CODE

import cv2

import numpy as np

from sklearn.linear\_model import LinearRegression

lx=[]

ly=[]

px=[]

py=[]

dist=10000 #this is the square of distance used in ransac(no need to take root for comparision)

lim=0.5

def fit\_line(bm,bc,img,q): #assume line y+mx+c

#function accepts m and c and fits line onto image

xf1 = -1

yf1 = -1

xf2=yf2=0

if bm == 0:

bm = bm + 0.01

if -bc > 0 and -bc < h:

xf1 = 0

yf1 = -bc

if -bm \* w - bc > 0 and -bm \* w - bc < h:

if xf1 != -1:

xf2 = xf1

yf2 = yf1

xf1 = w

yf1 = -bm \* w - bc

if (-h - bc) / bm > 0 and (-h - bc) / bm < w:

if xf1 != -1:

xf2 = xf1

yf2 = yf1

xf1 = (-h - bc) / bm

yf1 = h

if (- bc) / bm > 0 and (- bc) / bm < w:

if xf1 != -1:

xf2 = xf1

yf2 = yf1

xf1 = (- bc) / bm

yf1 = 0

print(bc, bm)

# print(xf1,yf1,xf2,yf2)

# print(h,w)

if q==-1:

cv2.line(img, (int(xf1), int(yf1)), (int(xf2), int(yf2)), [0,0,255], 5)

else:

cv2.line(img, (int(xf1), int(yf1)), (int(xf2), int(yf2)), [255, 255 \* q, 0], 5)

def getContours(img,imgBlankc):#finds contours and stores coordinates of centers of contours in lx and ly

contours,hierarchy = cv2.findContours(img,cv2.RETR\_LIST,cv2.CHAIN\_APPROX\_SIMPLE)

#print(len(contours))

n=0

for cnt in contours:

area=cv2.contourArea(cnt)

if area>10 and area<10000: #area>10 removes noise area<10000 removes the contour of external frame

#print(area)

n=n+1

M = cv2.moments(cnt)

x=M['m10']/(M['m00'])#x and y coordinates of moment of contour

y=M['m01']/(M['m00'])

lx.append(x)

ly.append(y)

#print(x,y)

#print("ghc")

cv2.drawContours(imgBlankc, cnt, -1, (0, 255, 0), 1)

print("n=",n)

return n

def distance(st,m,c): #find square of distance between point(lx[st],ly[st] from line y+mx+c=0

x=lx[st]

y=ly[st]

d=pow((y+m\*x+c),2)/(1+m\*m)

return d

def ransac(img,n):

rc = 0

bcount=0

bm=0

bc=0

for i in range(1,n):

for j in range(0,i):

y1=ly[i]

x1=lx[i]

y2=ly[j]

x2=lx[j]

if x1==x2:

continue

m=(y1-y2)/(x2-x1) #assume line passing thru (x1,y1) and (x2,y2)

c=(y1\*x2-y2\*x1)/(x1-x2)

count=0

rc=rc+1

for k in range(0,n): #find inliers for such a line

if distance(k,m,c)<dist:

count=count+1

if count>bcount: #pick line with highest inliers for the given dist value

bcount=count

bm=m

bc=c

px.clear()

py.clear()

for k in range(0, n):

if distance(k, m, c) < dist:

count = count + 1

px.append(lx[k]) #px,py stores inliers

py.append(ly[k])

#bm and bc are the lines that we get from ransac

fit\_line(bm, bc + pow(dist \* (1 + bm \* bm), 0.5), img, 0)

fit\_line(bm, bc - pow(dist \* (1 + bm \* bm), 0.5), img, 0)

#draw two lines to show inliers and outliers

def final\_plot(px,py,img):

nax=np.array(px).reshape((-1,1)) #apply linear regression on inliers and plot final line

nay=np.array(py)

model=LinearRegression()

model.fit(nax,nay)

m=-model.coef\_

c=-model.intercept\_

fit\_line(m,c,img,-1)

img=cv2.imread("Resources/dots.png")

dim=img.shape

h=dim[0]

w=dim[1]

imgBlankc = np.zeros\_like(img)

cv2.imshow("image", img)

#cv2.waitKey(10000)

img1=cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

n=getContours(img1,imgBlankc)

ransac(img,n)#call ransac to get px,py

print("number of inliers=",len(px))

final\_plot(px,py,img)#plot final line

cv2.imshow("image", img)

cv2.waitKey(0)