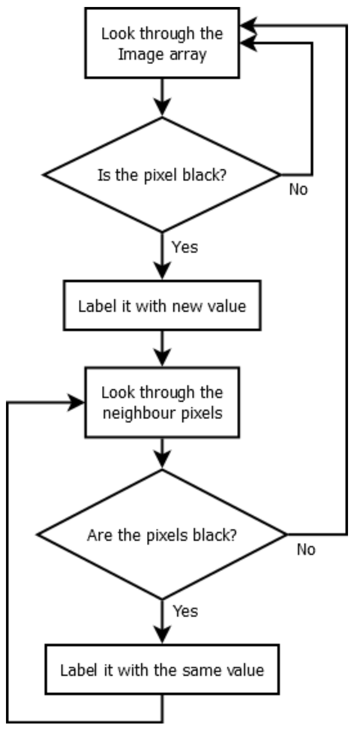

Look through the Image array
Is the pixel black?
No
Yes
Label it with new value
Look through the neighbour pixels
Are the pixels black?
No
Yes
Label it with the same value

Figure 5.1

The first for-loop shown in Figure 5.1 is performed in the following code. This stage is slightly different from the code provided. In the code provided, there was a *break* function inside the for-loop and variables *iseed* and *jseed*. In addition, the code was calling the function *mark4* outside the second for-loop. In my point of view, the function *mark4* or *mark8* should be called inside the second for-loop in order to guarantee that every pixel will be analysed.

```
value = 0;
for(i=0; i<100; i++) {
        for(j=0; j<100; j++) {
                if(Array[i][j] == 0) {
                        value = value + 1;
                        //Call function mark4 or mark8 to grow region
                        //in doing so all pixel values in region are altered to 'value'.
                        if (inOperation == 2) mark4(value, i, j); else mark8(value, i, j);
                }
        }
}
```

The function *mark4* is shown in the box below. In that function, the program analyses the neighbour pixels of the seed and, further, every pixel found is a new seed and the function is re-called. In order to make the code more understandable, there is a matrix *matrix* which each line is related with one position of the neighbour. In the for-loop, the position of a neighbour is determined adding a line of the matrix with the coordinates x and y of the seed.

```
void mark4 (int value, int iseed, int jseed) {
	int i, n, m;
	int matrix[4][2] = {	{-1, 0},
				{ 0, 1},
				{ 1, 0},
				{ 0,-1}};
	//This is not a black pixel! Stop.
	if (Array[iseed][jseed] !=0) return;
	//Set this pixel to the mark value
	Array[iseed][jseed] = value;
	//Now look for the four neighbours of the current pixel
	//The neighbours are related with each line of matrix[][]
	for (i=0; i<4; i++) {
		n = iseed + matrix[i][0];
		m = jseed + matrix[i][1];
		//Make sure that the pixel (n,m) belongs to the image
		if(n<0 || n>99) continue;
		if(m<0 || m>99) continue;
		//Found a black neighbour
		if(Array[n][m] == 0)
			mark4(value, n, m);
	}
}
```

A similar code was developed to analyse the 8-adjacent pixels. This function was called *mark8*. The changes are located in the initialization of the matrix *matrix* and in the parameters of the for-loop.

```
void mark8 (int value, int iseed, int jseed) {
	[ … ]
	int matrix[8][2] = {	{-1, 0},
				{-1, 1},
				{ 0, 1},
				{ 1, 1},
				{ 1, 0},
				{ 1,-1},
				{ 0,-1},
				{-1,-1} };

[ … ]
	for (i=0; i<8; i++) {
		[ … ]
		if(Array[n][m] == 0)
			mark8(value, n, m);
	}
}
```

At the end of the process, the image is saved in the disk and re-loaded. Each region is coloured in a different colour according to its value. An image is loaded and coloured by the following code.

```
myImage1 = imaqCreateImage (IMAQ_IMAGE_RGB, 0);//RGB image
```

```
imaqReadFile(myImage1, "C:\\IMAGEM.bmp", NULL, NULL ); // Read image file
imageArrayDisplay = imaqImageToArray (myImage1, IMAQ_NO_RECT, &cols,&rows);
for (i=0; i<(rows*cols); i+=1) {
        if (imageArrayDisplay[i].R == 1){imageArrayDisplay[i] = IMAQ_RGB_RED;}
        if (imageArrayDisplay[i].R == 2){imageArrayDisplay[i] = IMAQ_RGB_BLUE;}
        if (imageArrayDisplay[i].R == 3){imageArrayDisplay[i] = IMAQ_RGB_PINK;}
        if (imageArrayDisplay[i].R == 4){imageArrayDisplay[i] = IMAQ_RGB_GREEN;}
        if (imageArrayDisplay[i].R == 5){imageArrayDisplay[i] = IMAQ_RGB_YELLOW;}
}
imaqArrayToImage (myImage1, imageArrayDisplay, cols, rows);
imaqDisplayImage (myImage1, 0, TRUE);
```

5.3. Results and discussion

The following pictures show images that both types of neighbour (4-adjacent and 8-adjacent) have the same result. Each pixel which belongs to the same region is connected at least with one 4-adjacent neighbour.

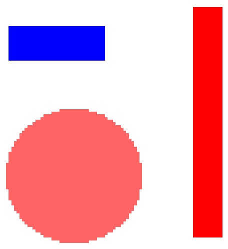


Figure 5.2

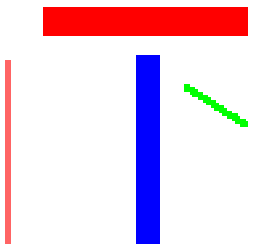


Figure 5.3

In the next example, the pictures show different results. Due to the way in which the pixels of the character "S" are connected, the 4-adjacent method cannot identify the whole character (Figure 5.4). When the 8-adjacent technique is applied (Figure 5.5), the character is completely recognised.

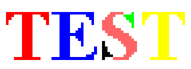


Figure 5.4

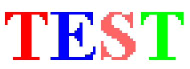


Figure 5.5

6. Erosion and Dilation

6.1. Objective

In this tutorial, the purpose is to develop a program which can perform 4 and 8 connect erosion in the image. This technique analyses the whole border of an object and removes its pixels. In addition, using a similar idea, the border of an object can be highlighted (inbound or outbound). Some pictures were supplied and the objective is to minimise all the undesired features within the images.

6.2. Methods

Both "erosion" and "dilation" operations are considered essential to morphological analysis of an image. In addition, the majority of the morphological methods are based on these two operations (Awcock & Thomas, 1995). The "skeletonization" is one of methods derived from them. This method has a great value as an identification tool. It can be a shape descriptor (invariant from scaling and rotating) and also can be used to reconstruct objects. When the interest is not the bulk shape of an object, such as a text analysis, the "skeletonization" can simplify the letters in order to facilitate their identification. Opening and closing are also operations which use erosion and dilation. They are important to boundary smoothing and noise elimination process.

Basically, the erosion method works removing all the pixels 4-adjacent or 8-adjecent to the background. In order to perform that, the program should pass through the image array twice. In the first search, the program selects all the neighbouring pixels to the background. In a next search, the program removes all the pixels previously selected. If the cells had been removed in the first search, the program would delete undesired pixels.

Instead of removing the neighbouring pixels to the background, every black pixel can be removed and the selected pixels can be returned to black colour. As a consequence, the program will highlight the border of the objects. The processes of erosion and boundary are shown in Figure 6.1.

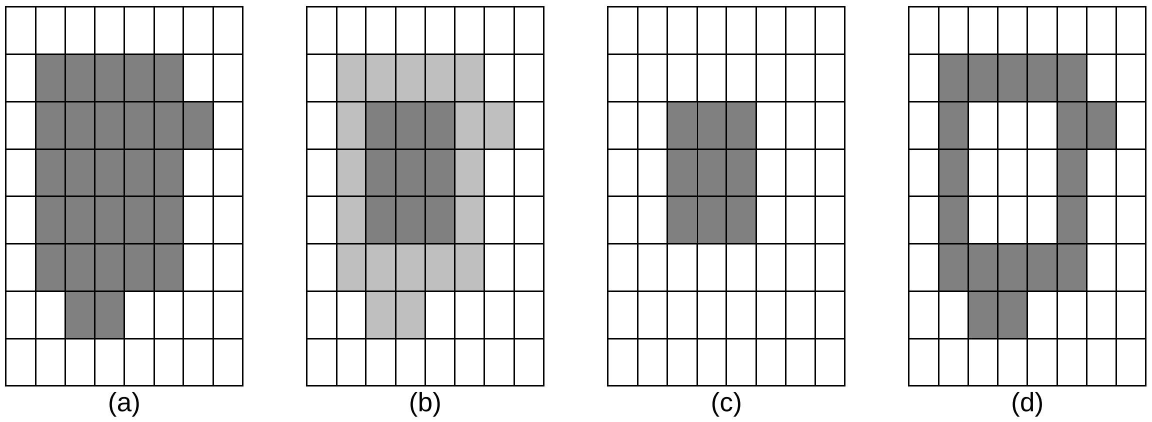


Figure 6.1: Erosion process: (a) the original image (dark gray is the object); (b) the neighbouring pixels to the background are selected (middle gray); (c) the erosion of (a); (d) boundary of (a)

The program was written in order to perform erosion and isolation of the border. The first step (selecting the neighbouring pixels) does not change in both operations. The second step is different, and depends in which operation has been selected by the user. The pseudo-code is illustrated bellow. The variable *inOperation* indicates the operation selected by the user. The table below shows all the possible values of inOperation and their meaning.

Table 6.1: Values and meanings for the variable "inOperation"

Value	**Meaning**
1	Display Image
2	Erosion 4-Connected
3	Erosion 8-Connected
4	Boundary 8-Connected

The pseudo-code for erosion and boundary operations is shown below.

```
Repeat the following instructions how many times the user had selected:
{
    Repeat the following instructions for every pixel in the image:
    {
        If ImageArray(index) = 0 then
            Repeat the following instructions for all 4 or 8-connected pixels:
            {
                If one of the neighbour pixel is white then ImageArray(index) = 2
                End If
            }
        End If
    }
    Repeat the following instructions for every pixel in the image:
    {
        If we are isolating the boundary (inOperation = 4) then
            If ImageArray(index) = 2 then
                ImageArray(index) = 0
            Else
                ImageArray(index) = 255
            End If
        Else
            If ImageArray(index) = 2 then
                ImageArray(index) = 255
        End If
    }
}
```

6.3. Results and Discussion

In the program, the user decides the number of erosions executed by the program. As a consequence, a large defect can be removed when erosions are executed consecutively. However, the user should be aware that it also can remove desired features of the object. An example of this issue will be shown in this report. In addition, this report will show the differences between 4-adjacent and 8-adjacent processes.

The first picture analysed is shown in Figure 6.2 (a). This picture contains a square surrounded with small defects in the four sides and small dots spread through the whole image. Using erosion, the objective is to remove these small defects reducing the size of the square. Afterwards, the dilation technique would restore the original size. The Figure 6.2 (b) and Figure 6.2 (c) indicate that the small defects around the square will still remain using the 4-adjacent process, even if erosions are processed consecutively. In order to remove these defects, the 8-adjacent process should be used. The Figure 6.2 (d) shows that in a single erosion, the defects can be completely removed.

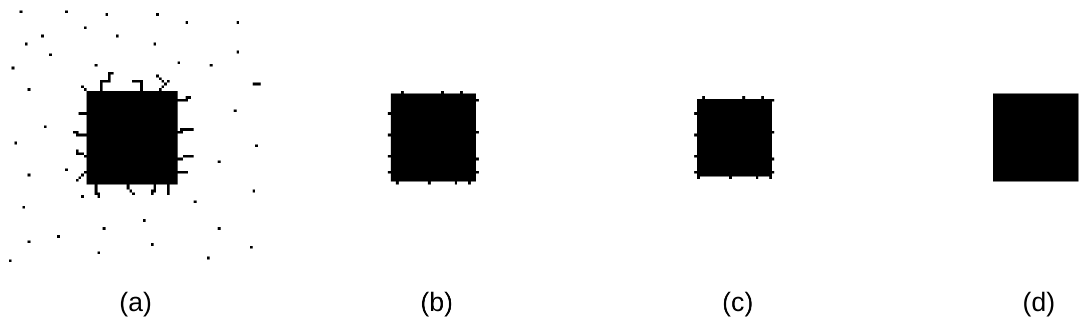

(a) (b) (c) (d)

Figure 6.2: (a) original image; (b) 4-connected erosion of (a); (c) three consecutive 4-connected erosion of (a); (d) 8-connected erosion

The next image is the Figure 6.3 (a). In the original image, there are connections between objects. These connections are regarded as defects in the image. The desired image is containing the objects without those links. When the erosion is applied in the original image, both 4-adjacent and 8-adjacent erosions can remove the undesired defects from the original image. However, the 8-connection erosion is more efficient in order to remove the connection between the circle and the square, due to neighbouring pixels in the diagonal line.

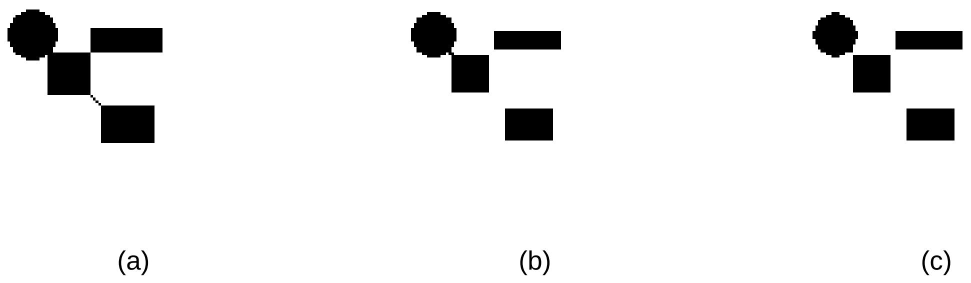
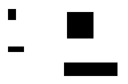
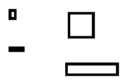

(a) (b) (c)

Figure 6.3: (a) original image; (b) 4-connected erosion of (a); (c) 8-connected erosion

Instead of removing the border of the objects, the next procedure will highlight these pixels. The Figure 6.4 (b) contains the border of the objects contained in the original image Figure 6.4 (a).



(a)



(b)

Figure 6.4: (a) original image; (b) boundary selection

7. Grey Level Edge Detection

7.1. Objective

In this work, the objective is to write a program which can enhance the edges of the image. The image is a 200 x 200 pixels resolution given by 0 to 255 grey level values. The program should load an image from the hard disk and process different techniques for edge enhancing. The resultant image should be displayed and saved in the hard disk. In this work, Kirch, Sobel and Laplacian operators will be studied and used to analyse some images.

7.2. Methods

The edges can be basically defined as an intense discontinuity in the image in one or more directions. Some methods of identifying these points are based on the determination of intensity gradients across the image. In order to highlight only these discontinuities, the pixels value are replaced with the finite-difference of their neighbours. As a consequence, the resultant pixel value will be a low value in a uniform region; on the order hand, the values will be high in a region containing edges. Using merely the finite-difference between pixels and their neighbours might generate noise in the resultant image. For that reason, there are some techniques applying weights to neighbours pixels. (Awcock & Thomas, 1995)

Convolution is one of the most important neighbourhood operator (Burdick, 1997). Using that mathematical tool in a digital imagery, a local area of pixels is used to calculate a desired value. The new value of a pixel is a weighted sum of its neighbouring grey-level values. The Equation 7.1 illustrates how the convolution is applied in this case. In that equation, *mask* is a 3x3 matrix containing the weight values, *image* is a 200x200 matrix containing all the grey-level values of the original image and *new_image* is a 200x200 matrix which stores the resultant values of the operation. As can be seen, each element of the mask is multiplied by a respective pixel value in the neighbourhood. All the multiplications are added together and the result is divided by the summation of the mask values. If the weight of the operator is equal to 0, the division by the weight should be ignored (Burdick, 1997).

A	B	C
D	E	F
G	H	I

Figure 7.1: Matrix "mask"

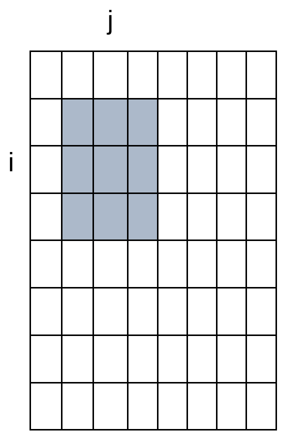


Figure 7.2: Matrix "image"

$$new\_image[i][j] = \frac{\sum_{n=-1}^{1}\sum_{m=-1}^{1} mask[n+1][m+1] * image[i+n][j+m]}{\sum_{n=-1}^{1}\sum_{m=-1}^{1} mask[n+1][m+1]}$$

Equation 7.1

The first method uses the Kirsch operators. This is a simple method, which highlights edges in specific directions (vertical, horizontal and diagonals). At the end, they can be performed consecutively in order to highlight edges from different directions. In Kirsch operators, the masks contain the following numbers.

-1	0	1
-1	0	1
-1	0	1

Figure 7.3: Vertical

1	1	1
0	0	0
-1	-1	-1

Figure 7.4: Horizontal

0	1	1
-1	0	1
-1	-1	0

Figure 7.5: 1st Diagonal

0	-1	-1
1	0	-1
1	1	0

Figure 7.6: 2nd Diagonal

The Laplacian and Sobel edge enhancement operations obtains the edges in all directions. Regions containing a constant brightness become black; on the other hand, changing on the brightness becomes white. This method is omnidirectional; in other words, all the black-to-white and white-to-black changes are highlighted for the same colour, regardless of their direction. The Sobel operator is less sensitive to image noise than the Laplacian operator, providing a stronger edge detection. (Baxes, 1994)

The next method is the Laplacian operator. This operator uses the following mask:

0	1	0
1	-4	1
0	1	0

Figure 7.7: Laplacian

The Sobel operation is not executed applying a single mask on the image grid. In this method, two values (s_x and s_y) are calculated from two different masks. The values are extracted from the following mask.

-1	0	1
-2	0	2
-1	0	1

Figure 7.8: 1st Value (s_x)

-1	-2	-1
0	0	0
1	2	1

Figure 7.9: 2nd Value (s_y)

The final value of the pixel is $\sqrt{{s_x}^2 + {s_y}^2}$.

All these operations can generate values less than 0 and greater than 255. Hence, before to display the image, every value need to be re-scaled into 0 and 255. Firstly, the program will look for the lowest and greatest value between the resultant values. Then, all the values are changed in function of these two extreme values.

The following flow diagram represents the algorithm used to execute the tasks.

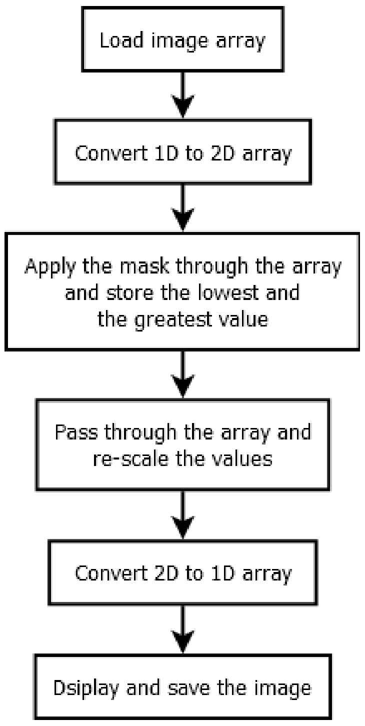

Load image array
Convert 1D to 2D array
Apply the mask through the array and store the lowest and the greatest value
Pass through the array and re-scale the values
Convert 2D to 1D array
Dsiplay and save the image

Figure 7.10

In the CVI program, the convolution is executed by a function called *conv*. This function receives the position *i* and *j* of the current pixel and the desired mask to be applied. The function looks through the neighbouring pixels and executes the convolution with the desired mask. The function is shown below.

```
int conv (int i, int j, int mask[3][3]){
        int result,m,n;
        result = 0;
        for(m=-1; m<=1; m++) {
                for(n=-1; n<=1; n++) {
                        result = result + mask[m+1][n+1]*ArrayRead[i+m][j+n];
                }
        }
        return result;
}
```

The following table shows the possible values and their meaning of the variable *inOperation*.

Table 7.1

Value	Meaning
1	Display Image
2	Vertical
3	Horizontal

4	1st Diagonal
5	2nd Diagonal
6	General Kirsch
7	General Laplacian
8	General Sobel

The following pseudo-code refers to the moment when the function *conv* is called. In this pseudo-code, *ArrayWrite* is the array which stores the resultant values, *i* and *j* are incremental variables for the row and the column position.

```
Repeat the following instructions for every pixel in the image
{
        Switch inOperation
        {
                Case 2: ArrayWrite[i][j] = conv (i,j,mask_vertical);
                Case 3: ArrayWrite [i][j] = conv (I,j,mask_horizontal);
                Case 4: ArrayWrite [i][j] = conv (I,j,mask_diagonal1);
                Case 5: ArrayWrite [i][j] = conv (I,j,mask_diagonal2);
                Case 6: ArrayWrite [i][j] = conv(i,j, mask_vertical) + conv(i,j, mask_horizontal) +
conv(i,j, mask_diagonal1) + conv(i,j, mask_diagonal2);
                Case 7: ArrayWrite [i][j] = conv(i,j,mask_laplacian);
                Case 8: sx = conv(i,j, mask_sobel1);
                        sy = conv(i,j, mask_sobel2);
                        ArrayWrite [i][j] = sqrt(sx*sx+sy*sy);
        }
        Look for the lowest value r_min and greatest value r_max;
}
Calculate range (difference between the lowest and the greatest).
```

The next step aims to re-scale all the values. The re-scale is performed in order to reallocate the numbers between 0 and 255. For that reason, the following equation is applied for every pixel in the image.

$$ArrayWrite[i][j] = \frac{ArrayWrite[i][j] - r_{min}}{range} * 255$$

Equation 7.2

7.3. Results and Discussion

The methods were tested in the images supplied by the tutor. From the resultant images, the differences between each technique can be seen. Each method has features that can be particularly useful for a case. The following image is a print screen of the program.

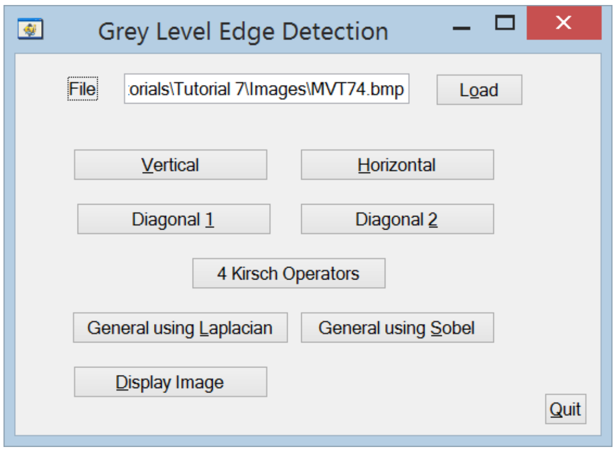


Figure 7.11

The first method uses the Kirsch operators. Each operator highlights the edges in a single direction (vertical, horizontal or diagonals). Hence, using this technique, the edges in a single direction can be intentionally highlighted (in order to measure width and height, for example). In addition, each mask can be performed consecutively and edges in all four directions can be highlighted.

For example, the following images show the horizontal and the vertical edge enhancement for the image depicted in Figure 7.12 (a). Using this technique, distances between edges can be calculated. Using the program developed in Tutorial 1, the Figure 7.12 (c) and the Figure 7.12 (e) are thresholded image from Figure 7.12 (b) and Figure 7.12 (d). The frequency histogram developed in Tutorial 3 was used to set proper threshold values. As expected, only horizontal edges appear in Figure 7.12 (c) and only vertical edges appear in Figure 7.12 (e).

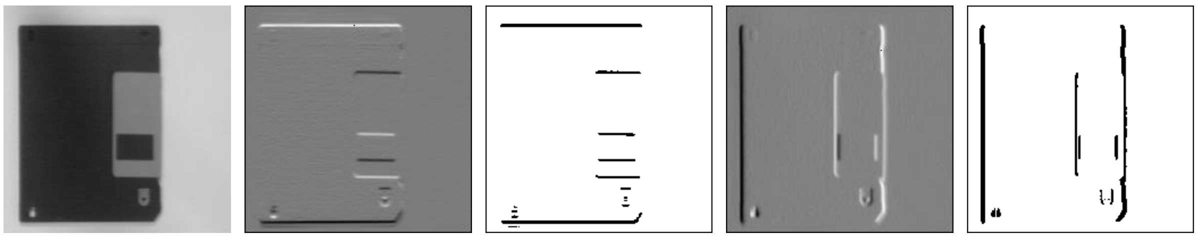


Figure 7.12

If all the Kirsch operators are applied in the original image, the resultant image contains the edges in all four directions. The Figure 7.13 (b) represents the resultant image after the convolutions be applied. The Figure 7.13 (c) is a thresholded image of Figure 7.13 (b). The image depicted in Figure 7.13 (c) highlights the edges of the original image.

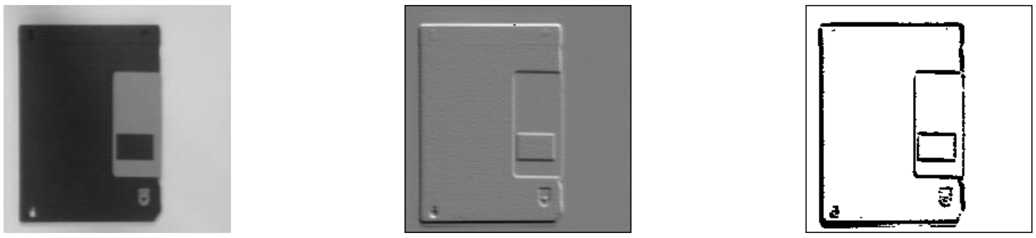

(a) (b) (c)

Figure 7.13

The next method is the Laplacian operator. The Figure 7.14 (a) is the original image and the Figure 7.14 (b) is the resultant image from the convolution. As performed previously, the program developed in Tutorial 1 was used to threshold the image and highlight the desired edges. As can be seen, this operation can contain a great deal of noise in the image. In order to remove the noise, this method is usually applied after another process for smoothing, such as the mean filter.

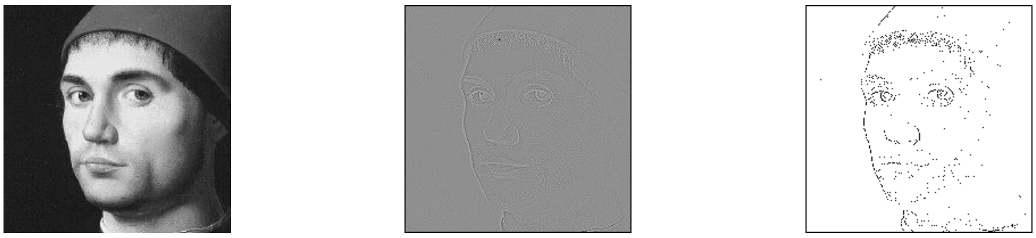

(a) (b) (c)

Figure 7.14

The Sobel operator will be used to enhance edges in the example below. The Figure 7.15 (a) and Figure 7.15 (c) are original images and the Figure 7.15 (b) and Figure 7.15 (d) are their resultant image. As can be seen, the edges are well-detected.

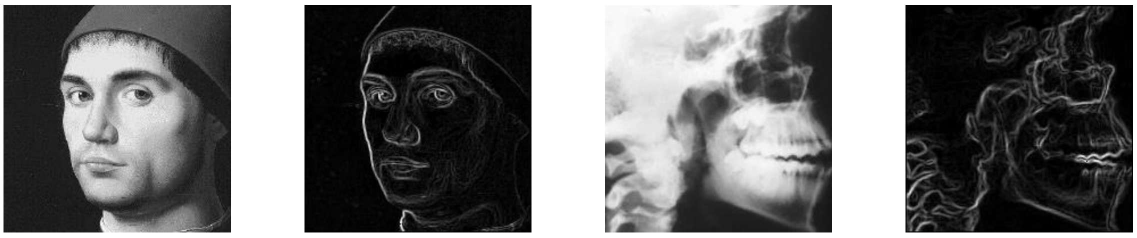

(a) (b) (c) (d)

Figure 7.15

The edge enhancement is important to exactly detect the border of an object. In this way, a sort of tasks can be performed. The program can measure certain features of a figure, such as the width and the height. The boundaries position can also be used to determine where the object is located. In addition, the edge enhancement can be useful to help the analysis of certain images, such as the X-Ray in Figure 7.15 (d).

In many applications, the environment is constructed in order to provide a huge contrast between the object and the background. For that reason, in some cases, there is not needs to use a sophisticated method to detect edges, due to the constrained environment. (Awcock & Thomas, 1995)

8. References

Awcock, G., & Thomas, R. (1995). *Applied Image Processing.* Macmillan Press.

Baxes, G. A. (1994). *Digital Image Processing.*

Burdick, H. E. (1997). *Digital Imaging.* Mc-Graw-Hill.