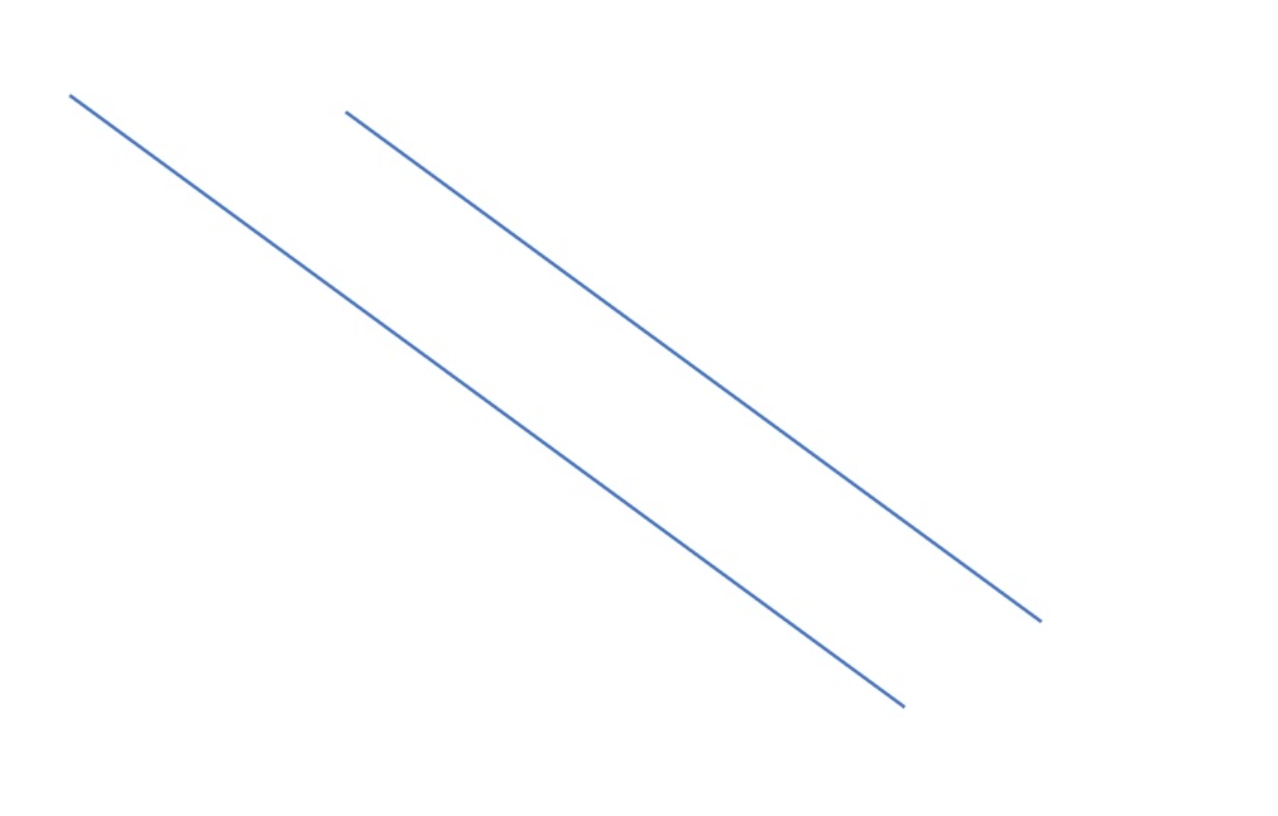
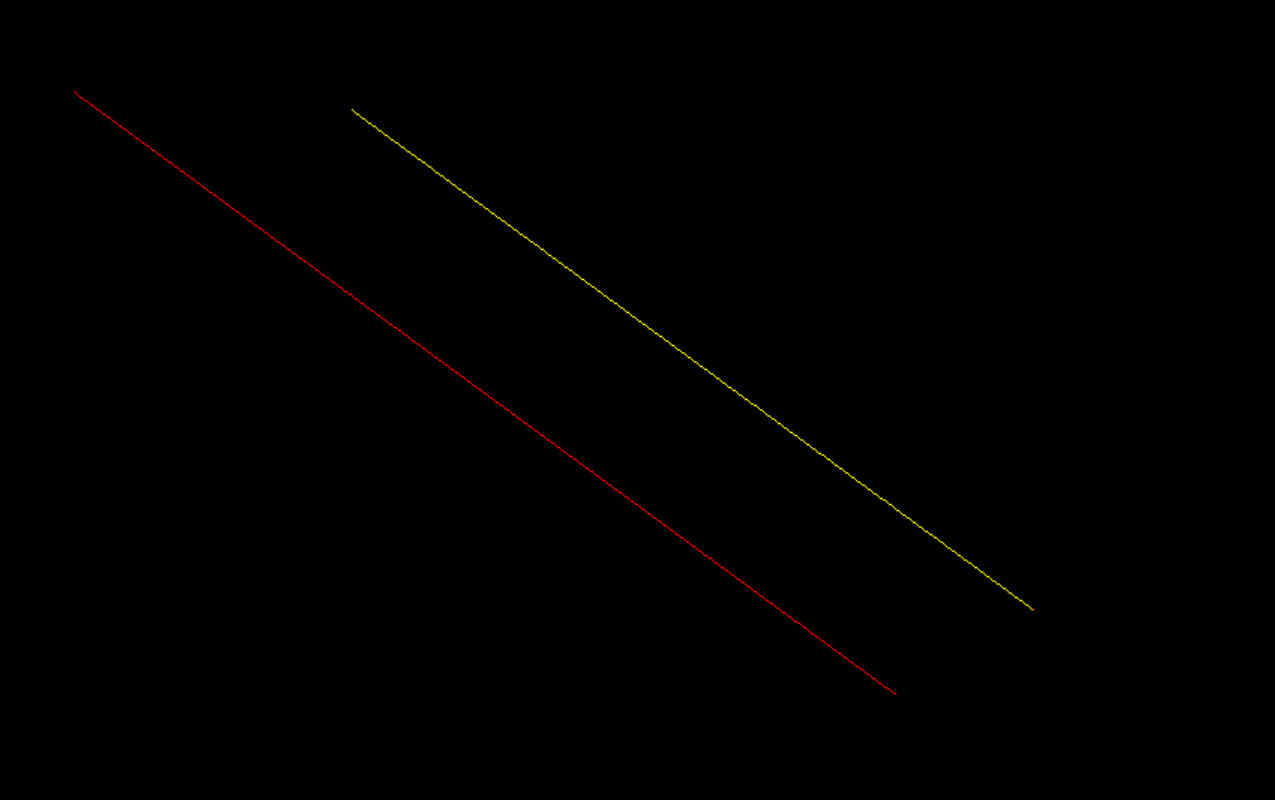
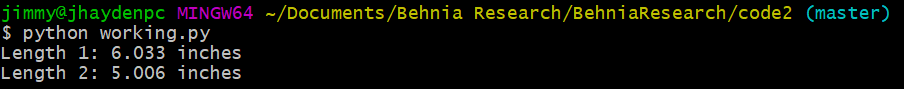
Multiple Lines Update:

I have updated the code so it will work with multiple lines. I am still working on getting the splitting point to work, it is close but there are a couple bugs still.



The above picture it the input and the picture below is the output. I used different colors to differentiate the two cracks.





#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

#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", img)

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 = [[]]

split = [[]]

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]

split.append([x,y])

#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

f= open("output.txt","w+")

#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

counter = 1

while(len(endpoints)>0):

tf = True

x = endpoints[0][1]

y = endpoints[0][0]

endpoints.remove([y,x])

while(tf):

if counter == 1:

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

elif counter == 2:

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

else:

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

info = getColor(img,y,x)

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

info.remove([])

info.remove(count)

check = 0

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 r==255:

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

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

info.remove([x,y])

check = check + 1

if len(info) == 0 or [y,x] in endpoints:

if [y,x] in endpoints:

endpoints.remove([y,x])

tf = False

elif len(info) == 1:

x1 = x

y1 = y

x = info[0][0]

y = info[0][1]

if x==x1:

l = l+w

elif y==y1:

l = l+h

else:

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

else:

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

tf = False

print("Length %d: %.3f %s" % (counter,l,units))

f.write("Length %d: %.3f %s\n" % (counter,l,units))

counter = counter + 1

l = 0

f.close()

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('two.png')

#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 = 7.0

heightUnits = 4.0

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 is transformed into a binary image

binary("cropped.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")