Development of an automatic tool for defect detection on ultrasonic testing images

Semester Project

Presented by

Youness El Houssaini

Born in Ouled Teima

University of Strasbourg

Faculty: UFR mathematics and informatics

Major: scientific computing and mathematics of information

Examiner: Dr. Caldini Queiros Céline

Freiburg Im Breisgau

Mai 2019

Confidentiality

The present work entitled

"Development of an automatic tool for defect detection on ultrasonic testing images"

Contains confidential data of the Fraunhofer Institute for Mechanics of Materials IWM.

The work may only be inspected by the first and second supervisors and is available for authorized members of the selection board.

A publication and duplication of the work is not permitted - even in excerpts.

An inspection of the work by unauthorized persons requires the explicit approval of the author and the Fraunhofer Institute for Mechanics of Materials IWM.

Abstract

Within the scope of the present work, a defect detection tool has been developed in order to assess the formed defects occurred during the manufacturing process of forged components. This tool is based on data derived out of available Ultrasonic Testing images. The tool defines position of defects and their shape in order to map the mechanism of fatigue damage. For this purpose, some approximations has been made from an expertise point of view, thus only severe damages has been treated within this work. Also, the accuracy of the detection tool is based on a fixed parameters e.g. the color threshold and the extrema of each defect.

The detection tool is developed partially using Python coding language with its image processing package Opencv for image pre-processing. Besides, some useful functions from the lecture of Dr. Vigon, Vincent. “Signal and graph” were implemented and further developed for image processing.

In this work it is shown that the detection tool gives better results when the UT images contains separated single defects and loses its accuracy when the defects are crossing each other in the given image. This crossing phenomenon is due to including defects of inner layers down the component and not only the outer surface.

Foreword

This report is part of the study program at the University of Strasbourg in the UFR Mathematics and informatics within the specialization of scientific computing and mathematics of information, CSMI. The work is considered as a fulfilment of the lecture "Project" taught by Dr. Céline Caldini Queiros and thanks to her priceless advices. It took place in the Fraunhofer Institute for Mechanics of Materials IWM from March 2019 to Mai 2019 (180 working hours). The topic was accepted as a semester project thanks to the agreement of the study program (CSMI) manager Prof. Dr. Prud'homme Christophe, to whom all my gratitude.

First and foremost I would like to thank the Fraunhofer Institute for Mechanics of Materials IWM which gave me this opportunity of enhancing both academic and professional knowledge. In addition, I would like to express my special thanks to my supervising professors, Dr. Vigon Vincent and Dr. Yannick Privat for the exemplary support by the University of Strasbourg. Also, my gratitude to my project supervisor M.sc. Aydin Ali, and Dipl.-Ing. Dittmann Florian for their enduring help and helpful insight.

Furthermore, my thanks to all employees in the project team for the helpfulness and friendliness shown to me during my work. The working conditions and the atmosphere I found extremely positive.

Contents

II. Nomenclature

Symbols	Meaning	Unit
l	length of the component	mm
w	Width of the component	mm
h	height of the component	mm
r	Radius of a disc-defect	MPa
(x, y)	The horizontal plan of the component	$-$
(x, z)	The vertical plan of the component	$-$
(i , j)	Coordinate of each pixel	$-$
n	Number of pixels in a single defect	$-$

III. Abbreviation

Abbreviation	Meaning
UT	**U**ltrasonic **T**esting
OpenCV	**O**pen source **C**omputer **V**ision
RGB	**R**ed **G**reen **B**lue
HSV	**H**ue **S**aturation **V**alue
LCF	**l**ow **c**ycle **f**atigue
maxVa	**Max**imum **Val**ue

1 Introduction

This chapter explains the motivation of this work. Furthermore, follows the concrete description of the task with the definition of the goals to be achieved and the prescribed plan of work.

1.1 Context

In mechanical engineering, the safe design of components is one of the most fundamental tasks, as their failure can have serious consequences. Known cases of damage in the recent past, such as for example, the catastrophic ICE accident in 1998 in Eschede and the tanker accidents of the tankers Erika 1999 and Prestige 2002 show that there is considerable need for research on the safe design of components. [1] Material fatigue plays a very important role in dynamically stressed components.

In fact, during the manufacturing process of forged components, the formation of defects is unavoidable. The failure rate of structural components is significantly influenced by these defects which can be detected by non-destructive methods. The aim of the overall research project is the optimization of the lifetime prediction of structural components by the consideration of nucleation process of forging defects. For this purpose low cycle fatigue (LCF) testing until fracture is carried out at specimens which are extracted from forged components in a way that a defect is positioned in the center. The fracture surfaces are then examined by means of scanning electron microscope (SEM) and energy dispersive X-ray spectroscopy (EDX). Within this procedure the defects can be recognized on the fracture surface. This test method provides images for probable defects as well as the chemical composition. The evaluation of these defects in the images is currently carried out manually and is very time-consuming, which shows the need of automation.

1.2 The task

The scope of the project is to establish an automatic tool for the detection as well as the classification of defects in fracture surfaces via UT images. This will have to be done in an iterative workflow including image pre-processing, definition of defects features and output

data processing. The programming language will be Python. As result, the tool should be able to detect defects in UT images and define its size and shape as well as the coordinates of its position so that a statistical distribution could be determined.

1.3 Methodology

The work has been conducted within a group of four members communicating the results of each other regularly via an internal network. In the process of developing the detection tool, iterative and incremental framework for project management was essential. Based on the Scrum approach the following workflow was made in order to manage meeting the time schedule objectives (see Figure 1).

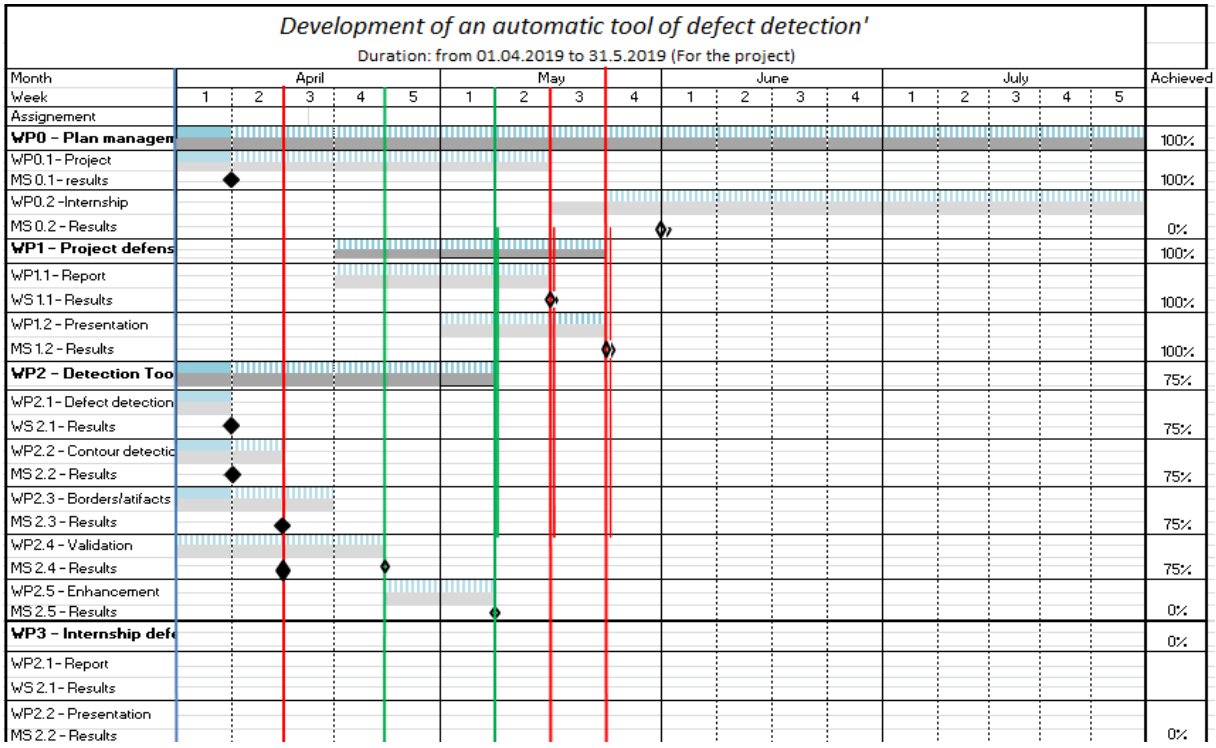

Figure 1 Screenshot of the project management plan

This method defines "a flexible, holistic product development strategy where a development team works as a unit to reach a common goal",[2] challenges assumptions of the "traditional, sequential approach"[2] to product development, and enables teams to self-organize by encouraging physical co-location or close online collaboration of all team members, as well as daily face-to-face communication among all team members and disciplines involved.

2 Encountered problems

In this chapter all obstacles in the process of developing the automatic detection of defects are explained. After selecting a suitable solution, the programmed tool is implemented in python coding language and presented in the following chapter. In particular, this chapter presents encountered problematics from the pre-processing of the UT images until the output data extraction.

2.1 UT image

It's similar to ultrasound imaging (sonography) which uses high-frequency sound waves to view inside the human body. UT imaging technique does not have only medical application but also is used in material science, namely in this project. It is used to create an image of internal material structures in order to assess the lifetime prediction of structural components by the consideration of nucleation process of forging defects. Its aim is often to show potential probable defects to exclude unsafe components. For a better insight, an example of UT image is presented below (see Figure 2).

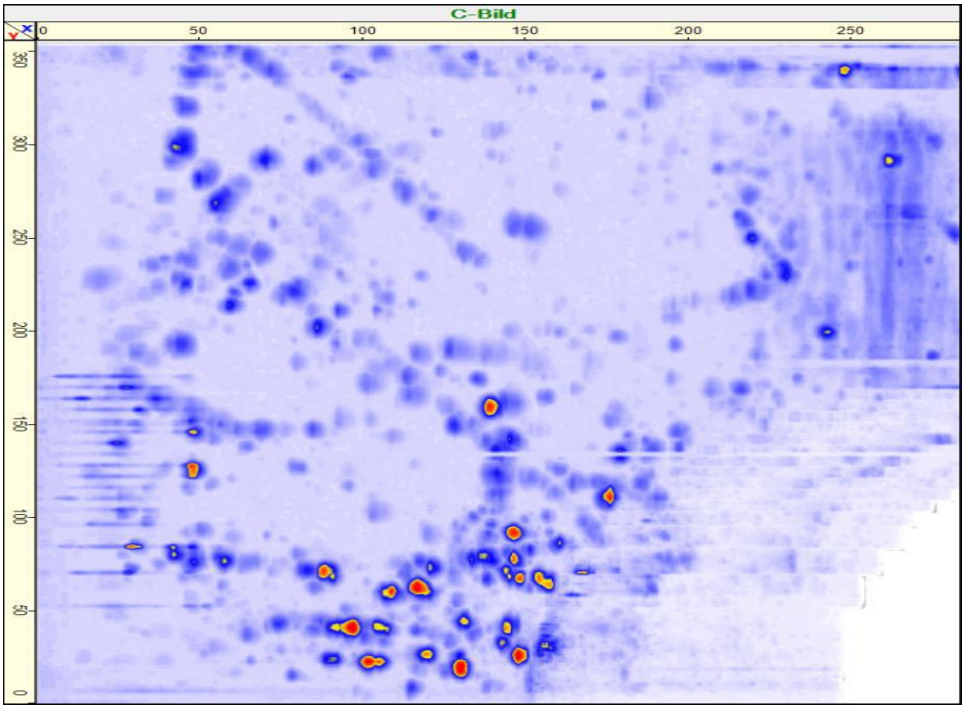

Figure 2 UT image of a project tested component [6].

2.2 Image pre-processing

The UT image preprocessing is carried out mainly using OpenCV. This is a library of programming functions mainly aimed at real-time computer vision. This library has been used in every step for pre-processing the UT images. The following chart illustrates the different steps before having the output image ready for data extraction (see Figure 3).

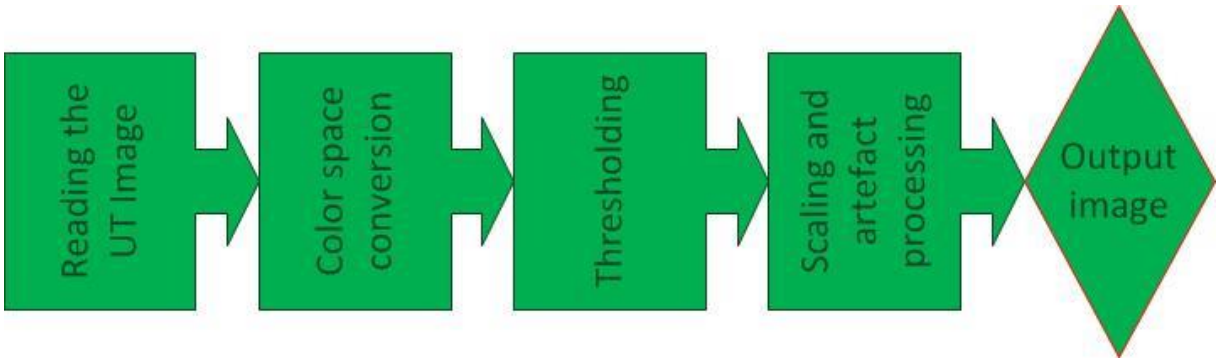

Figure 3 Scheme of UT image pre-processing.

2.2.1 Color space

In order to read the UT image and reproduce it in a digital form a color space has to be chosen first. Since "color space" identifies a particular combination of the color model and the mapping function, the word is often used informally to identify a color model. A "color model" is an abstract mathematical model describing the way colors can be represented as tuples of numbers (e.g. triples in RGB or quadruples in CMYK). When defining a color space, the usual reference standard is the CIELAB or CIEXYZ color spaces, which were specifically designed to encompass all colors the average human can see. [3]

Why HSV?

HSV are alternative representations of the RGB color model, designed in the 1970s by computer graphics researchers to more closely align with the way human vision perceives color-making attributes. In these models, colors of each hue are arranged in a radial slice, around a central axis of neutral colors which ranges from black at the bottom to white at the top. This “color model” allows a better manipulation of the colors using a single parameter called “Hue” (see Figure 4b). This is a key parameter in the UT image thresholding in the contrary to other “color models” when more than parameter is required. The RGB for instance

is a combination of three parameters or colors to determine a new resulting color (see figure 4a).

(a) **(b)**

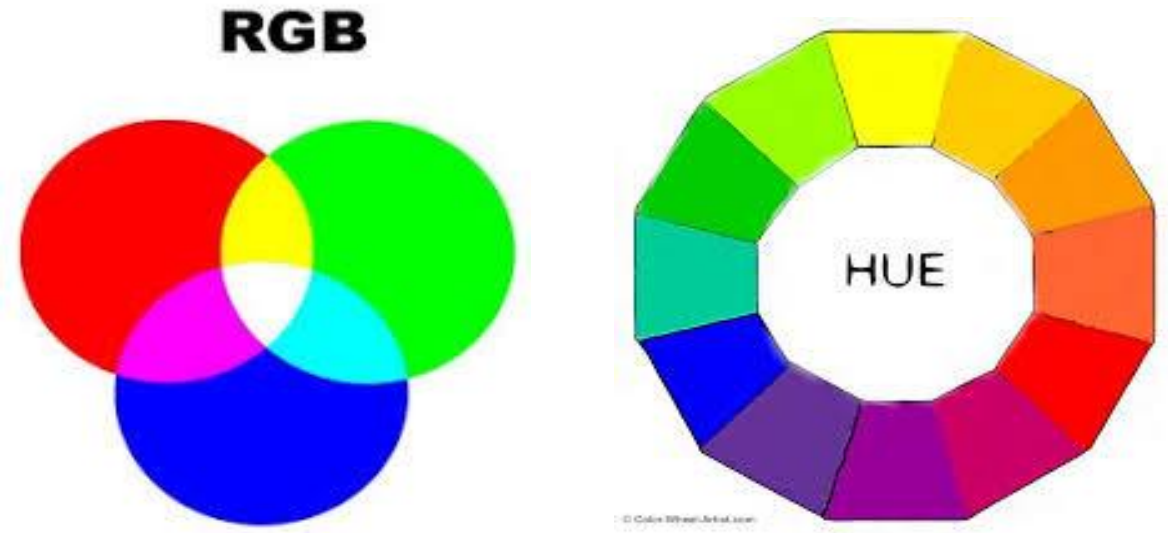

Figure 4 (a) RGB color space (b) Hue of HSV color space

Once the UT image is converted to HSV color model the yellow and red spots can be easily manipulated according to a certain value of the hue called “threshold”.

2.2.2 Image thresholding

The OpenCV library offers many thresholding functions which will be defined and compared in this section.

Simple Thresholding

Here, the matter is straight forward. If pixel value is greater than a threshold value, it is assigned one value (Black in our case), else it is assigned another value (no change in our case). The function used is cv.threshold. First argument is the source image, which should be a grayscale image. Second argument is the threshold value which is used to classify the pixel values. Third argument is the maxVal which represents the value to be given if pixel value is more than (sometimes less than) the threshold value. [4]

The following figure (See figure 5) is a result of the mentioned method above and the used code is in the appendices chapter (See figure 15).

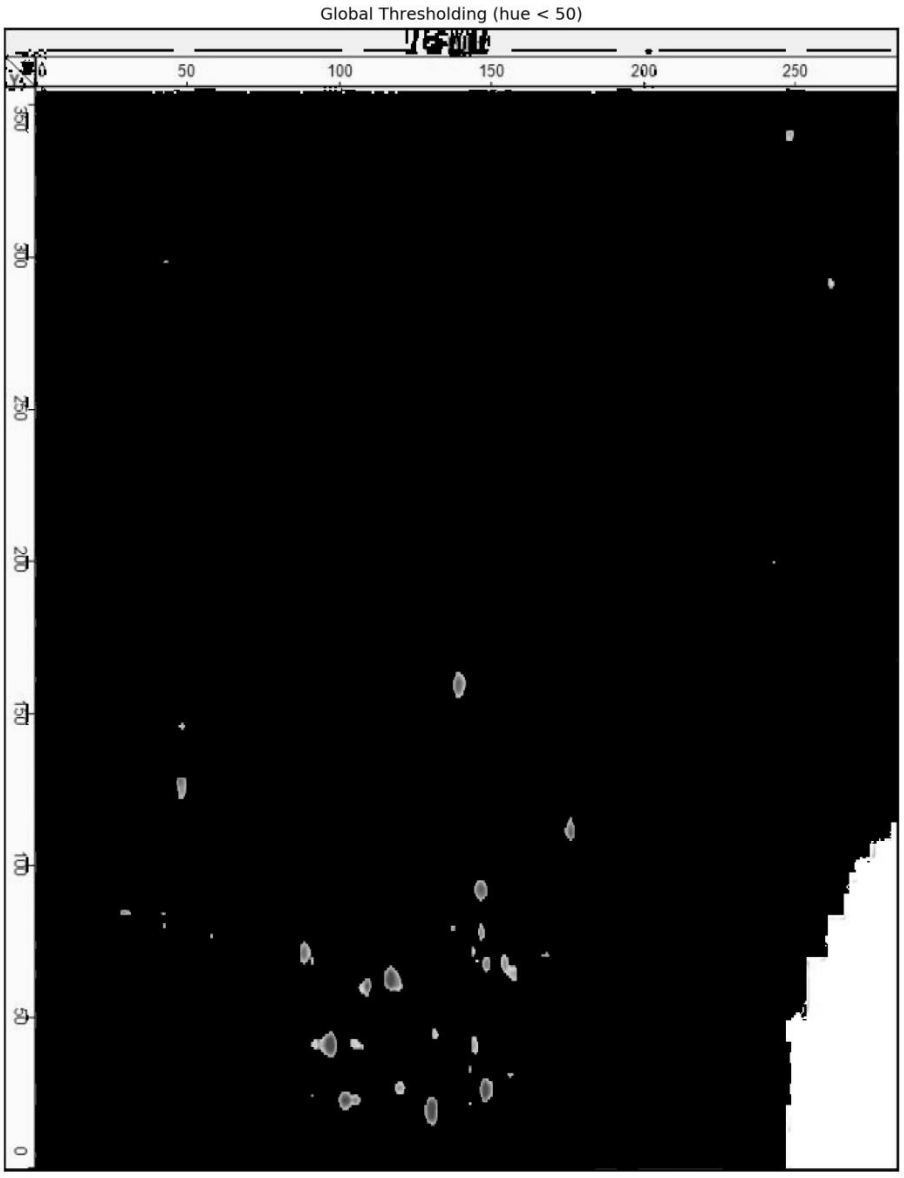

Figure 5 Obtained UT image after simple thresholding

Color space conversion is the translation of the representation of a color from one basis to another. This typically occurs in the context of converting an image that is represented in one color space to another color space, the goal being to make the translated image look as similar as possible to the original.

Adaptive Thresholding

In the previous section, we used a global value as threshold value. But it may not be good in all the conditions where image has different lighting conditions in different areas. In that case, we go for adaptive thresholding. In this, the algorithm calculates the threshold for small regions of the image. So we get different thresholds for different regions of the same image and it gives us better results for images with varying illumination.

It has three ‘special’ input parameters and only one output argument.

Adaptive Method - It decides how thresholding value is calculated.

- cv.ADAPTIVE_THRESH_MEAN_C : threshold value is the mean of neighbourhood area.
- cv.ADAPTIVE_THRESH_GAUSSIAN_C : threshold value is the weighted sum of neighbourhood values where weights are a gaussian window.

Otsu’s Binarization Thresholding

In global thresholding, we used an arbitrary value for threshold value. So, how can we know a value we selected is good or not? Answer is, trial and error method. But consider a bimodal image (In simple words, bimodal image is an image whose histogram has two peaks). For that image, we can approximately take a value in the middle of those peaks as threshold value. That is what Otsu binarization does. So in simple words, it automatically calculates a threshold value from image histogram for a bimodal image. (For images which are not bimodal, banarization won’t be accurate.) [4]

Adopted solution

In the following figure a comparison of the obtained images after using the three thresholding types mentioned above (see Figure 6). For better resolution, please check the appendices chapter.

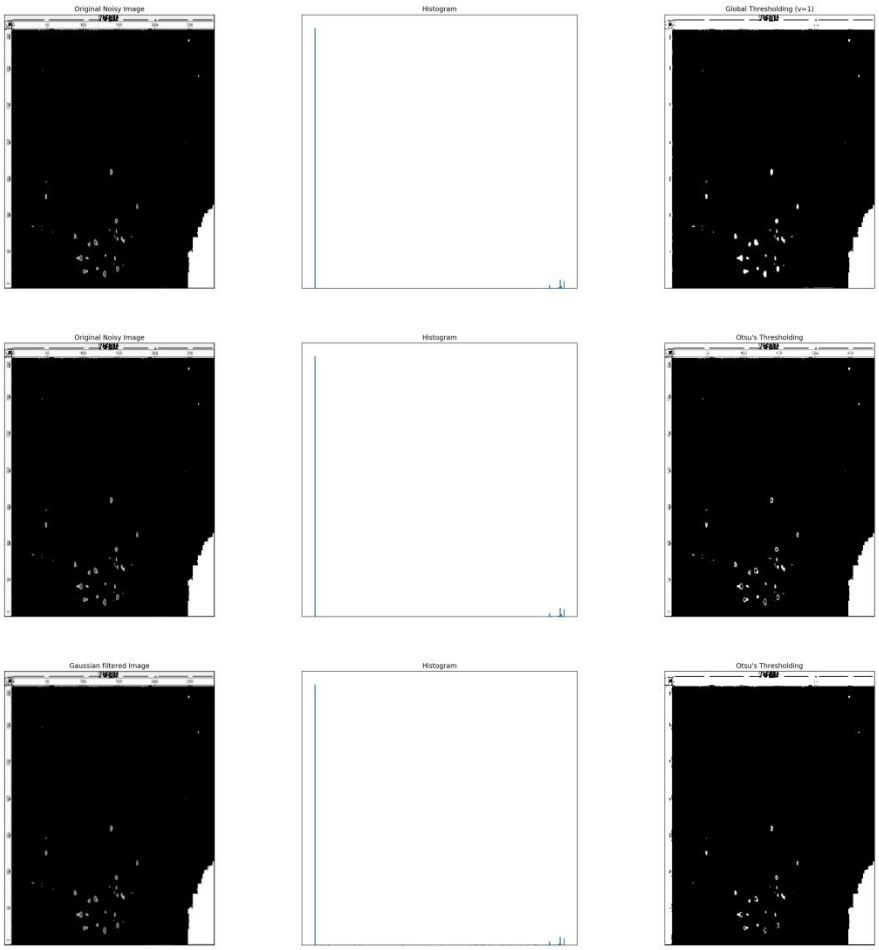

Figure 6 Original noisy image in the left and the color distribution in the middle and the thresholding output image on right side.

Given that the shown UT images doesn't have only two peaks and also they are colored image so the varying illumination is not the main factor the chosen thresholding is the global thresholding. Once the threshold is defined based on the color code of the HSV model it is applicable to all similar UT images.

2.2.3 Artifact and image scaling

Some of the UT images contain artifacts or artefacts which could be the source of any error in the perception or representation of the real image, reproduced by the involved equipment. The phenomenon is presented in the figure below (Figure 8).

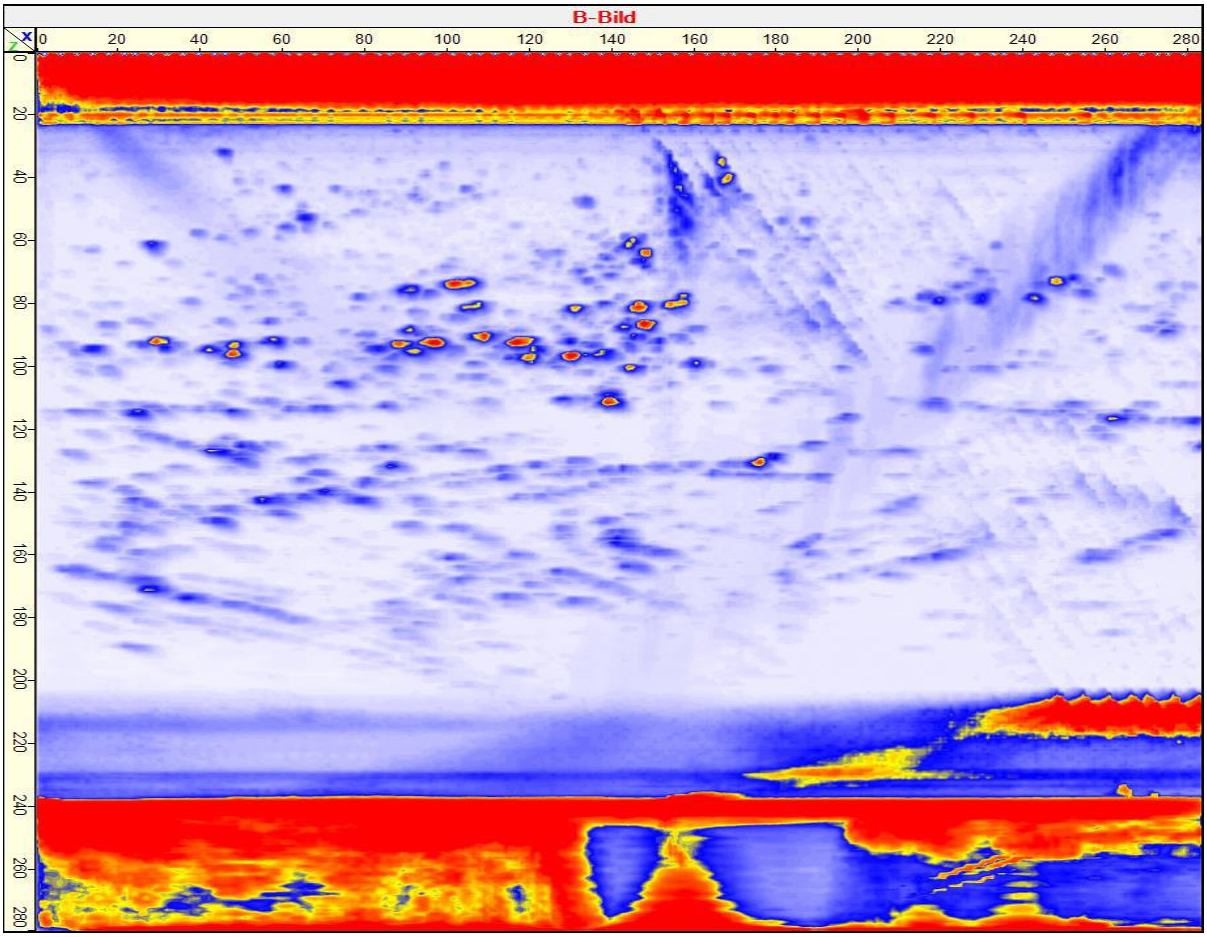

Figure 8 Vertical UT image with related artifacts [6].

This error is eliminated using global thresholding as well as the given dimensions of the component as shown in the following figure (Figure 9). Also, the real shape of the image given in -mm- according to the component dimension $l \times w \times h$. The used code is can be consulted in the appendices chapter (See Figure 16)

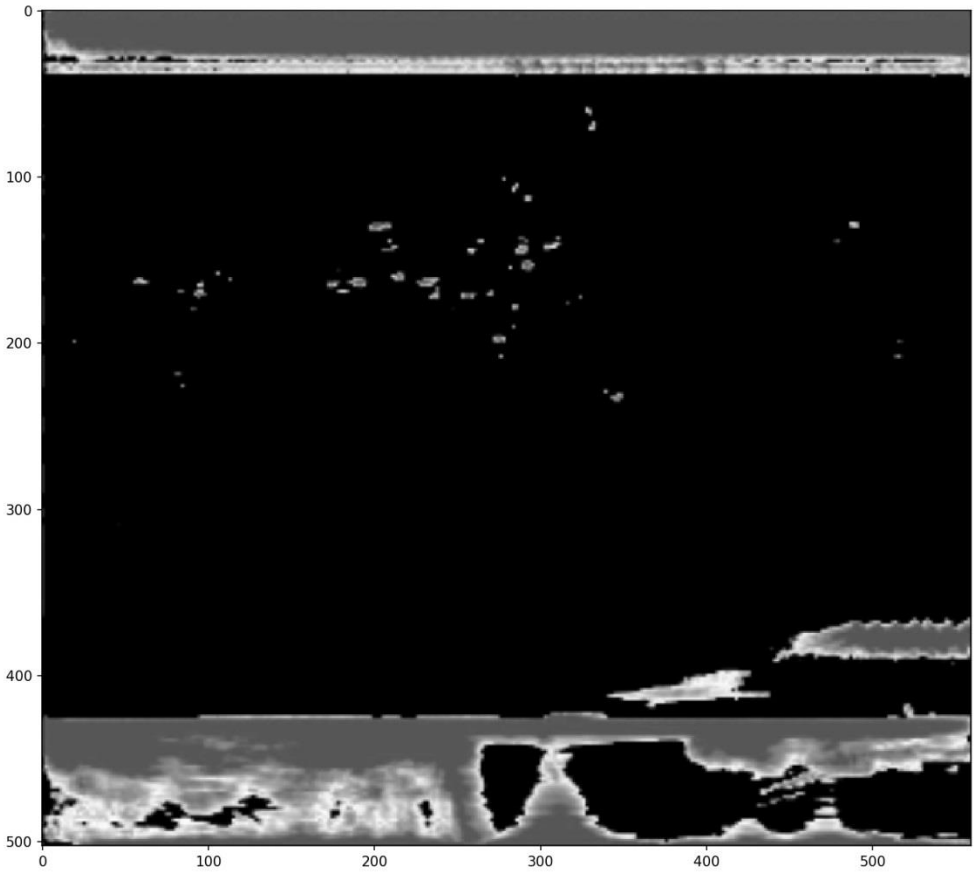

Figure 9 Output image after scaling the UT image.

The artifact is removed in the processing step using the "matplotlib.path" library and its function "contains_points". It takes four coordinates of a polygon and return nothing but the inside of the polygon. For more details one can check the appendices chapter (see Figure 17).

3 The tool development

In the previous chapter, the pre-processed UT image is already performed and ready for processing. In this chapter, the aim is to extract information such as the center and the shape of defects. Based on the chosen threshold, all defects are presented visually with certain accuracy shown below. Afterwards, an algorithm of output data extraction is implemented in Python programming language. Furthermore, obtained data of the extracted defects are processed to look like real component dimension. Finally, the defects centers are compared to the validation data.

3.1 Algorithm principle

The principle of the detection algorithm is based on the chosen threshold or maxval which is the highest value of the Hue matrix considered as a defect. This can be summarized as follows:

→ If "the pixel value" ≤ "maxval" then the pixel is considered as a defect. This line is repeated iteratively in all neighbours of each pixel (i,j) within a single connected defect until the condition is not valid any more. Here is an illustrative chart of the defect collecting algorithm (see figure 10).

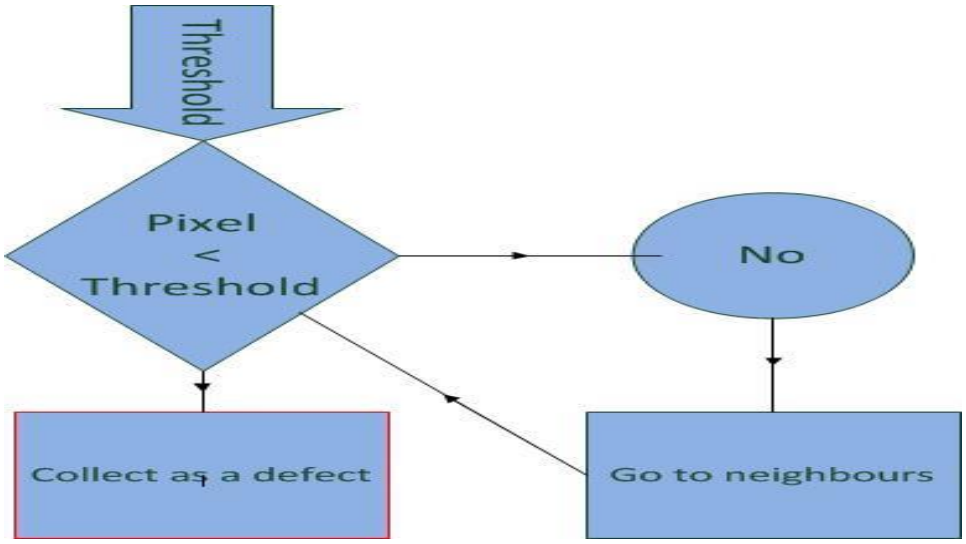

Figure 11 The algorithm flow chart of the image data extraction

The center of each defect is calculated then from the formula $\frac{\sum_{i,j}^{n}(i,j))}{n}$ as a barycenter. The shape of defects is considered in this chapter as a disc for simplification $\pi * r^2$. (see figure 13)

For more details about the code, please check the appendices chapters (See figure 18/19)

3.2 Validation

For the validation procedure, only the centers of defects are considered and thus compared with the validation data. This data is obtained manually by zooming into the defect and selecting its shape. The result of the validation should not be 100% identical to the validation data but at least satisfying. Here is a representation of the validation data and the obtained centers in the figure below (see Figure 12).

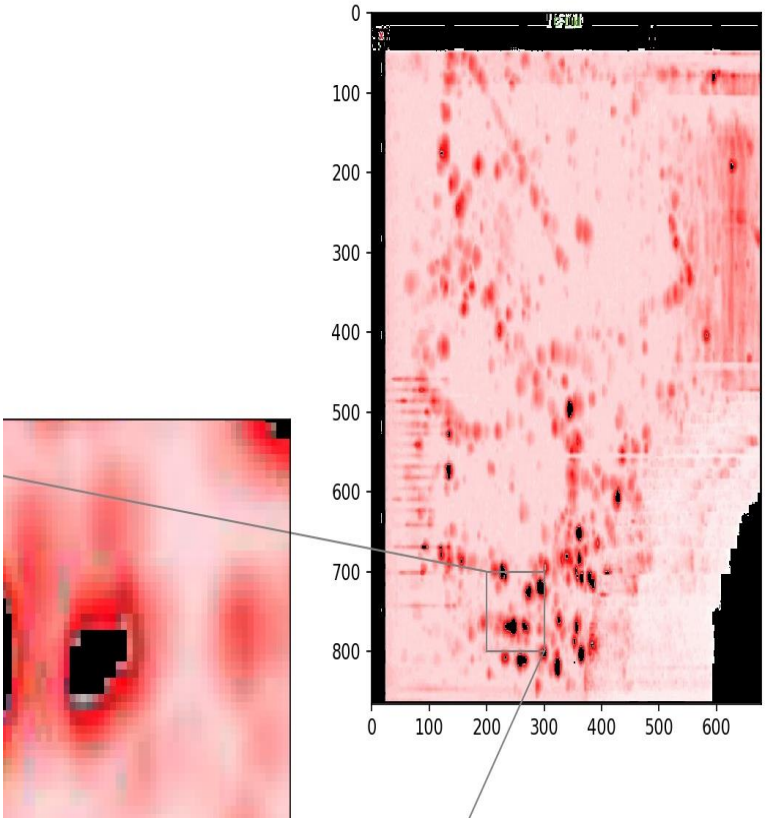

Figure 12 The image of the extracted data with a zoom to a single defect match.

(a) **(b)**

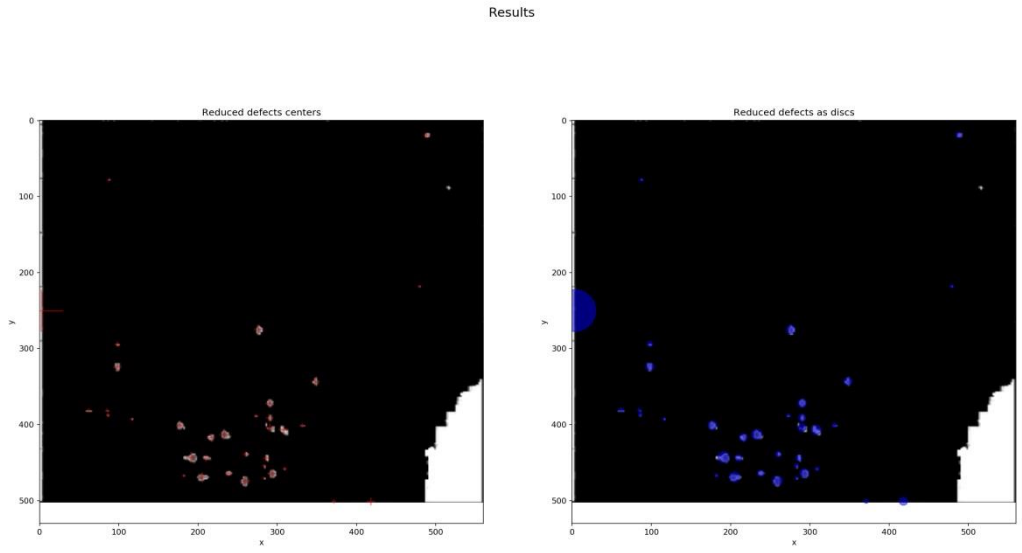

Figure 13 (a) Detected centers of defects (b) Estimated shape of the defect

3.3 Limitations

The results are satisfying for this UT image example, what about a different kind of UT image. Here is an example of more challenging images with multiple inter crossing defects (see Figure 14).

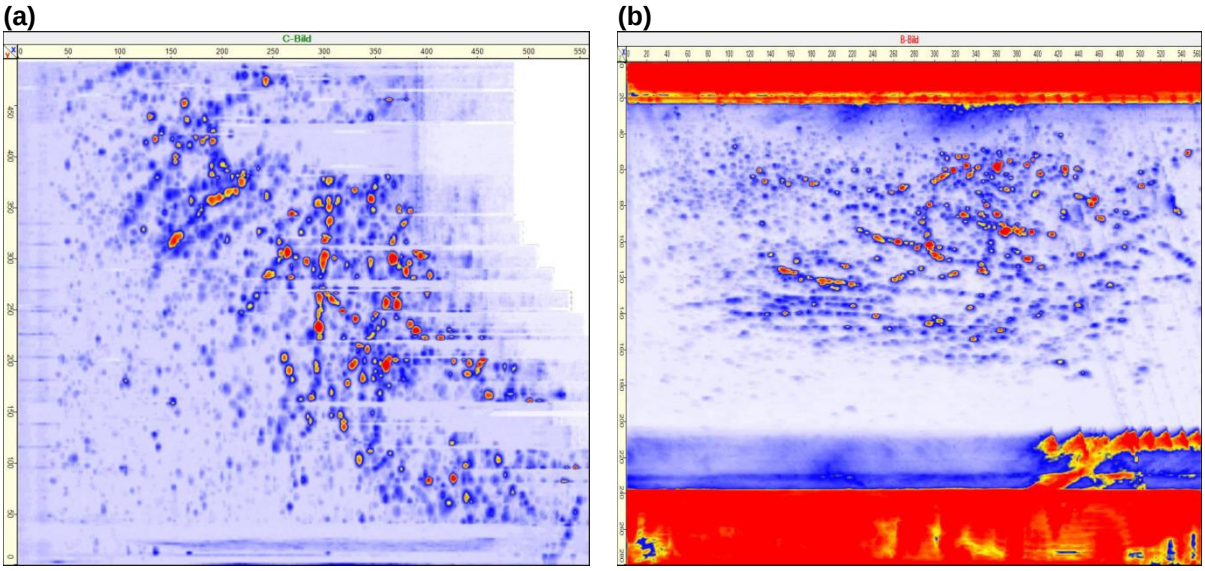

Figure 14 (a) UT Image of an xy plan (b) UT image of a xz plan

When applying the same algorithm, the multiple connected defects are considered as a single defect even though they are visually easy to distinguish. The result of the running algorithm is shown as follows (see Figure 15).

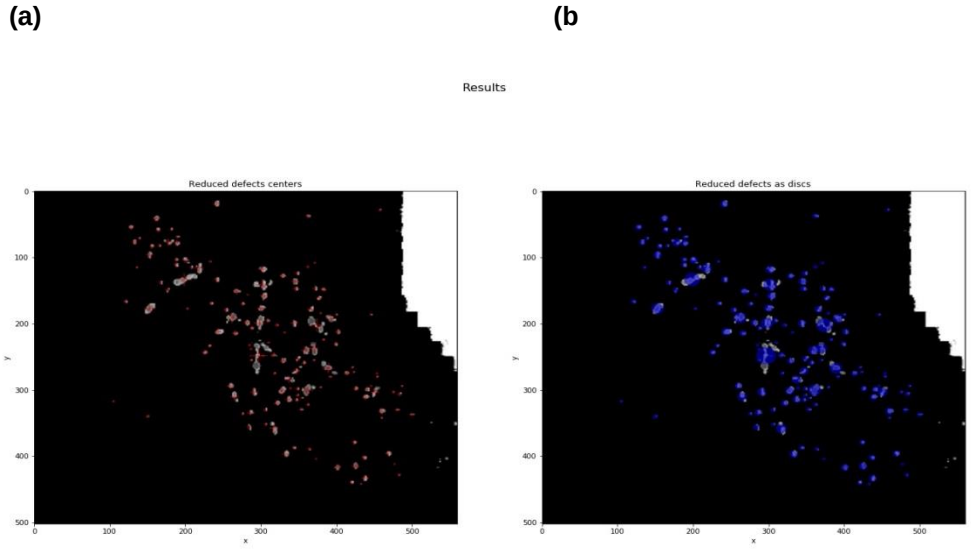

Figure 15 (a) Detected centers of defects of another component

(b) Estimated shape of the defect of another component

4 Detection tool enhancement

In previous chapter, the section of limitations has shown that the developed algorithm is not accurate when it comes to “crowded” UT images. Thus, further enhancement has to be pursued in this chapter. The algorithm is then an implementation of watershed algorithm using gradient algorithm for defining the so called markers as explained below.

4.1 Watershed algorithm

The watershed is a classical algorithm used for segmentation, that is, for separating different objects in an image.

Starting from user-defined markers or local extrema, the watershed algorithm treats pixels values as a local topography (elevation). The algorithm floods basins from the markers, until basins attributed to different markers meet on watershed lines. In many cases, markers are chosen as local minima of the image, from which basins are flooded.

In the example below (see Figure 16), two overlapping circles are to be separated (a). To do so, one computes an image that is the distance to the background (b). The maxima of this distance (i.e., the minima of the opposite of the distance) are chosen as markers, and the flooding of basins from such markers separates the two circles along a watershed line(c). [5]

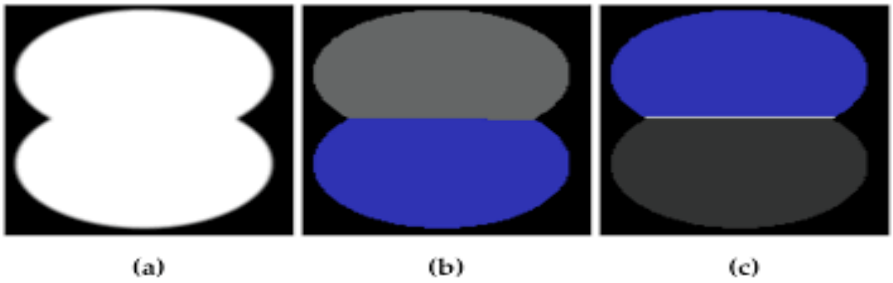

Figure 15 (a) Overlapping circles

(b) Different markers meeting on the watershed line

(c) separated circles along the watershed line

In order to implement this algorithm in our situation, the so called markers which have to be extrema are necessary. For that purpose, in the following section a basic algorithm is proposed.

4.2 Gradient algorithm

Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or approximate gradient) of the function at the

current point. If, instead, one takes steps proportional to the positive of the gradient, one approaches a local maximum of that function; the procedure is then known as gradient ascent.

Gradient descent is also known as steepest descent. However, gradient descent should not be confused with the method of steepest descent for approximating integrals.

5 Summary and outlook

The developed basic algorithm presented in chapter 3 is suitable for the detection of defects when the UT images are provided for every layer. This is a time consuming procedure until all defects 3D coordinates are determined. The alternative is use UT images for (x,y) plan and the (x,z) plan in order to determine the 3D distribution of defects. For this application the mentioned algorithm is not accurate and has been further enhanced as it was explained in chapter 4. It is the combination of the algorithm of chapter 3 and the watershed approach in order to separate multiple connected defects. As a preliminary step, extrema of the numerical color map of each defect has been determined first. It is an important input to the watershed algorithm in order to separate defects. Also, an approximation has been made regarding the shape of the separated defects. It consists on defining the half distance of extrema as a point of separation. This is an acceptable assumption as long as the position of the centers is accurate. The validation of the developed algorithm is ongoing, since manual labeling takes a lot of time.

In order to optimize the detection algorithm further development is necessary regarding the extrema determination. The gradient descent method is not the most adapted method for our case, other algorithm would be efficient according to experts. Some of those recommended algorithms are for instance:

- The greedy algorithm
- The ant colony optimization algorithm
- The genetic algorithm

After validating the detection tool, the next step is to optimize its running time by implementing an adequate algorithm of tracking down optimum. This will be the continuation of the work as a semester internship starting from June 2019 until end of July 2019.

6 Sources

[1] H. A. Richard und M. Sander. Ermüdungsrisse:Erkennen, sicher beurteilen, vermeiden. 2. Auflage. Berlin Heidelberg New York: Springer-Verlag. 2012.

[2] https://en.wikipedia.org/wiki/Scrum_(software_development)#Workflow

[3] https://en.wikipedia.org/wiki/Color_space

[4] https://opencv-python tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html

[5] https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_watershed.html

[6] Weber, F., Ultraschall-Untersuchung von Schmiedeteilen, Fraunhofer IZFP, Saarbrücken, 2017.

Other links

1. For colors code the following webpage was taken as a reference
   - http://colorizer.org/

2. To familiarize with image processing vocabulary
   - https://www.tutorialspoint.com/dip/concept_of_masks.htm
   - http://cns-alumni.bu.edu/~slehar/fourier/fourier.html

3. The coding is based on the package Opencv and its comprehensive documentation.
   - https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_colorspaces/py_colorspaces.html#converting-colorspaces
   - https://towardsdatascience.com/build-your-own-convolution-neural-network-in-5-mins-4217c2cf964f
   - https://www.youtube.com/watch?v=rTawFwUvnLE

4. The comprehension of the CNN approach was reached using basically the following webpages:
   - http://cs231n.github.io/convolutional-networks/#overview

5. From Git command lines to GitHub/local server
    - https://en.wikipedia.org/wiki/Git
    - https://en.wikipedia.org/wiki/Bash_(Unix_shell)

To emulate a bash environment on windows→

- https://stackoverflow.com/questions/45034549/difference-between-git-gui-git-bash-git-cmd
- https://www.quora.com/What-is-the-difference-between-Git-GUI-Git-Bash-and-Git-CMD

For image transformation

- https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_geometric_transformations/py_geometric_transformations.html#geometric-transformations

Appendices

A glance of the used codes for the detection tool development.

```
def detect(path,lower_bound,upper_bound,lower_bound_1 = [0,0,0],upper_bound_1 =[0,0,0]):

    img = cv2.imread(path,1)  # BGR color image reading and 0 flag for gray
    #img = cv2.medianBlur(img,5) #blur  to make the transition smooth from color to color
    #img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) #convert to RGB for real image plot

    # converting it into Hue, saturation, value (HSV)
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    '''Threshold'''

    # define range of red color in HSV
    lower = np.array(lower_bound)
    upper = np.array(upper_bound)

    # Threshold the HSV image to get only red colors
    mask_0 = cv2.inRange(hsv, lower, upper)
    mask_1 = cv2.inRange(hsv,np.array(lower_bound_1),np.array(upper_bound_1))
    mask =mask_0 + mask_1

    # Bitwise-AND mask and original image
    res = cv2.bitwise_and(img,img, mask = mask)


    return res
```

Figure 15 Thresholding code snippet

```
def shaping(image,i,j,resol = [300,300]):

    img = image
    rows,cols = img.shape

    frame = [[0,0],[cols,0],[0,rows],[cols,rows]]
    dimension = [[0,0],[resol[1],0],[0,resol[0]],[resol[1],resol[0]]]

    pts1 = np.float32(frame)
    pts2 = np.float32(dimension)

    M = cv2.getPerspectiveTransform(pts1,pts2)
    shaped = cv2.warpPerspective(img,M,(j,i))

    return shaped
```

Figure 16 Thresholding code snippet

```
def pre_shaping(image,resol = [305,366]):

    img = image
    rows,cols = img.shape

    frame = [[0,0],[cols,0],[0,rows],[cols,rows]]
    dimension = [[0,0],[resol[1],0],[0,resol[0]],[resol[1],resol[0]]]

    pts1 = np.float32(frame)
    pts2 = np.float32(dimension)

    M = cv2.getPerspectiveTransform(pts1,pts2)
    pre_shaped = cv2.warpPerspective(img,M,(resol[1],resol[0]))

    return pre_shaped


def real_shaping(normalized_image,rows_dim,cols_dim,i = 8,j = 17):

    img = normalized_image
    rows,cols = img.shape

    frame = [[i,j],[rows,j],[i,cols],[rows,cols]]
    dimension = [[0,0],[cols_dim,0],[0,rows_dim],[cols_dim,rows_dim]]

    pts1 = np.float32(frame)
    pts2 = np.float32(dimension)

    M = cv2.getPerspectiveTransform(pts1,pts2)

    real_shape = cv2.warpPerspective(img,M,(cols_dim,rows_dim))

    return real_shape
```

Figure 17 Image scaling code snippet

```
def data_reduction(polygon,coordinates,area):
    old_dim = len(coordinates)
    p = pth.Path(polygon)
    mask = p.contains_points(coordinates)
    reduced_coord = coordinates[mask]
    reduced_area = area[mask]
    new_dim = len(reduced_coord)
    print(old_dim-new_dim,' values were reduced')
    return reduced_coord,reduced_area
```

```
'''Function call'''

area = area
coordinates = coord
polygon = np.array([[0,35],[560,35],[560,200],[0,200]])
reduced_coord, reduced_area = data_reduction(polygon,coordinates,area)
```

Figure 18 Artifact elimination code snippet

```
def add_neighbours_in_stack(image, a, b, stack, i, j, inComponent):

    inComponent[i, j] = True
    neighbours = [(i+1,j),(i-1,j),(i,j-1),(i,j+1)]
    #neighbours = [(i-1,j-1),(i+1,j-1),(i+1,j),(i-1,j),(i,j-1),(i,j+1),(i-1,j+1),(i+1,j+1)]
    for (k,l) in neighbours:
        if 0<=k<image.shape[0] and 0<=l<image.shape[1]:
            if (image[k,l]>a and image[k,l]<b) and not inComponent[k, l]:
                inComponent[k, l] = True
                stack.append((k,l))
    return

def connected_component_nb(image, seuil, i0,j0, inComponent):
    stack = [(i0,j0)]
    defect = []
    nb=1
    barycentre = [i0,j0]
    while len(stack)>0:
        (i,j) = stack.pop()
        defect.append((i,j))
        nb+=1
        add_neighbours_in_stack(image, seuil, 256, stack, i, j, inComponent)

        barycentre[0] += i
        barycentre[1] += j

    barycentre[0] /= nb
    barycentre[1] /= nb
```

Figure 18 Connected defects extraction code part 1

```
def all_connected_components(image, seuil):

    inComponent = np.empty(image.shape, dtype=np.bool)
    inComponent[:, :] = False

    baryAndSizes = []
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            if image[i,j]>seuil and not inComponent[i,j]:
                baryAndSizes.append(connected_component_nb(image,seuil,i,j,inComponent))

    return baryAndSizes


def step2(image,defect_nb):

    imageR = image
    imageB = image

    baryAndSizesR = all_connected_components(imageR, 1)
    baryAndSizesB = all_connected_components(imageB, 1)

    xR, xB = np.array([]), np.array([])
    yR, yB = np.array([]), np.array([])
    sizeR, sizeB = np.array([]), np.array([])


    for i in range(len(baryAndSizesR)):
        xR = np.append(xR, baryAndSizesR[i][1][0])
        yR = np.append(yR, baryAndSizesR[i][1][1])
        sizeR = np.append(sizeR, baryAndSizesR[i][0])

    for i in range(len(baryAndSizesB)):
        xB = np.append(xB, baryAndSizesB[i][1][0])
        yB = np.append(yB, baryAndSizesB[i][1][1])
        sizeB = np.append(sizeB, baryAndSizesB[i][0])

    '''plot'''
```

Figure 19 Connected defects extraction code part 2