A Project Report

On

Crack detection using Image processing

BY

BALAJI SAI KUMAR B

15XJ1A0108

Under the supervision of

DILIP KUMAR, PRAFULLA

SUBMITTED IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS OF

SE 422: PROJECT TYPE COURSE

MAHINDRA ECOLE CENTRALE

COLLEGE OF ENGINEERING

HYDERABAD

(NOVEMBER 2018)

Mahindra Ecole Centrale

Hyderabad

Certificate

This is to certify that the project report entitled "**Crack detection using Image processing**" submitted by Mr. BALAJI SAI KUMAR B (ID No. 15XJ1A0108) in partial fulfillment of the requirements of the course SE421/SE422, Project Course, embodies the work done by him/her under my supervision and guidance.

(SUPERVISOR NAME & Signature)	**(EXTERNAL NAME & Signature)**
Mahindra Ecole Centrale, Hyderabad.	Mahindra Ecole Centrale, Hyderabad.
Date:	22-11-2018

Acknowledgement

I am deeply obliged to my mentor Prof. Dilip Kumar and Prof. Prafulla for having given me the opportunity to undertake Crack detection using Image processing as my final year phase-1 project, for providing me with valuable guidance and necessary help for my successful completion of training I would also extend my gratitude to the institution. It has been an excellent learning experience for me as I could get to know the pertinent issues related to cracks and their effect on the structure. I have been immensely benefited by the exposure gained in the field of Structural health monitoring under Prof. Dilip and Prof Prafulla.

I would like to convey my sincere thanks to Mr. Prabhakar, Head of Department. I sincerely extend my gratitude to the director of the Institute.

Abstract

The earliest indications of degradation of the structure is formation of Cracks on the concrete surface which is critical for the maintenance as the continuous exposure will lead to the severe damage. Manual inspection is the acclaimed method for the crack inspection. In the manual inspection, the sketch of the crack is prepared manually, and the conditions of the irregularities are noted. Since the manual approach completely depends on the specialist's knowledge and experience, it lacks objectivity in the quantitative analysis. So, automatic image-based crack detection is proposed as a replacement. Literature presents different techniques to automatically identify the crack and its depth using image processing techniques. Different algorithms have been tested on an image containing cracks and the modules by MATLAB has been used in order to obtain the algorithm used by MATLAB and have been compared. Some algorithms threshold limit has been manipulated and different functions have been tested to obtain an output with noise filtered.

Contents

0.1 Introduction

A structure when subjected to stress undergoes deformation. Excessive deformation leads to formation of cracks. These cracks play vital role in designing the member. Crack propagation can occur due to various reasons. A structure's strength is compromised when it starts cracking. The entire design of reinforced concrete is based on the fact that the concrete when in tension develops cracks and then will not be able to take the load and the steel reinforcement will be used to take the load in tension. Formation of crack will declare the member to be unfit and cause discomfort among the residents even though the member can still carry the load without failing. Detecting the crack before it fails is vital and will reduce the risk of failure. Cracks can range from a thousandth of μ to few mm. Image processing aids at finding the crack width, length and all other data accurately just by using a simple algorithm. Few drones have been developed to obtain few photographs and then identify the crack without the sight engineer evaluate it. Few theories of photometry are used.
The total cost of maintenance, rehabilitation, and replacement of a structure is related to the average age of the entire the structure. Many factors tend to accelerate the deterioration of structure; for instance, for bridge truck weights and traffic counts have increased dramatically, causing premature physical weathering on all networks, while available funds for MR&R have not been able to keep up with the declining situation.

Need

Engineering structures like concrete surface, beams are often subjected to fatigue stress, cyclic loading, that leads to the cracks that usually initiate at the microscopic level on the structure's surface. Not only these cracks cause material discontinuity but also reduce the material stiffness of the material. To prevent prevent damage and possible failure, early detection allows to take measures. Crack detection is the process of detecting the crack in the structures using any of the processing techniques. The two ways crack detection can be classified is Destructive Testing and Non-Destructive testing. The objective of the type, number, width and length of the cracks on the structural surface shows the earliest degradation level and carrying capacity of the concrete structures Below image shows the process of obtaining required parameters.
Image acquisition ▶ Pre processing ▶ Image processing ▶ Crack detection ▶ Parameters estimation

Image acqusition

The automated crack detection can be done using some of the Non-destructive testing techniques like
(i) Infrared and ther- mal testing,
(ii) Ultrasonic testing,
(iii) Laser testing, and
(iv) Radiographic testing
There is an increasing interest in image-based crack detection for non-destructive inspection. Some of the difficulties in the image based detection are because of the random shape and irregular size of cracks and various noises such as irregularly illuminated conditions, shading, blemishes and concrete spall in the acquired images. Because of its simplicity in the processing, many of the image processing detection methods were proposed. These methods are classified into four categories, namely integrated algorithm, morphological approach, percolation-based method, and practical technique

0.1.1 Theories in Image processing

The image processing is a vast subject, and some of the theories are applied to obtain crack image. Below are some of the theories that have been used and their packages given in OpenCV

0.1.2 Kernel convolution

Image is a matrix which has the location of a pixel along with intensities in three bands of colours. The image on convolution with the kernel matrix produces the desired effect. A kernel is usually a 3X3 matrix and higher and it is operated on pixel and its adjacency to obtain certain modifications. Below are some matrices given which are used as kernel matrix for convolution. For mean blur,

$$k = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

For Gaussian blur

$$k = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

0.1.3 RGB to Grayscale Image

The image is basically a two dimensional array with all the information of the location of the pixel stored as rows and columns and the information about the intensities of the three bands of the colour is linked to those location of the pixel. It is basically considered as a cell, and this will be extracted in image and this data is converted into grayscale where only one band of intensities will be present. This is obtained using a formula given by OpenCV. The formula to convert from RGB image to grayscale image is given below

$$gray = 0.299 * r + 0.587 * g + 0.114 * b. \tag{1}$$

This method is called weighted average method to obtain the intensities in grayscale.

0.1.4 Sobel method

The Sobel edge detector is known for its simplicity and speed, compared to other algorithms that are computationally complex, and is based on the spatial gradient algorithm. However, Sobel is very susceptible to image noise, which may result in false positive detections when attempting to identify edges in real-world situations. This is inherent in all gradient-based algorithms. Filtering to smooth the noise out can reduce the false edge identification but does not eliminate the problem The operator uses two 3X3 kernels which are convolved with the original image to calculate approximations of the derivatives – one for horizontal changes, and one for vertical. If we define A as the source image, and Gx and Gy are two images which at each point contain the vertical and horizontal derivative approximations respectively, the computations are as follows:-

$$Gx = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

$$Gy = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

These matrices are convoluted with the matrix elements and the value is stored in the pixel. At each point in the image, the resulting gradient approximations can be combined to give the gradient magnitude, using:

$$G = \sqrt{Gx^2 + Gy^2} \tag{2}$$

Using this information, we can also calculate the gradient's direction.

$$\theta = \arctan(\frac{Gy}{Gx})$$

0.1.5 Canny method

The Process of Canny edge detection algorithm can be broken down to 5 different steps:
Apply Gaussian filter to smooth the image in order to remove the noise
Find the intensity gradients of the image
Apply non-maximum suppression to get rid of spurious response to edge detection
Apply double threshold to determine potential edges
Track edge by hysteresis: Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.

0.2 MATLAB packages

The image processing has been carried out with the help certain MATLAB packages. These packages have been addressed below.
Bwboundaries:- This package returns the boundaries of object present in the image. This package returns the rows, columns of the pixels which correspond to the boundaries of the image.
Bwlabel:- bwlabel(BW,N) returns a matrix L, of the same size as BW, containing labels for the connected components in BW. N can have a value of either 4 or 8, where 4 specifies 4-connected objects and 8 specifies 8-connected objects; if the argument is omitted, it defaults to 8.
Imtool:- displays the binary image BW. Values of 0 display as black, and values of 1 display as white.
Graythresh:- computes a global threshold (LEVEL) that can be used to convert an intensity image to a binary image with IMBINARIZE.
Im2bw:- converts an image from grayscale to binary image. it binarizes the image
Imfill:- Fill image regions and holes.
bwareopen:- Remove small objects from binary image.
Improfile:- Pixel-value cross-sections along line segments.
regionprops:- Measure properties of image regions.

0.3 Results

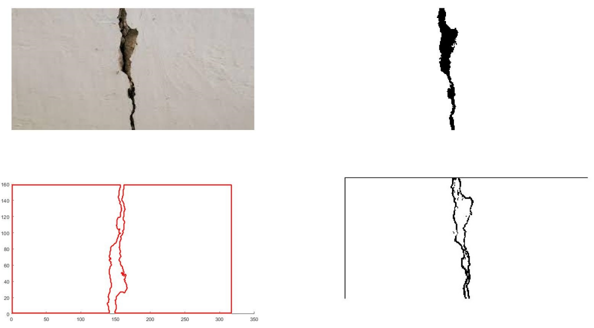

Figure 1: original image, Binary image, Boundary of the object in the image, image obtained after processing using my code

The above Fig.2 is the result obtained after image has been processed. The below image is for another image. The length of the crack has been mention in the last page which includes code.

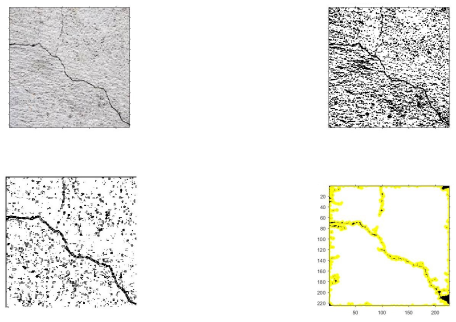

Figure 2: original image, Binary image, image obtained after processing using my code, Boundary of the object in the image,

0.4 Conclusion

Certain algorithms provide better image of cracks over the other. I have personally developed a threshold value formula which will scape out all the unnecessary data and leave out the important ones while binarizing the image (noise filtering). The formula has been made using trial and error process

in the starting stage of the project.
The error function has been used as a threshold to remove intensities below that threshold. It basically functions as a low pass filter to the image intensities. The mean and standard deviation of image intensities in error function is used to obtain the threshold.

$$threshold = erf(\frac{\mu}{\sigma}) \tag{3}$$

where μ is the mean of image intensities and σ is mean deviation of the image intensities. This formula scrapes out all the unnecessary data, which can be considered as noise in the image. The cracks that will be removed will be relatively smaller when compared to the cracks obtained after the filtering.
Drawbacks:- This code doesn't work for images which is larger in size(image dimensions nominal:- <500*500). The larger the size of the image, the larger is the amount of time taken to obtain the data. The image data is skewed if it is taken in low light conditions. Code has to be improved in order to obtain results for all type of image resolutions.
Obtaining the crack width and detection of the crack width becomes difficult as the size of the crack decreases. An image of the wall in MEC dorms has been taken to demonstrate the problems due to smaller cracks and usage of algorithms like this. The cracks have not been observed in the binary image when compared to other images which have cracks visible to naked eye.

0.5 Plan ahead

Using python and OPENCV packages and develop a kernel operator which will blur the images so as to obtain the cracks with desired width.
Comparing the results with different algorithms used to obtain the length of the crack.
Developing a function like bwboundaries(package present in MATLAB) to obtain boundaries.
Increasing the efficiency of in binarising the image.
Using IS456 and other code of provisions and identifying allowable cracks formation and detecting a crack and alarming as soon as the crack width and length crosses nominal crack width and length.

0.6 References

Mohan, A., &Poobal, S.(2017). *Crack detection using image process: A critical review and analysis.* Alecandria Engineering Jounral

Ikhlas Abdel-Qader, P.E., Osama Abudayyeh, P.E., M.ASCE and Michael E. Kelly *Analysis of Edge-Detection Techniques for Crack Identification in Bridges*

MATLAB Code

```
clc;clear all;
originalImage =imread('D:\downloads\education\project\final year project\1.jpg');%reads image
greyimg=im2double(rgb2gray(originalImage));%coverts into black and white
a=max(size(greyimg));
b=min(size(greyimg));
k1=[1 0 -1; 2 0 -2; 1 0 -1];
k2=[1 2 1;0 0 0; -1 -2 -1];
img=[];%modified image matrix
imgf=[];
for i=2:b-2
    for j=2:a-2
        img=greyimg(i-1:i+1,j-1:j+1);
        img1=sum(sum(k1.*img));
        img2=sum(sum(k2.*img));
        img=sqrt(img1^2+img2^2);
        imgf(i,j)=img;
    end
end
for i=2:b-2
    for j=2:a-2
    if imgf(i,j)<erf(mean(mean(imgf))/std(std(imgf)))%i tried to look at the normal distribut
ion and find an upper limit.(a random guess)
            imgf(i,j)=1;
        else
            imgf(i,j)=0;
    end
    end
end
subplot(2, 2, 4);
imshow(imgf);
hFig = figure;
subplot(2, 2, 1);
imshow(originalImage, []);
[rows, columns, numberOfColorBands] = size(originalImage);
if numberOfColorBands > 1
        grayImage = rgb2gray(originalImage); % Take green channel.
else
        grayImage = originalImage;
end
level = graythresh(grayImage);
binaryImage = im2bw(grayImage, level);
binaryImage = imfill(binaryImage, 'holes');
binaryImage = bwareaopen(binaryImage, round(0.1*numel(binaryImage)));
labeledImage = bwlabel(binaryImage);
% Measure the area
measurements = regionprops(labeledImage, 'Area');
boundaries = bwboundaries(binaryImage);
numberOfBoundaries = size(boundaries, 1);
for blobIndex = 1 : numberOfBoundaries
        thisBoundary = boundaries{blobIndex};
        x = thisBoundary(:, 2); % x = columns.
        y = thisBoundary(:, 1); % y = rows.
    maxDistance = -inf;
    for k = 1 : length(x)
                distances = sqrt( (x(k) - x) .^ 2 + (y(k) - y) .^ 2 );
                [thisMaxDistance, indexOfMaxDistance] = max(distances);
        if thisMaxDistance > maxDistance
                        maxDistance = thisMaxDistance;
                        index1 = k;
                        index2 = indexOfMaxDistance;
        end
    end
    xMidPoint = mean([x(index1), x(index2)]);
        yMidPoint = mean([y(index1), y(index2)]);
        longSlope = (y(index1) - y(index2)) / (x(index1) - x(index2))
        perpendicularSlope = -1/longSlope
    y1 = perpendicularSlope * (1 - xMidPoint) + yMidPoint;
        y2 = perpendicularSlope * (columns - xMidPoint) + yMidPoint;
        [cx,cy,c] = improfile(binaryImage,[1, columns], [y1, y2], 1000);
        c(isnan(c)) = 0;
        firstIndex = find(c, 1, 'first');
        lastIndex = find(c, 1, 'last');
        perpendicularWidth = sqrt( (cx(firstIndex) - cx(lastIndex)) .^ 2 + (cy(firstIndex) -
cy(lastIndex)) .^ 2 );
        averageWidth = measurements(blobIndex).Area / maxDistance;
end
subplot(2, 2, 2);
imshow(binaryImage);
subplot(2, 2, 3);
visboundaries(binaryImage);
```

```
longSlope =

    1.0128


perpendicularSlope =

   -0.9873


longSlope =

    0.9405


perpendicularSlope =

   -1.0633
```