Final Report

Automatic Crack Monitoring System

Richard Liu
Associate Professor, Principle Investigator

Xiemin Chen
Research Associate

Min Wu
Research Assistant

Project No.: TxDOT 7-3997

Conducted in Cooperation With The
Texas Department of Transportation

TxDOT Project Director: Dar-Hao Chen
TxDOT Project Coordinator: Ed Oshinsky

DISCLAIMERS

The contents of this report reflect the views of the authors who are responsible for the facts and accuracy of the data presented herein. The contents do not necessarily reflect the official views or policies of the Texas Department of Transportation. This report does not constitute a standard, specification or regulation.

University of Houston

Houston, TX77004-4005

Report No. TxDOT 7-3997	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle **Automatic Crack Monitoring System**		5. Report Date October 2001 6. Performing Organization Code
7. Author(s) Richard Liu, Xuemin Chen, and Min Wu		8. Performing Organization Report No.
9. Performing Organization Name and Address Department of Electrical and Computer Engineering, University of Houston Houston, TX 77004-4005		10. Work Unit No. 11. Contract or Grant No.
12. Sponsoring Agency Name and Address Texas Department of Transportation 4203 Bull Creek, Bldg. #39 Austin, TX 78731		13. Type of Report and Period Covered Final Report 14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract:

This report discusses the design and implementation of a Highway Crack Monitoring System (HCMS) used in a Texas multi-purpose load simulation (TxMLS) test site. The HCMS is a black-and-white CCD camera-based image-processing device that processes the image of the cracked pavements, extracts crack information and characterizes the cracks in terms of crack length and width. In this project, both hardware and software are developed. The hardware system is fully automatic with computer controlled x-y motor motion devices and Hall-effect sensors. Many lab tests were conducted. Processed results show that an accuracy of more than 90% is achieved for crack width and length classification. The crack-processing algorithm developed in this project uses the method of feature extraction and object characterization. A user-friendly, Windows-based user interface is developed, which allows user to control the camera motion, calibrate the system, acquire crack images, process images, and store data. The accuracy of the object characterization largely depends on the reliability of the features extracted from the original image data. The HCMS system implements recurring thresholding as the basic segmentation algorithm, which adaptively determines the gray level threshold through an estimation-verification process. Distinct block operation is applied to improve the performance on the non-uniform property of the road image. Connected-component object recognition and other criteria are also implemented to distinguish the crack objects and remove the noise. The crack object in the binary image is characterized by object boundary processing. The perimeter of the object can be obtained by accumulating the moves of the pixels on the border contour. Border convolution is implemented to classify the border pixels and to determine the corresponding move length. With the area of the object, the length and width information can be obtained. The limitation of the algorithm is also discussed.

17. Key Words Laser, micro texture, macro texture, laser triangulation, position sensitive device, microwave remote sensing, millimeter waves, inversion	18. Distribution Statement No restrictions.		
19. Security Classification (of this report) Unclassified	20. Security Classification (of this page) Unclassified	21. No. of Pages	22. Price

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

This form was electrically by Elite Federal Forms Inc.

TABLE OF CONTENTS

List of Figures vi

List of Tables vii

CHAPTER 1 Introduction 1

1.1 Background 1

1.2 The Highway Crack Monitoring System 2

1.3 Organization of This Report 6

CHAPTER 2 Recurring Image Segmentation 7

2.1 Introduction 7

2.2 The Crack Images 7

2.3 Recurring Thresholding 9

2.4 Connected Component Object Identification 18

2.5 Distinct Block Operation 21

CHAPTER 3 Crack Object Characterization Algorithm 23

3.1 Introduction 23

3.2 Basic Definition 23

3.3 Border of a Crack Object 25

3.4 Crack Length Calculation 28

3.5 Crack Width Classification 35

3.6 Crack Growth 38

3.7 Calibration 40

CHAPTER 4 Illustrative Examples **42**

4.1 Example 1 42

4.2 Example 2 46

CHAPTER 5 System Software Application Design **49**

5.1 Introduction 49

5.2 Windows Framework 49

5.3 MATLAB Interface 53

5.4 CCD Camera Control 54

5.5 Motion System Control 55

5.6 Database Management 56

CHAPTER 6 System Implementation and Applications **58**

6.1 System Implementation 58

6.2 Applications 64

CHAPTER 7 Conclusions **72**

REFERENCES **74**

APPENDICES **75**

Appendix I Matlab Image Processing Source Code 75

Appendix II CCD Camera Control Class Declaration 82

Appendix III RS232 I/O Implementation Source Code 84

Appendix IV Pin Assignment of the Motion System Control Box 89

List of Figures

Figure 1.1 The block diagram of the HCMS 3

Figure 1.2 System program layout 4

Figure 1.3 The motion system 5

Figure 1.4 The TxMLS 6

Figure 2.1 Examples of the crack images 8

Figure 2.2 Histograms of the crack images in figure 2.1 11

Figure 2.3 Threshold estimation 13

Figure 2.4 Object extraction using the segmentation result 14

Figure 2.5 Intensity transformation 16

Figure 2.6 Image intensity adjustment 17

Figure 2.7 Result binary image after segmentation 18

Figure 2.8 Binary image labeling 19

Figure 2.9 Segmentation result 21

Figure 3.1 Square lattice of the digital image 24

Figure 3.2 Neighborhoods on the square lattice 24

Figure 3.3 Borders of a binary image object 25

Figure 3.4 Border extraction 28

Figure 3.5 Discrete distance: (A) $d_4(P, Q)$ (B) $d_8(P, Q)$ 30

Figure 3.6 Turning pixels 31

Figure 3.7 Calculation of d_D in the first octant 34

Figure 3.8 Processing result 37

Figure 3.9 Image Grids 38

Figure 3.10 Crack growth detection 39

Figure 4.1 Example I: Source image 42

Figure 4.2 Example I: Processing result 43

Figure 4.3 Example II 47

Figure 6.1 CCD camera used in this project 59

Figure 6.2 Control board and DC power supply 60

Figure 6.3 (a) Schematic of the controller-driver circuit 62

Figure 6.4 (b) Schematic of the controller-microprocessor and interface 63

Figure 6.5 Processed results of a horizontal crack. Processed area is shown by the rectangular box. 65

Figure 6.6 The image processing software can not identify object from cracks. The ruler image is being processed as a vertical crack in this example. 66

Figure 6.7 Asphalt cracks with fine branches 67

Figure 6.8 An example of alligator cracks and processed results 68

Figure 6.9 Processed results of a cluster of cracks on an old-pavement 69

Figure 6.10 Processed result of scattered cracks on an asphalt surface 70

Figure 6.11 Processed results of asphalt-sealed cracks 71

List of Tables

Table 4.1 Parameters of the Image Objects 43

Table 4.2 Calculation results 44

Table 4.3 Calculation results of the rotated image 44

Table 4.4 Crack coverage of example II 48

Chapter 1

Introduction

1.1 Background

In the last 10 years, the United States has witnessed a continuing rise in population and the longest peacetime expansion of the economy in the nation's history. As a result, travel and transportation rise proportionately. In terms of passenger travel, Americans used the automobile for more than 90 percent of their travel (by mileage) in 1975. Today, highway travel (in passenger miles) and automobiles have continued to dominate--still accounting for 90 percent of travel--while air travel accounts for another 9 percent, and the other modes together account for the last one percent. In terms of freight transportation, there has been a large increase in highway and truck transportation--an increase of its modal share from 23 to 30 percent in just 10 years--accounting for the major part of the increase of freight transportation over the past 10 years.

However, the nation’s highway system is aging, while the volume of the traffic that it supports continues to increase dramatically. Road maintenance technology, however, has remained virtually stagnant for many years. It typically involves small-scale, dispersed activities performed under traffic conditions by relatively low-skilled laborers with basic equipment. Conventional road maintenance methods will be seriously strained to meet the increasing demands of the future.

Automation of the road maintenance operations has a tremendous potential to improve this situation. Highway crack monitoring is especially well suited for this purpose since it is such a widespread, costly, and labor-intensive operation. Accuracy and efficiency are always the problems, and as traffic volumes increase, crack monitoring operations become increasingly disruptive to the travelling public. Automation of crack monitoring can significantly reduce labor costs, improve work quality, provide better records, and reduce worker exposure to road hazards.

The Highway Crack Monitoring System (HCMS) presented in this report is capable of automatic-image acquisition and processing. The cracks will be extracted from the high-noise background and will be also classified based on its width. The overall statistical information of the road will be obtained for analysis as well.

1.2 The Highway Crack Monitoring System

The main purpose of the HCMS system is to obtain the information about the crack and its coverage of the road image. Based on the width of the crack, it is classified into five categories. The major tasks of the HCMS system are the following:

I. Road image acquisition

II. Image processing

III. Crack object classification

IV. Statistical report

The HCMS consists of three major parts; the computer, the image acquisition device, and the motion system. The block diagram of the HCMS system is shown in Figure 1.1.

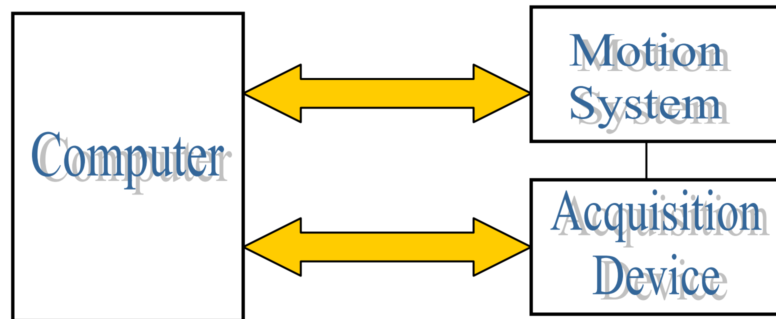


Figure 1.1 the block diagram of the HCMS

The computer is the center for both system control and image processing. Because of the huge amount of image processing and storage, the computer should have high speed, large physical memory, massive storage space, and fast video processing power. Currently we are using a Pentium III 550MHz workstation, which has a physical memory of 256 MB and a 15GB hard disk drive. Faster computers are desired in the future.

The HCMS application software is programmed with Microsoft Foundation Class (MFC) on the 32-bit Windows NT operation system. Figure 1.2 shows the layout of this application program. It integrates the functionality of the image acquisition, image processing and motion-system control in one system. The double-view style enables the user to view both the original image and processing result at the same time. The image processing uses the tool of MATLAB Image Processing Toolbox version 5.3 Multithread programming is implemented for the CCD camera control. A component is

designed for the RS232 serial port communication using the Windows APIs. Multiple database support is also used in the software design to store the statistical data and improve the user interface. More detail will be introduced in Chapter 6.

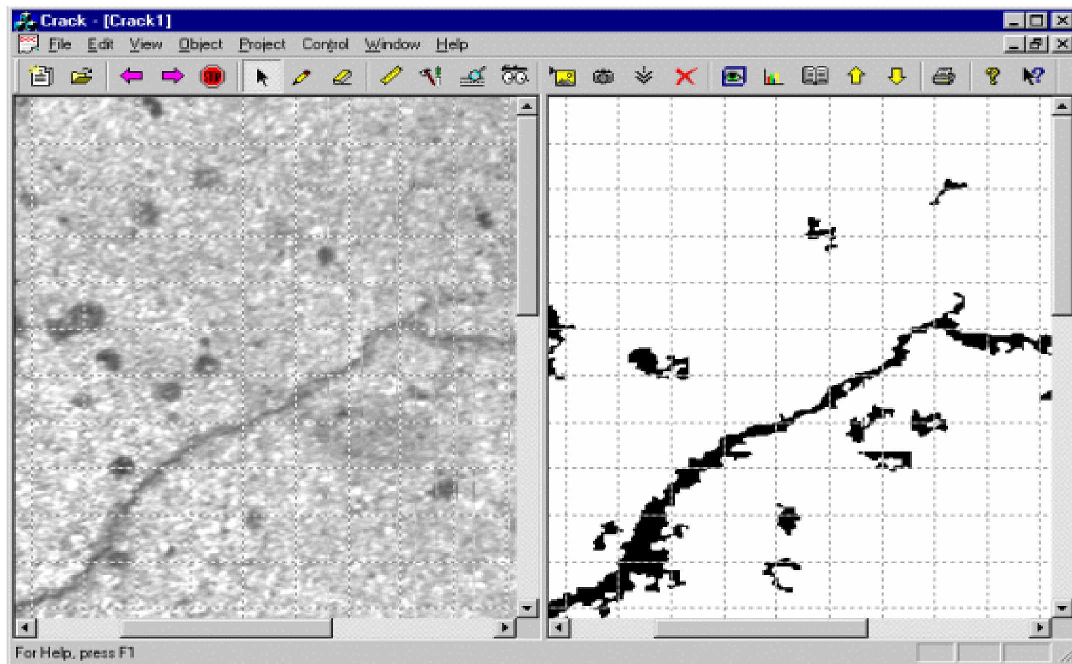


Figure 1.2 System program layout

The CCD camera is used as the image acquisition device in the HCMS system. It can acquire images at 256 gray-scale, with resolution of 640-by-480 pixel format. The camera is placed on the frame of a motion system (Figure 1.3), about one meter above the ground, which makes the image area of 1.2-by-1.2 meters with the 6-mm lens. The acquired image is stored in the Windows bitmap format. The typical size of the image file is about 320 KB. Examples of such images are shown in Figure 2.1 in Chapter 2.

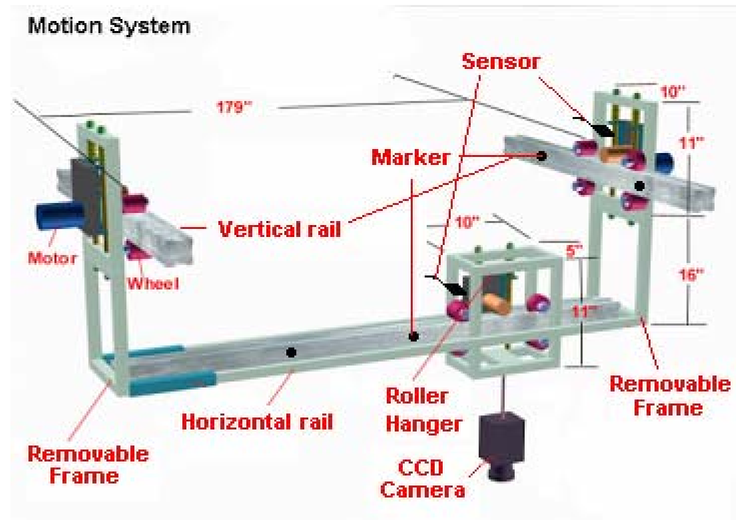


Figure 1.3 The motion system

The motion system of the HCMS is shown in Figure 1.3. These two vertical rails are fixed on both sides of the Texas Multipurpose Loading System (TxMLS), which is shown in Figure 1.4. The TxMLS has a huge vehicle designed by the Texas Department of Transportation, for the purpose of highway monitoring and maintenance. The motion system is capable of 2-dimensional movements to let the camera work continuously in either direction, so that full image coverage of the road is achieved. The control signal of the motion system is sent and received through the RS232 serial port of the computer. There are some magnetic markers set along the vertical and horizontal rail. The sensors are placed on both the removable frame and the roller hanger, so that they can move together with the motion system. Whenever the sensor moves close enough to the marker, the two wires attached to the sensor are short-circuited. Once the system detects this signal, the motion system stops and the camera takes an image. The markers are

precisely set so that the minimum overlapping between two adjacent images is achieved. Software debouncing should be considered to remove the spurious sensor signal.

Figure 1.4 The TxMLS

1.3 Organization of This Report

The image segmentation algorithm will be presented in detail in Chapter 2. It is a very important step to extract cracks from the original road image. After the segmentation, a binary image is obtained for crack analysis. Chapter 3 will discuss the crack's length and width calculation algorithm. As the width information is obtained for each individual crack, the cracks will be classified based on that information. Two examples of the field test will be listed and analyzed in Chapter 4. Chapter 5 will give a brief introduction to the system software. Chapter 6 illustrates some examples. The conclusion and future work will be given in the last chapter.

Chapter 2

Recurring Image Segmentation

2.1 Introduction

Image segmentation is one of the most important steps leading to the analysis of processed image data—its main purpose is to divide an image into parts that have a strong correlation with objects or areas of the real world contained in the image. [1]

Typically, digital images often come up with huge quantities of data with respect to the size and depth of color or gray scale. With image segmentation, substantial data volume reduction is immediately gained, and more importantly, the image objects of our interest are separated from the rest of the image. According to the complexity of the road images, the recurring thresholding method is implemented to improve the sensitivity and accuracy of the processing algorithm. Usually, entirely correct and complete segmentation of complex scenes cannot be achieved. Connected component-object identification is implemented to remove noise objects at a higher level of processing.

2.2 The Crack Images

2.2.1 Intensity Image Data

For an intensity image, the image data can be stored in a single two-dimensional matrix, with each element of the matrix corresponding to one image pixel. For a 256 gray scale image, the image data is within the range of [0,255]. The size of the matrix is

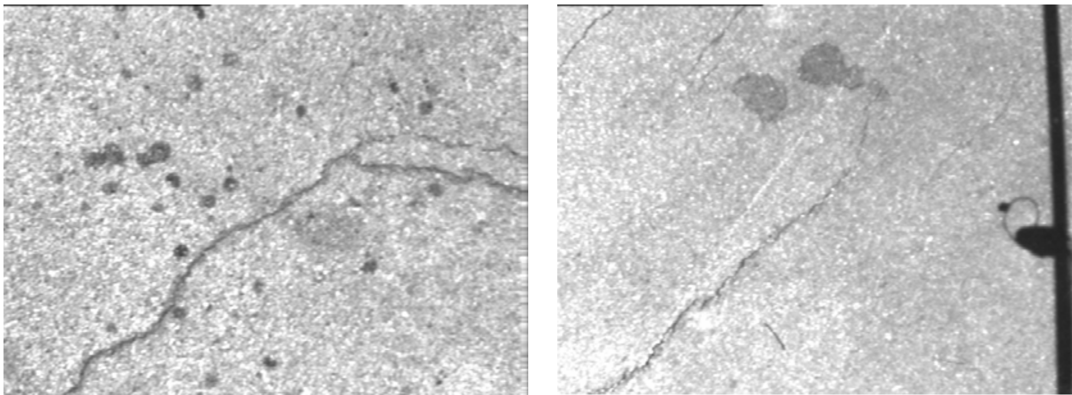
the size of the image by pixel. Therefore the acquired image in the HCMS system, which is in the format of 256 gray scale and the size of 640-by-480 pixel, can be defined as

$$f(i, j) \in [0,255] \tag{2.1}$$

where $0 < i \leq 640$, $0 < j \leq 480$.

Be aware that the elements in the intensity matrix represent the gray level at this pixel, "0" represents black and "255" represents full intensity, or white.

2.2.2 Characteristics of the Crack Image



(A) (B)

Figure 2.1 Examples of the crack images

Image data ambiguity is the major problem of segmentation, especially for crack images. The texture of the pavement always causes noises. The shadow and other surface objects can be other sources of the noise. Figure 2.1 shows two typical samples taken from the highway. In figure 2.1 (A), the region of the cracks is relatively easy to distinguish. The gray level of the crack objects is clearly distinct from the background.

There are several oil spots that need to be identified. The image in figure 2.1 (B) unfortunately represents a more complicated situation that is commonly found in general practice. The image background varies at different parts of the image due to the reflection and non-uniform lighting. There is a shadow region on the right, which is incurred by the camera frame.

2.3 Recurring Thresholding

Gray-level thresholding is one of the more commonly used methods of segmentation. The objects or image regions are characterized by reflectivity or light absorption of their surfaces. A brightness constant or threshold can be determined to segment objects and background. [1]

The recurring thresholding method uses gray-level thresholding as the basic method. Therefore, correct threshold selection is crucial for successful segmentation. It is impossible to use a single threshold for all road images, since there are gray-level variations among different images and even objects and background in the same image. However, it can be adaptively estimated based on the average gray level of the specific image or image fraction. This estimated threshold is set high so that most of the crack objects and some noise objects are allowed as objects in the resulting binary image. With this result as a mask, we can obtain a new “crack” image by simply clearing all the pixels, which are marked as background in the binary result from the original road image. It is reasonable to see that most of the background is removed in this image, which results in an image with a more regular histogram with a local peak representing

the crack pixel distribution. More accurate segmentation can be achieved by analyzing this image and its histogram.

2.3.1 Histogram and Threshold

The crack objects can be characterized by light absorption of the highway surface, which appear at a different gray scale from the neighbors or background. From Figure 2.1, it is noticeable that the cracks are slightly darker than the neighboring areas, which indicates that they usually have a lower gray value to distinguish themselves from the background.

Gray-level thresholding is the transformation of an input image $f(i, j)$ to an output image $g(i, j)$ as follows:

$$g(i, j) = \begin{cases} 1 & \text{for} \quad f(i, j) \geq T \\ 0 & \text{for} \quad f(i, j) < T \end{cases} \tag{2.2}$$

where T is the threshold constant, $g(i, j) = 1$ for image pixels of the image objects, and $g(i, j) = 0$ for background pixels (or vice versa). With thresholding, the original intensity image is converted to a binary image. Only the pixels of particular interest are left as image objects.

Correct threshold T selection is crucial for successful threshold segmentation. The direct method of gray-level thresholding is based on the histogram-shape analysis. The histogram is the chart that shows the distribution of intensities in an intensity image. Figure 2.2 shows the histograms of the two corresponding images in Figure 2.1. In both histograms, there is a local peak near the intensity 200, which indicates most of the crack pixels, as well as some of the noise background. The sharp rise near gray level 255

in both images is caused by the large quantity of the white pixels on the image background. The lower range ([0,250]) of the histogram reflects the distribution of cracks.

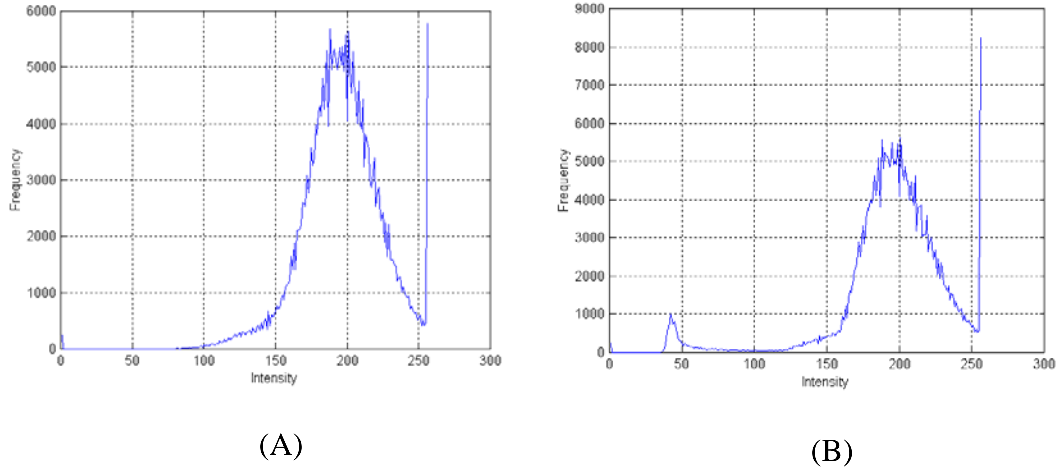


Figure 2.2 Histograms of the images in Figure 2.1

The different shape of the histogram reflects the difference in the property of the images. The left peak in Figure 2.2 (B) is incurred not because of crack objects, but the shadow area on the right part of the image instead. The far distance between the two peaks indicating the gray level of the shadow differs greatly from that of the crack objects.

The typical image consisting of objects with approximately the same gray -level differs from the gray -level of the background, which results in the bi-modal of the histogram. The image has two local peaks in its histogram. The objects form one of them while the background forms the other. In this case, it is easy to find the threshold that includes all the object gray distribution, but all of the background pixels fall outside.

[1] Unfortunately, this situation is not common for crack images. The crack pixels only occupy a small percentage of the total pixels. The background has different gray levels. Some of these background pixels may have very close gray levels to the crack objects. Considering the non-uniform lighting conditions, there may be fake peaks in the histogram. From Figure 2.2, we can see that the histogram of the crack image does not appear as the bi-modal. It depends not only on the image objects, but also many other physical conditions. It is almost impossible to determine the correct threshold value directly from the histogram. In addition, the threshold must be determined based on the local image content, which varies at different parts of the image. Therefore, we cannot set a global threshold for the entire image.

2.3.2 Threshold Estimation

As discussed in the previous sections, it is very difficult to precisely determine the threshold. However, it is possible for the threshold to be estimated to detect the crack objects. Obviously, this threshold should follow these facts below:

(1) For 256 gray-level image, the value of threshold constant is within the range [0,255];

(2) When the image grows darker, the average gray level of the image pixels becomes lower. The value of the threshold constant also becomes lower. And vice versa. It is a monotonic relationship

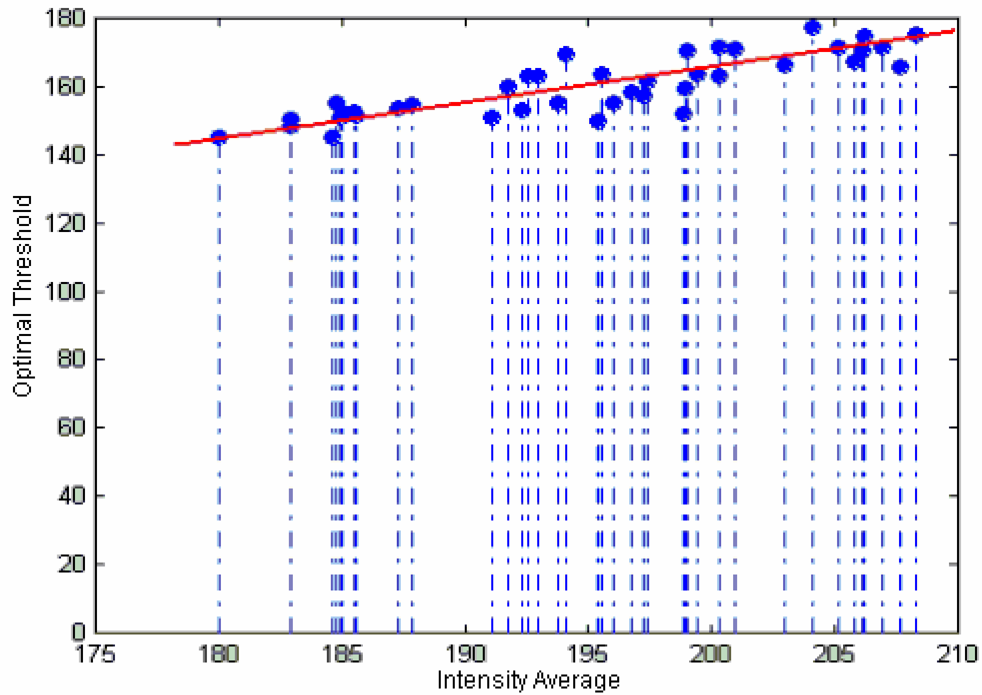


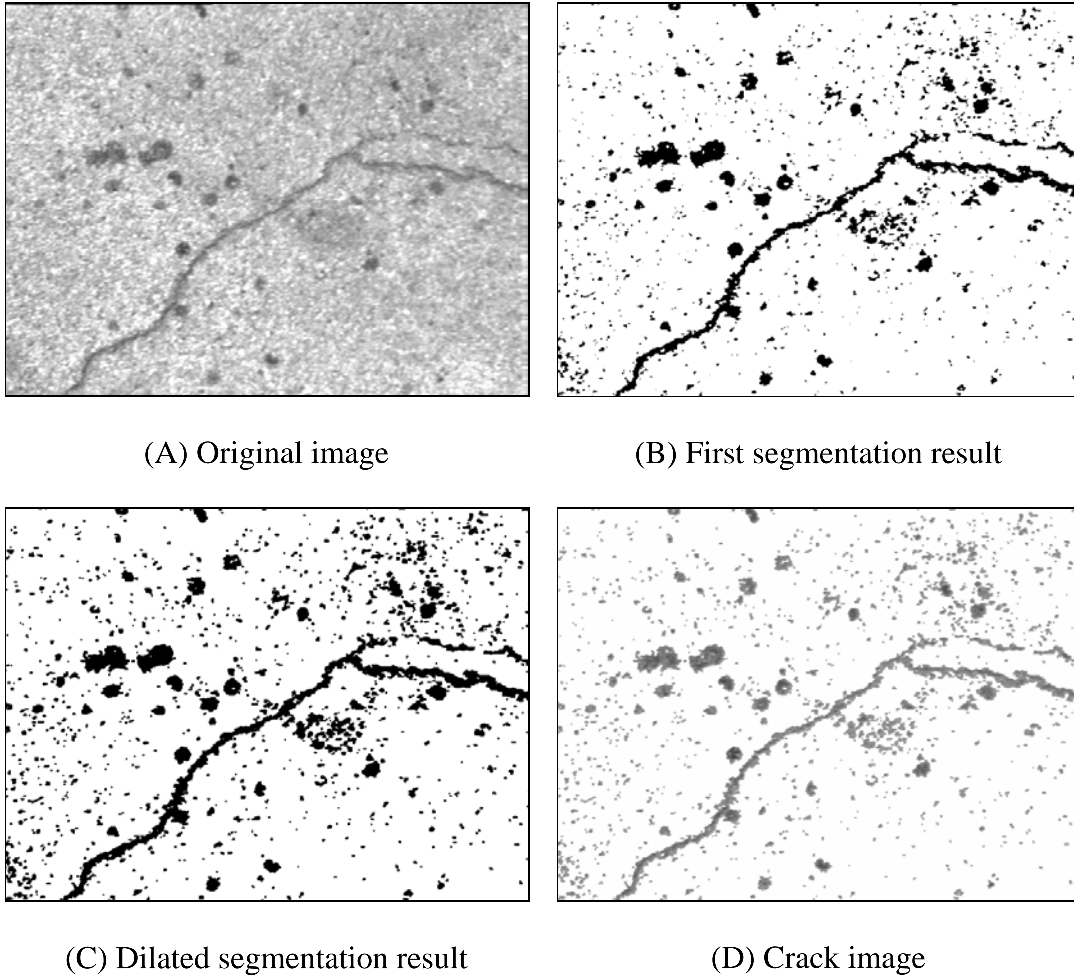
Figure 2.3 Threshold estimation

The average gray level of the image can be a suitable parameter for the threshold estimation. It is obtained by averaging all the elements of the image data matrix. Because there is a monotonic relationship between the threshold and the darkness, we can assume it is simply a linear function, so that:

$$T = k \bullet mean(I) + \Delta \tag{2.3}$$

where T is the estimated threshold and I is the image data matrix. The parameter k and Δ can be estimated by real samples. This is shown in Figure 2.3. The dots in the figure are obtained by averaging a large quantity of samples. The red line is the estimated threshold function with $k=1$, Δ=-32. When this function is applied to segmentation, the result is very good. Figure 2.4 shows the segmentation results of the

image in Figure 2.1(A). The resulting binary image of this segmentation process is not the final result. Instead, it leads to a more specific crack image shown in Figure 2.4(D).



(A) Original image (B) First segmentation result

(C) Dilated segmentation result (D) Crack image

Figure 2.4 Object extraction using the segmentation result

2.3.3 Recurring Threshold Determination

As a result of threshold estimation, a binary image [Figure 2.4 (B)] is obtained by applying the function (2.2) to the original crack image. However, some noise is also included in the result because the threshold is set high, and any pixel whose gray level is below the threshold will be recognized as an object pixel. From this result, it is very difficult to distinguish crack objects from noise. However, we can go back to remove all

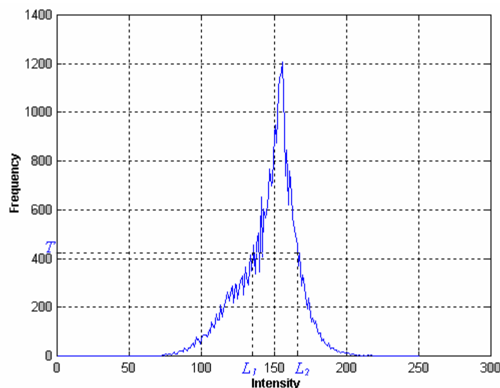
the background pixels from the original image, where the segmentation result works as a mask. As the result, a new crack image can be obtained with most of the background being removed.

To avoid the loss of the crack edge, a dilation operation is applied to the segmentation result in Figure 2.4 (B). For binary images, the dilation adds pixels to the boundaries of the objects in which the pixels have the value of “1.” Therefore, only the objects can be enlarged. The dilation operation uses a specified neighborhood, which is a two- dimensional binary matrix. The neighborhood for a dilation operation can be of an arbitrary shape and size. It is represented by a structure element, which is a matrix consisting of only 0’s and 1’s. The center pixel of the structure element represents the pixel of interest, while the elements in the matrix that are “1” define the neighborhood. The state of any pixel in the output image is determined by applying a rule to the neighborhood of the corresponding pixel in the input image. The rule is defined as follows:

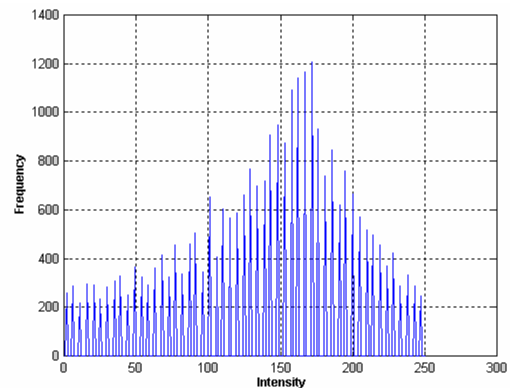
If any pixel in the input pixel’s neighborhood is “1,” the output pixels is “1.” Otherwise, the output pixels is “0”. [5]

Figure 2.4 (C) shows the dilated result of the image in Figure 2.4 (B), and Figure 2.4 (D) shows the result masked from the original image with Figure 2.4 (C). As we can see, most of the background in the original image has been removed. The corresponding histogram will have a major peak for the crack object pixels, which is shown in figure 2.5 (A). There is also a high peak at gray level 255 for the white background, which has been removed in the figure.

When we look back at the original image in Figure 2.4 (A), we can see that not all the pixels of the crack objects are at exactly the same gray level. This is caused by the boundary effect which shows that the border pixels of the crack have a higher gray level than the center pixels of the crack. This makes them look a little brighter and closer to the background. Therefore, the pixels of the crack objects exist in a narrow range of the gray-level histogram. We are not able to determine the threshold simply at the gray level of the maximum point of the histogram. The band thresholding method is implemented instead. In order to do that, further image transformation is needed to get the object pixels in the correct intensity range. The goal of the image intensity transformation is to map an image's intensity value from one range to another. We can increase the contrast of the image and eliminate the noise at different gray levels by expanding a small range of intensity data to the entire intensity range. [5]



(A) Histogram of the masked image



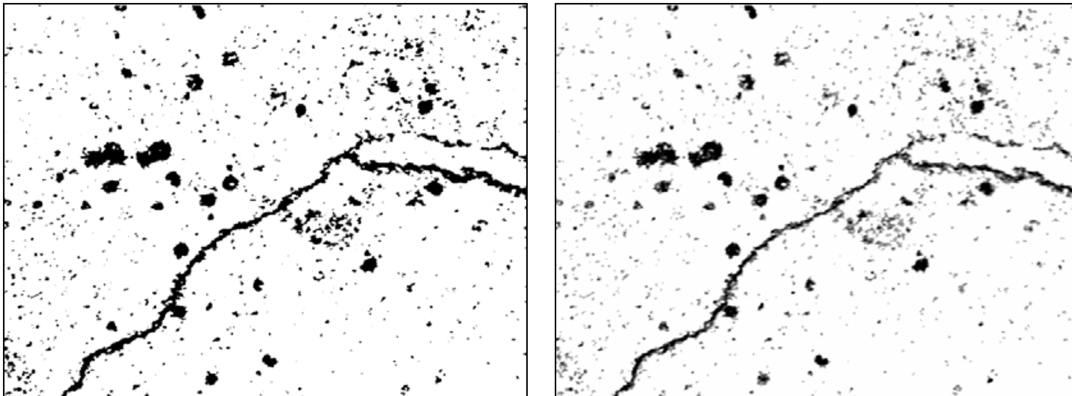
(B) Histogram of the image after intensity transformation

Figure 2.5 Intensity transformation

The source intensity range, which is the bandwidth of the band -thresholding, is chosen so that most crack boundary pixels are included and that the least noise is existing in the result. Figure 2.5 (A) shows how to determine the bandwidth. T is a parameter which varies with the content of the image. Generally, it is defined as:

$$T = \alpha \bullet max(I) \qquad (\alpha<1) \tag{2.4}$$

where I is the histogramand α is the bandwidth coefficient. When α is large, T becomes high. The selected range of intensity transformation becomes narrow, so that some of the boundary pixels will be lost. When α is small, T becomes low. The selected range of intensity transformation becomes wide, so that some of the background pixels will also be included in the result. Therefore, it is very efficient to compensate the boundary effect by adjusting the bandwidth coefficient α.



(A) $T=80\%*max(I)$ (B) $T=50\%*max(I)$

Figure 2.6 Image intensity adjustment

Once α is set, the boundary points of the intensity range L_1 and L_2 are straightforward. Figure 2.5(B) shows the histogram of the image after the intensity

transformation. It is the expansion of the selected portion [L_1, L_2] of the histogram in Figure 2.5(A). Different image transformation results with the given bandwidth coefficient α are shown in Figure 2.6. $\alpha=50\%$ is applied to the algorithm according to the actual result. After the image transformation, the resulting image is still a 256 gray-level intensity image. Since it is assumed that most of the image objects remaining are crack objects, we can set a high threshold to convert it to the binary image as the final segmentation result. Figure 2.7 shows this binary image.

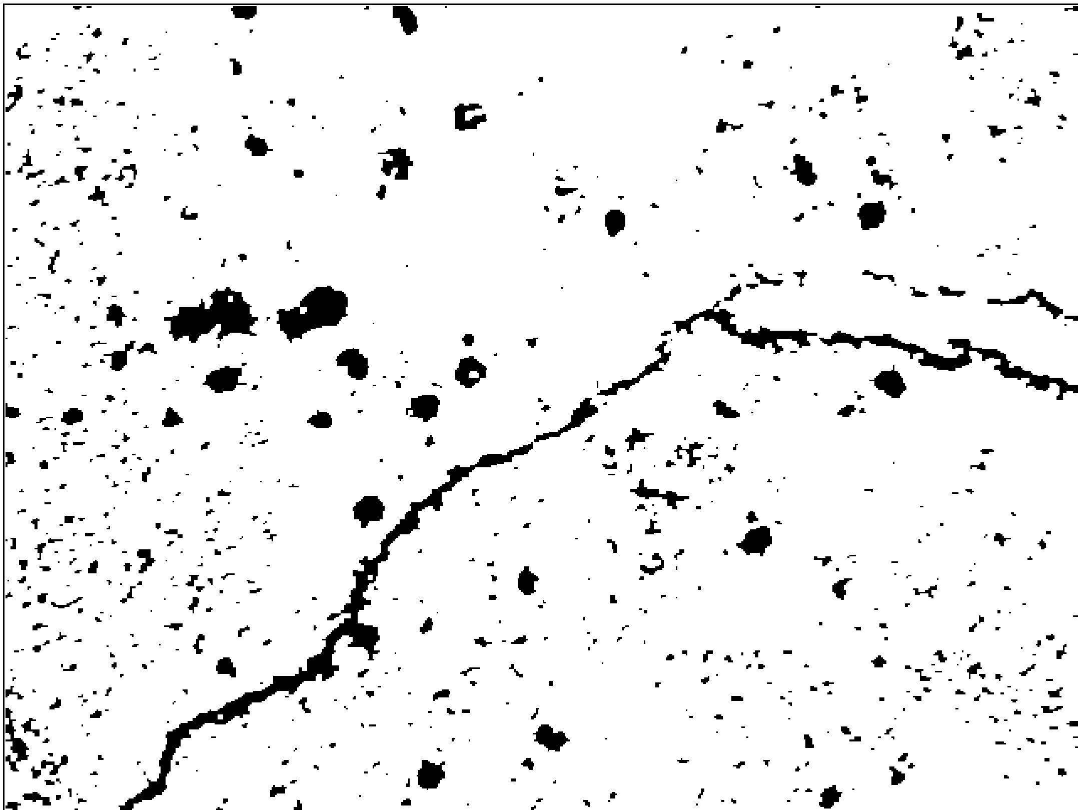
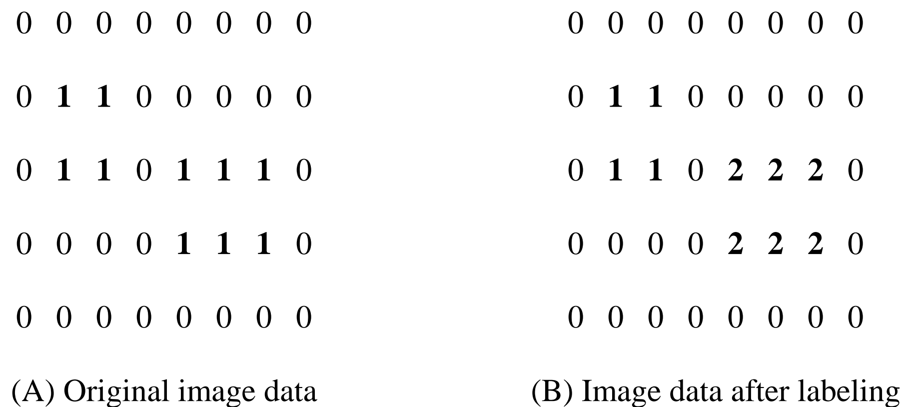


Figure 2.7 Result binary image after segmentation

2.4 Connected-Component Object Identification

As mentioned before, entirely correct and complete segmentation of complex scenes cannot be achieved. It is inevitable to have some noise pixels included in the segmentation result shown in Figure 2.8. These noise pixels cannot be removed during the segmentation because these pixels have gray levels very close to the crack objects. To distinguish these pixels, additional methods must be implemented according to the characteristics of the cracks.

2.4.1 Binary Image Labeling



(A) Original image data (B) Image data after labeling

Figure 2.8 Binary image labeling

In order to distinguish each object in the binary image, it must be isolated from the rest of the image for further analysis. With the binary image-labeling method, the pixels in different connected objects of a binary image are labeled with different integer values. The binary image data is labeled according to different neighborhood relationships that will be discussed in the next chapter. Individual objects can be extracted from the image by selecting the pixels with specific values. Figure 2.8 shows an example of the binary

image labeling. In Figure 2.8 (A), there is a 5X8 binary image data fragment. The pixels with a value of “1” are the object and others are the background. The two separate objects are labeled with “1” and “2” after the binary image-labeling operation, which is shown in Figure 2.8 (B). After this, the first object can be extracted by selecting the pixels with the value of “1,” and the other one by selecting the pixels with “2,”

2.4.2 Connected-Component Object Identification

Segmentation cannot remove the noise pixels with gray levels close to the crack objects. Higher level-image processing is necessary to distinguish the crack objects based on the characteristics of cracks. However the definition of “crack” has much ambiguity, which adds difficulty to this process of discrimination. Two preliminary criteria are set here to distinguish the crack.

(1) The object is not a crack if the number of the object pixels is less than a certain limit N_L.

(2) The object is not a crack if the number of the object pixels is higher than a certain limit N_H.

The first one can be applied to remove the small objects incurred by sand, pebbles, or the texture of the background. The other one can be applied to remove large objects caused by shadow or paint. In my program, $N_L = 25$, and N_H is set as one third of the total number of pixels of the whole image. The final binary crack image is obtained after this step. It is shown in Figure 2.9.

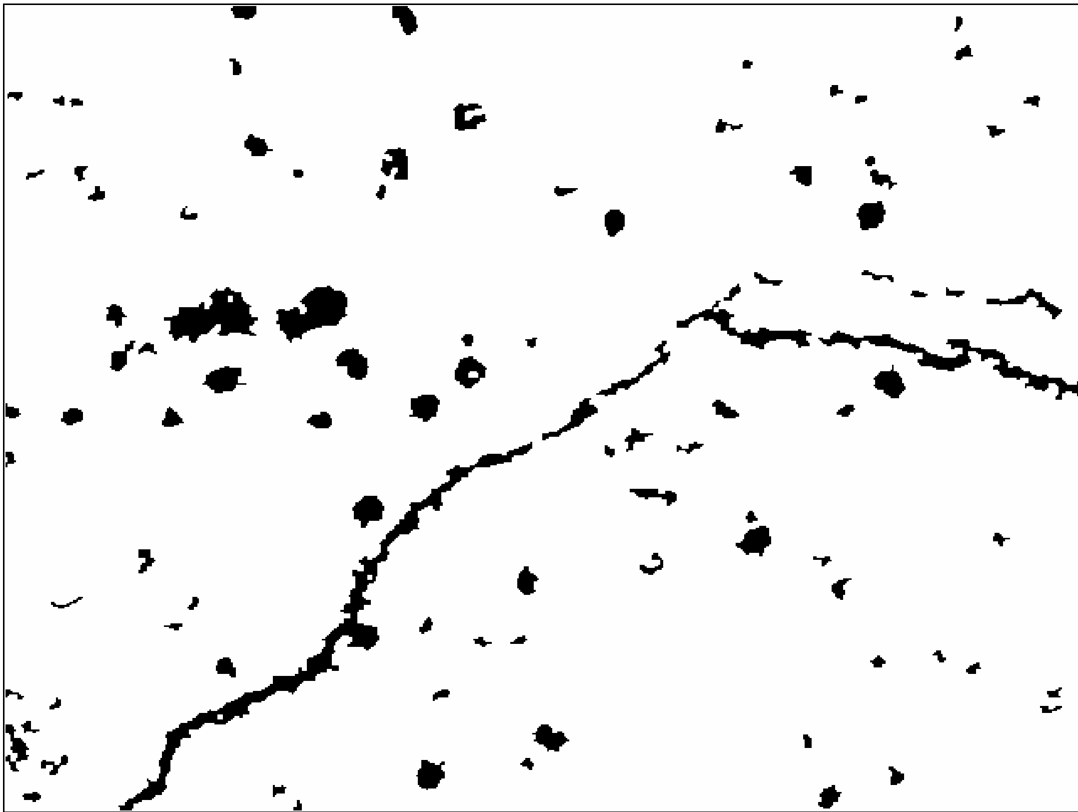


Figure 2.9 Segmentation result

2.5 Distinct Block Operation

To solve the problem of non-uniform property of the road image, the distinct block operation is applied. The original image is divided into distinct blocks. Within the same block, the lighting condition can be regarded as uniform. The processing is performed on each block individually to get the results in the corresponding block of the output image.

The optimal block size can be determined with the image size and the maximum block dimension. Given the image size m-by-n, the maximum row and column

dimensions for the block as k, the optimal block dimension can be determined with the following algorithm. [5]

(1) If m is no greater than k, $k = m$.

(2) If m is greater than k, consider all values between min($m/10$, $k/2$) and k. The optimal column size is the value that minimizes the padding required.

(3) Similar steps for block row size.

For an image of 640-by-480, the optimal block size is 80-by-96.

Chapter 3

Crack Object Characterization Algorithm

3.1 Introduction

After the image segmentation in the previous chapter, a binary image is obtained from the original 256 gray-level image. The image data becomes a discrete set of "1" and "0" (black and white), which represents the crack object (foreground) and background respectively. The crack object can be characterized on the basis of pixel neighborhood relationships to the definition of the discrete set.

3.2 Basic Definitions

In order to define further algorithms to characterize the image object, such as the calculation of length and area, several robust basic definitions are needed. Connectivity and neighborhood of digital images are discussed in this section.

3.2.1 Square Lattice

In the mathematical model of a digital image, the image can be mapped to a set of discrete pixels on a two-dimensional square lattice. The pixel area is defined with the center pixel, leading to the representation of pixels as discrete points in the plane. As shown in Figure 3.1, a square lattice is built with black dots (•) and continuous lines. The dots represents center pixels in the lattice, and the line implies that the two pixels

are connected with each other in the lattice. With this definition, the image pixels become the points on the square lattice.

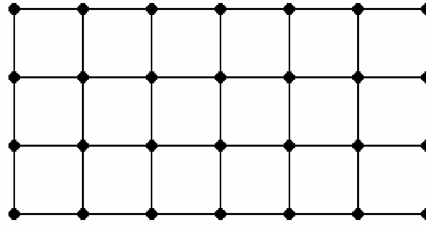


Figure 3.1 Square lattice of the digital image

3.2.2 Neighborhood

The neighborhood is defined by referring to the considered square lattice. With the structure of a square lattice, the 4-connected neighborhood ($N_4(\boldsymbol{O})$) and 8-connected neighborhood ($N_8(\boldsymbol{O})$) are derived, which is shown in Figure 3.2. The 4-connected neighborhood includes the four direct neighbors of the point $\boldsymbol{O}$ in question. The 8-connected neighborhood is completed using the four direct neighbors and other four corner points for a given point $\boldsymbol{O}$.

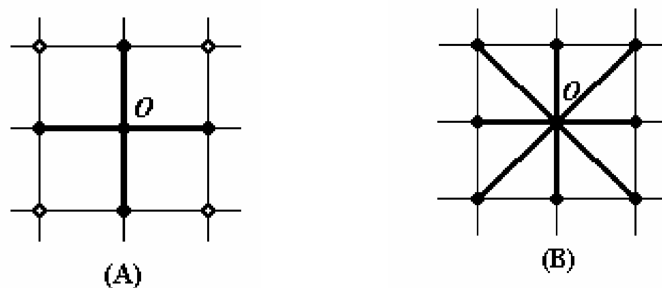


Figure 3.2 Neighborhoods on the square lattice

The definition of the neighborhood sets the foundation for a digital image processing algorithm in this chapter.

3.3 Border of a Crack Object

In the binary image of the cracks, an important subset of pixels in the crack object is the set of the boundary pixels, which separates the crack objects from the background. The border describes the shape, perimeter, area, etc. about the crack object. Further calculation of the crack can be obtained by analyzing these border pixels.

3.3.1 Definition: Border of image object

Given a set of points P, the complement of P denoted as P^C. The 8-connected border of P is the set of points Γ that have at least one 4-connected neighbor in P^C. The 4-connected border of P is the set of points Γ that have at least one 8-connected neighbor in P^C. [2]

The border of the object is defined with a neighborhood relationship. Figure 3.3 illustrates an example of the 4-connected border (B) and 8-connected border (C) for the object in (A).

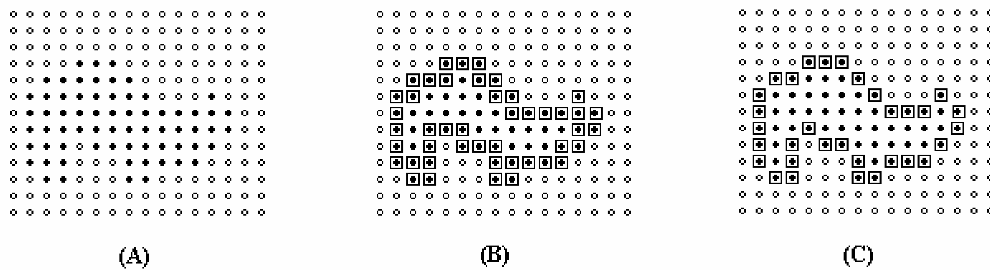


Figure 3.3 Borders of a binary image object

3.3.2 Border Determination

A pixel in a binary image can be determined as a border pixel if it satisfies both of the following criteria [2]:

(1) It is a foreground pixel.

(2) One (or more) of the pixels in its given neighborhood (4- or 8-connected) is a background pixel.

3.3.3 Border Smoothing

Correct border extraction is very important for further calculation. The purpose of the smoothing is to remove the irregularities of the crack objects that will significantly affect the accuracy of the length calculation. The approach is based on the operators from mathematical morphology, which is given by *Minkowski algebra*. These operators have been proven efficient in noise removal on binary images. There are two basic morphological operations, *dilation* and *erosion*, which are defined as follows: [2]

Given a set of pixels F and its border Γ with respect to a neighborhood relationship. For any pixel $p \in F$, $N_D(p)$ *denotes the neighborhood of p.*

(i) The dilation operator dilation(.) applied to F results in the set

$$dilation(F) = F \cup \bigcup_{p \in F} N_D(p) \qquad (3.1)$$

(ii) The erosion operator erosion(.) applied to F results in the set

$$erosion(F) = F \setminus \Gamma \qquad (3.2)$$

Advanced morphological operators *closing* and *opening* can be obtained by combining these basic operators:

The *closing(.)* and *opening(.)* operators are defined by [2]:

$$closing(F)=erosion(dilation(F)) \qquad (3.3)$$

$$opening(F)=dilation(erosion(F)) \qquad (3.4)$$

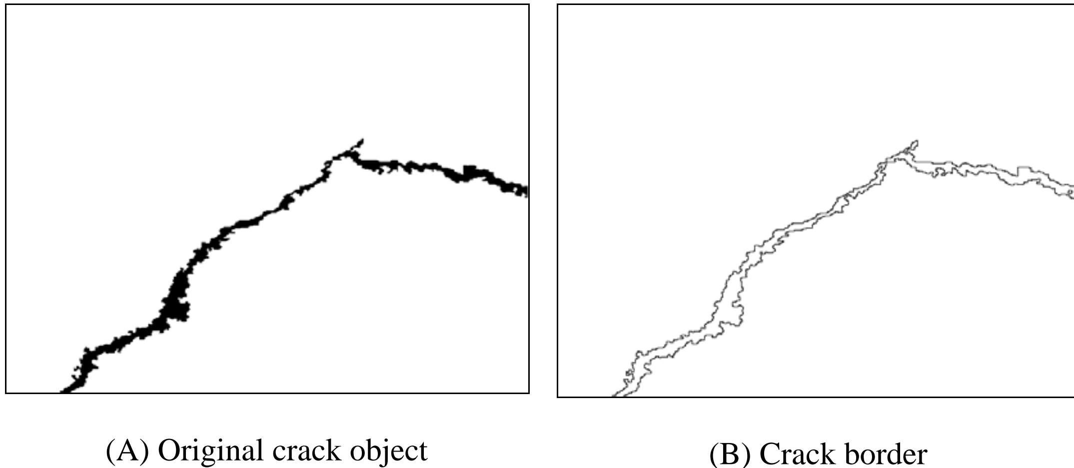
Both *closing* and *opening* operators can be used for binary object smoothing. The closing operator can fill spurious holes, whereas the opening operator smoothes the irregularities by removing spurious components. They can only remove the one-pixel size spurious smoothing. However, different levels of object smoothing can be obtained by using different combinations of dilation and erosion operators. [2] For example, two-pixel sized holes can be filled using the operator *erosion[erosion(dilation[dilation(.)])]*. Similarly, two-pixel sized spurious components can be removed using *dilation[dilation(erosion[erosion(.)])]*. According to the binary result of segmentation, a three-pixel level of smoothing is applied to the processing here. To avoid the possible image loss caused by the erosion on the image border (three-pixel depth), the object pixels within the three-pixel range of the image border have been preserved before the border smoothing operation and restored after the smoothing process.

3.3.3 Crack Object Dilation

Since most of the cracks are very long and slim, direct border extraction will break the close arc of the crack object border. A dilation operation, discussed in the previous chapter, is applied prior to the border extraction. Furthermore, the dilation operation can also remove some of the spurious noise pixels of the crack objects.

The error incurred by this dilation for the further calculation is limited because the dilation can only slightly increase the perimeter of the crack. The matrix used in the dilation is a 2-by-2 matrix with all “1” element, for the smallest dilation operation;

however, it is enough to preserve the close border arc. Figure 3.4 shows the border extraction with dilation.



(A) Original crack object (B) Crack border

Figure 3.4 Border extraction

3.4 Crack Length Calculation

When the crack's border is extracted, it appears as a close arc in the partition. The length of the crack can be computed from its perimeter. Most cracks have very large length-to-width ratios. Half of the perimeter can be a good approximation to its length.

3.4.1 Move and Move Length

A move on the lattice is the displacement from a point to one of its neighbors. A move length is the value of the local distance between a point and its neighbors.

According to the definition above, we can see that there is only one type of move in a 4-connected neighborhood, defined as Δ for a unit length. For the 8-connected

neighborhood, the moves are Δ_1 and Δ_2, which represent the distance from the center point to the direct 4 neighbors and to the 4 corner points.

3.4.2 Euclidean Distance

Euclidean distance is the basic precise definition of distance between two points in a two-dimensional plane. It is defined as follows: [2]

Given two points $P(x_p, y_p)$ and $Q(x_q, y_q)$, the Euclidean distance of PQ is defined by

$$d_E(P,Q) = \sqrt{(x_q - x_p)^2 + (y_q - y_p)^2} \quad (3.5)$$

Other discrete distances are nothing but approximations of the Euclidean distance.

3.4.3 Arc Length and Discrete Distance

The length of a digital arc is the sum of the length of all the moves that compose it. The discrete distance between two points P and Q is the length of the shortest digital arc from P to Q. [2]

The definition above leads to the 4-connected discrete distance denoted *as* $d_4(P, Q)$ and 8-connected discrete distance denoted *as* $d8(P, Q)$. $d_4(P, Q)$ is the length of the shortest 4-connected arc, and $d_8(P, Q)$ is the length of the shortest 8-connected arc. Figure 3.5 shows the difference of $d_4(P, Q)$ and $d_8(P, Q)$ between P and Q. Obviously, $d_8(P, Q)$ is a better approximation of the actual length than $d_4(P, Q)$. The approximation error will be discussed later.

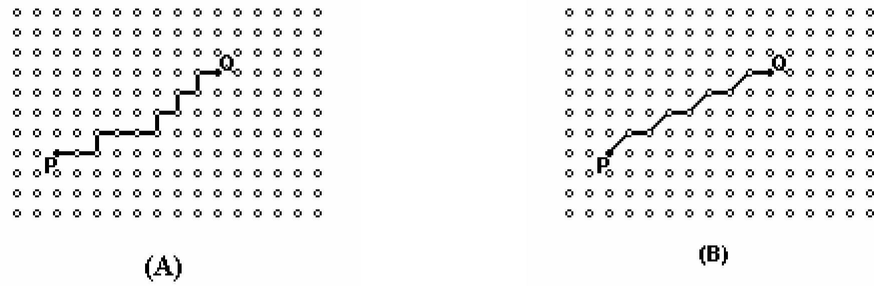


Figure 3.5 Discrete distance: (A) $d_4(P, Q)$ (B) $d_8(P, Q)$

3.4.4 Crack Length Calculation Algorithm

The 8-connected neighborhood is used in the length calculation. Based on the discussion in the previous section, the length is computed by summing up all the move lengths of the close arc of the crack border. In most cases, there are dozens of crack objects in one image and each crack may be a collection of thousands of pixels. A fast algorithm is desired even for the fastest computer.

Since there are only two types of moves for the 8-connected neighborhood, all we need to do is to find out how many pixels on the border have the move of Δ_1 and the other will have the move of Δ_2. It is easy to count the total number of border pixels. Given the total number of border pixels N_8, K pixels has the move of Δ_1, the length of the curve L is

$$L = K \bullet \Delta_1 + (N_8 - K) \bullet \Delta_2 \qquad (3.6)$$

To simplify the calculation, we can also start with the 4-connected borders. There is only one type of move Δ in a 4-connected neighborhood, with Δ equal to the Δ_1 of the 8-connected border. For any Δ_2 move on the 8-connected border, it is associated with two

turning pixels on the 4-connected border. The turning point can be determined by the fact that the two closest neighbors are not on a straight line. Figure 3.6 shows these turning pixels. From the figure, for the arc from pixel **A** to **B**, the 4-connected arc length is $4\Delta(4\Delta_1)$, and the 8-connected arc length is $2\Delta_1 + \Delta_2$. Therefore, for each turning point on the 4-connected arc, the length adjustment will be $\frac{2\Delta_1 - \Delta_2}{2}$. Therefore, the equation (3.2) becomes:

$$L = N_4 \bullet \Delta_1 - K \bullet \frac{2\Delta_1 - \Delta_2}{2} \tag{3.7}$$

where N_4 is the total number of pixels and K is the number of turning pixels on the 4-connected border.

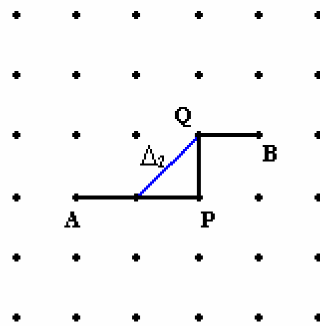


Figure 3.6 Turning pixels

The remaining issue is to find out these turning points in a fast way. Sorting pixel by pixel is not a good idea, because it will be too tedious with so many pixels existing in a crack object.

The Border Convolution method is introduced to solve this problem. This method is based on matrix processing. With modern computer technologies, matrix computation

becomes easier and faster. New mathematical software tools are also designed to apply to this trend, such as MATLAB and MATHCAD.

The two-dimensional convolution is a neighborhood operation. The value of an output pixel is computed by multiplying elements of two matrices and summing for the results. One of the matrices represents the image data, and the other matrix is known as a convolution kernel. Let *I* be the binary image data matrix. The convolution is:

$$I^{'} = I \otimes SE \tag{3.8}$$

where *SE* is the convolution kernel. To find out the turning pixels, two convolution kernels are defined as:

$$SE_1 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \qquad SE_2 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{3.9}$$

In the resulting image data matrix $I^{'}{}_1$ and $I^{'}{}_2$, the element has the value of 3 for all non-turning pixels. Counting on the data in matrix $I^{'}{}_1$ and $I^{'}{}_2$, it is easy to get the exact number of the turning pixels on a 4-connected border. The length of the border is then straightforward with equation (3.4).

Integer values 3 and 4 are assigned to move length Δ_1 and Δ_2 for better computer preservation and higher processing speed. In this case, the length of the diagonal move $\sqrt{2}$ is approximated by $\frac{4}{3}$.

3.4.5 Approximation Errors

Based on the model set up in the proceeding discussions, the continuous concepts such as continuity and distance have been mapped onto discrete space as the concepts of

connectivity and discrete length. It is important to find out the approximation error made during this process.

The relative error E_D is defined with the given discrete length d_D and the Euclidean distance d_E between two points *P* and *Q* as [2]

$$E_D(P,Q) = \frac{(1/\varepsilon)d_D(P,Q) - d_E(P,Q)}{d_E(P,Q)} = \frac{1}{\varepsilon}(\frac{d_D(P,Q)}{d_E(P,Q)}) - 1 \quad (3.10)$$

The parameter ε ($\varepsilon > 0$) is the scale factor used to maintain consistency between radii of discrete length and Euclidean distance. In the 8-connected neighborhood, let $\varepsilon = \Delta_1$.

Let $d_D = d_{\Delta_1,\Delta_2}$ and point *P* be the origin. Considering the first octant in Figure 3.7, we see that

$$d_{\Delta_1,\Delta_2} = (x_Q - y_Q)\Delta_1 + y_Q\Delta_2 \quad (3.11)$$

The error E_D is measured along the line (*x=L*) with (*L>0*). Therefore, for all *Q* that $x_Q = L$ and $0 \le y_Q \le L$, the relative error at point *Q* is given by [2]

$$E_D(P,Q) = \frac{(L - y_Q)\Delta_1 + y_Q\Delta_2}{\varepsilon\sqrt{L^2 + y_Q^2}} - 1 \quad (3.12)$$

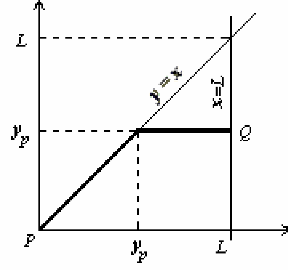


Figure 3.7 Calculation of d_D in the first octant

Since E_D is a convex function in $[0, L]$, its local extreme can be obtained so that $\frac{\partial E_D(P,Q)}{\partial y} = 0.$

$$\frac{\partial E_D(P,Q)}{\partial y} = \frac{1}{\varepsilon\sqrt{L^2 + y_Q^2}}\left((\Delta_2 - \Delta_1) - \frac{((L - y_Q)\Delta_1 + y_Q\Delta_2)y_Q}{L^2 + y_Q^2}\right). = 0 \quad (3.13)$$

In this case, $E_D(P,Q) = \frac{\sqrt{\Delta_1^2 + (\Delta_2 - \Delta_1)^2}}{\varepsilon} - 1$, where $y_Q = \frac{\Delta_2 - \Delta_1}{\Delta_1} L$. The two bound points at $P(0,0)$ and $Q(0,L)$ should also be considered as the possible peak point. Therefore, the maximum relative error is defined as the point that $E_D^{MAX}(P,Q)$ =max{$|E_D(P,Q)|$, E_D^P, E_D^Q }.

Recall (3.13), $E_D^{MAX}(P,Q) = \max\left(\left|\frac{\Delta_1}{\varepsilon} - 1\right|, \left|\frac{\sqrt{\Delta_1^2 + (\Delta_2 - \Delta_1)^2}}{\varepsilon} - 1\right|, \left|\frac{\Delta_2}{\varepsilon\sqrt{2}} - 1\right|\right).$ As Δ_1 =3 and Δ_2 =4, the maximal relative error is

$$E_{3,4}^{MAX} = \left|\frac{4}{3\sqrt{2}} - 1\right| \approx 5.73\% \quad (3.14)$$

3.5 Crack Width Classification

3.5.1 Area Calculation

A good estimation for the area of the crack object in binary images is found by summing the areas of each pixel. Let's define the *on* pixel in the binary image, which represents the pixel with the value of 1. The area of an individual pixel is determined by looking at its four 2-by-2 neighborhoods. There are six different distinguishable patterns for a 2-by-2 neighborhood, each representing a different area: [6]

- Zero on pixels (area=0)
- One on pixel (area=1/4)
- Two adjacent on pixels (area=1/2)
- Two diagonals on pixels (area=3/4)
- Three on pixels (area=7/8)
- All four on pixels (area=1)

Because each pixel is part of four different 2-by-2 neighborhoods, a single "1" pixel surrounded by "0" pixels has a total area of 1.

3.5.2 Width Calculation

As the area and the length of the crack object are both obtained, its average width is computed by dividing the area by its length. There is an important step necessary to isolate each crack object and do the calculation individually. The connected-component labeling and selection methods are used, which have been discussed in the previous chapter.

So far, we suppose that all the existing objects in the resulting binary image are cracks we are interested in. In fact, since we have used gray-level segmentation, which distinguishes the crack objects based on their gray value difference from the background, additional higher-level processing methods are needed to distinguish the crack objects. Certain criteria are necessary to remove noise at this final step. Three additional criteria are set for the crack discrimination:

(1) Width limit: No object is a crack object when its width is larger than this limit.

(2) Length limit: No object is a crack object when its length is smaller than this limit.

(3) Length-to-width ratio limit: No object is a crack object when its length-to-width-ratio is smaller than this limit.

The actual parameter is obtained from a large quantity of real examples. The width limit is set as 20 pixels, the length limit as 10 pixels, and the length-to-width ratio limit as 3. Figure 3.8 shows the final result image of the cracks.

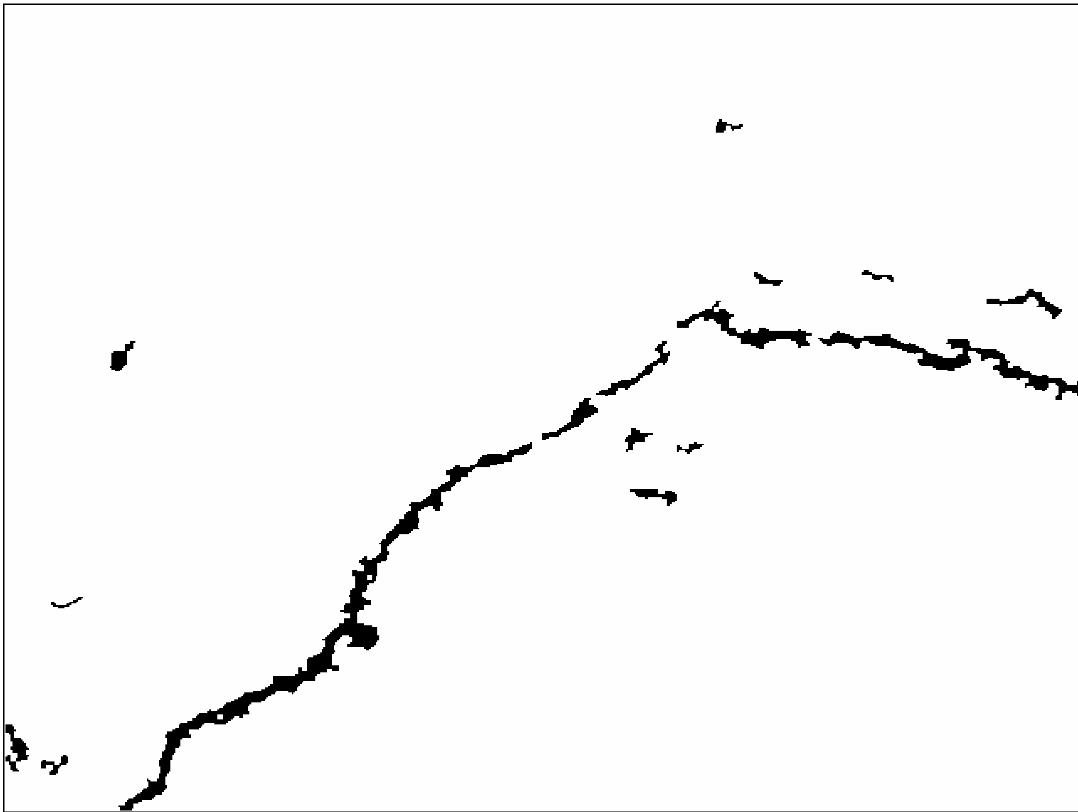
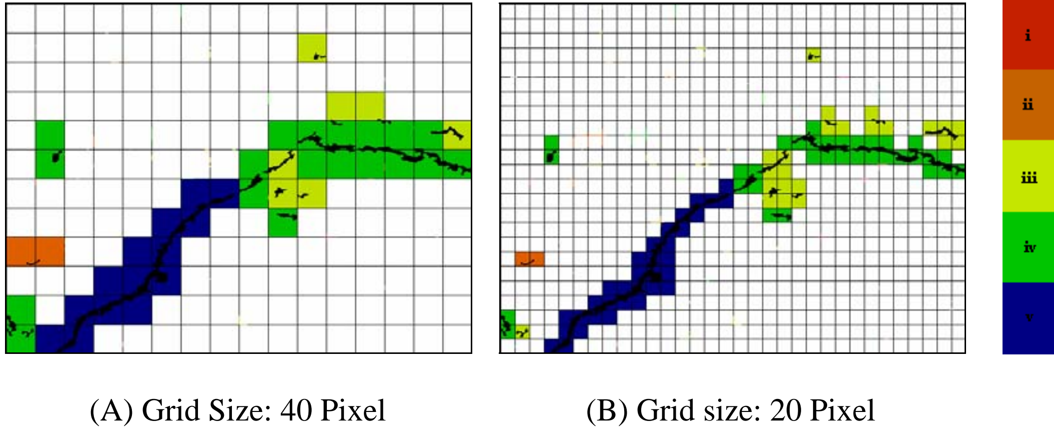


Figure 3.8 Processing result

3.5.4 Grid Classification

In the crack characterization, we are not only interested in the properties of an individual crack but, more importantly, in the overall percentage of the crack coverage on the road. A grid format image is created for this purpose. The whole image is divided into small identical grids. The grid size can be user-defined. According to the width, all the cracks are classified into five categories. The range of the crack width increases from category one to five. The grids are classified by the widest crack inside, and filled with a specific color. With the grid classification, we can get the visualized information of the crack coverage, which is very important to determine the quality of the pavement.

Figure 3.8 shows two examples of the grid image on a same result crack image. It is easily seen that the result is more accurate with smaller grid size.



(A) Grid Size: 40 Pixel (B) Grid size: 20 Pixel

Figure 3.8 Image Grids

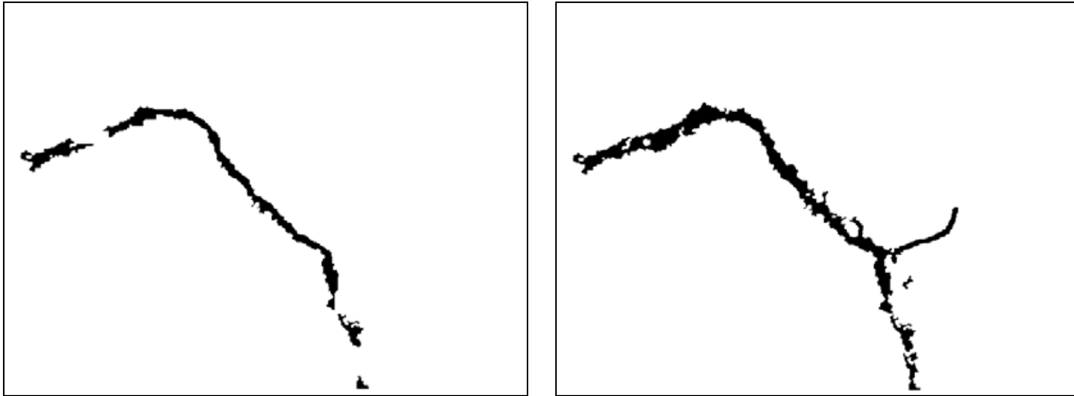
3.6 Crack Growth

Crack growth is a very important issue of automatic crack monitoring. It can be determined by comparing different processed result images of the same location. Since the images are binary images, the difference can be easily obtained by an XOR ($\oplus$) operation. However, in most of the cases, the two images cannot be perfectly overlapped. There is shifting or even rotation between the two images. To solve this problem, the theory of correlation is applied. The correlation coefficient r of image A and B is defined as

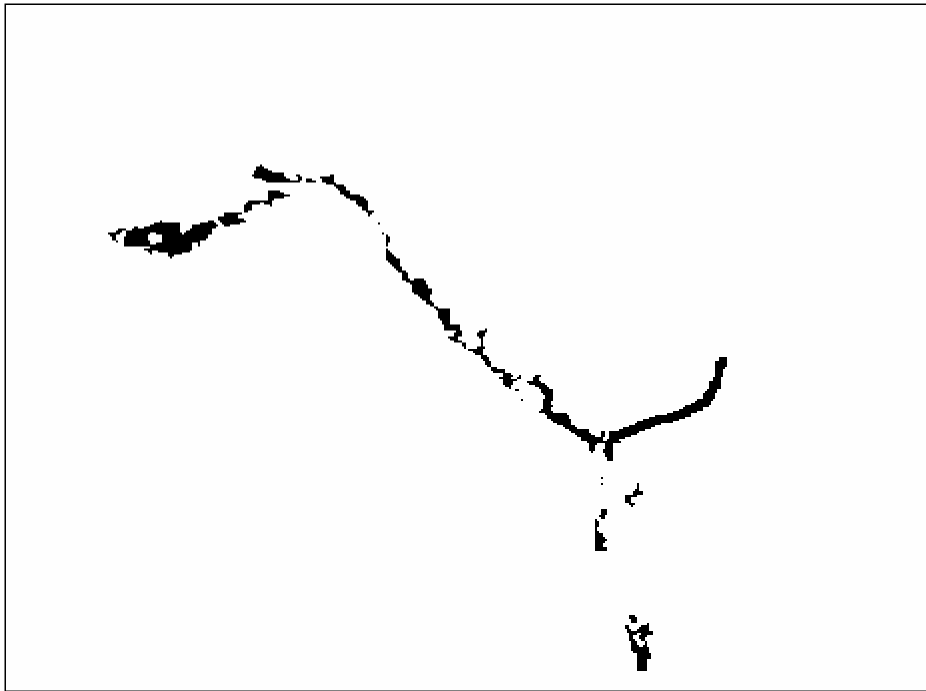
$$r = \frac{\sum_m \sum_n (A_{mn} - \overline{A})(B_{mn} - \overline{B})}{\sqrt{(\sum_m \sum_n (A_{mn} - \overline{A})^2)(\sum_m \sum_n (B_{mn} - \overline{B})^2)}} \tag{3.15}$$

The equation above can be simplified as (3.16) if A and B are binary images.

$$r = \sum_{m} \sum_{n} A \oplus B \qquad (3.16)$$



(A) Before (B) After



(C) Crack Growth

Figure 3.9 Crack growth detection

Considering the arbitrary 2-dimensional displacement between *A* and *B*, the correlation coefficient is calculated in a loop. The most overlapping occurs when the coefficient r reaches the maximum. When the laser pointer is used for the positioning system, the precision can be under 10 mm. Compared with an image size of 1.2-by-1.2 meters, the image rotation can be neglected. Figure 3.9 shows an example of crack-growth detection. The horizontal displacement between (A) and (B) is 14 pixel/26mm, and the vertical displacement is 11pixel/21 mm.

3.7 Calibration

So far, all the calculations are achieved with the unit of *pixel*. However, the physical length and width is required in practice. It could be computed based on the parameters such as the height and the angle of the image-acquisition devices. However, the most direct method is the system calibration. The basic idea is that we get the calibration parameter based on the processing result of an object whose parameters are already known. This calibration parameter is the ratio of the processing result to the physical size of the object.

A special calibration board is designed for this purpose. On this board, there are several equally-spaced lines with the same length. These line objects are long and dark so that they are very easy to distinguish. They have the same length for the object identification. When we obtain more than two objects in the processing result that have very close values, they are recognized as the calibration objects and the average value is used to calculate the calibration parameter. Or, they will be neglected as noise objects to

avoid error. Each time the calibration parameter is reserved in the system so that it can be reused after the current session. The user can do the calibration any time to obtain the most recent parameter for measurement.

Chapter 4

Illustrative Examples

4.1 Example I

This example is designed to test the sensitivity of the crack-detection algorithm and the accuracy of the crack-calculation algorithm. The target is a board with several lines that have different lengths and widths. Figure 4.1 shows the acquired image of this board.

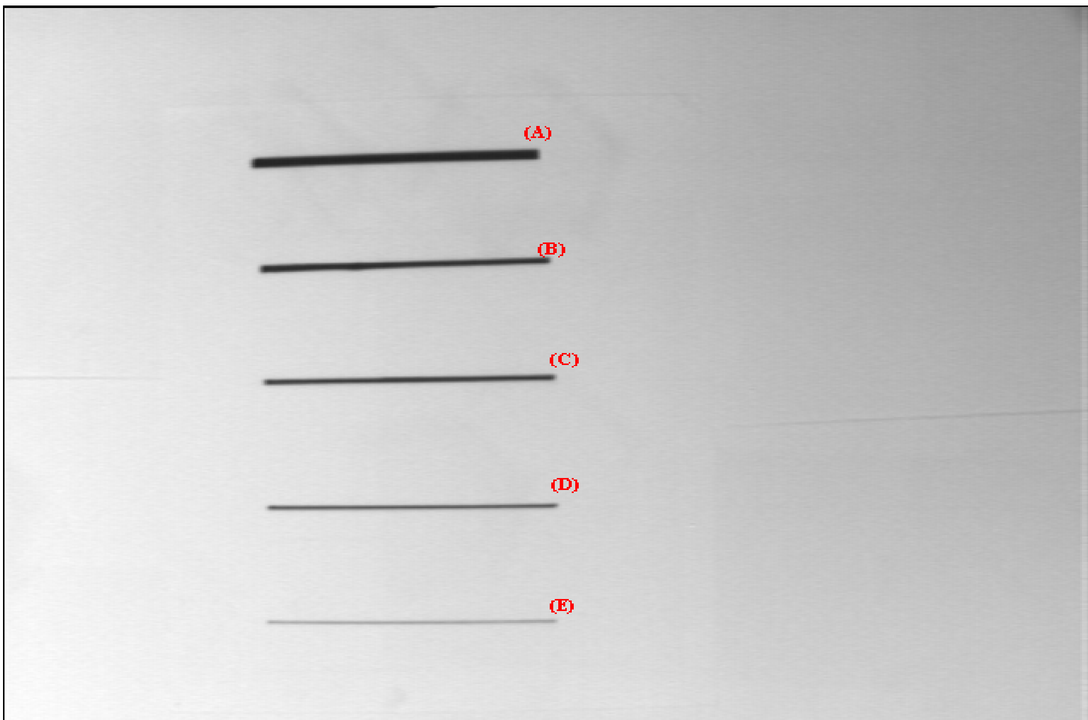


Figure 4.1 Example I : Source image

This image is taken inside the lab, with a general light source. Compared with natural sunlight, it is relatively weak and non-uniform. The camera is placed at the height of about a meter, which is able to cover the area of about 1.2 meters by 1.2 meters on the ground. From figure 4.1, we can see that the light extraction is considerable, especially for small and slim objects like (D) and (E). The parameters of the objects from (A) to (E) are given in Table 4.1.

	(A)	**(B)**	**(C)**	**(D)**	**(E)**
Width (mm)	12.1	7.5	6.2	3.5	1.7
Length (mm)	295	293	293	293	292

Table 4.1 Parameters of the Image Objects

Figure 4.2 shows the processing result of the binary crack image and Table 4.2 gives the calculation result for width and length of the objects.

(B) Binary output (B) Overlapped output

Figure 4.2 Example I: Processing result

		(A)	(B)	(C)	(D)	(E)
Width	**Actual (mm)**	12.1	7.5	6.2	3.5	1.7
	Result (mm)	11.9	7.4	7.0	5.2	3.2
	Error (%)	1.65%	1.33%	12.9%	48.6%	88.2%
Length	**Actual (mm)**	295	293	293	293	292
	Result (mm)	291	288	284	268	264
	Error (%)	1.36%	1.71%	3.1%	8.87%	9.90%

Table 4.2 Calculation results

To further verify the accuracy of the algorithm, the source image is rotated by a 45° angle. Table 4.3 shows the calculation results for this image.

		(A)	(B)	(C)	(D)	(E)
Width	**Actual (mm)**	12.1	7.5	6.2	3.5	1.7
	Result (mm)	11.5	7.0	7.0	4.8	3.0
	Error (%)	4.96%	6.67%	12.9%	37.1%	76.5%
Length	**Actual(mm)**	295	293	293	293	292
	Result (mm)	288	289	286	280	284
	Error (%)	2.37%	1.37%	2.39%	4.44%	2.74%

Table 4.2 Calculation results of the rotated image

From the results, we can see that the algorithm is very efficient in detecting the crack objects, even when the light condition is poor and the objects are very slim. The length-calculation algorithm is also proven to match with the object very well. The maximum error is less than 10%, which is a special case caused by the distinct block operation in Chapter 2. When we look at the overlapped result in Figure 4.2 (C), we can see that the start and end of the lines (D) and (E) are cut off, because that's exactly the border of the sub-image divided from the whole image when the distinct block operation is applied. In such sub-images, the object is so small that it is removed as noise. This problem can be corrected by overlapping with the distinct block operation.

The width calculation is also effective when the object width is above 5mm under the current testing environment. When the width is below 5mm, the calculation is limited by the light conditions and the resolution of the camera. The light has the major impact on the image segmentation. The boundary effect of the image object is great because of the flat reflection of the light and the sensitivity of the CCD camera. In the source image, the gray value varies greatly from the border to the center of the line objects.

The two results from the images with different angles are well matched with each other, which proves the effectiveness of the whole algorithm of calculation.

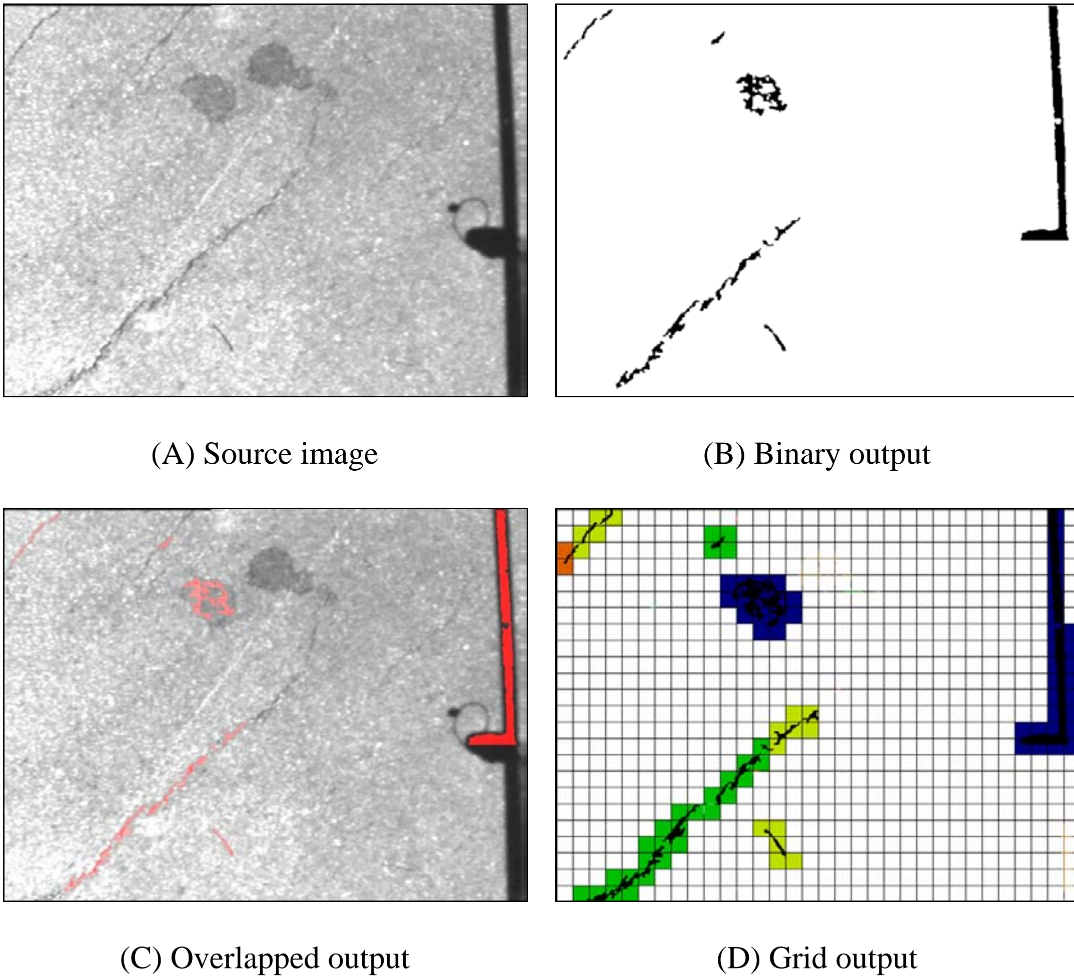
To improve the lighting situation, an artificial lighting system is in design to provide the optimal lighting condition. The camera flashlight can be a solution. It can provide strong light, and avoid heavy equipment of the regular light system at the same time. There are a couple of pins on the flashlight circuit. It is triggered once these pins are

short-circuited. The flashlight control signal can be sent through the RS232 port of the computer. However, the timing is an important issue for control. Additional driving circuit is needed.

Considering the camera resolution of 640-by-480 pixel, and the fact that it covers the area of 1.2-by-1.2 meters, it is easy to estimate the calibration factor by dividing the image size with the camera resolution. Therefore, each pixel in the processing result is compared to the actual length of about 1.87mm. In this experiment, the factor is calculated as 1.74mm by calibration, which is also the highest resolution of the algorithm.

4.2 Example II

Experiment II demonstrates the performance of image processing in the real environment. Figure 4.4 (A) shows the source image acquired on the real highway pavement. The binary output, overlapped output and grid output images are shown in (B), (C) and (D) respectively.



(A) Source image (B) Binary output

(C) Overlapped output (D) Grid output

Figure 4.3 Example II

With the grid output image in Figure 4.4 (D), the information about the crack distribution is obtained, which is shown in Table 4.3. In the result, most of the crack objects are successfully detected with different light conditions. Some of the noise objects, such as the paint and shadow, are removed. The calculation results match the real objects. Table 4.3 shows the statistical result of the crack coverage of Example II.

	Class I (0~3mm)	**Class II (3~5mm)**	**Class III (5~7mm)**	**Class IV (7~10mm)**	**Class V (>10mm)**
Road coverage (%)	0	0.2	2	3.6	5.4
Total length (mm)	0	89	304	545	832

Table 4.3 Crack coverage of example II

Chapter 5

Software Design

5.1 Introduction

The system application software is programmed with Microsoft Fundamental Class (MFC) on the operation system of Windows NT 4.0. To control the external hardware and process the acquired image in one system, the system software design is very complicated. There are five major components:

(1) Windows framework of views and documents,

(2) MATLAB image processing,

(3) CCD camera driver,

(4) Motion system control ,

(5) System database management.

Several advanced Windows programming techniques, such as multiple view and multithread are implemented in the software design.

5.2 Windows Framework

The system application has a user-friendly interface. Figure 1.4 in Chapter 1 shows the frame of the program. It implements the splitter windows and double-view-single-document architecture. The left pane shows the original image and the right pane shows

the result image after processing. Drawing and erasing functions enable users to modify the processing result and re-process the crack image.

5.2.1 The Splitter Windows

The splitter window appears as a special type of frame that holds multiple views in panes. The user can move the splitter bar to adjust the relative size of the panes.

Programming with Microsoft Foundation Class (MFC), the framework of the splitter window application project can be created using the Visual C++ Application Wizard. An object of the MFC class *CSplitterWnd* represents the splitter window. It is a data member of the *CChildFrame* class. The *CChildFrame* class also has to override the virtual member function of *OnCreateClient* to create the splitter window, and the splitter window creates the views. The function of *OnCreateClient* is shown below.

```
BOOL CChildFrame::OnCreateClient(LPCREATESTRUCT lpcs,
    CCreateContext* pContext)
{
    // Create static splitter
    if (!m_wndSplitter.CreateStatic(this,1,2,WS_CHILD))
    {
                return FALSE;
    }
    if (!m_wndSplitter.CreateView(0, 0, RUNTIME_CLASS(CCrackView),
            CSize(0, 0), pContext))
    {
```

```
            return FALSE;
        }
        if (!m_wndSplitter.CreateView(0, 1, RUNTIME_CLASS(CCrackView),
             CSize(0, 0), pContext))
        {
            return FALSE;
        }
        m_bSplitter = TRUE;
        m_wndSplitter.ShowWindow(SW_SHOWNORMAL);
        m_wndSplitter.UpdateWindow();
        return TRUE;
    }
```

5.2.2 The *CCrackObj* Class

The HCMS software application enables the user to modify the image processing result and recalculate the crack information. Any drawing or erasing that the user makes is an object of the class *CCrackObj*. This class is derived from the MFC base class *CObject*. It has the member variables to describe the properties of the object of the user modification, such as crack width, crack color, the position of the object located, as well as the view and document object pointer that the object belongs to.

Whether the user draws or erases, a line object is created to store such information. Another class *CCrackStroke* is derived from the *CCrackObj*, which includes most of the lower level operations on such line objects. It has a *CArray* member variable that stores

coordinates of all the points on the line trail. The member function *AddPoints*() adds new points to the array whenever the line object extends. Together with the properties in its *CCrackObj* base class, the user is able to modify the image of the processing result under precise control.

5.2.3 The Document and View

By using the splitter window, there are more than one view objects existing in the frame. The HCMS software application implements the double-view-single-document architecture for the document and view relation. Two MFC functions are helpful for the object of the document class to iterate through the multiple views: *CDocument:: GetFirstViewPosition*() and *CDocument::GetNextView*(). The first function returns a POSITION value for the first view in the system. The *GetNextView*() function takes this value to retrieve the view object pointer. The following piece of code is an example of how to iterate through the views and repaint.

```
//To get the first view in the list of views:
POSITION pos = GetFirstViewPosition();
CView* pFirstView = GetNextView(pos );
// This example uses CDocument::GetFirstViewPosition
// and GetNextView to repaint each view.
void CMyDoc::OnRepaintAllViews()
{
   POSITION pos = GetFirstViewPosition();
   while (pos != NULL)
   {
```

```
            CView* pView = GetNextView(pos);
            pView->UpdateWindow();
        }
    }
```

There are also some graphic functions that enable the user to modify the processing result image. The modification is under perfect control by drawing, erasing and line width definition. The system is able to update the processing result in a Windows bitmap image, reprocess the image and update the crack statistical information of this image.

5.3 MATLAB Interface

All the image-processing code is written with the MATLAB Image Processing toolbox. It processes the 256 gray-level intensity images acquired by the CCD camera. A binary crack image, an overlapped image, and a color-grid image are created as the outputs. Since we use the latest version 5.3 of MATLAB, which is not accompanied with a C++ compiler, a special piece of MATLAB interface code is designed to implement the MATLAB source code in the C++ program. The detailed procedure is listed below.

- Start a MATLAB engine session: ***Engine *engOpen(const char *startcmd)***. "stmartcmd" must be NULL for Windows. A pointer to the engine handle is returned.
- Execute a MATLAB program piece: ***int engEvalString(Engine *ep, const char *string)***. The "string" should be a valid MATLAB command to be executed. Zero is returned if MATLAB runs successfully.

- Quit a MATLAB engine session: ***int engClose(Engine *ep)***. It returns "0" on success and "1" otherwise.

In addition, the MATLAB C++ interface header file "***engine.h***", which can be found in the MATLAB system directory, must be included in the C++ project. The user also needs to generate an external library file to control the MATLAB engine import and export. For details, please refer to the MATLAB reference book. The MATLAB image processing code is listed in Appendix I.

5.4 CCD Camera Control

The CCD camera has its own API functions for implementation. All the camera operations are encapsulated into a camera class. In the camera initialization, about 600KB memory is allocated to store two frames of the 640-by-480 image data, so that the animated motion pictures can be acquired by refreshing the content of the two frames consecutively. The memory will be released when the camera event ends.

Since the camera requires the system to allocate a large block of memory and it works at a relatively lower speed, multithread programming is suitable for the camera operation. The camera thread is started by initializing this pointer, and the MFC function *afxBeginThread*() is called. The camera is suspended by calling the MFC function *afxSuspendThread*(). The thread can only be terminated by calling the *afxEndThread*() inside the thread. To do so, generally we can set an EndThread event, which will trigger the function to destroy the thread itself. The header file of this camera thread is listed in Appendix II.

5.5 Motion System Control

The motion system is controlled through the ADR101 RS232 interface board that is connected to the COM1 port of the computer. The ADR101 board converts the 8-bit serial data through the serial port to the 8-bit parallel data and latches it. Direct access to the hardware is not allowed under Windows NT. Interaction with the serial port is achieved through a file handle and various WIN32 communication API's.

The first step in accessing the serial port is setting up a file handle.

```
m_hCom = CreateFile("Com1",
             GENERIC_READ | GENERIC_WRITE,
             0, // exclusive access
             NULL, // no security
             OPEN_EXISTING,
             0, // no overlapped I/O
             NULL); // null template
```

Check the returned handle for INVALID_HANDLE_VALUE and then set the buffer sizes.

```
m_bPortReady = SetupComm(m_hCom, 128, 128); // set buffer sizes
```

Port settings are specified in a Data Communication Block (DCB). The easiest way to initialize a DCB is to call GetCommState to fill in its default values, override the values that you want to change and then call SetCommState to set the values.

```
m_bPortReady = GetCommState(m_hCom, &m_dcb);
m_dcb.BaudRate = 9600;
m_dcb.ByteSize = 8;
m_dcb.Parity = NOPARITY;
m_dcb.StopBits = ONESTOPBIT;
```

```
m_dcb.fAbortOnError = TRUE;
m_bPortReady = SetCommState(m_hCom, &m_dcb);
```

Communication timeouts are optional but can be set similarly to DCB values:

```
m_bPortReady = GetCommTimeouts (m_hCom, &m_CommTimeouts);
m_CommTimeouts.ReadIntervalTimeout = 50;
m_CommTimeouts.ReadTotalTimeoutConstant = 50;
m_CommTimeouts.ReadTotalTimeoutMultiplier = 10;
m_CommTimeouts.WriteTotalTimeoutConstant = 50;
m_CommTimeouts.WriteTotalTimeoutMultiplier = 10;
m_bPortReady = SetCommTimeouts (m_hCom, &m_CommTimeouts);
```

If all of these API's are successful then the port is ready for use. Signal I/O through the ADR101 is achieved by writing the specific ADR101 command. A C++ serial port control class is generated. The detail code segment is listed in Appendix III.

5.6 Database Management

Since a very large amount of images are acquired and processed, a database is suitable for the result storage and processing. The Microsoft Database Access Object (DAO) data access system is implemented in the program. The processing results are stored in a Microsoft Access database file. There are two tables in this file, which store the results for individual images and the total statistical information respectively. When a new image is processed, a new record is created for this image in the first table and the record of the total statistics in the second table is renewed by a recursive function based on the total number of records. There are many useful SQL functions encapsulated in the C++ DAO database access class, which enable the users to navigate among the records.

To use the Microsoft DAO database access, the header file **afxdao.h** must be included in the MFC project header file **StdAfx.h**.

Chapter 6

System Implementation and Applications

6.1 System Implementation

Figure 6.1 shows a detailed block diagram of the HCMS. The digital camera is a black and white CCD camera made by Hitachi (KP-160U) as shown in Figure 6.1. The main specs of the CCD camera are as follows:

Model: KP-160U, black and white CCD camera

Power: DC 12V, 300mA

Resolution: 640X480

Connection: BNC connecter coaxial cable.

An auto-focus lens is attached to the camera. The lens is electrically controlled by the camera circuit. Information on the lens is listed below:

Model: H612E (C60625)

Manufacturer: Asahi Precision Co., Ltd.

Power: 8V~12V DC, less than 45mA

Focal length: 6mm

Iris range: 1.2~360

Mount: C

The image is acquired by an image acquisition board (Model: H612E by Asahi Precision Co., Ltd) inserted in the PCI slot inside a Pentium II, 400 MHz computer. The specifications of the frame grabber are listed as follows:

Model: PX510

Bus: PCI Bus

Manufactory: Imagenation Co.

Power: 5VDC, 650mA

Resolution: 640X480 pixels

Input composite video format: Monochrome, RS-170 (NTSC) or CCIR (PAL), auto detect

Input video: 1 V peak to peak, 75 Ohm

Capture time: Real time video capture; RS-170 (NTSC), 1/30 second per frame; CCIR (PAL), 1/25 second per frame

Supported operating systems**:** Win 98, 98-SE, 2000, ME, NT4

Supported languages**:** Visual C/C++

Figure 6.1 CCD camera used in this project

The computer is installed with a serial port expansion board so that more than 4 RS-232 serial ports are available. The control board to the camera motion and sensor system is designed and manufactured by Subsurface Sensing Lab at the University of Houston. Figure 6.2 shows power supply for the control, sensor, and motor and the

control board. A microprocessor (Microchip PIC16C71) is used for the system control. Communication with the computer is via RS-232 serial port. The electrical characteristics of the control board, power supply and motors are summarized as follows:

Control board:

Manufacturer: Subsurface Sensing Lab, University of Houston

Power: 5VDC, 100mA; 12VDC, 200mA

Microprocessor: Microchip PIC16C71

Communication port: RS232

Software: Assembly language

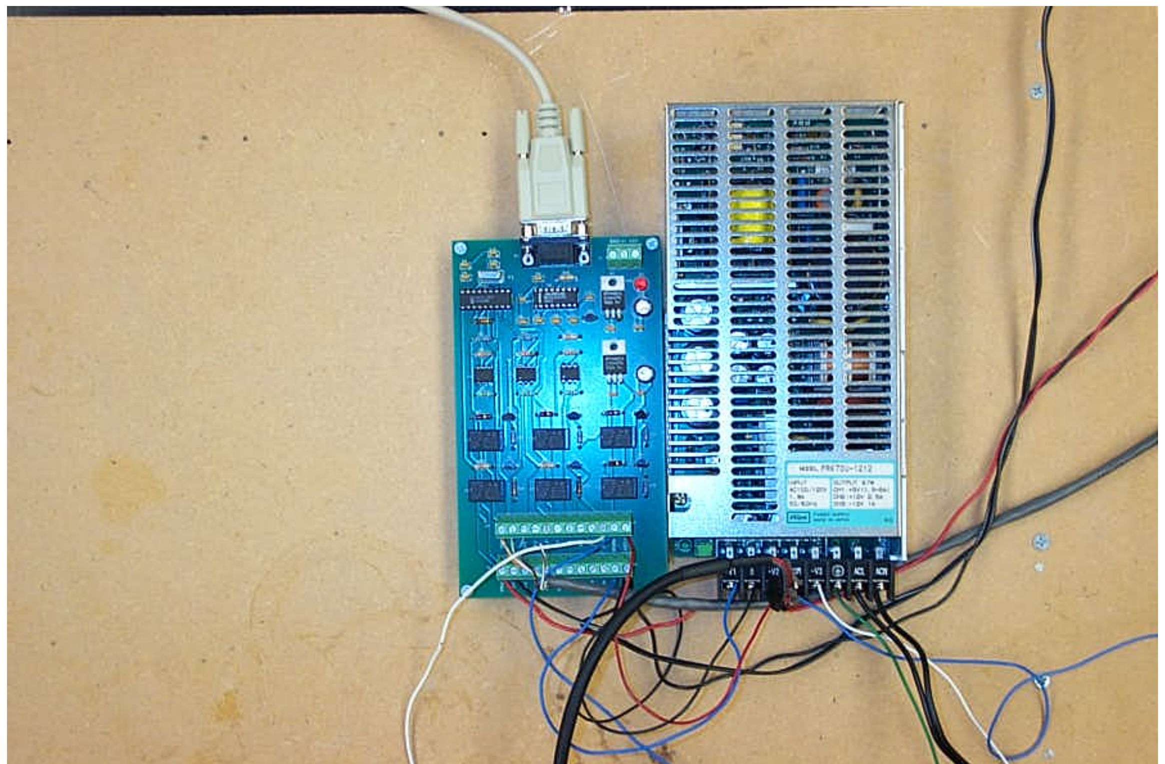


Figure 6.2 Control board and DC power supply

DC Power supply:

Model: PRK70U-1212

Manufactory: Volgen

Input: AC 100/120V, 1.8A, 50/60Hz

Output: 67W

CH1: +5V, 1.5~5A

CH2: +12V, 2.5A

CH3: -12V, 1A

DC Motor

Model: 4Z839

Manufactory: Dayton Electric Mfg. Co.

Power: 12VDC, 1.25A

F/L Torque: 15ln. –Lbs.

Input Motor H.P.: 1/160

Ratio: 95.7:1

In order to achieve smooth motion, three motors are used: two synchronous motors for x motion and one motor for y motion. The synchronization between the two motors are achieved by using the microcontroller. Synchronization algorithm is developed. At the point where an image is to be taken, a magnet is installed. Hall-effect sensors are used for sensing the position of the camera. Figure 6.3 shows the schematic of the controller board. Detailed connection information is listed in Appendix IV. Software used for motor control and position sensing can be found in Appendix V. Figure 6.4 is an overview of the control system.

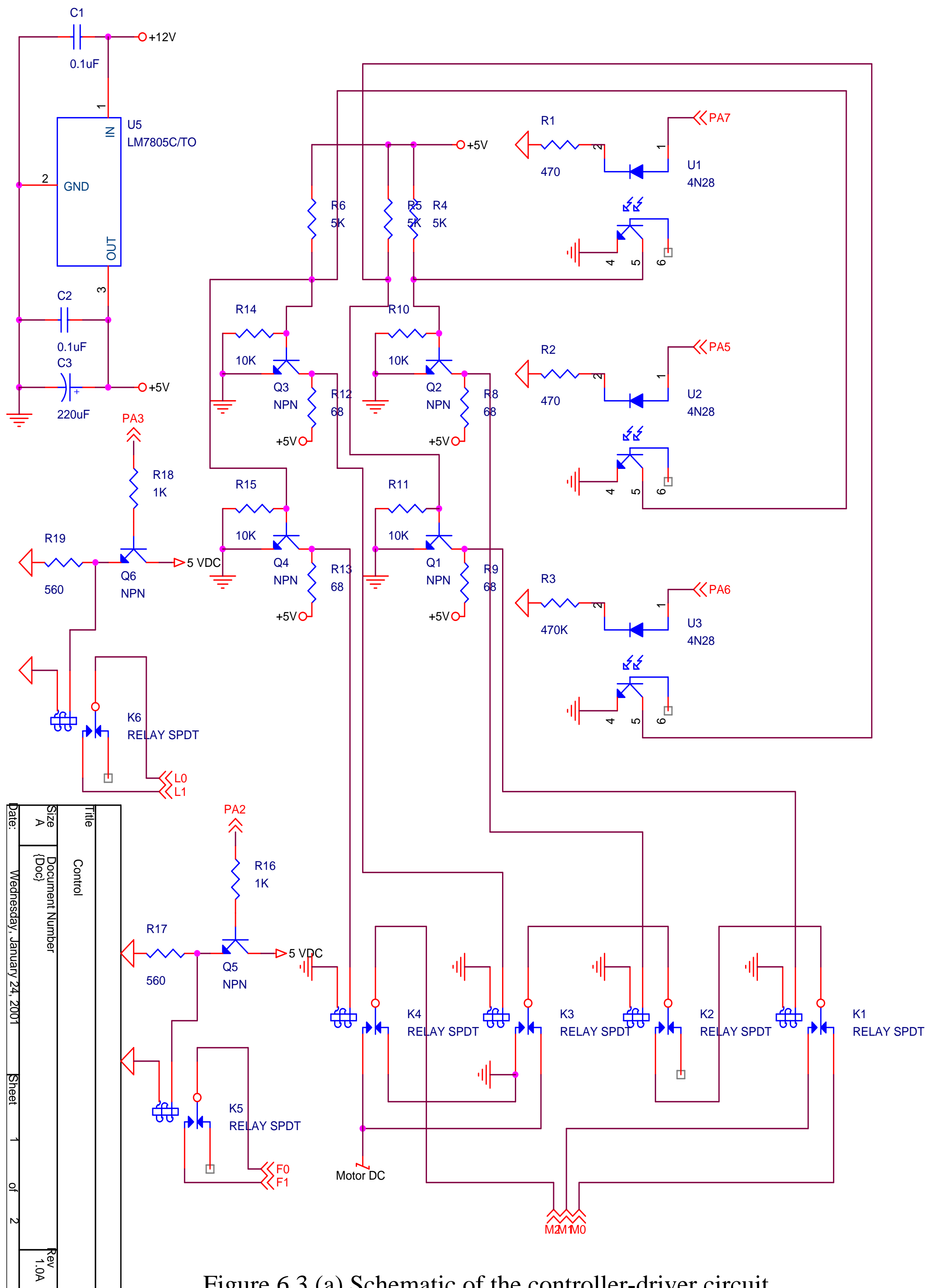


Figure 6.3 (a) Schematic of the controller-driver circuit

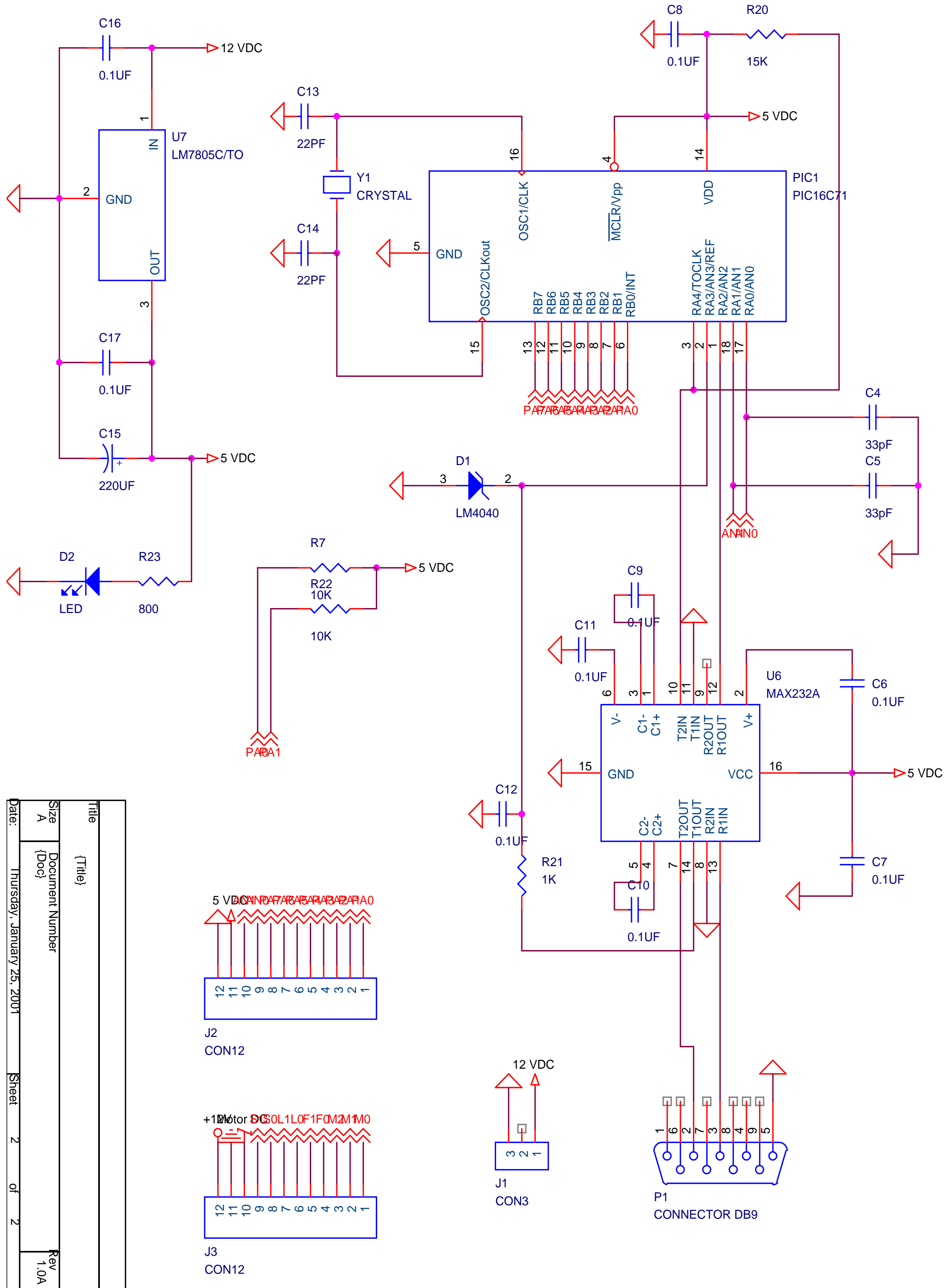


Figure 6.3 (b) Schematic of the controller-microprocessor and interface

Figure 6.4 Overview of the motion system developed in this project.

6.2 Applications

This system has been ready for field installation for more than 6 months now. Due to the unavailability of the TxMLS, we are unable to do field tests to the developed crack-monitoring system. However, many lab tests were done and many images taken from the field using a portable digital camera which were processed using the software developed in this project. In this section, some of the processed results are presented.

Figure 6.5 is the processed results of a horizontal crack. Statistical information is directly displayed in the widow next to the processed image.

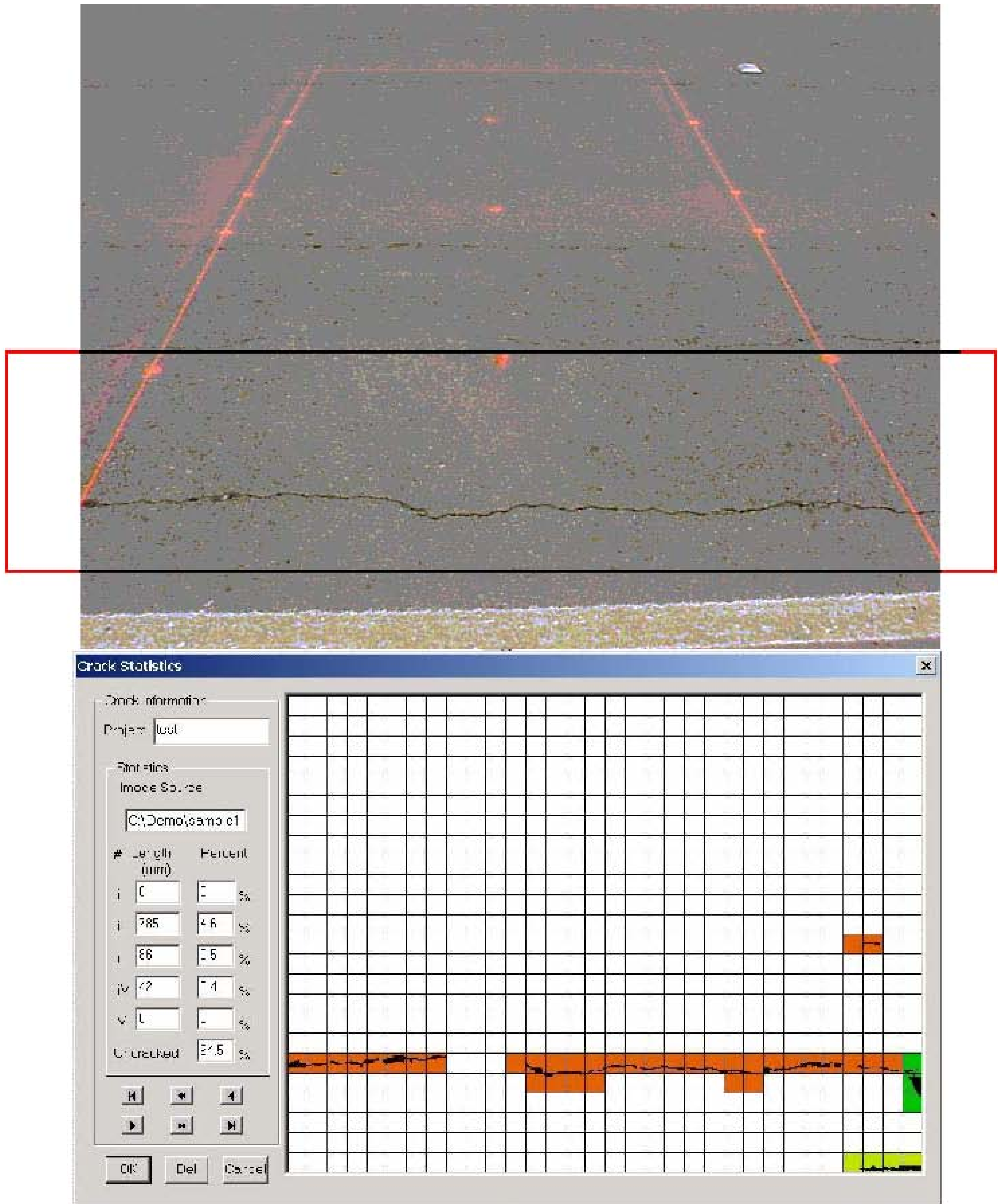
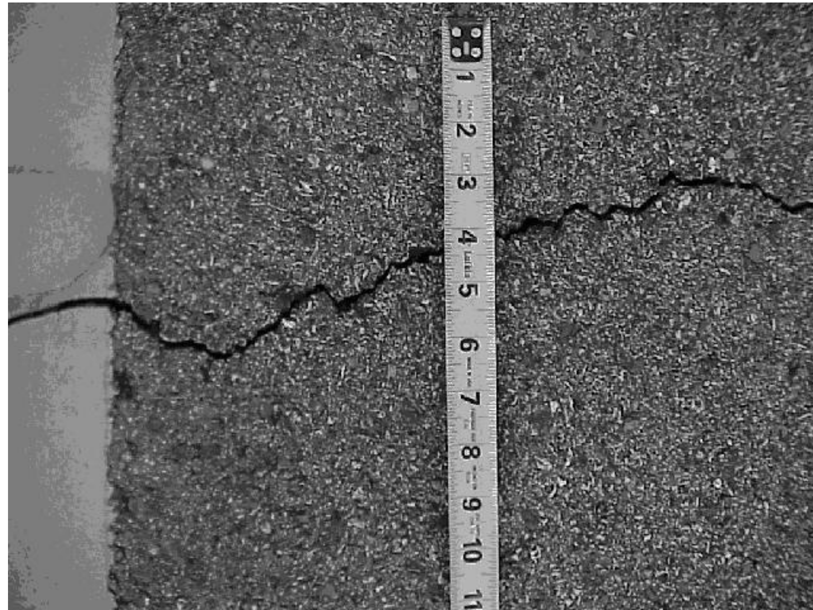


Figure 6.5. Processed results of a horizontal crack. Processed area is shown by the rectangular box.

Figure 6.6 shows an asphalt crack with a ruler giving actual dimensions with the processed results. It is seen that the image processing software thinks of the ruler as a vertical crack.



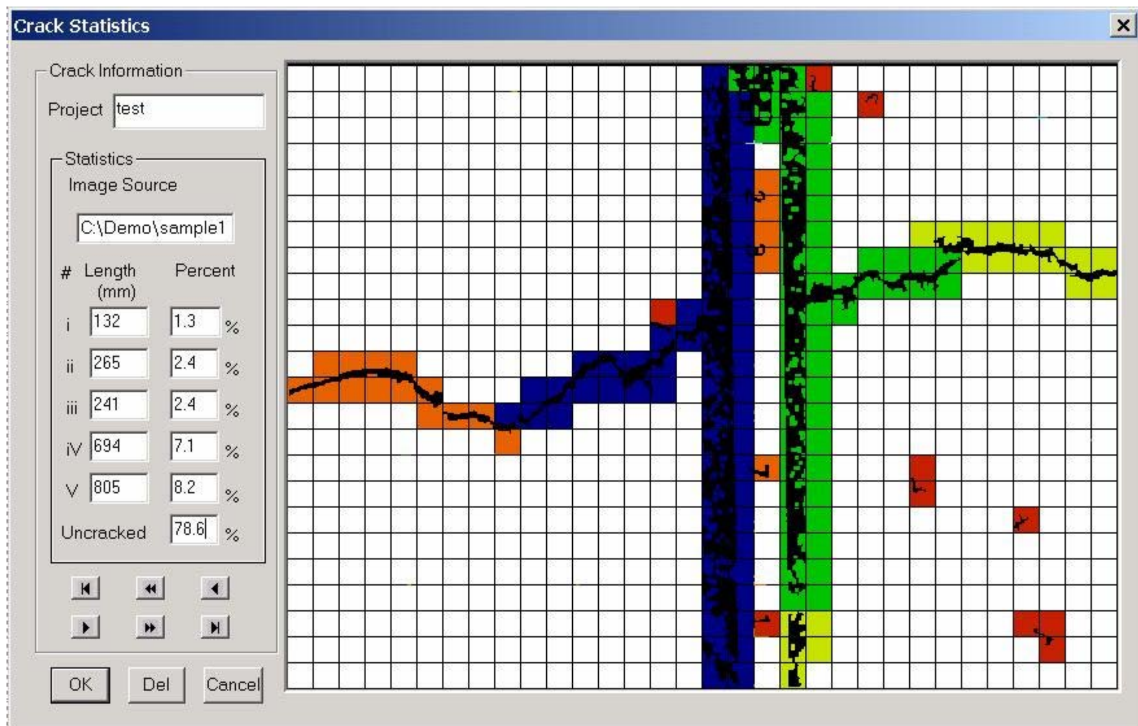
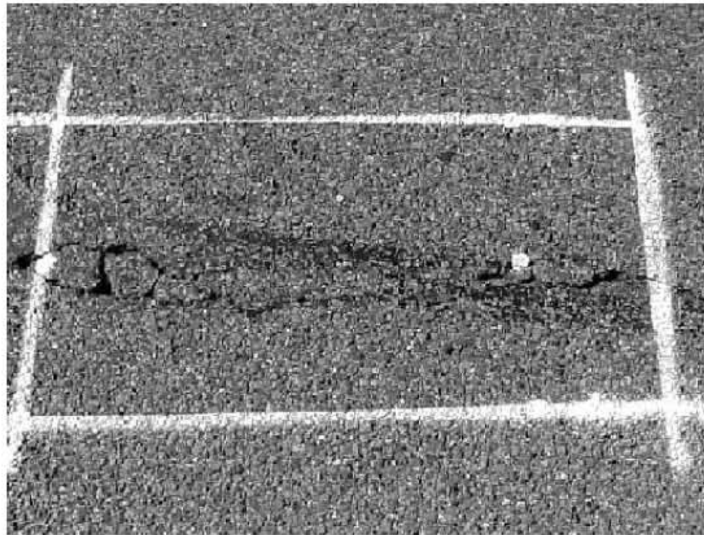


Figure 6.6 The image processing software can not identify object from cracks. The ruler image is being processed as a vertical crack in this example.

Figure 6.7 is a picture of asphalt cracks on a relatively new pavement. Cracks are mostly unconnected with fine branches. After processing the image, the software system is able to categorize cracks satisfactorily.



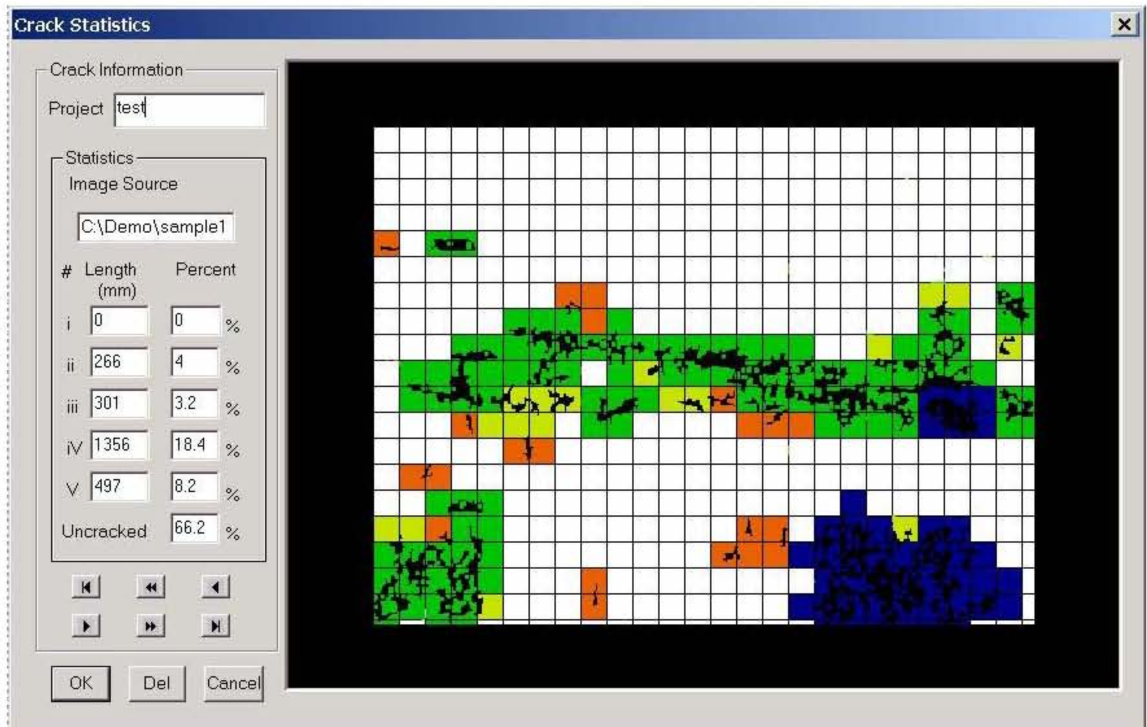


Figure 6.7 Asphalt cracks with fine branches

Figure 6.8 is an example of an alligator crack. Even with the dark image, the software is able to identify most of the cracks.

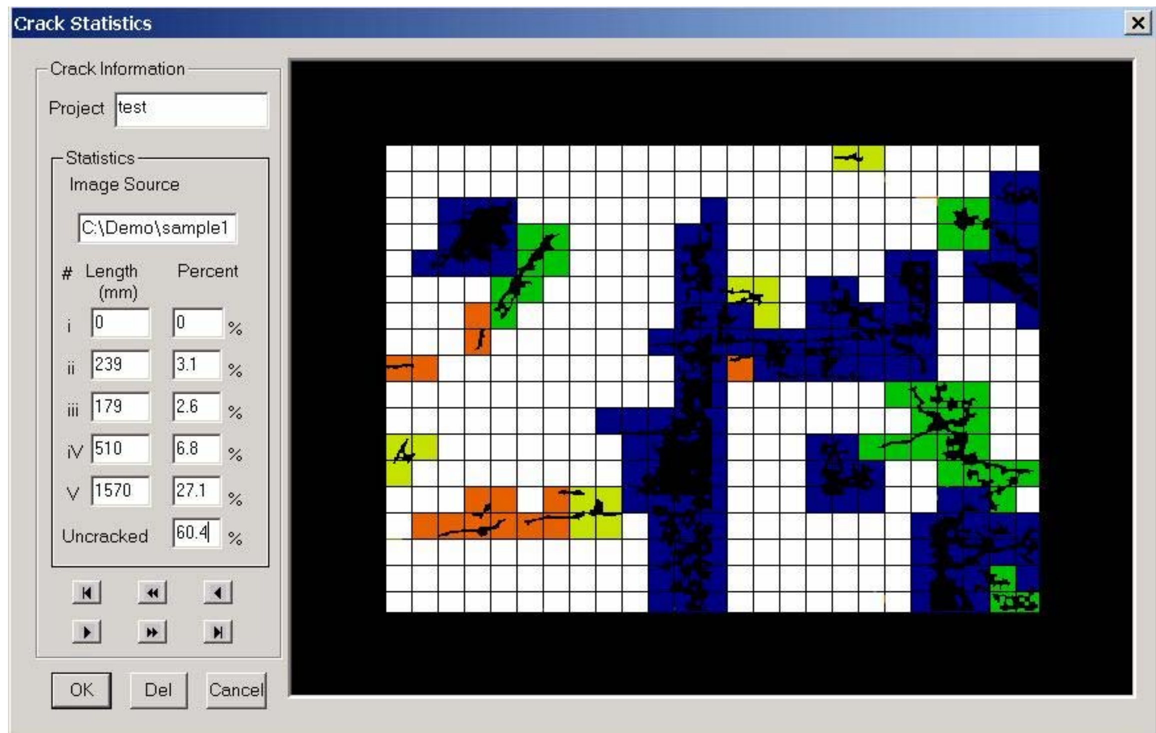


Figure 6.8 An example of alligator cracks and processed results.

Figure 6.9 shows a cluster of cracks forming a "hole" on an old pavement. The picture below is the processed image and statistic results.

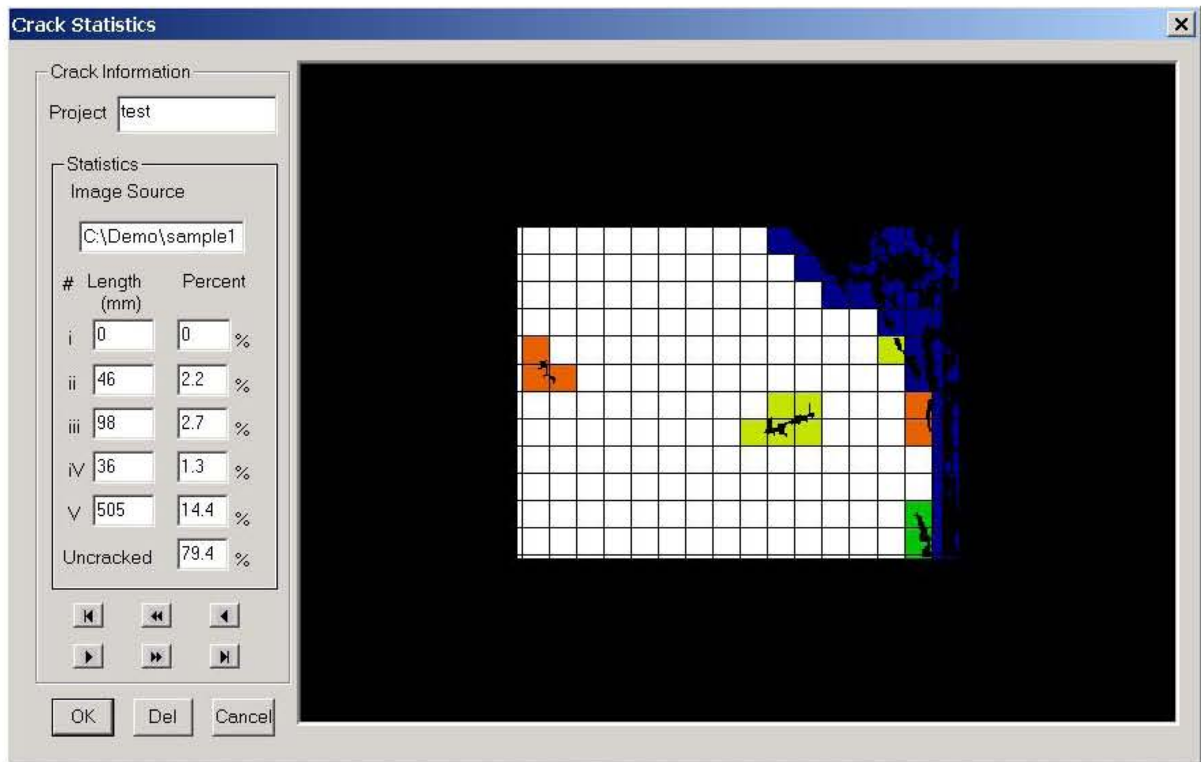
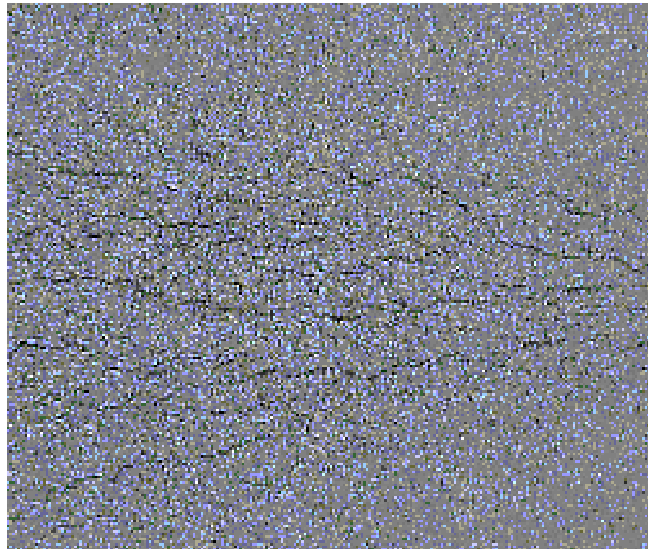


Figure 6.9 Processed results of a cluster of cracks on an old-pavement

Figure 6.10 is the image and processed result of a scattered crack on an asphalt pavement. It shows that the image processing software is able to acquire most of the cracks even with fine width.



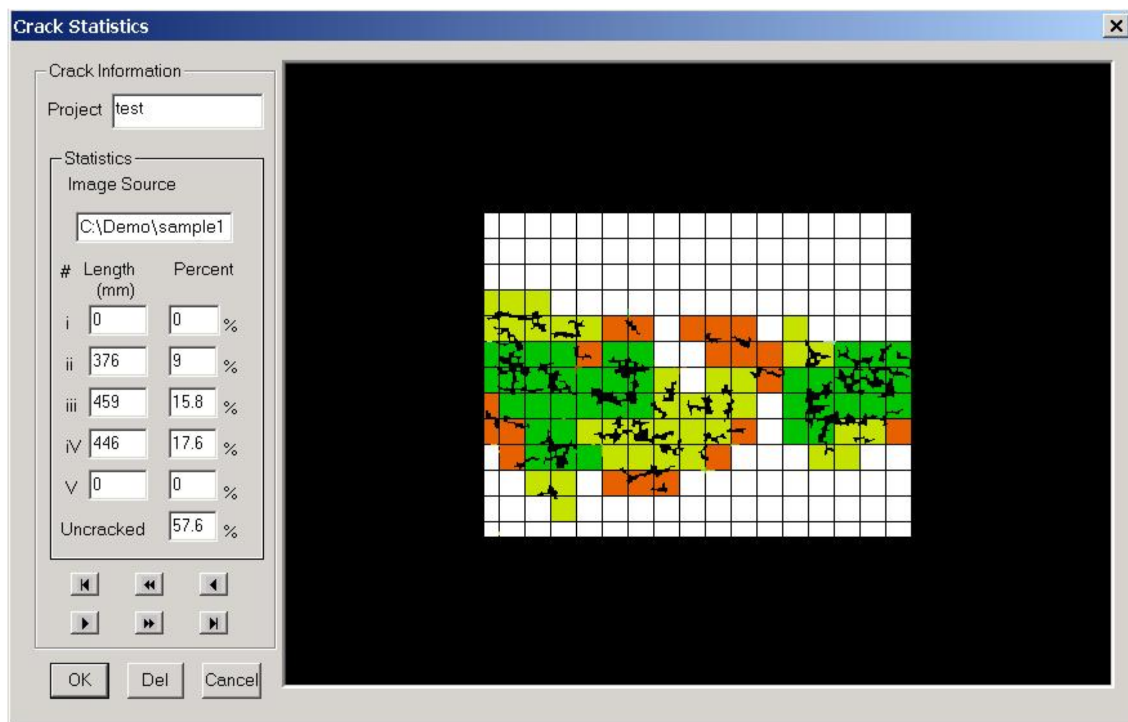


Figure 6.10 Processed result of scattered cracks on an asphalt surface

Figure 6.11 shows that the software is unable to identify the difference between an asphalt-sealed crack and a crack without sealant.

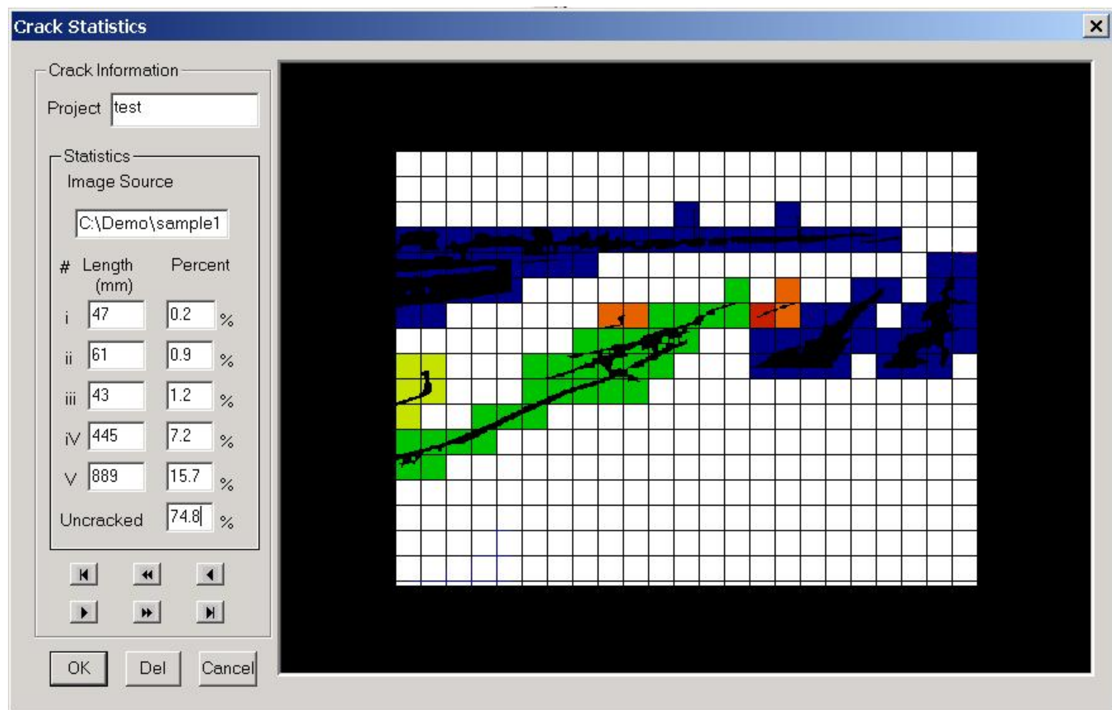


Figure 6.11 Processed results of asphalt-sealed cracks

Chapter 7

Conclusions

The Highway Crack Monitoring System provides a complete platform for automatic highway crack detection and analysis. It is still under development.

The crack image-processing algorithm is proven to be highly accurate and efficient. The recurring thresholding method demonstrates very high sensitivity on the crack object detection. The gray-level threshold is adaptively determined from an estimation-verification process, which is based on the local image content. The band thresholding is implemented to compensate for the boundary effect. The bandwidth is adjustable to achieve the best processing precision.

Multiple noise removal methods are implemented during the image processing. The connected-component-object identification is implemented to remove the background noise and small fake objects. A large fake object is distinguished by setting certain thresholds according to the properties of the crack object.

The crack object in the binary image is characterized by the object boundary processing. Morphological operations, such as dilation and erosion, are applied to remove the border irregularity, thus improving the accuracy of the calculation. Border convolution is a much more effective alternative of border tracing to determine and classify the border pixels. Two convolution-kernel matrices are designed to find out the turning pixels on the border contour. The perimeter of the object is obtained by accumulating the corresponding move lengths of the border pixels. The length and width of the object are therefore straightforward.

The HCMS hardware system demonstrates great reliability and efficiency. The CCD camera is under precise control. The motion system works with great flexibility. It has multiple working modes to provide great freedom.

Much work remains to improve the performance of the system. Higher image quality is desired for image acquisition in future systems. A camera with higher resolution and sensitivity is under consideration. Further improvement is also needed on the recurring segmentation algorithm. Now the algorithm is restricted to detect the crack objects that have the similar gray level within the same local image, which results in a single peak appearing in the histogram. Although with distinct block processing, the local image is restricted to a small part of the whole image, there might be cracks existing in the same image with a different gray level. Multiple layer segmentation can be the solution for it.

When TxMLS is in operational condition, we can install this system on to the TxMLS and do more field tests and improve the performance according to the results in the field.

References

[1] Milan Sonka, Vaclav Hlavac and Roger Boyle. Image Processing, Analysis, and Machine Vision. PWS Publishing, CA, 1998.

[2] S. Marchand-Maillet and Y.M.Sharaiha. Binary Digital Image Processing. Academic Press, San Diego, 2000.

[3] Kenneth R. CastleMan. Digital Image Processing. Prentice-Hall, New Jersey, 1979.

[4] John C. Russ. The Image Processing Handbook. IEEE Press, 1995.

[5] MATLAB Image Processing ToolBox, the Math Works, 1997.

[6] Gerhard X. Ritter and Joseph N. Wilson. Handbook of Algorithms in Image Algebra, 2nd Edition. CRC Press, 2000.

[7] David J. Kruglinski, George Shepherd and Scot Wingo. Programming Visual C++. Microsoft Press, 1998.

APPENDIXES

APPENDIX I Matlab Image Processing Sources Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Module Name : IP()
%Designed by Min WU, SSL Lab, University of Houston, May 31, 2000
%This program is designed to extract the crack information from the
%real images taken from the highway. It filters off the background
%noise and gets the output BMP file. It also characterizes the
%cracks in the image, such as the length and width.
%The input file is 256 or lower gray level. The output file is in
%mono format.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function IP(SrcFile, grid, factor, DestFile, OvlpFile, GridFile,
            Cal1, Cal2, Cal3, Cal4)
%function IP()

%Preset variables for testing
%SrcFile='d:\demo\sample1.bmp';
%DestFile='d:\demo\sample1R.bmp';
%GridFile='d:\demo\sample1G.bmp';
%grid=20;                %%grid size
%factor=168;             %%calibration parameter
%Cal1=300;               %%3mm
%Cal2=500;               %%5mm
%Cal3=700;               %%7mm
%Cal4=1000;              %%10mm
%Preset variables for testing

%Pre-defined parameters
Len_thd =10;
Len_width_ratio=5/factor;
Width_thd=2000;         %%20mm
Size_len=640;           %%image column size
Size_height=480;        %%image row size
%Pre-defined parameters

[BW, map] = imread(SrcFile, 'bmp'); %%read source image
BW=erase_border(BW);          %%erase border caused by the camera

%%distinct block operation
BW1(1:Size_height,1:Size_len)=1;
for i=1:5
   for j=1:8
      r1=96*(i-1)+1;
      c1=80*(j-1)+1;
      sel=imcrop(BW, [c1 r1 79 95]);
      sel1=core(sel);
      BW1(r1:(r1+95), c1:(c1+79))=sel1;
   end
```

```
end
BW1=bwmorph(BW1, 'close'); %%border smoothing

length=[0 0 0 0 0];      %%length vector
PCT=[0 0 0 0 0];         %%percentage vector

%%connected component labeling for object identifying
[crack, n]=bwlabel(BW1, 8);
BW=crack;
if (n>0)
   delta=(2-sqrt(2))/2;
   for i=1:n
      [col,row] = find(crack==i);
      %%select the object in smaller image
      sel = bwselect(BW1, row, col, 8 );
       r1=min(row);
      if (r1>3)
         r1=r1-5;
      end
      r2=max(row);
      if (r2<475)
         r2=r2+5;
      end
      h=r2-r1;
      c1=min(col);
      if (c1>3)
         c1=c1-5;
      end
      c2=max(col);
      if (c2<635)
         c2=c2+5;
      end
      w=c2-c1;
      sel1=imcrop(sel, [r1 c1 h w]);

      ar=bwarea(sel1);  %%calculate the area of the object
      %%object border smoothing at 3-pixel level
      sel1=bwmorph(sel1, 'dilate', 3);
      sel1=bwmorph(sel1, 'erode', 3);
      %%object dilation for border extraction
      SE=ones(2);
      sel1=dilate(sel1, SE);
      sel1=bwperim(sel1, 8);
      [col,row] = find(sel1>0);
      [a,b]=size(col);
      %%length calculation
      x=adjust(sel1);
      pix=a-(a-x)*delta;
      len=pix/2-4;
      %%width calculation
      Wid=factor*(ar/len);

      %%object identification using length-width criterion
      if ((len<Len_thd) | (len/Wid<Len_width_ratio) |
```

```
            Wid>Width_thd)
         BW1=BW1&(~sel);
      else
         if (Wid<Cal1)                     %%This program segment is to
            BW=FillCrack(BW, i, 150);  %%categorize the cracks
            length(1)=length(1)+len;
         elseif(Wid<Cal2)                  %%based on their width,
            BW=FillCrack(BW, i, 160);  %%and refill the cracks
            length(2)=length(2)+len;
         elseif(Wid<Cal3)                  %%with a new value
            BW=FillCrack(BW, i, 170);  %% representing different
            length(3)=length(3)+len;   %%categories.
         elseif(Wid<Cal4)
            BW=FillCrack(BW, i, 180);
            length(4)=length(4)+len;
         else
            BW=FillCrack(BW, i, 190);
            length(5)=length(5)+len;
         end
      end
   end
end

%%create the color map for output image
map =[0 0 0; jet(2)];
imwrite(~BW1,map,DestFile,'bmp');   %%write the output bitmap file

%%set backgound of the grid image
[BW, PCT]=SetGrid(BW, grid);
fid = fopen('C:\crack\report.dat','w');
for i=1:5
   fprintf(fid,'%d\t%d\t',fix(length(i)),fix(PCT(i)))
end
fclose(fid);

%%this segment is to re-draw the cracks
[BW1,n]=bwlabel(BW1, 8);
[col,row] = find(BW1>0);

[m,k]=size(col);
if (m>0)
   for i=1:m
      BW(col(i),row(i)) = 0;
   end
end
imwrite(BW,mapR,GridFile,'bmp'); %%write the output bitmap file


%%erase_border():  erase the fixed
%%noise incurred by the CCD camera on the image border
function [X]=erase_border(BW)

BW(1,:)=255;
BW(:, 640)=255;
```

```
X=BW;


%%adjust(): turning pixel adjustment for the length
%%calculation for binary image object
function [count]=adjust(BW)

SE1=[0 0 0; 1 1 1; 0 0 0];     %%convolution kernal
SE2=[0 1 0; 0 1 0; 0 1 0];     %%convolution kernal

BW1=fix(filter2(SE1,BW)/3);    %%convolution to determine
BW2=fix(filter2(SE2,BW)/3);    %%the turning pixels

count=0;
v=sum(BW1);
count=count+sum(v);            %%count the total number of
v=sum(BW2);                    %%the turning pixel
count=count+sum(v);


%%core(): core function for image segmention
function [BW1]=core(BW)

sum1=sum(BW);                  %%calculate the average
avg=sum(sum1)/(96*80);         %%gray value of the pixels

x=avg-33;                      %%threshold estimation
BW1=BW<x;                      %%first segmentation

[crack, n] = bwlabel(BW1, 8);
for i = 1 : n
   [col,row] = find(crack==i);
   [m, k]=size(col);
   if (m < 35)                 %%object idetification
      BW1(col,row) = 0;        %%lower limit
   end
   if (m > 2500)               %%object idetification
      BW1(col,row) = 0;        %%upper limit
   end
end

BW1=double(BW1);
BW2=double(BW);
BW2=uint8(BW1.*BW2);           %%masking original image

%%histogram analysis
[hstgrm, x]=imhist(BW2);       %%get the histogram
hstgrm(1:5)=0;                 %%clear white pixel distribution
[posL, posH]=get_pos(hstgrm, 0.3);  %%get the crack pixel range
BW1=imadjust(BW, [posL/255 posH/255+0.005], [0 1]);
                               %%intensity transformation
BW1=BW1<208;                   %%segmentation

[crack, n] = bwlabel(BW1, 8);
```

```
for i = 1 : n
   [col,row] = find(crack==i);
   [m, k]=size(col);
   if (m < 25)                    %%object identification
      BW1(col,row) = 0;
   end
   if (m > 2800)                  %%object identification
     BW2 = bwselect(BW1, row, col, 8);
     BW1 = BW1 & (~BW2);
   end                            %%
end

MK(1:96, 1:80)=1;
MK(2:95,2:79)=0;            %%preserve object pixels near
MK=MK&BW1;                  %%the border for recovery

BW1=bwmorph(BW1, 'close');
BW1=BW1 | MK;               %%recover the border pixels


%%get_pos(): determine the gray value range for transformation
function [pL, pH]=get_pos(hstgrm, pct)

hstd=pct*max(hstgrm);
m_hst=find(hstgrm>hstd);
[sm1, sm2]=size(m_hst);
if (sm1>0)
   pL=m_hst(1);
   pH=m_hst(sm1);
else
   pL=0;
   pH=0;
end


%%FillCrack(): assign an individual gray value to the crack
%%object based on the crack width category
function [xImage]=FillCrack(crack, label, CAT)

[col,row] = find(crack==label);
[k1,k2]=size(col);
m=k1;
for k=1:k1
   crack(col(k), row(k))=CAT;
end
xImage=crack;

%%SetGrid(): set the grid background of grid output
%%and calculate the crack occupancy
function [Block, GRP]=SetGrid(BW, grid)

xSize=640;
ySize=480;
```

```
PCT=[0 0 0 0 0];

xBlock=ceil(xSize/grid);
yBlock=ceil(ySize/grid);
blocks=xBlock*yBlock;      %%count the total grids

for m=0:(yBlock-1)
   for n=0:(xBlock-1)
      xStart=fix(grid*n)+1;
      yStart=fix(grid*m)+1;

      xStop=fix(grid*n+grid);
      yStop=fix(grid*m+grid);

      lengthX=fix(grid);
      lengthY=lengthX;

      if ((xStart+grid)>xSize)
         lengthX=xSize-xStart;
         xStop=xSize;
      end

      if ((yStart+grid)>ySize)
         lengthY=ySize-yStart;
         yStop=ySize;
      end

      col=xStart+1:xStop;  %%get the grid column parameter
      row=yStart+1:yStop;  %%get the grid row parameter

      square = imcrop(BW, [xStart yStart lengthX lengthY]);

      c=max(square);       %%determind the maximum element, which
      v=max(c);            %%is the widest crack label value
      if (v==150)
         BW(row, col)=15;  %%assign "red" color to the class I
         PCT(1)=PCT(1)+1;
      elseif (v==160)
         BW(row, col)=32;  %%assign "orange" color to the class II
         PCT(2)=PCT(2)+1;
      elseif (v==170)
         BW(row, col)=63;  %%assign "yellow" color to the class III
         PCT(3)=PCT(3)+1;
      elseif (v==180)
         BW(row, col)=49;  %%assign "green" color to the class IV
         PCT(4)=PCT(4)+1;
      elseif (v==190)
         BW(row, col)=5;   %%assign "blue" color to the class V
         PCT(5)=PCT(5)+1;
      else
         BW(row, col)=256; %%others for black.
      end
   end
end
```

```
GRP=1000*PCT/blocks;
   Block=BW;
```

APPENDIX II CCD Camera Control Class Declaration Code

```
/////////////////////////////////////////////////////////////////////
// CameraThread.h : header file
//

#ifndef __CAMERATHREAD_H__
#define __CAMERATHREAD_H__

#include "wpx5_NT.h"    //CCD camera API

typedef struct
{
      BITMAPINFOHEADER head;
      RGBQUAD colors[256];
} MAPHEAD;

/////////////////////////////////////////////////////////////////////
// CCameraThread thread

class CCameraThread : public CWinThread
{
public:
      DECLARE_DYNCREATE(CCameraThread)
      CCameraThread(CWnd* pWnd);

// Attributes
public:
      CRect m_rectBorder;
      HANDLE m_hEventKill;
      HANDLE m_hEventDead;
      static HANDLE m_hAnotherDead;

      static CRITICAL_SECTION m_csCameraLock;


// Operations
public:
      void OnScrollBy(CSize sizeScroll, BOOL bScroll = TRUE);
      int ImageMaxY;
      int ImageMaxX;
      void KillThread();
      virtual void SingleFrame();

// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CCameraThread)
      //}}AFX_VIRTUAL

// Implementation
      virtual ~CCameraThread();
      virtual void Delete();
      BOOL pxWriteCurrentFile(LPSTR fileName);
```

```
protected:

	BOOL AllocBuffer();
	void SetBitMapHead();
	void CreateGrayPalette();
	MAPHEAD m_mapHead;
	BYTE * gpBits;
	void GetImage(FRAMEHANDLE frh);
	void Paint(HDC hDC);
	HPALETTE hpalette;
	FRAMEHANDLE frh[2];
	int frhIdx;
	int tagQ[2];
	HANDLE hBuf;
	FGHANDLE fgh;
	BOOL CameraInit();
	void CameraExit();
	virtual BOOL InitInstance();

	// Generated message map functions
	//{{AFX_MSG(CCameraThread)
		// NOTE - the ClassWizard will add and remove member
functions here.
	//}}AFX_MSG

	DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////
#endif
```

APPENDIX III RS232 I/O Implementation Source Code

```
/////////////////////////////////
// initialize the com port
/////////////////////////////////
BOOL CComPort::Initialize()
{
      DWORD dwRC;
      DWORD dwError;
      char  sMsg[512];

      m_bPortReady = TRUE; // everything is OK so far

      m_hCom = CreateFile(m_sComPort,
            GENERIC_READ | GENERIC_WRITE,
            0, // exclusive access
            NULL, // no security
            OPEN_EXISTING,
            0, // no overlapped I/O
            NULL); // null template

      if (m_hCom == INVALID_HANDLE_VALUE)
      {
            m_bPortReady = FALSE;
            dwError = GetLastError();

            // example error code expansion follows
            LPVOID lpMsgBuf;
            lpMsgBuf = NULL;
            dwRC = FormatMessage(
                  FORMAT_MESSAGE_ALLOCATE_BUFFER |
                  FORMAT_MESSAGE_FROM_SYSTEM |
                  FORMAT_MESSAGE_IGNORE_INSERTS,
                   NULL,
                  dwError, //  from GetLastError(),
                  MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
                  (LPTSTR) &lpMsgBuf,
                  0,
                  NULL);

            if (dwRC && lpMsgBuf)
            {
                  sprintf(sMsg, "COM open failed: Port=%s Error=%d
                        - %s", m_sComPort, dwError, lpMsgBuf);
                  AfxMessageBox(sMsg);
            }
            else
            {
                  sprintf(sMsg,"COM open failed: Port=%s Error=%d",
                        m_sComPort, dwError);
                  AfxMessageBox(sMsg);
            } // end if
```

```
	if (dwRC && lpMsgBuf)
	{
		LocalFree( lpMsgBuf );
	} // end if

} // end if

if (m_bPortReady)
{
	m_bPortReady = SetupComm(m_hCom,
		128, 128); // set buffer sizes
	if (!m_bPortReady)
	{
		dwError = GetLastError();
		sprintf(sMsg, "SetupComm failed: Port=%s
			Error=%d",  m_sComPort, dwError);
		AfxMessageBox(sMsg);
	} // end if
} // end if

if (m_bPortReady)
{
	m_bPortReady = GetCommState(m_hCom, &m_dcb);
	if (!m_bPortReady)
	{
		dwError = GetLastError();
		sprintf(sMsg, "GetCommState failed: Port=%s
			Error=%d",  m_sComPort, dwError);
		AfxMessageBox(sMsg);
	} // end if
} // end if

if (m_bPortReady)
{
	m_dcb.BaudRate = 9600;
	m_dcb.ByteSize = 8;
	m_dcb.Parity = NOPARITY;
	m_dcb.StopBits = ONESTOPBIT;
	m_dcb.fAbortOnError = TRUE;

	m_bPortReady = SetCommState(m_hCom, &m_dcb);
	if (!m_bPortReady)
	{
		dwError = GetLastError();
		sprintf(sMsg, "SetCommState failed: Port=%s Error
			= %d", m_sComPort, dwError);
		AfxMessageBox(sMsg);
	}
} // end if

if (m_bPortReady)
{
	m_bPortReady = GetCommTimeouts (m_hCom,
		&m_CommTimeouts);
```

```
			if (!m_bPortReady)
			{
				dwError = GetLastError();
				sprintf(sMsg, "GetCommTimeouts failed: Port=%s
					Error = %d", m_sComPort, dwError);
				AfxMessageBox(sMsg);
			} // end if
		} // end if


		if (m_bPortReady)
		{
			m_CommTimeouts.ReadIntervalTimeout = 50;
			m_CommTimeouts.ReadTotalTimeoutConstant = 50;
			m_CommTimeouts.ReadTotalTimeoutMultiplier = 10;
			m_CommTimeouts.WriteTotalTimeoutConstant = 50;
			m_CommTimeouts.WriteTotalTimeoutMultiplier = 10;
			m_bPortReady = SetCommTimeouts (m_hCom,
				&m_CommTimeouts);
			if (!m_bPortReady)
			{
				dwError = GetLastError();
				sprintf(sMsg, "SetCommTimeouts failed: Port=%s
					Error = %d", m_sComPort, dwError);
				AfxMessageBox(sMsg);
			} // end if
		} // end if

		if (m_bPortReady)
		{
			BOOL  bWriteRC;
			DWORD iBytesWritten;

			//Configure port A: Bit A0 is the pin for signal input
			//Bit A7 is the output pin for motor control

			iBytesWritten = 0;

			bWriteRC = WriteFile(m_hCom, "CPA00000001\r",
				12, &iBytesWritten, NULL);
			if (!bWriteRC || iBytesWritten == 0)
			{
				AfxMessageBox("Fail to initialize the port!");

			}
		}

		return m_bPortReady;
	} // end CComPort::Initialize

	/////////////////////////////////
	// terminate the com port
	/////////////////////////////////
	void CComPort::Terminate()
```

```
{
      CloseHandle(m_hCom);
} // end CComPort::Terminate


/////////////////////////////////
// read data from the com port
/////////////////////////////////
BOOL CComPort::Read(CString& sResult)
{
      BOOL  bWriteRC;
      BOOL  bReadRC;
      DWORD iBytesWritten;
      DWORD iBytesRead;
      char  sBuffer[32];

      iBytesWritten = 0;
      bWriteRC = WriteFile(m_hCom, "RPA\r",4,&iBytesWritten,NULL);
      if (!bWriteRC || iBytesWritten == 0)
      {
            return FALSE;
      } // end if
      memset(sBuffer,0,sizeof(sBuffer));
      bReadRC = ReadFile(m_hCom, &sBuffer, 15, &iBytesRead, NULL);

      if (bReadRC && iBytesRead > 0)
      {
            sResult = sBuffer;
      }
      else
      {
            return FALSE;
      } // end if
      return TRUE;
} // end CComPort::Read

/////////////////////////////////
// write command to the com port
/////////////////////////////////
void CComPort::Write(const int& m_iCommand)
{
      BOOL  bWriteRC;
      DWORD iBytesWritten;
      DWORD dwError;
      char  sMsg[128];

      iBytesWritten = 0;
      switch (m_iCommand){
      case 1:
            bWriteRC = WriteFile(m_hCom, "SPA11100011\r",
                  15, &iBytesWritten,NULL);
            break;
      case 2:
            bWriteRC = WriteFile(m_hCom, "SPA10000011\r",
```

```
                15, &iBytesWritten,NULL);
            break;
        default:
            bWriteRC = WriteFile(m_hCom, "SPA00000011\r",
                15, &iBytesWritten,NULL);
            break;
        }

        if (!bWriteRC || iBytesWritten == 0)
        {
            dwError = GetLastError();

            sprintf(sMsg, "Write of length query failed: RC=%d, "
                        "Bytes Written=%d, Error=%d",
                        bWriteRC, iBytesWritten, dwError);
            AfxMessageBox(sMsg);
        } // end if
}
```

APPENDIX IV Pin Configuration of the Motion System Control Box

1. Interior

PIN #	DIP5	DIP8
1	PA6 (ADR101)	PA7 (ADR101)
2	F0 (FLASH)	PA5 (ADR101)
3	PA2 (ADR101)	GND (Vcc)
4	F1 (FLASH)	PA1 (ADR101)
5	PA0 (ADR101)	Vcc (+5V)
6	--	V_{DD} (+12V)
7	--	GND (V_{DD}, V_{EE})
8	--	V_{EE} (-12V)

2. Exterior

PIN #	Assignment	PIN #	Assignment
1	DIP8 (6)	**7**	Motor B
2	DIP8 (7)	**8**	Motor C
3	DIP8 (8)	**9**	Sensor A(0)
4	DIP5 (2)	**10**	Sensor A(1)
5	DIP5 (4)	**11**	Sensor B(0)
6	Motor A	**12**	Sensor B(1)