Send Orders for Reprints to reprints@benthamscience.ae

846 *The Open Civil Engineering Journal,* **2015**, *9,* 846-851

Open Access

Concrete Crack Width Detecting System for Android Platform

Chen Shuang-rui*, Shi Zheng* and Yan Quan-sheng*

School of Civil Engineering and Transportation, South China University of Technology, Guangzhou 510640, Guangdong, China

Abstract: In order to measure crack width accurately and automatically, an Android-based Automatic Crack Width Measuring System (ACWMS) has been developed, taking advantage of the high portability of Android devices. After capturing the image using mobile phone camera, the image is processed by image processing techniques, including graying, binaryzation, denoising and edge recognition. A specified algorithm is executed to calculate the crack width according to the provided edge data. Measurements has been done to each of the 10 cracks in the same concrete beam, using Samsung Galaxy S3 mobile phone and WYSX-40X Crack detector, respectively. Test result shows that the maximum crack width accuracy reaches 95.26%, which satisfies the construction needs. Therefore, this system can greatly improve the efficiency and accuracy during crack width measurement.

Keywords: Android, crack, image processing, OTSU algorithm, width measuring.

1. INTRODUCTION

During the bridge's operation stage, the generation of cracks is almost inevitable. The width of crack is usually regarded as an important basis during the evaluation of serviceability limit state [1]. Therefore, the width of cracks should be measured accurately. At present, there are several methods for the evaluation of cracks, including artificial crack microscopy, crack image presentation with artificial interpretation, crack image presentation with automatic interpretation [2]. Artificial interpretation has low efficiency and may introduce personal errors; Crack image presentation with automatic interpretation requires the support of PC, which is not portable. Based on the above shortages, an automatic crack width measuring system based on Android device has been developed, taking the unique advantage of high portability of Android devices.

2. SYSTEM DESIGN

The overall design idea of the Android-based Automatic Cracking Measuring System (ACWMS) is to capture the image of the crack by mobile phone camera, process the image using digital image processing technique, and then execute a specified algorithm according to the edge data to calculate the crack width [3]. The hierarchical structure of the system is shown in Fig. (**1**).

The system includes two modes: Capture mode and Open mode. Capture mode indicates a new measure of crack width.

When this mode is activated, the process of capturing image is done by built-in camera application, which automatically executes operations including focusing, exposure, taking picture and preview.

After the image is captured, the image is processed by graying, binarizing, denoising, edge-recognition, etc [4]. After the image processing procedures are completed, the image is stored in a specified folder. Open mode is similar to capture mode, the slight difference lies in the source of image. The image in open mode comes from existing files.

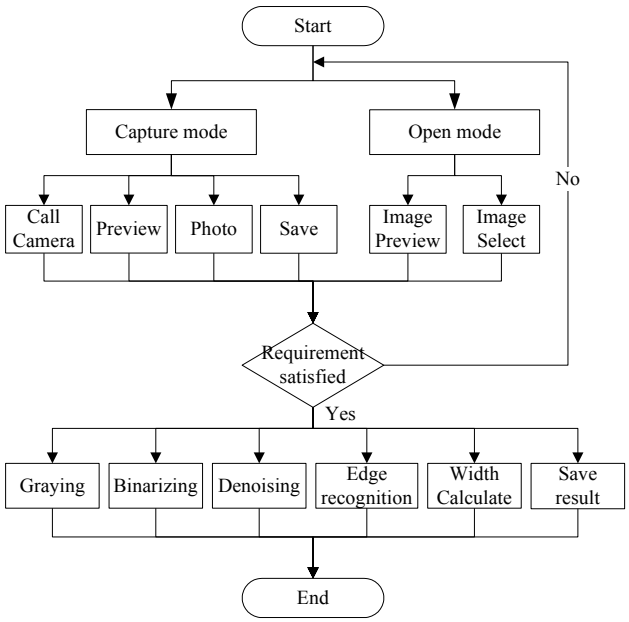


Fig. (1). System hierarchical structure.

3 REALIZATION OF THE SYSTEM

This system mainly consists of 3 parts: Crack image capturing, image processing and crack width calculation. The

key techniques of the system will be introduced below in this chapter.

There are two ways to capture image in Android system. The first one is to customize camera interface to realize camera functions. Despite the high degree of customization, however, the preview, focus, picturing, rendering functions should be implemented by the programmer. Another way is calling the built-in camera application of the Android system, so that the image data can be acquired after photographing [5]. The system only requires the image data rather than the customization of UI in photographing, therefore the second method is adopted by the system.

The code below calls the built-in camera application [6]:

```
// Call system camera
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
// Set directory of the image data
File f = new File(dir, "crack.jpg");
Uri u = Uri.fromFile(f);
// Activate the system camera intent.putExtra(MediaStore.Images.Media.ORIENTATION, 0);
intent.putExtra(MediaStore.EXTRA_OUTPUT, u);
startActivityForResult(intent, 1);
```

After the camera finished picturing, the method onActivityResult (int requestCode, int resultCode, Intent data) is called, the image data can be acquired by calling method data.getData(), however it returns the thumbnail of the image, the error of the calculation result is relatively big, Therefore only the module class CrackChart in onActivityResult() should be activated, the original image data is processed in this class.

```
//Activate image processing module
Intent intent = new Intent();
intent.setClass(this, CrackChart.class);
startActivity(intent);
```

The entire image processing algorithm includes 5 parts: graying, binaryzation, filtering, edge recognition, and width calculation, among which edge recognition is the key part of the system, because the effect of it determines the accuracy of the algorithm.

The captured image is color image, but the post processing is based on gray image. The crack recognition is based on the difference between the grayscale of the crack and the background, therefore it is necessary to convert color image to grayscale image. In a RGB image, every pixel consists of R(Red), G(Green) and B(Blue) three color components [7]. According to the psychological principle of color brightness, the transformation between color and grayscale can be done by the following formula:

$$\text{Gray} = \text{R}*0.299 + \text{G}*0.587 + \text{B}*0.114 \quad (1)$$

However, CPU calculates much faster using integer arithmetic rather than using floating point arithmetic, and considering that shifting operation is much faster than division operation in CPU, therefore the right expression is multiplied by 2^{16}, namely 65536 in integer, and then the coefficients of R,G,B are calculated, with truncation error included in order to maintain 3 digit precision and achieve higher accuracy.

$0.299 * 65536 = 19595.264 \approx 19595$

$0.587 * 65536 + (0.264) = 38469.632 + 0.264 = 38469.896 \approx 38469$

$0.114 * 65536 + (0.896) = 7471.104 + 0.896 \approx 7472$

The calculation above uses truncation method in integer convertion, whichis more accurate than using rounding method directly. And then, right shift 16 digits to the right expression to divide 2^{16}. After the above processing, the grayscale conversion formula becomes

$$\text{Gray} = (\text{R}*19595 + \text{G}*38469 + \text{B}*7472) >> 16 \quad (2)$$

The effect of graying is shown in Fig. **5**(**a**).

In order to identify the crack information, the target area must be binarized. Image binarizing is a process that divides an image into several small areas of mutually disjoint and separate the target and background. Image binarizing is the basis of image recognition analysis. Assume the raw image is gray(x,y) and the threshold value is *T*. Set pixels which gray value is greater than *T* for 1, else for 0[8], dividing the source image into two parts. The formula is as follows:

$$p(x,y) = \begin{cases} 1, gray(x,y) > T \\ 0, gray(x,y) \le T \end{cases} \quad (3)$$

OTSU method is regarded as the best algorithm in determining the threshold value in images segmentation. It performs well with least calculation and no interference by image brightness and contrast. The method has been widely used in digital image processing. The method is adopted by this system.

The basic idea of OTSU is [9], assume that the image has *L* levels of grayscale, there are n_i pixels whose grayscale is i, the total number of pixels is

$$N = \sum_{i=0}^{L-1} n_i ,$$

the probability of each grayscale level is $p_i = n_i / N$, it's evident that $p_i \ge 0$ and

$$\sum_{i=0}^{L-1} p_i = 1 .$$

The image is divided into 2 parts, A and B, according to the threshold value T. Part A is the background class where the grayscale of pixels lies between 1~t, and part B is the target class where the grayscale lies between t+1~L-1, the probability and the mean value grayscale level of class A,B are as follows

$$p_A = \sum_{i=0}^{t} p_i , p_B = \sum_{i=t+1}^{L-1} p_i = 1 - p_A \quad (3)$$

$$\omega_A = \sum_{i=0}^{t} ip_i / p_A, \omega_B = \sum_{i=t+1}^{L-1} ip_i / p_B \quad (4)$$

The mean value of the overall grayscale value of the image

$$\omega_0 = p_A\omega_A + p_B\omega_B = \sum_{i=0}^{L-1} ip_i \quad (5)$$

The variance between class A and B can be obtained as follows

$$\sigma^2(t) = p_A(\omega_A - \omega_0)^2 + p_B(\omega_A - \omega_0)^2 \quad (6)$$

In order to get the best threshold value, the variance between classes A and B is used as judging criterion, the best threshold value T makes the variance achieve its maximum value.

$$\max\left[\sigma^2(t)\right] = \max_{0\le t\le L-1}\left[p_A(\omega_A - \omega_0)^2 + p_B(\omega_A - \omega_0)^2\right] \quad (7)$$

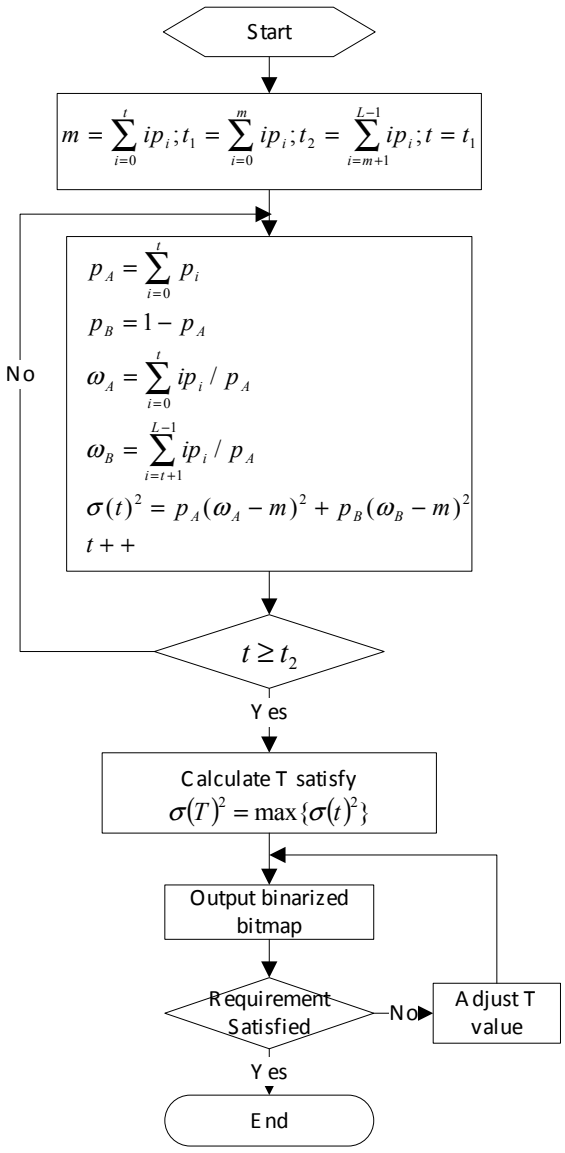


Fig. (2). Flow chart of OTSU algorithm.

Because the variance indicates the uniformity of grayscale distribution, the larger the variance is, the bigger difference is between A and B. If the target is categorized as background or background is categorized as target by mistake, the variance between A and B will become smaller. The probability of categoring mistake will get its lowest value when $\sigma^2(t)$ achieves its highest value [10], which is OTSU criterion. The flow chart is shown in **Fig. (2)**. The effect of binarizing is shown in Fig. **5**(**b**).

It is inevitable that the image will get disturbed in the process of capturing, transferring and storing, resulting in image degradation. However, the preprocessing algorithm will directly affect the post processing operations, such as target identification, edge recognition, etc. Therefore, image denoising is quite necessary to maintain raw information and eliminate useless information, thus acquiring a high quality image.

After binarizing, the source image transfers into monochrome image. In most cases, large amount of noise also exists in the image besides the target, which interferes the process of edge recognition and crack extraction, therefore all noise should be cleared in order to extract cracks accurately, which leads to the necessity of denoising. Denoising methods includes median filtering, wavelet threshold denoising based on wavelet domain, PDE based image denoising, total variation(TV) image denoising, etc [11, 12]. Despite the differences, one thing in common is that after processing these methods, only the noise intensity is lowered but noise is not totally cleared with only cracks remained.

One of the characteristics of source image is that the crack appears as a large area of black pixels, and noise appears as isolated, scattered little black dots, therefore a threshold value T could be set, and then the number of black pixels in each black block can be calculated. The block will be preserved if the number of pixels is greater than T, else it will be eliminated if the number of pixels is less than T. The key to the denoising algorithm is the enumeration of black blocks. The whole image could be considered as a graph, a path exists between adjacent pixels with the same color [13]. A rule should be set to determine whether a collection of pixels is a black block or not: a black block should satisfy the condition that for any pixel inside the block, consider the pixels that locates up, down, left, or right to it, at least one of them should be black. According to this rule, Fig. **3**(a) has 1 black block, Fig. **3** (b)(c) has 2 black blocks.

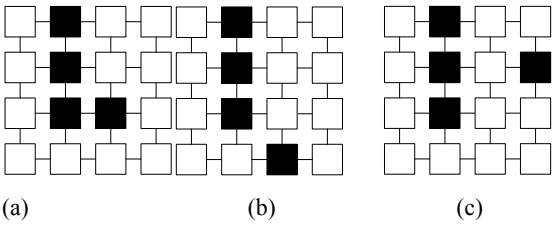


Fig. (3). Black block recognition.

Traversal of the graph can get all the connected components. Graph traversal means starting from any of the vertices in the graph, and visit all the vertices in the graph for only once [14]. As a basic operation of the chart, lots of oth-

er operations of the chart is based on the traversal operation. Recursive algorithm is implemented in traversal.

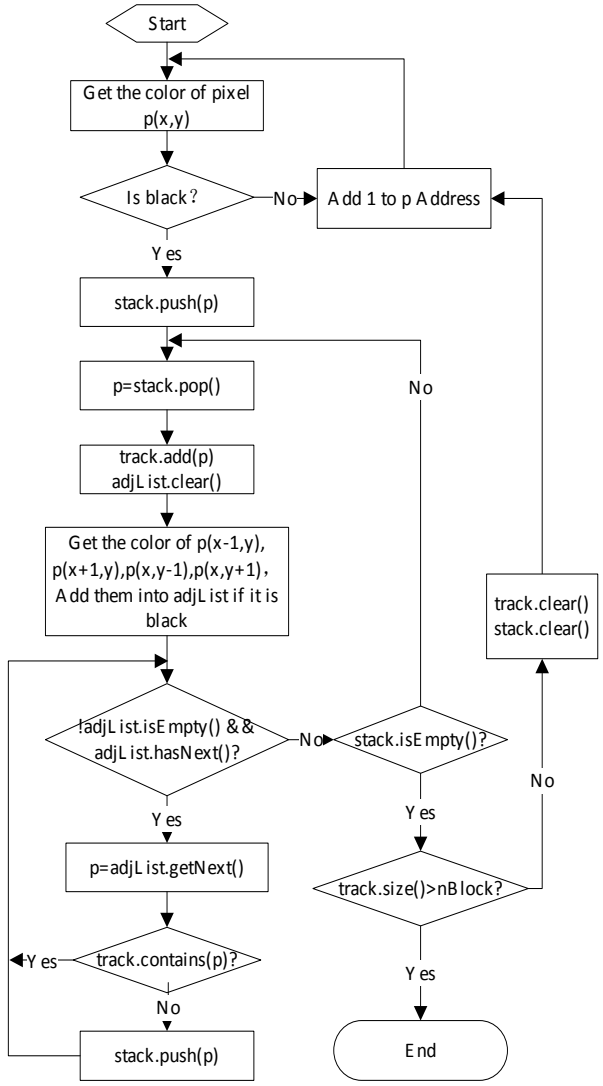
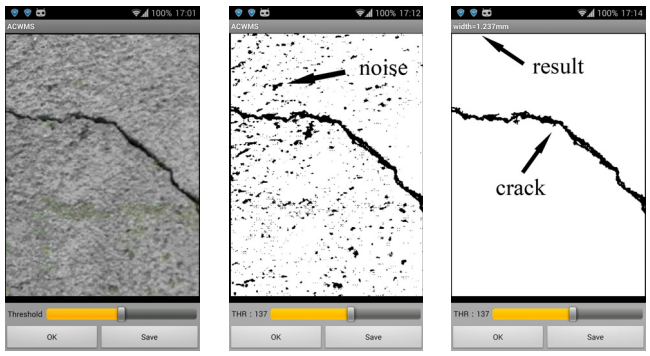


Fig. (4). Flow chart of the BFS traversal algorithm.

The overall idea of algorithm is: Create a global variable 'track' and a local variable 'stack', the type of track is Hashmap, while the type of stack is Stack. For every vertex in the graph, visit the adjacent vertices following the sequence of A->A right->A left->A up->A down, the current visited vertex should be added in the Linked list adjList if it is black, else it should be skipped. After all the adjacent vertices of A have been traversed, push the vertices that are not in 'track' onto 'stack'. After the traversal in order, if 'stack' is not empty, then pop out a vertex and add it into Linked list Track, and then continue traversal until the stack gets empty. Till now, the vertices in track form a complete black block.

Set a threshold value nBlock, clear the black block if the number of black pixels is less than nBlock, else keep it if the number is greater than nBlock. The value of nBlock is relatively wide, because the number of pixels in the crack is far greater than that in the isolated noise. A suitable range of nBlock can be acquired after several times of experiments, therefore the value can be fixed and written in the source code. Noise can be cleared after nBlock is determined, therefore, when searching for black blocks, if the number of pixels is less than nBlock, then do not proceed, if it is greater than nBlock, it indicates that the crack is found. Save the pixels of the crack, clear the source bitmap (set it to white) and draw the crack at the original position. Because the crack usually locates in the middle of the image, thus the scan should start in the middle of the image and should be continued if the crack is not found, the speed of crack finding can be further accelerated in this way. The flow chart of the algorithm is shown in Fig**. (4)**. Fig. **5**(c) shows the effect of elimination of isolation block.



(a) Raw image (b) Binary image (c)Denoised Image

Fig. (5). Image processing.

After denoising, only the crack remains in the image. By progressive scanning of the bitmap, the coordinate of the points on the left/right corner can be extracted and stored in lists named LeftPointList and RightPointList with the type of point. The flow chart is shown in Fig. **(6)**.

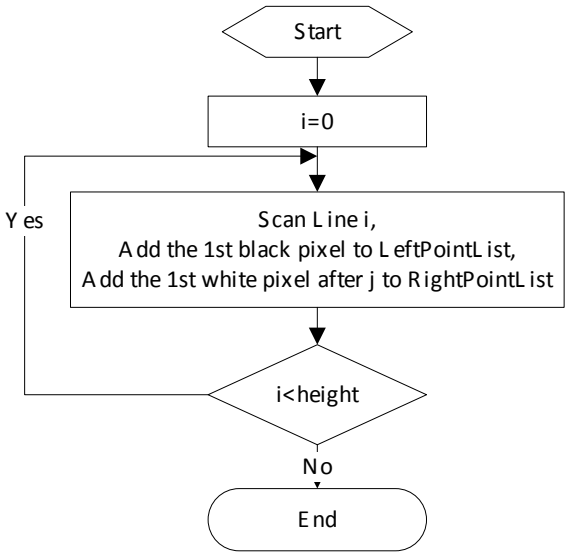


Fig. (6). Flow chart of crack extraction.

Assume the coordinate of the points on the left edge is (x_l, y_l), while for points on the right edge is (x_r, y_r), the distance between two points on the plane is [14]:

$$d = \sqrt{(x_r - x_l)^2 + (y_r - y_l)^2} \quad (8)$$

For any point on the left edge, traverse all points on the right edge using the above formula, set the value of crack width to the minimum value of d, namely d_{min}, the crack width is the maximum value of d_{min}, namely $max\ \{d_{min}\}$ [15]. Because the crack width at any point on the left edge is calculated by this point and a nearby point on the right edge, therefore when the point on right edge is traversed, only nearby area of the specified point on the left edge needs to be traversed. The flow chart is shown in Fig. **(7)**. The calculation result is displayed on the activity title as shown in Fig. **5**(**c**).

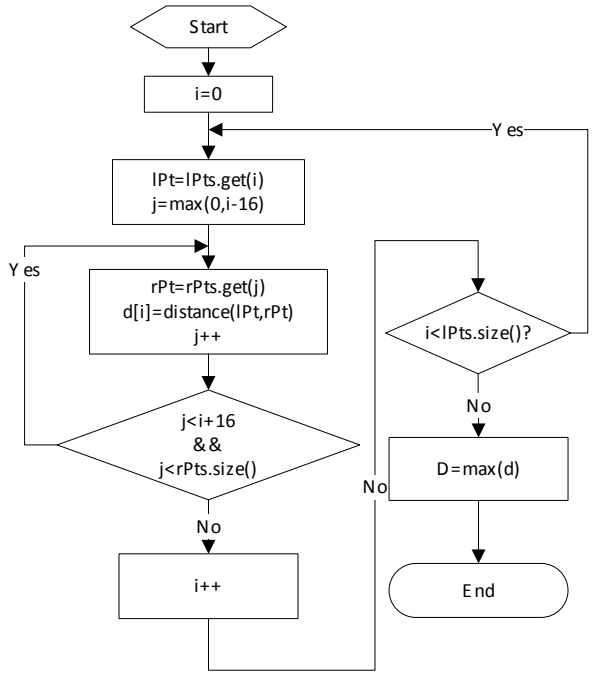


Fig. (7). Flow chart of width calculating.

The unit of crack width calculated above is pixel, the calibration coefficient is needed in the calculation of actual crack width. Because of the tiny size of the crack on the direction of width, the camera should be placed to the crack as close as possible in order to have the crack occupying more pixels in the image. However, restricted by the limit of the camera's focal distance, too close will end up in a blurry image and the light will be impeded thus affecting the image quality effect. The optimized distance should be 10cm between crack and camera after trial and error (shown as Fig. **8**). the calibration coefficient is also based on this distance [16].

Draw a 10cm line on the white paper using a black pen, place the camera at the top surface of a rectangular block with 10cm thickness while photographing the line. The coordinate of both ends of the line (x_1,y_1) and (x_2,y_2) should be determined by the system, and the length of the line in bitmap could be calculated as follows

$$d_p = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (9)$$

Then the calibration coefficient a will be:

$$a=100/d_p \quad (10)$$

the crack's actual width will be

$$D=a*d_p \quad (11)$$

The detection of the width of 10 cracks on a cracked concrete beam has been done. To each of the 10 cracks, the crack width is measured by Samsung Galaxy S3 mobile phone with an integrated 8-million pixel camera. The results are compared with the done by WYSX-40X crack detector. The main technical parameters of WYSX-40X includes: the amplication factor: 40, the high value of range: 4mm, the minimum scale: 0.05mm. Comparison results are displayed in Table **1** and Fig. (**9**).

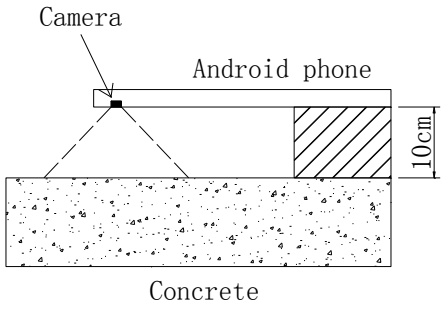


Fig. (8). Schematic of capturing.

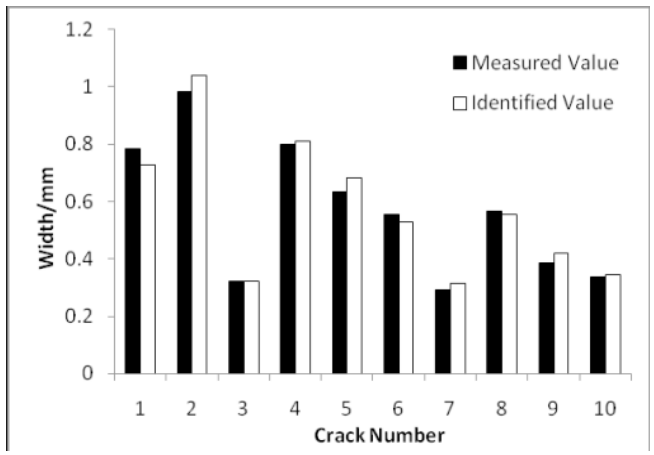


Fig. (9). Result comparison between identified value and measured value.

It is known from the result that the mean absolute error is 0.027mm, the mean relative error is 4.74%; identification accuracy reaches 95.26%; the standard deviation of absolute error is 0.021mm; the standard deviation of relative error is 3.03%. Identified values agree well with measured values, with precision high enough to meet the engineering requirements.

CONCLUSION

From the results above, conclusions can be drawn as follows:

(1) The ACWMS developed in this paper uses android device to capture crack image and directly determines the

Table 1. Comparison between measured value and identified value of crack width.

Image No.	Measured Value (mm)	Identified Value (mm)	Absolute Error (mm)	Relative Error (%)
1	0.785	0.726	0.059	7.54%
2	0.982	1.040	0.058	5.87%
3	0.324	0.322	0.002	0.71%
4	0.800	0.809	0.009	1.11%
5	0.635	0.681	0.046	7.21%
6	0.555	0.528	0.027	4.83%
7	0.292	0.313	0.021	7.30%
8	0.567	0.557	0.010	1.85%
9	0.385	0.419	0.034	8.78%
10	0.336	0.343	0.007	2.16%
Average			0.027	4.74%
Standard Deviation			0.021	3.03%

maximum crack width on the image after a series of processing including graying, binaryzation, and denoising, which performs well in detecting cracks on concrete surface. Test result shows that the identification accuracy of the system reaches 95.26%, which is high enough to meet the engineering requirements.

(2) Using this system, crack width can be easily calculated with high efficiency and convenience, the only requirement is a portable device with Android system. Besides, it is a good way to apply ACWMS to the measurement practice that requires no contact.

CONFLICT OF INTEREST

The authors confirm that this article content has no conflict of interest.

ACKNOWLEDGEMENTS

Declared none.

REFERENCES

[1] F. Zhi, and P. Hai-tao, "Crack width detection on the concrete surface of bridge based on image analysis technology",*Journal of Hunan University (Natural Sciences)*, vol. 39, no. 1, pp.7-12, 2012.

[2] Z. Hong-yi, Z. Qiang-xin and Y. Hai-qing, "Concrete structural damage apparent crack identification and model research", *Digital Technology and Application*, vol. 1, pp. 139-141, 2009.

[3] C. Li-hua, and D. Zhi-xue, "Design and implementation of crack width detection system based on android", *Computer Engineering and Design*, vol. 34, no. 9, pp. 3195-3199, 2013.

[4] Y. Wenjun, S. Xiaomei, "The image acquisition system based on android", *Journal of Xi'an polytechnic University*, vol. 26, no. 4, pp. 494-501, 2012.

[5] S. Rui-hong, "The application of camera interface on android", *Computer Programming Skills & Maintenance,* vol. 23, pp. 10-14, 2013.

[6] E. Burnette, "Hello android 3rd edition", *Pragmatic Programmers*, LLC, North Carolina, 2010.

[7] Y.. Da-zhang, Z. Jian-gang, and G. Yong, "A way of graying in the image of a bitmap and the realization of programming", *Journal of Guangxi University of Technology*, vol. 15, no. 1, pp. 23-26, 2004.

[8] J. Ming, L. Hui, and H. Huan, "Study on image binarizing", *Software Guide*, vol. 8, no. 4, pp. 175-177, 2009.

[9] Q. Li-na, Z. Bo, and W. Zhan-kai, "Application of OTSU method in image processing", *Radio Engineering*, vol. 36, no. 7, pp. 25-26, 2006.

[10] C. Liang, "The study of image segmentation algorithm based on OTSU Method", *Wuhan University of Technology*, 2008.

[11] L. Hui, "Research on Image Denoising Based on Wavelet Transform", *Hunan Normal University*, 2012.

[12] P. Gu, "Method research on image denoising and edge detection", *Central South University of Technology*, 2012.

[13] M.A. Weiss, "Data structures and algorithm analysis in C++", *Addison Wesley*, vol. 3, 2006.

[14] T. H. Cormen and C. E. Leiserson, "Introduction to algorithms", The MIT Press, London, vol. 3, 2006.

[15] D. Rui-ling, and L. Qing-xiang, and L. Yu-he, "Summary of image edge detection", *Optical Technique*, vol. 31, no. 3, pp. 415-419, 2005.

[16] R. Wen-jie, "The research on methods of edge detection", *Shandong University*, 2008.

Received: May 26, 2015 Revised: July 14, 2015 Accepted: August 10, 2015