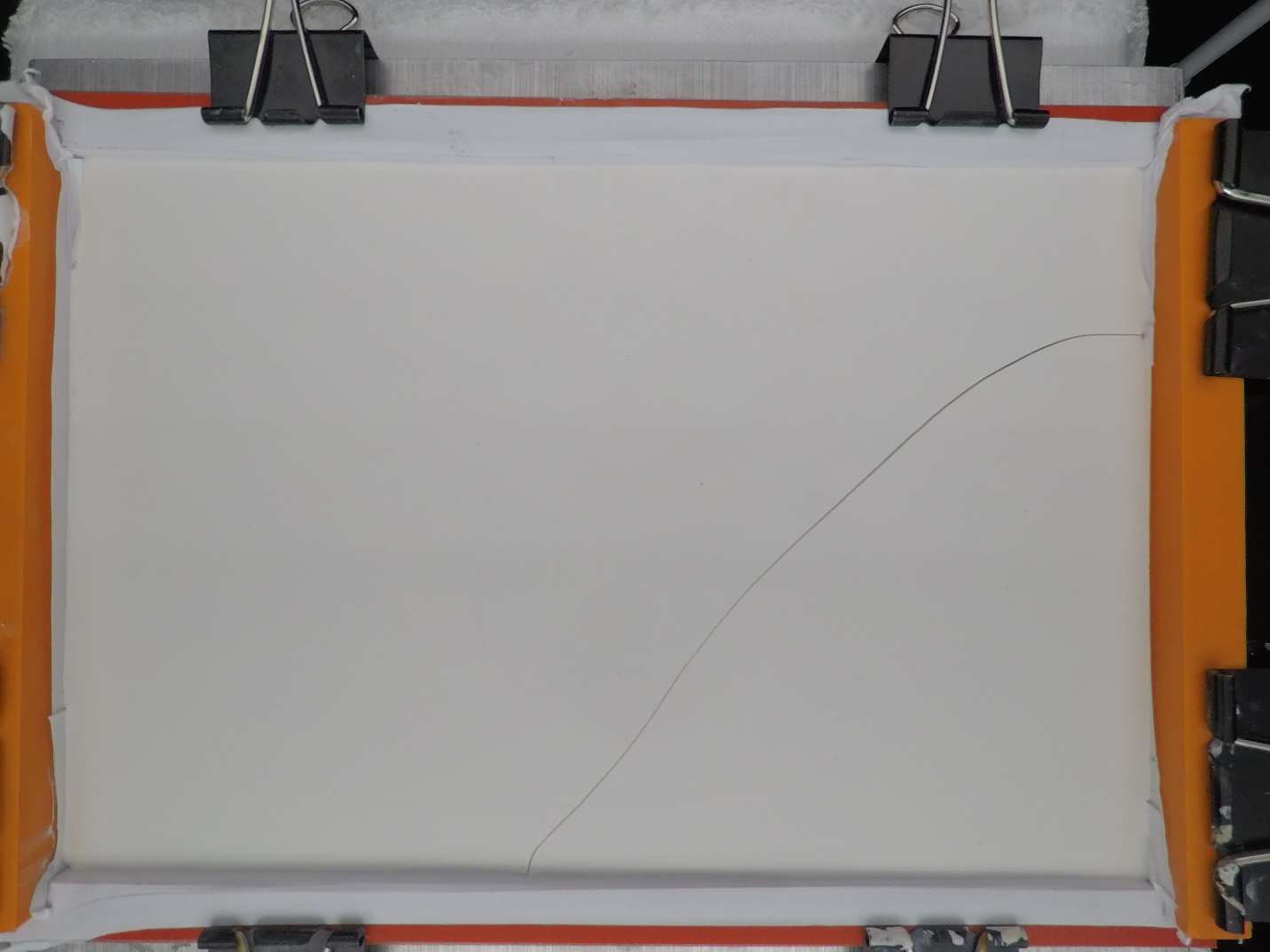
Crack Analysis Paper

Jimmy Hayden

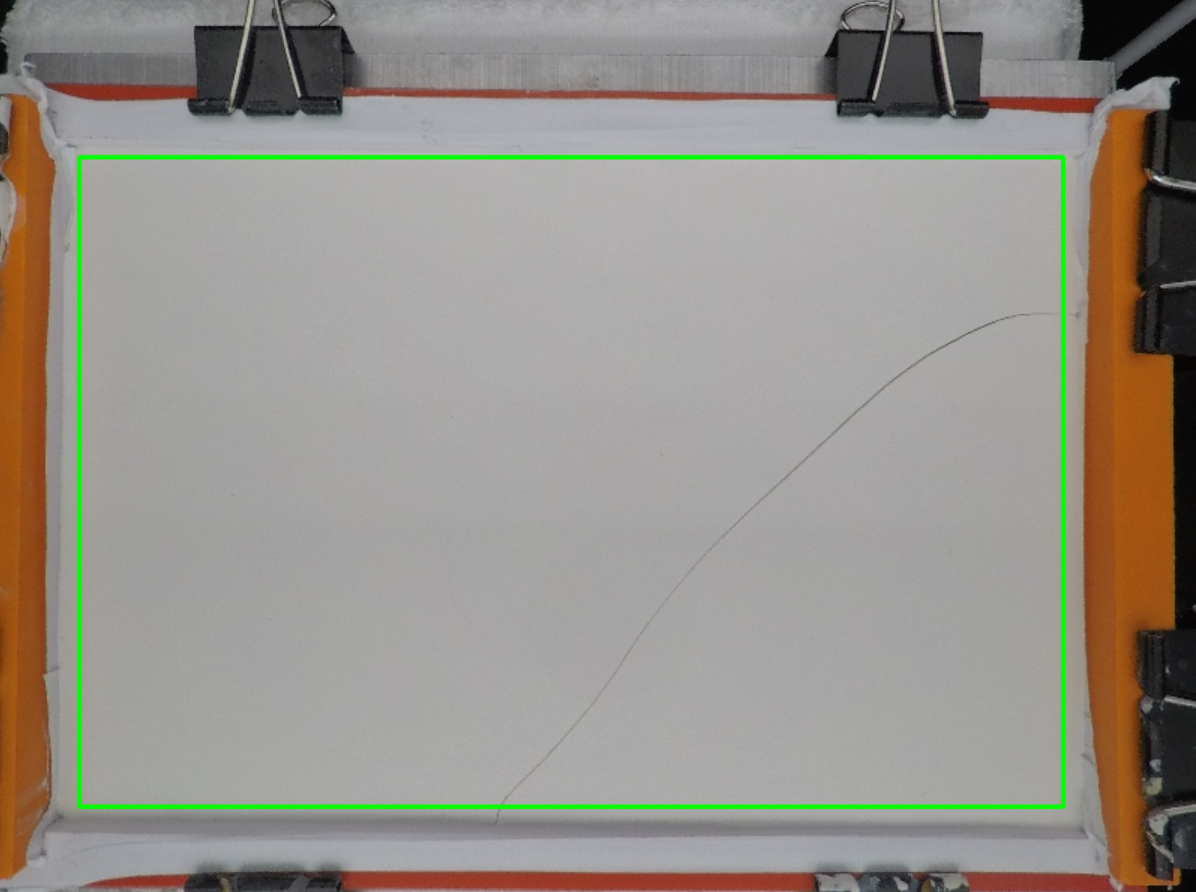
Our team designed a program with the objective of using image processing to analyze the cracking of a material. The program is programmed in python and using libraries such as cv2, morphology and skimage. The program is organized with all of the different functions in the top half of the program and the main function that runs the program in the bottom half of the program.

We start out the program by placing the image that we will be analyzing in the same directory as the python file. The image we will be testing is called Trial33.jpg, and it is a clay sample a member of our research group created and monitored using a GoPro.



The image is then read in by the program using the cv2 command imread. Using the command img.shape, we are able to obtain the height and width of the image in pixels. The next section of the program is used to edit the height and width to scale the height and the width to fit the computer screen. The user then adds the dimensions of the length and width and units we will be using. This will allow us to accurately add up the length and present the length in the units the user would like.

The ROI section of the program allows the user to crop the image to get rid of any extra parts of the image that they do not want. For example, in our image that we tested, the tray that holds the clay sample is in the image and we would like it not to be. By clicking one corner, and dragging the mouse, the user can select only the portion of the image they want analyzed.

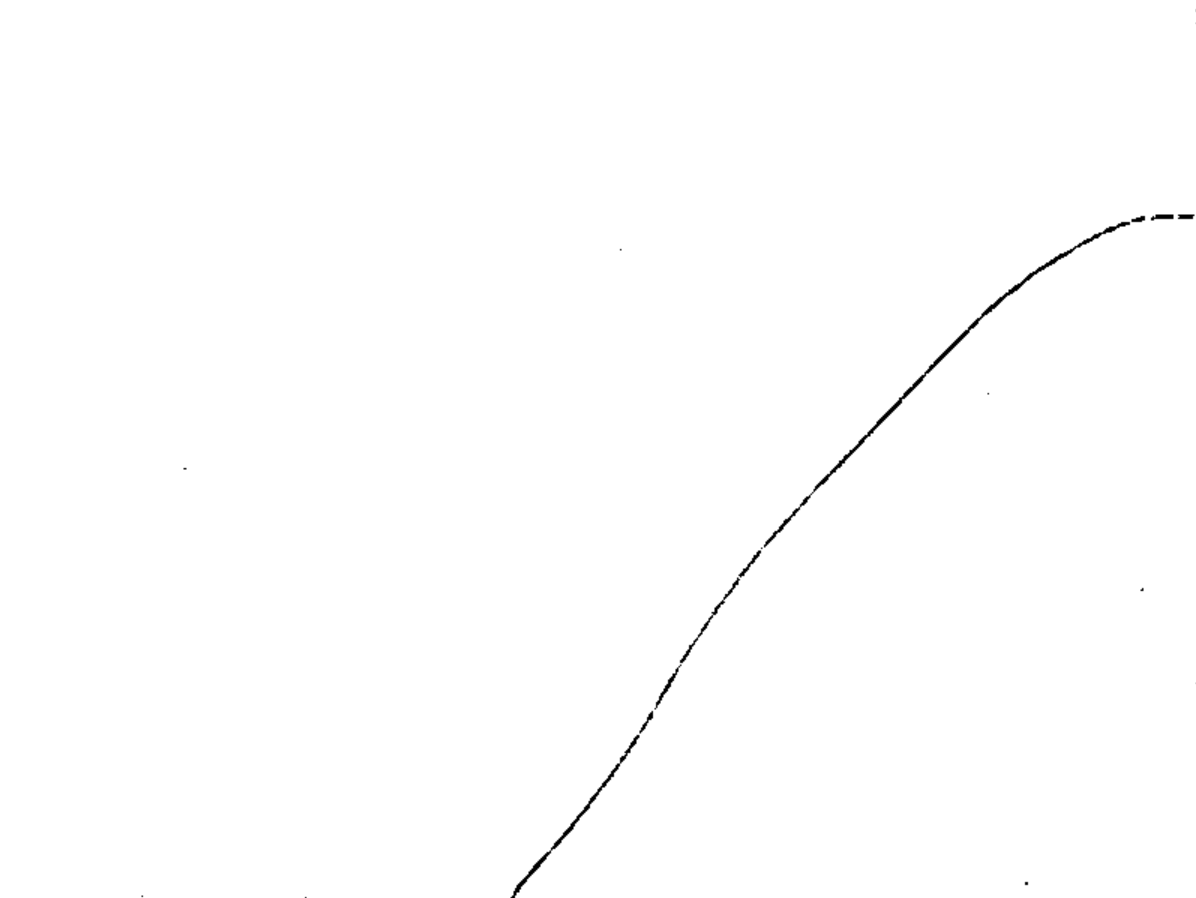


Once the image is cropped to the area the user would like, the user will then press the Esc button to see the cropped image.



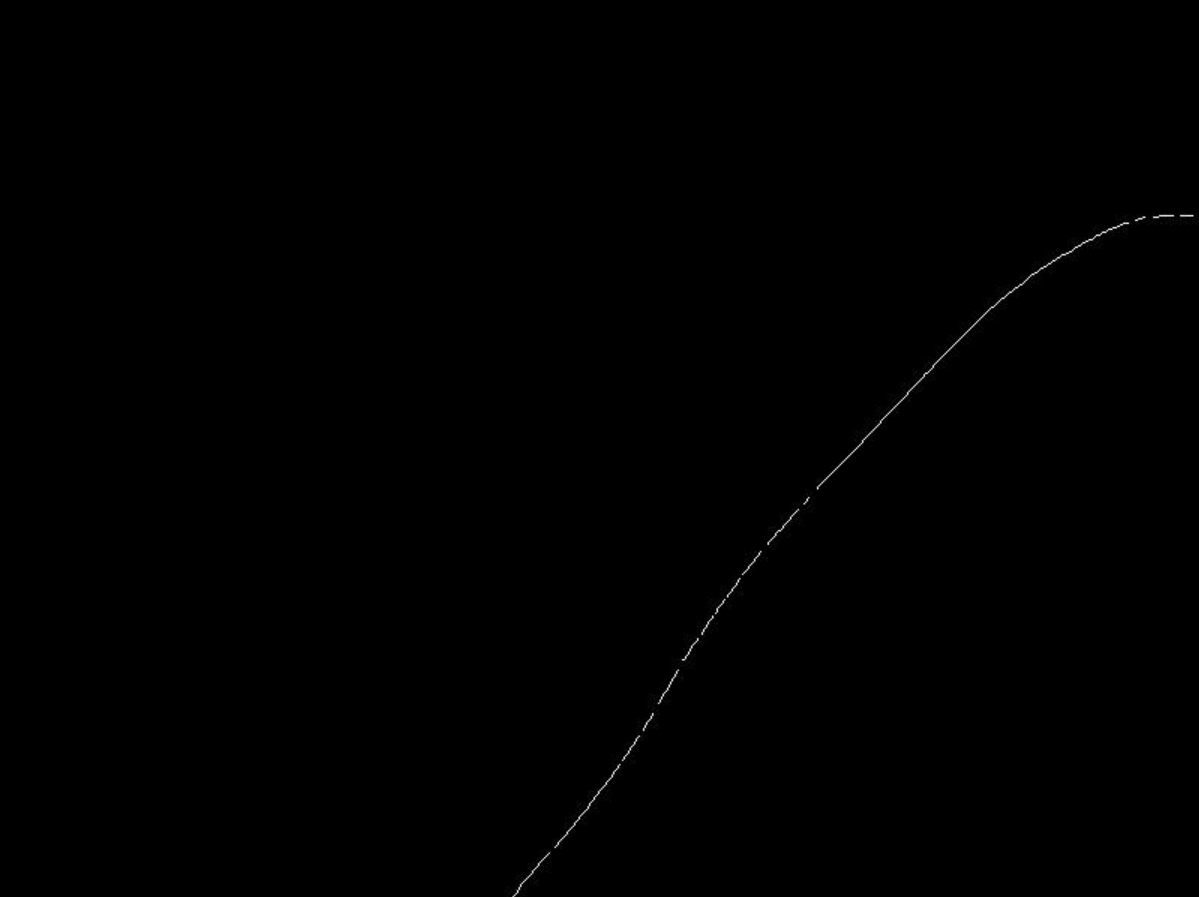
After the image has been cropped, it will be saved as cropped.jpg and that image will be found in the current directory. The enlarge function is then called which will keep the image resized to the dimensions that the user will like.

The function that is then called is binary. The binary function takes in the image and makes it grayscale. A median filter is then applied to the image and computer vision then uses the adaptiveThreshold function to make each pixel either black or white.

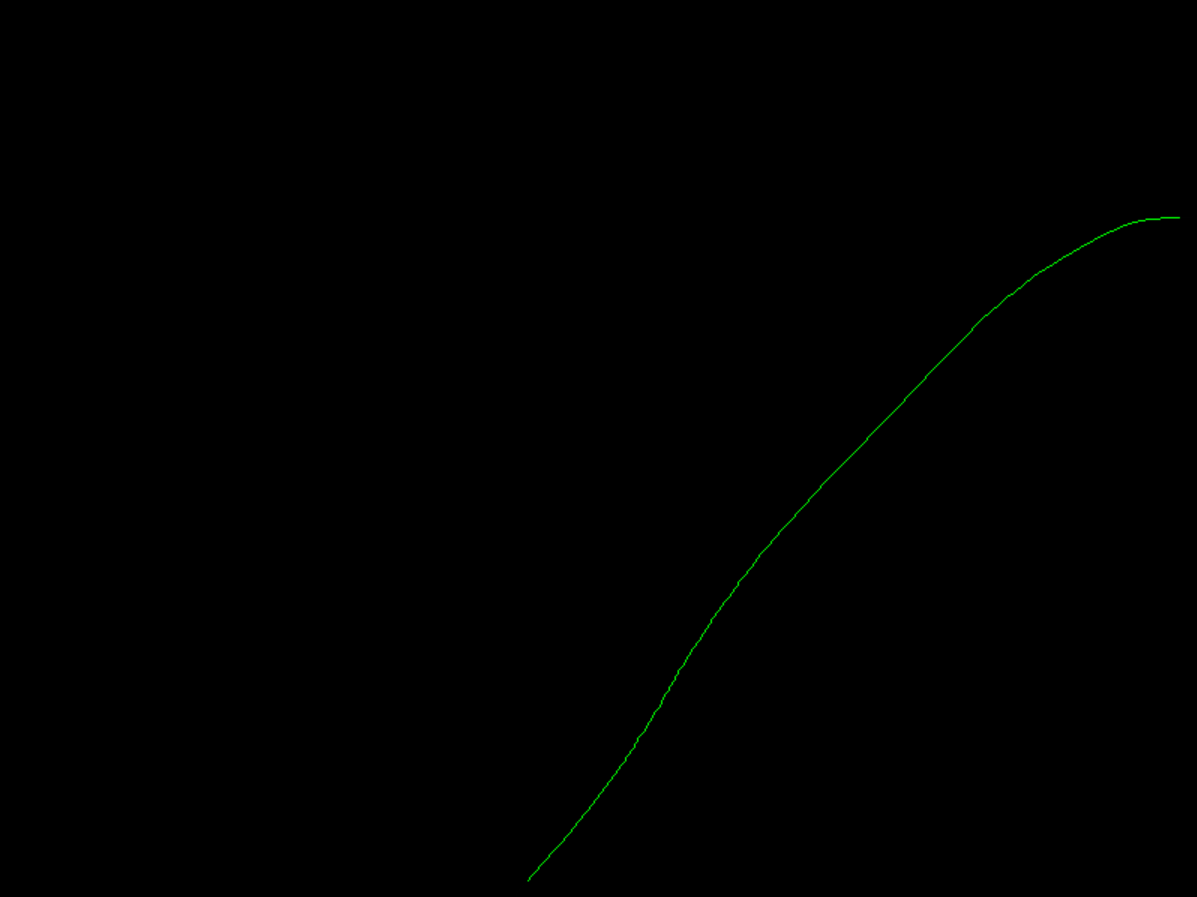


The next function used in the median function. This function reads in the binary image and applies a median filter of 3 to the image.

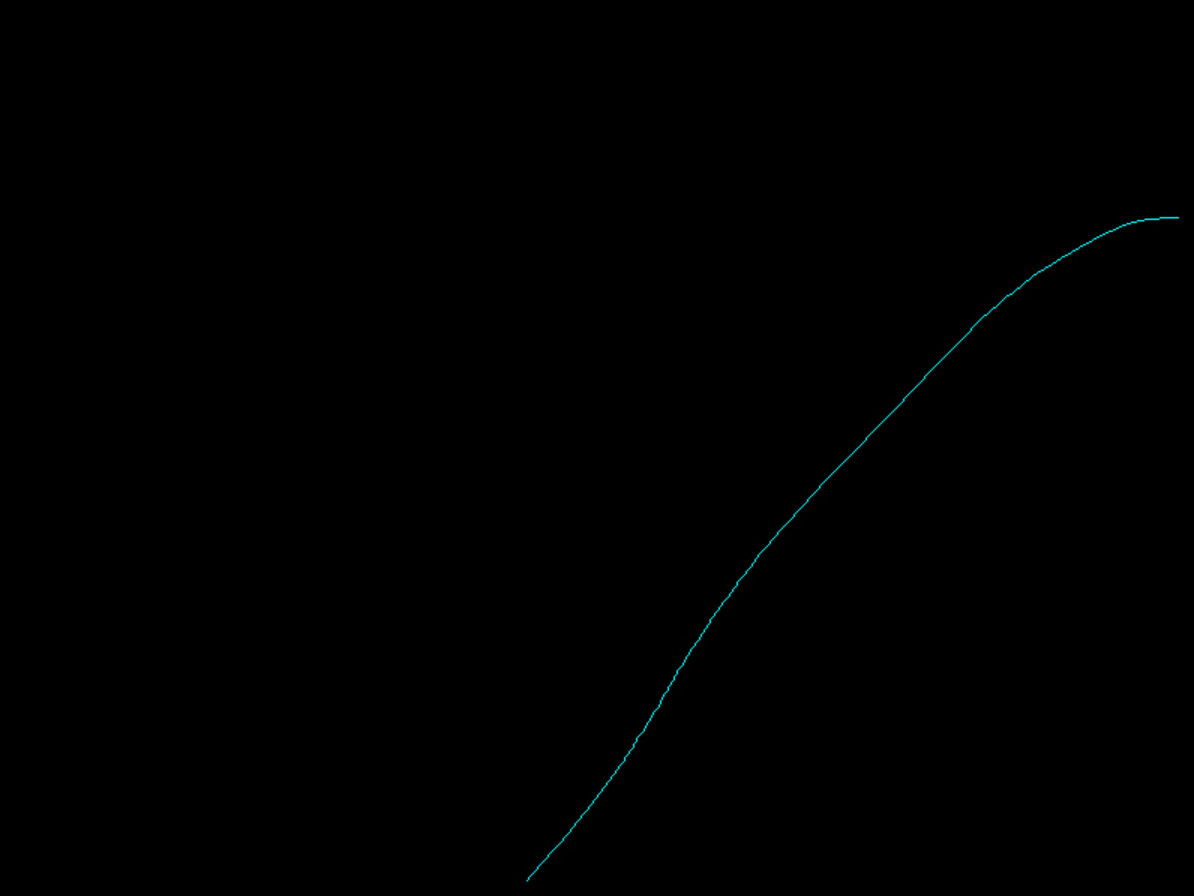
The next function is called thinning2, and takes in the median image. This function uses a morphology function skeletonization to take the image and make the width of the cracking one pixel wide with keeping the shape. This image is saved as gaps.jpg and then outputted on the screen.



After the thinning function has completed, the complete function is called. This function is used to fill in the gaps in the skeletonized photo.



The bandw function is called which takes our new image and makes it black and white again. Then using this image, the getLength function is called. This function loops through all the pixels and then categorizes them into three groups. The first group is an endpoint, which means that it is a pixel that is white so it is part of the cracking, but it is at the end. The second group is just an normal cracking pixel, which means that it is in the cracking, and there are pixels on either side of it that are also part of the cracking. The last group is a splitting point pixel, meaning that multiple cracks slit off of this pixel. To determine the groupings, each pixel is first determined to be part of the cracking or not. This is determined by the red, green and blue values of the pixel. If they are all higher than 150, they are a part of the crack. Once that is determined, the getColor function is used to take in the pixel location and determine if any of the eight pixels around it are part of the cracking. Based on the information returned from the function, the selected pixel is categorized as an endpoint, normal cracking pixel or splitting point. All pixels categorized as endpoints are then added to an array called endpoints. This array is then looped through to calculate the length. Starting at one endpoint, the getColor function is used to see what pixels around the selected pixel are part of the cracking. Then, the pixels already counted are taken out of this array of points, and the selected point is taken out. If there is one pixel left, the selected pixel is updated to that one. The length is updated based on where the pixel is in relation to the selected one. If the new pixel is horizontal, we take the total width that the user enters and divide it by the total number of pixels wide and that is the number added to the length. If the new pixel is vertical, we take the total height that the user enters and divide it by the total number of pixels high and that is the number added to the length. If the new pixel is diagonal, we take both the numbers we get from the previous two cases, square them, add them together and then get the square root of that number. Once we get to a selected pixel that has no other pixels that are part of the cracking that are around the selected one, we have reached an endpoint and the length is outputted.



Once the final image is outputted on the screen, the user will press Esc, and all windows will disappear, and the total length will be outputted on the screen.



crackAnalysis.py

#Python Code that will take in a picture that contains cracking

#Outputs the length of the cracking

import math

import scipy.ndimage.morphology as m

import cv2

import numpy as np

from skimage import img\_as\_float

from skimage import io, color, morphology

from skimage import io, morphology, img\_as\_bool, segmentation

from scipy import ndimage as ndi

import matplotlib.pyplot as plt

#Resizes the image

def enlarge(fileName):

img = cv2.imread(fileName)

cv2.imwrite("file3.jpg", cv2.resize(img, (a,b)))

#Makes each pixel of the image black or white

def binary(img):

img = cv2.imread(img,0)

img = cv2.medianBlur(img,5)

th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C,\

cv2.THRESH\_BINARY,11,2)

cv2.imwrite("binary.jpg", th3)

cv2.imshow("Binary",th3)

cv2.waitKey(0)

#Applies median filtering to get rid of noise

def median(img1):

img = cv2.imread(img1)

median = cv2.medianBlur(img, 3)

cv2.imwrite("median.jpg",median)

cv2.waitKey(0)

#Thins the image to one pixel wide using the skeletonize function from morphology

def thinning2(name):

image = img\_as\_float(color.rgb2gray(io.imread(name)))

image\_binary = image < 0.5

out\_skeletonize = morphology.skeletonize(image\_binary)

out\_thin = morphology.thin(image\_binary)

plt.imsave('gaps.jpg', out\_skeletonize, cmap='gray')

img = cv2.imread("gaps.jpg")

cv2.imshow("Thinning2", img2)

cv2.waitKey(0)

#Fills in gaps in the skeleton image

def complete(img):

image = img\_as\_bool(io.imread(img))

out = ndi.distance\_transform\_edt(~image)

out = out < 0.02 \* out.max()

out = morphology.skeletonize(out)

out = segmentation.clear\_border(out)

out = out | image

cv2.imshow("out",out)

cv2.waitKey(0)

cv2.imwrite('gaps\_filled.jpg', out)

#Makes all green pixels white

def bandw(img1):

image = cv2.imread(img1)

height, width, channels = image.shape

for y in range(0,height):

for x in range(0,width):

color = image[y,x]

b = color[0]

g = color[1]

r = color[2]

if b>0 and g>100 and r>0:

image[y,x] = [255,255,255]

else:

image[y,x] = [0,0,0]

cv2.imwrite("bandw.jpg", image)

#Function to calculate the length

def getLength(img,wU,hU,units):

image = cv2.imread(img)

height, width, channels = image.shape

w = wU/width

h = hU/height

#Array created that will contain all endpoints of the cracks

endpoints = [[]]

for y in range(0,height):

for x in range(0,width):

color = image[y,x]

#RGB values gotten for the selected pixel

b = color[0]

g = color[1]

r = color[2]

#Checks to see if all RGBs are part of the crack

if b>150 and g>150 and r>150:

#Cracks to see if pixel is not part of the border

if y>0 and x>0 and y<height-1 and x<width-1:

#The getColor function takes in a selected pixel and checks the eight pixels around it to see if it is part of the cracking

info = getColor(img,x,y)

#Count is the number of pixels around the selected one that are part of the cracking

count = info[[len(info)-1][0]]

#If there are more than 2 pixels in count, the selected pixel is a splitting point

if count>2:

image[y,x]=[255,0,0]

#If there are exactly two pixels in count, the selected pixel is in the middle of the cracking

if count==2:

image[y,x]=[0,255,0]

#If there is exactly one pixel in count, the selected pixel is an endpoint and added to the endpoint array

if count==1:

image[y,x]=[0,0,255]

endpoints.append([x,y])

#Categorizes the pixel as an endpoint if the selected pixel is on the border of the image

if y==0 or x==0 or y==height or x==width:

image[y,x]=[0,0,255]

endpoints.append([x,y])

endpoints.remove([])

#l is the variable to keep track of the length

l = 0.0

#Loops through all the endpoints

#For each endpoint we start with that pixel and move throughout the crack, adding to the length for each pixel until we get to another endpoint

for a in range(0,len(endpoints)):

tf = True

x = endpoints[a][0]

y = endpoints[a][1]

endpoints.remove([x,y])

while(tf):

image[y,x]=[255,255,0]

info = getColor(img,x,y)

count = info[[len(info)-1][0]]

info.remove([])

info.remove(count)

check = 0

x1=(info[0][1])

y1=(info[0][1])

for a1 in range(0,len(info)):

color = image[info[a1-check][0],info[a1-check][1]]

b = color[0]

g = color[1]

r = color[2]

if b==255 and g==255 and r==0:

x=(info[a1-check][0])

y=(info[a1-check][1])

info.remove([x,y])

check = check + 1

#info.remove!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

#An endpoint has been reached so the length with be printed

if len(info)==0:

print("TOTAL LENGTH: %.3f %s" % (l,units))

tf = False

#The pixel is in the middle of the crack and the length will be updated

elif len(info)==1:

x1 = x

y1 = y

x = info[0][1]

y = info[0][0]

if x==x1 or y==y1:

l = l+w

elif y==y1:

l = l+h

else:

l = l+math.sqrt(math.pow(w,2)+math.pow(h,2))

#The pixel is a splitting point

else:

#info.add([x,y])

print("Splitting Point")

print("TOTAL LENGTH: %d pixels" % l)

#break

break

cv2.imwrite('end.jpg',image)

cv2.imshow("final",image)

cv2.waitKey(0)

#Returns an array with the pixels that are part of the cracking from the eight surrounding pixels

def getColor(img,x,y):

image = cv2.imread(img)

height, width, channels = image.shape

info = [[]]

count = 0

color = image[y,x]

b = color[0]

g = color[1]

r = color[2]

color1 = image[y-1,x-1]

b1 = color1[0]

g1 = color1[1]

r1 = color1[2]

if b1>150 and g1>150 and r1>150:

count = count+1

info.append([y-1,x-1])

color2 = image[y-1,x]

b2 = color2[0]

g2 = color2[1]

r2 = color2[2]

if b2>150 and g2>150 and r2>150:

count = count+1

info.append([y-1,x])

color3 = image[y-1,x+1]

b3 = color3[0]

g3 = color3[1]

r3 = color3[2]

if b3>150 and g3>150 and r3>150:

count = count+1

info.append([y-1,x+1])

color4 = image[y,x-1]

b4 = color4[0]

g4 = color4[1]

r4 = color4[2]

if b4>150 and g4>150 and r4>150:

count = count+1

info.append([y,x-1])

color5 = image[y,x+1]

b5 = color5[0]

g5 = color5[1]

r5 = color5[2]

if b5>150 and g5>150 and r5>150:

count = count+1

info.append([y,x+1])

color6 = image[y+1,x-1]

b6 = color6[0]

g6 = color6[1]

r6 = color6[2]

if b6>150 and g6>150 and r6>150:

count = count+1

info.append([y+1,x-1])

color7 = image[y+1,x]

b7 = color7[0]

g7 = color7[1]

r7 = color7[2]

if b7>150 and g7>150 and r7>150:

count = count+1

info.append([y+1,x])

color8 = image[y+1,x+1]

b8 = color8[0]

g8 = color8[1]

r8 = color8[2]

if b8>150 and g8>150 and r8>150:

count = count+1

info.append([y+1,x+1])

info.append(count)

return info

#############################################MAIN###################################

#User enters the file name

img = cv2.imread('Trial33.jpg')

#Height and Width obtained in number of pixels

height, width, channels = img.shape

#Height and Width are adjusted to fit on the screen

a = int(width/5)

#a = width

b = int(height/5)

#b = height

#User enters the height and width and units

widthUnits = 9.0

heightUnits = 6.0

units = "inches"

#Image is resized

cv2.imwrite("resize.jpg", cv2.resize(img, (a,b)))

filename = 'resize.jpg'

selection = False

roi = []

#Uses ROI to allow the user to crop the image by dragging their mouse

def roi\_selection(event, x, y, flags, param):

global selection, roi

if event == cv2.EVENT\_LBUTTONDOWN:

selection = True

roi = [x, y, x, y]

elif event == cv2.EVENT\_MOUSEMOVE:

if selection == True:

roi[2] = x

roi[3] = y

elif event == cv2.EVENT\_LBUTTONUP:

selection = False

roi[2] = x

roi[3] = y

image\_read\_path=filename

window\_name='Input Image'

window\_crop\_name='Cropped Image'

esc\_keycode=27

wait\_time=1

input\_img = cv2.imread(image\_read\_path,cv2.IMREAD\_UNCHANGED)

if input\_img is not None:

clone = input\_img.copy()

cv2.namedWindow(window\_name,cv2.WINDOW\_AUTOSIZE)

cv2.setMouseCallback(window\_name, roi\_selection)

while True:

cv2.imshow(window\_name,input\_img)

if len(roi) == 4:

input\_img = clone.copy()

roi = [0 if i < 0 else i for i in roi]

cv2.rectangle(input\_img, (roi[0],roi[1]), (roi[2],roi[3]), (0, 255, 0), 2)

if roi[0] > roi[2]:

x1 = roi[2]

x2 = roi[0]

else:

x1 = roi[0]

x2 = roi[2]

if roi[1] > roi[3]:

y1 = roi[3]

y2 = roi[1]

else:

y1 = roi[1]

y2 = roi[3]

crop\_img = clone[y1 : y2 , x1 : x2]

if len(crop\_img):

cv2.namedWindow(window\_crop\_name,cv2.WINDOW\_AUTOSIZE)

cv2.imshow(window\_crop\_name,crop\_img)

k = cv2.waitKey(wait\_time)

if k == esc\_keycode:

#Created file cropped.jpg that saves the newly cropped image

cv2.imwrite("cropped.jpg", crop\_img)

#Image size is updated

enlarge("cropped.jpg")

#Image is transformed into a binary image

binary("file3.jpg")

#Median Filtering is used to get rid of access points

median("binary.jpg")

#Thinning is used to make the cracking one pixel wide

thinning2("median.jpg")

#Complete fills in gaps in the cracking

complete("gaps.jpg")

#Cracking RGB values are 255

bandw("gaps\_filled.jpg")

#Length is calculated using demensions and units given

getLength("bandw.jpg",widthUnits, heightUnits, units)

#Windows are removed

cv2.destroyAllWindows()

break

else:

print("Please Check The Path of Input File")