from scipy.spatial.distance import euclidean

from imutils import perspective

from imutils import contours

from PIL import Image

import numpy as np

import imutils

import cv2

from skimage import io

# Function to show array of images (intermediate results)

def show\_images(images):

for i, img in enumerate(images):

cv2.imshow("image\_" + str(i), img)

cv2.waitKey(0)

cv2.destroyAllWindows()

img\_path = "images/28.73378 image.JPG"

# Read image and preprocess

image = cv2.imread(img\_path)

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

blur = cv2.GaussianBlur(gray, (9, 9), 0)

edged = cv2.Canny(blur, 210, 210)

edged = cv2.dilate(edged, None, iterations=1)

edged = cv2.erode(edged, None, iterations=1)

#show\_images([blur, edged])

# Find contours

cnts = cv2.findContours(edged.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

cnts = imutils.grab\_contours(cnts)

# Sort contours from left to right as leftmost contour is reference object

(cnts, \_) = contours.sort\_contours(cnts)

# Remove contours which are not large enough

cnts = [x for x in cnts if cv2.contourArea(x) > 100]

#cv2.drawContours(image, cnts, -1, (0,255,0), 3)

#show\_images([image, edged])

#print(len(cnts))

# Reference object dimensions

# Here for reference I have used a 2cm x 2cm square

ref\_object = cnts[0]

box = cv2.minAreaRect(ref\_object)

box = cv2.boxPoints(box)

box = np.array(box, dtype="int")

box = perspective.order\_points(box)

(tl, tr, br, bl) = box

print(tl,tr,br,bl)

print(br[0]-bl[0])

# Draw remaining contours

for cnt in cnts:

box = cv2.minAreaRect(cnt)

box = cv2.boxPoints(box)

box = np.array(box, dtype="int")

box = perspective.order\_points(box)

(tl, tr, br, bl) = box

cv2.drawContours(image, [box.astype("int")], -1, (0, 0, 255), 2)

mid\_pt\_horizontal = (bl[0] + int(abs(br[0] - bl[0])/2), bl[1] + int(abs(br[1] - bl[1])/2))

mid\_pt\_verticle = (tr[0] + int(abs(tr[0] - br[0])/2), tr[1] + int(abs(tr[1] - br[1])/2))

wid = euclidean(tl, tr)

ht = euclidean(tr, br)

cv2.putText(image, "{:.1f}px".format(wid), (int(mid\_pt\_horizontal[0] - 15), int(mid\_pt\_horizontal[1] - 10)), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 0), 2)

cv2.putText(image, "{:.1f}px".format(ht), (int(mid\_pt\_verticle[0] + 10), int(mid\_pt\_verticle[1])),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 0), 2)

de=wid

print("de=",de,"px")

#cv2.imshow("im",image)

show\_images([image])

oriimg=cv2.imread(img\_path)

crop\_img = oriimg[int(tl[1]):int(bl[1]-(tr[0]-tl[0])),int(tl[0]):int(tr[0])]

abvpor=cv2.Canny(crop\_img,1000,700)

#cv2.imshow("edge",abvpor)

# Reading image

font = cv2.FONT\_HERSHEY\_COMPLEX

img2 = crop\_img

# Reading same image in another

# variable and converting to gray scale.

img = abvpor

# Converting image to a binary image

# ( black and white only image).

\_, threshold = cv2.threshold(img, 90, 115, cv2.THRESH\_BINARY)

# Detecting contours in image.

contours, \_= cv2.findContours(threshold, cv2.RETR\_TREE,

cv2.CHAIN\_APPROX\_SIMPLE)

# Going through every contours found in the image.

mxn=0

mnn=10000000000000000000000

tmx=0

tmn=10000000000000000000000

for cnt in contours :

approx = cv2.approxPolyDP(cnt, 0.009 \* cv2.arcLength(cnt, True), True)

# draws boundary of contours.

cv2.drawContours(img2, [approx], 0, (0, 0, 255), 5)

# Used to flatted the array containing

# the co-ordinates of the vertices.

n = approx.ravel()

print(n)

i = 0

for j in n :

if(i % 2 == 0):

x = n[i]

y = n[i + 1]

if y==0:

if tmn>x:

tmn=x

if tmx<x:

tmx=x

if x>mxn:

mxn=x

if x<mnn:

mnn=x

# String containing the co-ordinates.

string = str(x) + " " + str(y)

print(string)

if(i == 0):

# text on topmost co-ordinate.

cv2.putText(img2, "Arrow tip", (x, y),

font, 0.5, (255, 0, 0))

else:

# text on remaining co-ordinates.

cv2.putText(img2, string, (x, y),

font, 0.5, (255, 0, 0))

i = i + 1

#Calculating ds

print(n)

print("here",mxn,mnn)

ds=mxn-mnn

print("ds=",ds,"px")

S=ds/de

print("S=",S)

print("msrment",tmx,tmn)

dIMG=tmx-tmn

scale=(1.2/dIMG)

denew=scale\*de

print(denew,'mm')

rho=input("input ∆ρ in g/cc ")

rho=float(rho)

if ((S>=0.3) and (S<=0.4)):

Hin=(0.34074/(S\*\*2.52303))+(123.9495\*(S\*\*5))-(72.82991\*(S\*\*4))+(0.01320\*(S\*\*3))-(3.38210\*(S\*\*2))+(5.52969\*(S))-1.07260

if ((S>0.4) and (S<=0.46)):

Hin=(0.32720/(S\*\*2.56651))-(0.97553\*(S\*\*2))+(0.84059\*S)-(0.18069)

if ((S>0.46) and (S<=0.59)):

Hin=(0.31968/(S\*\*2.59725))-(0.46898\*(S\*\*2))+(0.50059\*S)-(0.13261)

if ((S>0.59) and (S<=0.68)):

Hin=(0.31522/(S\*\*2.62435))-(0.11714\*(S\*\*2))+(0.15756\*S)-(0.05285)

if ((S>0.68) and (S<=0.9)):

Hin=(0.31345/(S\*\*2.64267))-(0.09155\*(S\*\*2))+(0.14701\*S)-(0.05877)

if ((S>0.9) and (S<=1)):

Hin=(0.30715/(S\*\*2.84636))-(0.69116\*(S\*\*3))+(1.08315\*(S\*\*2))-(0.18341\*S)-(0.20970)

IFT=(rho\*9.81\*(denew\*\*2)\*Hin)

print("σ = ",IFT,"dyne/cm")

# Showing the final image.

#cv2.imshow('image2', img2)

# Exiting the window if 'q' is pressed on the keyboard.

if cv2.waitKey(0) & 0xFF == ord('q'):

cv2.destroyAllWindows()

#cv2.imshow("cropped", crop\_img)

cv2.waitKey(0)