Recursos para o Python:

Canais no youtube

Universo discreto

<https://www.youtube.com/channel/UCEn6kONg6EC_Ylh0RlInsMw>

Sentdex

<https://www.youtube.com/user/sentdex>

Pysource

<https://www.youtube.com/channel/UC5hHNks012Ca2o_MPLRUuJw>

onde encontrar materiais de estudo, exemplos, foruns e aquivos para python

<https://github.com/>

<https://pt.stackoverflow.com/>

<https://morioh.com/>

##IMPORTANTE##

#esse código foi feito com intensão de iniciar o desafio.

#Ele apresenta funções chaves que podem ajudar no desenvolvimento do código de cada um.

#Esse programa pode conter erros (como apontados em nosso encontra do python).

#Não é necessário se prender à essas funções/métodos. Provavelmente existem formas mais eficientes de se calcular a área que queremos encontrar.

#Existe muita documentação online sobre cada uma das funções. Caso tenha dúvida sobre quais parametros devem ser fornecidos em cada função, basta procurar por sua documentação online.

#https://docs.opencv.org/2.4/ esse é o link para a documentação do openCV

#https://docs.scipy.org/doc/ esse é o link para a documentação do numpy

#Nossa ideia inicial para o desafio é determinar a área do tricoderma em placa de Petri. Vamos tentar quebrar o objetivo principal em pequenos objetivos para discutirmos semanalmente. A pergunta da vez é: qual a melhor forma de determinar o contorno do tricoderma? Fazer borramento na imagem? algum tipo de preenchimento? Não fazer nenhum tratamento? etc...

#inicialmente, são importadas todas as bibliotecas que serão utilizadas no programa

#para importar uma biblioteca, basta digitar "import" e na frente colocar o nome da biblioteca.

#o python segue um padrão ao utilizar funções que estão dentro de uma bibliteca. Essa padrão é sempre escrito como nomeDaBiblioteca.nomeDaFunção(parametros)

#Por exemplo. Dentro da biblioteca openCV (chamada de cv2) existe a função para ler uma imagem (imread), então escreve-se essa função como "cv2.imread('nomedaimagem.extensão').

#existem alguns nomes de bibliotecas que são longos e como no padrão do python sempre precisamos escrever o nome da bibliteca ao utlizar a função, para deixar esse nome mais curto podemos utilizar o comando "as" para "dar um apelido à função". Como feito abaixo, ao colocarmos "import numpy as np" quer dizer que sempre que precisamos acionar a biblioteca numpy, basta escrever np ao invés de escrever o nome inteiro da biblioteca.

import cv2 #importando a biblioteca openCV

import numpy as np #importando a biblioteca numpy com a abreviação "np". A bibliteca numpy possui funções matemáticas e algebra linear (vetores e matrizes)

r\_petri=4.5 #definindo o raio real da placa de petri

area\_real\_petri = np.pi\*r\_petri\*\*2 #calculando a área real da placa de petri utilizando o raio definido

#np.pi é uma função presenta na biblioteca numpy que retorna o valor da constante pi (3,14...)

imagem=cv2.imread('imagem1.jpg',0) # carregando (lendo) uma imagem e atribuindo a ela o nome de "imagem". É importante ter em mente que o nome atribuido à

# imagem (antes do sinal de igual) será o nome de que deve ser utilizado ao longo do programa. O nome da imagem entre

# aspas simples é o nome do arquivo que está salva no computador e pode ter formatos diferentes (neste caso, jpg). Caso a

# extensão seja diferente, será necessário alterar. Ainda, o segundo argumento na função é uma tag que representa se a

# imagem será lida colorida ou em escala de cinza. O valor 1 representa imagem colorida e o valor 0 representa imagem em

# escala de cinza. Por exemplo, se queremos carregar uma imagem em escala de cinza que está salva em nosso computador com

# o nome de "imagem3" salvo em formato ".tif" e atribuir à essa imagem o nome "figura\_original", nosso comando deverá ser

# "figura\_original = cv2.imread('imagem3.tif', 0)

cv2.imshow('imagem original', imagem) # plotando a imagem em uma nova janela. Entre aspas simples fica o nome da janela que será aberta para mostrar a

# imagem. Depois da virgula, vem o nome da imagem que será exibida.

imagem = cv2.GaussianBlur(imagem,(21,21),0) # suavizando a imagem utilizando filtro gaussiano. Esse foi o método que eu encontrei para tentar obter o melhor

# contorno do tricoderma na placa de Petri a título de exemplo. Como discutimos no encontro, não é a melhor forma

# nosso desafio agora é encontrar como fazer isso de forma mais confiável, de modo que a área dentro desse

# contorno seja suficientemente próxima do valor da área real.

imagem=255-imagem # invertendo a imagem (negativo)para evitar problemas com contorno extra. As vezes essa etapa é necessária

# (em outros casos não, como também foi visto durante nosso encontro na segunda-feira). Temos sempre que manter em mente que para o

# python, o cor preta representa fundo da imagem.

imagem\_bi = cv2.adaptiveThreshold(imagem,255, cv2.ADAPTIVE\_THRESH\_MEAN\_C, cv2.THRESH\_BINARY, 45, 10) #Essa função é utilizada (threshold) para limiarizar a

# imagem, ou seja, transformar a imagem em apenas duas

#cores (preto e branco). As intensidades de cores em

#uma

#imagem em escala de cinza variam de 0 (preto)

#até 255

#(branco), ou seja, existem 256 níveis de intensidade.

# Para limiarizar uma imagem, uma valor limiar

#(threshold)

#é fornecido e baseado nesse limiar, todos os valores

# acima viram 255 (banco) e todos os valores abaixo

# recebem intensidade

# 0 (preto). Existem duas formas básicas de realizar a

#limiarização. A primeira é a limiarização adaptativa

#(utilizada nesse programa), que leva em consideração

#as

#variações de cores (gradientes) em cada parte da

#imagem. Quando utilizamos a limiarização adaptativa,

#não é preciso fornecer uma valor para o limiar, pois a

#função irá determinar qual o valor mais apropriado. A

#segunda forma, seria a limiarização simples, onde o

#programador fornece o valor do limiar. Essa função é

# chamada de cv2.Threshold(), procurar para ver os

# parametros necessários

cv2.imshow('imagem binarizada', imagem\_bi) #plotando a imagem binarizada

img\_cont, contorno, hierarchy = cv2.findContours(imagem\_bi,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_NONE) #determinando os pontos de contorno. A fundação

#cv2.findContours devolve todos os contornos encontrados

#na imagem. É importante notar que essa função retorna

#3 parâmetros (aqui chamados de img\_cont, contorno e

#hierarchy). O primeiro parâmetro é uma matriz com os

#valores de intensidade da imagem original, o segundo

#parâmetro é um vetor contendo todo os contornos

#encontrados. Cada contorno encontrado é um vetor.

#Por fim, hierarchy retorna a hierarquia dos contornos,

#de forma geral, o contorno maior é o contorno 0,

#o segundo maior é o contorno 1 e assim por diante.

print(len(contorno)) # o comando print mostra no console o valor desejado. O comando "len" vem da palavra lengh em ingles (comprimento/tamanho)

# e esse comando retorna o tamanho de um vetor ou matriz, ou seja, quanto elementos existem dentro dele. Por exemplo, se temos um vetor

# x = [0,1,2,3,4], e fizermos print(len(x)), irá mostrar no console o valor 5 que é o comprimento do vetor x.

# as próximas linhas de comando só foram possível, pois eu conheço quantos contornos eu tenho na imagem.

# é necessário automatizar a forma de encontrar o contorno desejado, para que funcione em qualquer imagem.

# as próximas linhas atribui os diferentes contornos encontrados à variáveis.

# quando temos um vetor e queremos acessar um componente dentro dele, utilizamos a indexação (como feito nas linhas 83-87 e 90-94).

# para utilizar a indexação basta colocar o nome do vetor seguido por [posição dentro do vetor].

# por exemplo, se temos novamente um vetor x = [0,1,2,3,4] e queremos pegar apenas o valor que está na segunda posição dentro desse vetor (neste caso, na segunda posição temos o valor 1). Então temos que escrever "x[1]. Note que a primeira posição é a posição 0, a segunda posição é a posição 1 e assim por diante. Outro exemplo, se temos um vetor com valores de áreas: areas = [100, 130, 100, 240, 400, 250] e queremos acessar apenas o terceiro valor dentro desse vetor, nossa indexação será areas[2].

#de forma similar, ao utilizarmos o cv2.findContours(), a função nos retorna um vetor (aqui chamado de 'contorno') contendo todos os contornos encontrados na imagem (no total foram 5). Abaixo cada um desses contornos (posição 1, posição 2, posição 3, posição 4, posição 5) é atribuido à uma variável.

cont\_0 = contorno[0] #contorno 0

cont\_1 = contorno[1] #contorno 1

cont\_2 = contorno[2] #contorno 2

cont\_3 = contorno[3] #contorno 3

cont\_4 = contorno[4] #contorno 4

#a função cv2.contourArea(parametro), calcula a área dentro de um dado contorno.

#utilizamos então a indexação para calcular a área de cada um de nossos 5 contornos.

#atribui agora variáveis à área relativa à cada contorno encontrado, neste caso, foram encontrados 4 contornos na imagem.

area\_0 = cv2.contourArea(contorno[0])

area\_1 = cv2.contourArea(contorno[1])

area\_2 = cv2.contourArea(contorno[2])

area\_3 = cv2.contourArea(contorno[3])

area\_4 = cv2.contourArea(contorno[4])

#desenhando os contornos na imagem original chamda de "imagem"

#cv2.drawContours(nomeDaImagem, nomeDoVetorDeContornos, posiçãoDentroDoVetor, (intensidades), grossura da linha no desenho)

a=cv2.drawContours(imagem, contorno, 1, (0,0,255), 2)

b=cv2.drawContours(imagem, contorno, 3, (0,0,255), 2)

#calcula a área da tricodarma por diferença

#neste caso, eu sei por visualização que minha área é a "area\_3", fica como parte do desafio automatizar essa cálculo para casos onde não é posível determinar por visualização.

area\_tr=area\_3

#como a área até o momento foi dada em número de pixels, calcula-se a relação entre o número de pixels de toda a placa de Petri e o número de pixels do tricodarma. Esse passo é importante para conveter número de pixels em unidades físicas.

area\_ratio = area\_3/area\_1

#uma vez conhecendo a relação entre placa de Petri e área da tricoderma e conhecendo também a área real do prato de Petri (medida manualmente), por proporção é determinadada a área da tricoderma.

area\_tr\_real = area\_ratio\*area\_real\_petri

#mostra o valor da área da placa de Petri

print('área placa petri', area\_real\_petri, 'cm2')

#mostra o valor da area da tricoderma

print('área thricoderma',area\_tr\_real, 'cm2')

#próximos cv2.imshow's mostram os contornos nas imagens. Note que apenas plotei 2 contornos, mas existem 5 contornos na figura.

cv2.imshow('imagem 1',a)

cv2.imshow('imagem 2',b)

#IMPOTANTE!! Sempre que for carregar uma imagem, colocar os comandos abaixo como mostrado

cv2.waitKey(0)

cv2.destroyAllWindows()

#Esses comandos fazem com que o sistema "pare" de ler a imagem ao acionar qualquer tecla do teclado.

#caso a imagem seja carregada sem uma forma de finalizar sua leitura, o programa irá travar por entrar em um "loop" infinito.