departamento de eletrónica, telecomunicações e informática

Curso	8309 - Mestrado Integrado em Engenharia Eletrónica e Telecomunicações
Disciplina	41500 – Visão por Computadores na Indústria
Ano letivo	2020/21

Deliverable 2

Deteção de cor e tamanho de Legos com a raspberry pi 4

Autores:

77848	Fábio Almeida		
90367	Ana Sousa		
89833	Micael Ramos		
Turma	P1	Grupo	10

Data	31/05/2021
Docente	António Neves
	Daniel Canedo

Resumo: O presente relatório descreve a milestone 2, isto é, deteção da cor e tamanho de legos com recurso à *raspberry pi 4*. Também é apresentado o *setup* feito, alguns resultados obtidos e algumas conclusões desses mesmos resultados.

Introdução

Nesta *milestone,* e relativamente à anterior, o objetivo é fazer a deteção de cores e tamanho de legos com recurso à raspberry pi 4. Para isso, o algoritmo feito anteriormente com o *dataset* do professor foi adaptado para o nosso, de modo a obter os resultados esperados nesta fase do projeto.

Assim, foi criado o *setup* que se encontra na figura 1. Este *setup* é constituído por uma caixa fechada com uma fita led no interior, de modo a simular um ambiente de luz controlada. A intensidade da luz pode ser regulada, de modo a minimizar reflexos da luz nas peças. É de notar que, por cima da fita led foi colocada uma folha de papel vegetal, que tem como objetivo tentar diminuir os reflexos de luz que poderão aparecer nos legos.

Figura 1 - Setup utilizado no projeto

Resultados e Análise de Resultados

O algoritmo desenvolvido na *milestone* anterior tinha como objetivo definir os limites das 3 componentes do HSV para as diferentes cores dos diversos legos. Após termos as várias gamas de valores do HSV de modo a identificar as diferentes cores, passamos para a identificação do contorno dos legos. Para esta fase, de modo a obter melhores resultados, foi necessário tratar as "máscaras", ou seja, as imagens binárias de cada cor num conjunto de legos, de modo a preencher e/ou eliminar pequenas falhas que possam ter resultado de uma má definição dos limites de HSV, ou de zonas demasiados iluminadas. Para isso foram usados processos morfológicos, nomeadamente os processos de *Opening* e *Closing*. Na figura 2, 3 e 4 são mostradas, respetivamente, a "máscara" original, após *Opening* e após *Closing.*

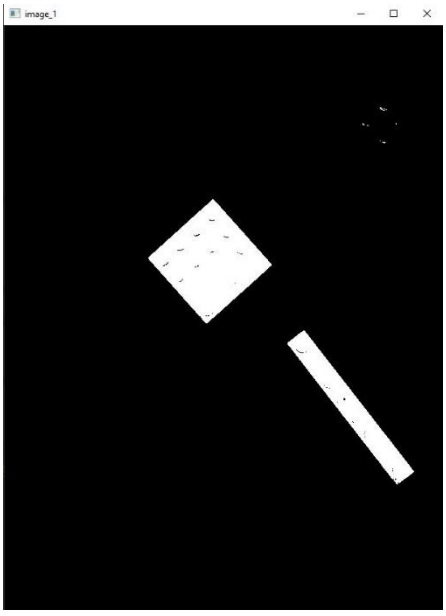

Figura 2 - "Máscara Original"

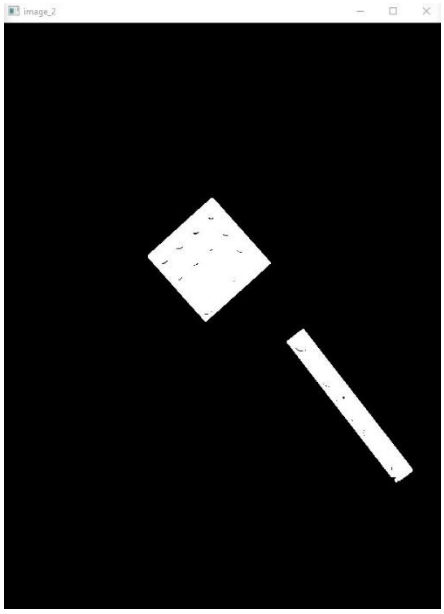

Figura 3 - "Máscara" após Opening

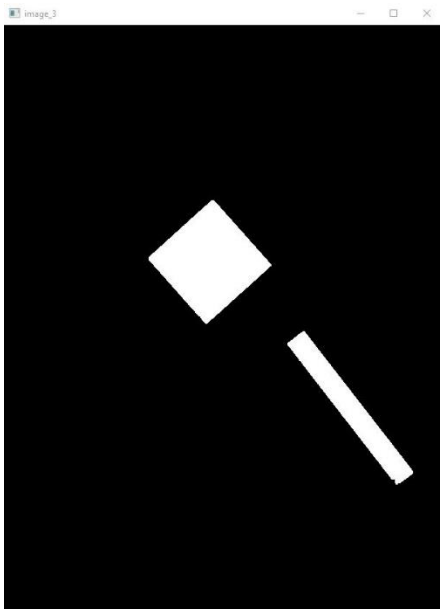

Figure 4 - "Máscara" após Closing

Após efetuar este pós processamento, aplicou-se o Canny() seguido pelo findContours(). Com o resultado obtido, e de modo a garantir que erros não corrigidos pelo processo de pós processamento são minimizados, é calculada a área de cada uma das "zonas brancas" da máscara, e apenas serão consideradas como sendo um lego, as áreas superiores a um determinado valor. De seguida é desenhado o contorno de cada lego e calculado seu tamanho. O tamanho de cada lego é calculado em função do tamanho do contorno desenhado, e da sua relação com um valor encontrado pelo grupo de forma experimental. É ainda feita uma contagem para determinar o número de legos identificados. Depois de todos os legos de uma determinada cor serem identificados é repetido todo o processo descrito para as diferentes cores configuradas. No caso de uma dada cor estar configurada, mas não existir nenhum lego dessa cor na imagem, o sistema indica que a quantidade de legos dessa cor é 0.

Inicialmente utilizamos o *dataset* fornecido pelo professor para verificar se o algoritmo implementado anteriormente funcionava na *raspberry pi 4*. Assim, obtemos a figura 5, onde podemos ver que os resultados foram os esperados.

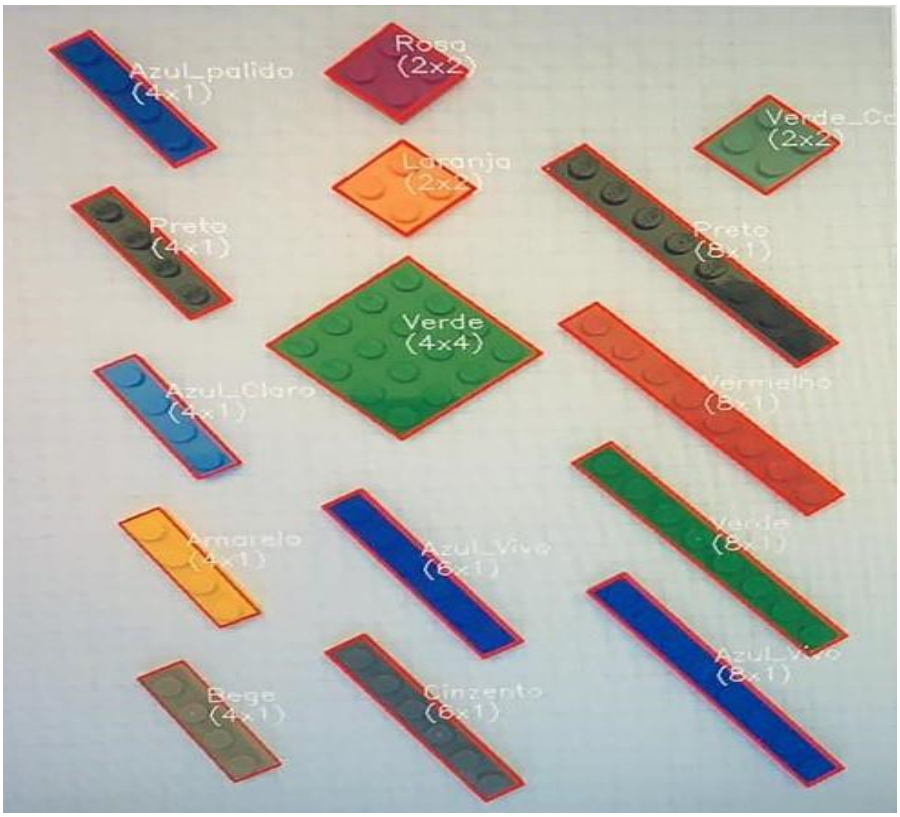

Figura 5- Resultado obtido com o dataset do professor

Depois de experimentar o algoritmo anterior, começamos por configurar as várias cores a serem identificadas, isto é, definimos as gamas dos valores de HSV das várias cores, usando a *raspberry pi* e a sua câmara e o algoritmo desenvolvido na *milestone* anterior.

Inicialmente começamos por definir as máscaras de cada cor individualmente, como mostram as figuras 6 a 10. Depois de alguns testes, chegámos à conclusão de que os resultados obtidos não estavam a ser os esperados com a "calibração" de uma cor por cada imagem. Assim, passamos para a definição das máscaras de todas as cores de uma vez, tal como mostra a figura 11. De notar que, a resolução pré-definida da câmara da *Raspberry pi* não era a mais adequada e então, decidimos acrescentar ao nosso código a definição de uma resolução mais adequada, para melhorar o resultado obtido.

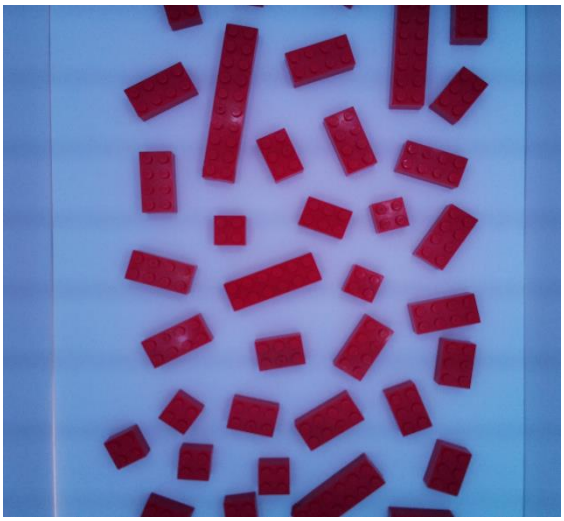

Figura 6- Máscara para a cor vermelha

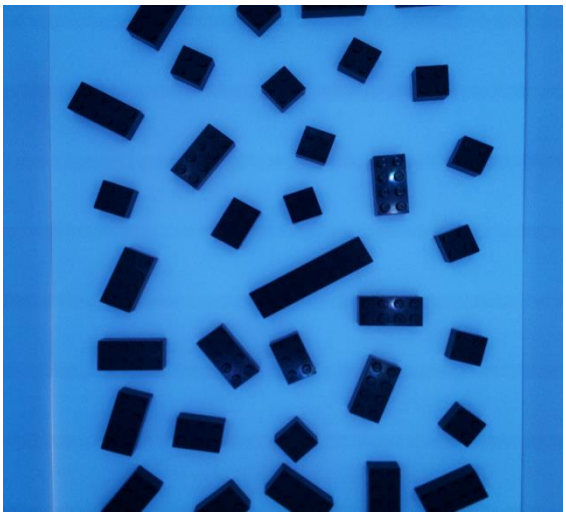

Figura 7- Máscara para a cor preta

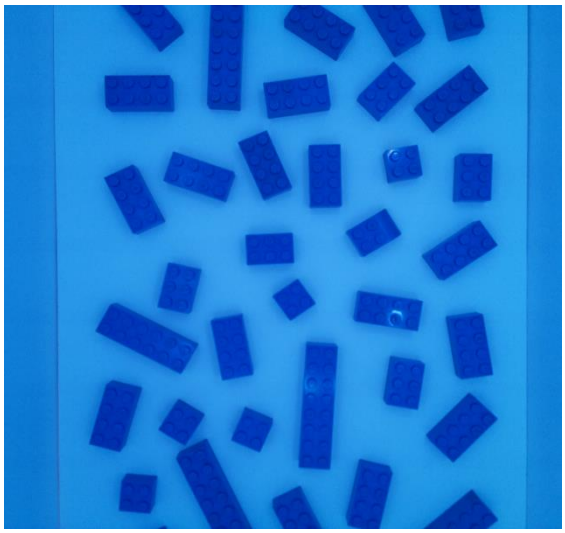

Figura 8- Máscara para a cor azul

Figura 9- Máscara para a cor amarela

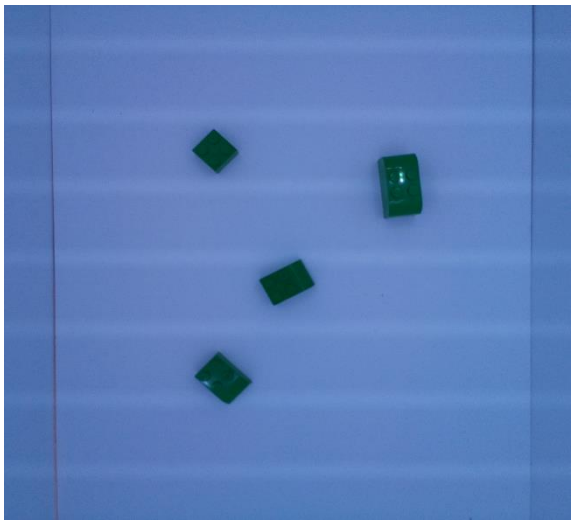

Figura 10- Máscara para a cor verde

Figura 11- Máscara final para várias cores

Depois de feita a máscara para algumas cores, conseguimos obter a cor e o tamanho de cada peça de lego em tempo real, tal como mostram as figuras 12, 13 e 14.

Figura 12- Resultado obtido em tempo real

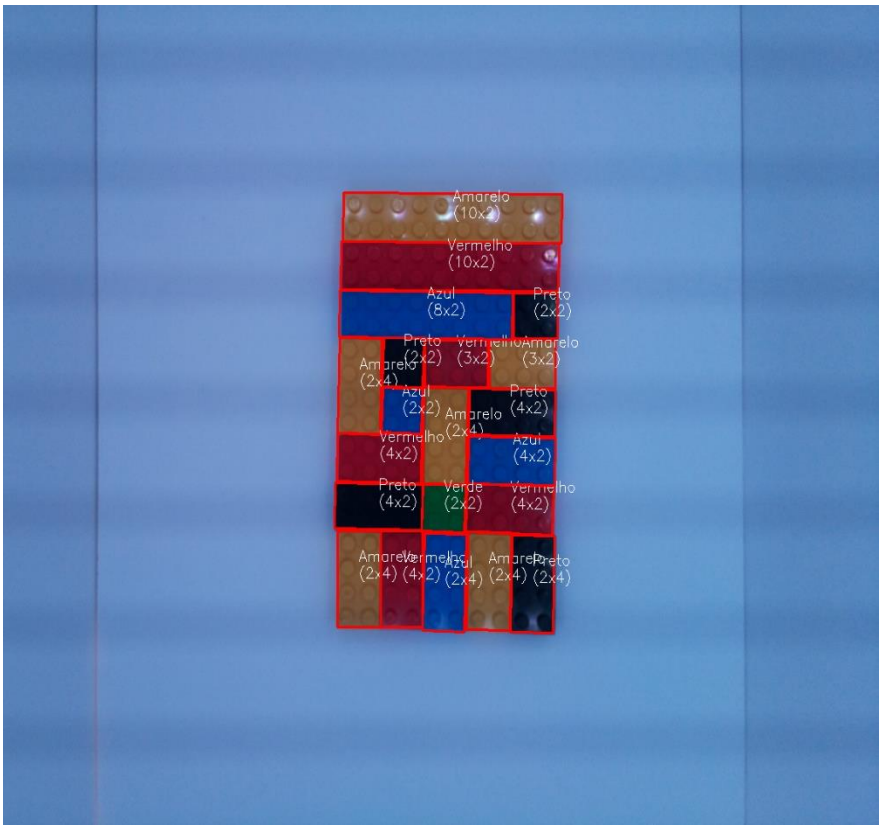

Figura 13- Resultado obtido em tempo real

Figura 14- Resultado obtido em tempo real

Tal como referido anteriormente, o sistema é ainda capaz de fazer a contagem dos legos de cada cor, tal como mostra a Figura 15.

Figura 15 - Contagem dos legos de cada cor

Após a análise dos resultados obtidos com recurso à *Raspberry pi*, reparamos que as figuras apresentam uma espécie de riscas horizontais por cima dos legos. Também, dependendo da posição das peças dos legos, podemos ver nas figuras acima que em certos sítios a luz é mais incidente. Estes dois fatores, por vezes, geram erros no sistema.

Como a calibração da câmara ainda não foi feita, esta poderá ser uma razão para estes erros e por isso, a próxima tarefa será fazer a calibração da câmara e outras possíveis alterações para melhorar os resultados obtidos.

É de notar que, as peças de cor branca não conseguem ainda ser detetadas, uma vez que devido ao *background* e às condições de luminosidade, os valores do HSV para esta cor e para o *background*, ficam dentro da mesma gama.

Conclusões

No que diz respeito a esta fase do projeto, os objetivos definidos no início foram em grande parte superados. Como já foi referido anteriormente, os próximos passos serão fazer a calibração da câmara, bem como tentar melhorar a incidência da luz em certos pontos, quer ajustando a intensidade e/ou a distribuição da luz no *setup,* quer por vi da *software.*

Deste modo, com o trabalho feito até ao momento, consideramos estar num bom caminho em vista dos requisitos definidos para a elaboração do projeto.

Anexos

```python
def mouse_click():
    import cv2
    global init

    # Variables
    init = [0,0,0]

    # Load image
    img = cv2.imread(cv2.samples.findFile("legos.png"))


    # Reduce the image
    height, width = img.shape[:2]
    size = (int(width * 0.5), int(height * 0.5))  # bgr
    img = cv2.resize(img, size, interpolation=cv2.INTER_AREA)

    # Convert BGR to HSV
    HSV = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    # Mouse left button action
    def getposHsv(event, x, y, flags, param):
        global init
        if event == cv2.EVENT_LBUTTONDOWN:
            print("HSV is", HSV[y, x])
            init[0]=HSV[y,x][0]
            init[1] = HSV[y, x][1]
            init[2] = HSV[y, x][2]

    # Display the result image
    cv2.imshow('image', img)

    # Call mouse left button action
    cv2.setMouseCallback("image", getposHsv)

    while(1):
        k = cv2.waitKey(20) & 0xFF
        if k == 27:
            break
        elif k == ord('s'):
            print('init is',init)
            return init
    cv2.destroyAllWindows()
```

```python
# min_max.py
#
# System Calibration program
# This program allows us to creat all the mask for the color that the system can identify
#
# Running process:
#   - Pick a point with the left mouse button
#   - Press 's' to save the point picked
#   / A new window opens
#   - Click in a point of the image to include it in the mask (if click in a point and realize that you don't want the result,
#   - After the mask is fine press 's' to save it
#   - Write the name of the file where the masks will be saved
#   - Write the color of the maks created
#   - Check the message with the indication that the color has been recorded
#   - Repeat the process for all desired colors

import cv2
import numpy as np
import ast
from mouse import mouse_click
from time import sleep
from picamera import PiCamera, Color

#capture_image()
camera= PiCamera()
camera.resolution = (2000, 1944)
camera.start_preview()
sleep(2)
camera.capture('/home/pi/Desktop/legos.png')
camera.stop_preview()

# Variables
global init
init = [0,0,0]
cores = {}

# Call mouse_click() function in order to obtaining the initial limit values of the mask
init_val = mouse_click()

def nothing(x):
    pass

# Load image
image = cv2.imread(cv2.samples.findFile("legos.png"))

```

```python
# Mouse right button action
def getposHsv_right(event, x, y, flags, param):
    global hMin,sMin,vMin,hMax,sMax,vMax,hMin_p,sMin_p,vMin_p,hMax_p,sMax_p,vMax_p
    if event == cv2.EVENT_RBUTTONDOWN:
        print("HSV is", hsv[y, x])

        #   Save the current values before update to allows us to return to previus mask state
        hMin_p = hMin
        sMin_p = sMin
        vMin_p = vMin
        hMax_p = hMax
        sMax_p = sMax
        vMax_p = vMax

        #   See if the HSV values of the picked pixel is outside the limits of the mask, and if so, update them
        if(hsv[y, x][0] < hMin):
            hMin=hsv[y, x][0]
        if(hsv[y, x][1] < sMin):
            sMin = hsv[y, x][1]
        if(hsv[y, x][2] < vMin):
            vMin = hsv[y, x][2]
        if(hsv[y, x][0] > hMax):
            hMax = hsv[y, x][0]
        if(hsv[y, x][1] > sMax):
            sMax = hsv[y, x][1]
        if(hsv[y, x][2] > vMax):
            vMax = hsv[y, x][2]



# Reduce the image
height, width = image.shape[:2]
size = (int(width * 0.5), int(height * 0.3))  # bgr
image = cv2.resize(image, size, interpolation=cv2.INTER_AREA)

# Create a window
cv2.namedWindow('image')

# Create the trackbars for see the color change
cv2.createTrackbar('HMin', 'image', 0, 179, nothing)
cv2.createTrackbar('SMin', 'image', 0, 255, nothing)
cv2.createTrackbar('VMin', 'image', 0, 255, nothing)
cv2.createTrackbar('HMax', 'image', 0, 179, nothing)
cv2.createTrackbar('SMax', 'image', 0, 255, nothing)
cv2.createTrackbar('VMax', 'image', 0, 255, nothing)

```

```
# Initialize HSV min/max values with the returned by the mouse_click function
hMin = init_val[0]-5
sMin = init_val[1]-1
vMin = init_val[2]-1
hMax = init_val[0]+5
sMax = init_val[1]+1
vMax = init_val[2]+1


while(1):

    # Call mouse right button action
    cv2.setMouseCallback("image", getposHsv_right)

    #  Set the limit values of the mask in to trackbars
    cv2.setTrackbarPos('HMin', 'image', hMin)
    cv2.setTrackbarPos('SMin', 'image', sMin)
    cv2.setTrackbarPos('VMin', 'image', vMin)
    cv2.setTrackbarPos('HMax', 'image', hMax)
    cv2.setTrackbarPos('VMax', 'image', sMax)
    cv2.setTrackbarPos('SMax', 'image', vMax)

    # Set minimum and maximum HSV values to display
    lower = np.array([int(hMin), int(sMin), int(vMin)])
    upper = np.array([int(hMax), int(sMax), int(vMax)])
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # Create the mak
    mask = cv2.inRange(hsv, lower, upper)

    # Create an image using the mask
    result = cv2.bitwise_and(image, image, mask=mask)

    # Display the result image
    cv2.imshow('image', result)
    k = cv2.waitKey(10) & 0xFF
    if k == ord('q'):
        break
    # if you press 'r' the mask return to the previous state
    elif k == ord('r'):
        print(hMin_p,sMin_p,vMin_p,hMax_p,sMax_p,vMax_p)
        hMin = hMin_p
        sMin = sMin_p
        vMin = vMin_p
        hMax = hMax_p
        sMax = sMax_p
        vMax = vMax_p
```

```
    # if maks is fine and you wont to save it
    elif k == ord('s'):

        # read the name of the file, open it and read the dictionary
        try:
            nome_arquivo = input('Nome do ficheiro:')
            arquivo = open(nome_arquivo, 'r+')
            contente = arquivo.read()
            cores = ast.literal_eval(contente)
            arquivo.seek(0)
            arquivo.truncate(0)
        # if file does not exist, create it
        except FileNotFoundError:
            arquivo = open(nome_arquivo, 'w+')



        print("Cor a gravar:")
        color = input()

        # see if the color already exists
        if(str(color+'_l') in cores):
            print('Cor já existe')
            break
        else:
            texto_l = (str(hMin) + ',' + str(sMin) + ',' + str(vMin))
            texto_u = (str(hMax) + ',' + str(sMax) + ',' + str(vMax))
            cores[color+'_l'] = texto_l
            cores[color+'_u'] = texto_u

        # write the dictionary in to the file
        arquivo.write(str(cores))
        arquivo.close()
        print("Cor gravada:", color)

        # call the mouse_click function again to repeat the all the process for every color you want
        init_val=mouse_click()
        hMin = init_val[0] - 10
        sMin = init_val[1] - 5
        vMin = init_val[2] - 5
        hMax = init_val[0] + 10
        sMax = init_val[1] + 5
        vMax = init_val[2] + 5

cv2.destroyAllWindows()
```

```python
import cv2 as cv
import numpy as np
import ast
from sympy import *
#from skimage.morphology import watershed
import math
from time import sleep
from picamera import PiCamera, Color

from capture_img import capture_image

array_cores = []
array_valores = []

#capture_image()
camera= PiCamera()
camera.resolution = (2592, 1944)
camera.start_preview()
sleep(2)
camera.capture('/home/pi/Desktop/legos.png')
camera.stop_preview()
# Load image
image = cv.imread(cv.samples.findFile("legos.png"))

# Reduce the image
height, width = image.shape[:2]
size = (int(width * 0.50), int(height * 0.50))  # bgr
image = cv.resize(image, size, interpolation=cv.INTER_AREA)


# Open file
arquivo = open("cores_3.txt", 'r+')
contente = arquivo.read()

# Import Dictionary
cores = ast.literal_eval(contente)

for x in cores:

    array_cores.append(str(x[:-2]))

#print(array_cores)

```

```
for x in cores.values():

    array_valores.append(x)

#print(array_valores)

count = 0

while count<len(array_valores):

    #print(array_cores[count])
    cor = array_cores[count]

    lower = np.fromstring(array_valores[count], sep=',')
    upper = np.fromstring(array_valores[count+1], sep=',')
    count = count + 2

    hsv = cv.cvtColor(image, cv.COLOR_BGR2HSV)

    # Create mask
    mask = cv.inRange(hsv, lower, upper)

    # Pos-processing
    kernel = cv.getStructuringElement(cv.MORPH_RECT, (3, 3))
    opening = cv.morphologyEx(mask, cv.MORPH_OPEN, kernel)
    closing = cv.morphologyEx(opening, cv.MORPH_CLOSE, kernel)


    #cv.imshow('mask',mask)
    #cv.imshow('opening',closing)
    #cv.imshow('closing',closing)

    edges = cv.Canny(closing, 100, 200)

    contours, hierarchy = cv.findContours(edges, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)

    n_legos = 0;

    for contour in contours:
        approx = cv.approxPolyDP(contour, 0.01 * cv.arcLength(contour, True), True)
        area = cv.contourArea(contour)
        if area > 1000:
            n_legos = n_legos + 1;
            #print("Area:", area)

```

```
            # Retangulo com inclinacao
            rect = cv.minAreaRect(contour)
            box = cv.boxPoints(rect)
            box = np.int0(box)
            cv.drawContours(image, [box], 0, (0, 0, 255), 2)
            #cv2.drawContours(image, [contour], 0, (0, 0, 255), 2)
            lado_1 = round(rect[1][1]/29)
            lado_2 = round(rect[1][0]/29)

            #print('Tamanho em pixeis: '+str(rect[1][1])+' x '+str(rect[1][0]))
            #print('Tamanho em encaixes: ' + str(lado_1) + ' x ' + str(lado_2))

            # Retangulo sem inclinacao
            #x, y, w, h = cv2.boundingRect(contour)
            #cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)

            #cv2.drawContours(image, [contour], 0, (0, 0, 255), 2)

            # finding center point of shape
            M = cv.moments(contour)
            if M['m00'] != 0.0:
                x = int(M['m10'] / M['m00'])
                y = int(M['m01'] / M['m00'])
                #print("Centroide: "+str(x)+' ; '+str(y))
                #print(x, y)

            perimetro = cv.arcLength(contour, True)
            #print("Perimetro:", perimetro)


            cv.putText(image, cor, (x , y - 20), cv.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 1)
            cv.putText(image, '('+str(lado_1)+'x'+str(lado_2)+')', (x, y), cv.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 1)
    print(' ')
    print("Numero de legos "+ str(cor)+" = "+str(n_legos))
    print(' ')


while (1):

    # Display result image
    cv.imshow('image', image)
    #img_name = "result.png".format(0)
    cv.imwrite('Result.png', image)
    k = cv.waitKey(10) & 0xFF
    if k == ord('q'):
        break
```