KKY/ZDO

Dokumentace ke kódu na detekci kleštíků

Pavlína Täglová

Veronika Osminová

29.5.2021

Nároky na vstupní data:

Pro správnou funkčnost kódu je nutná přítomnost QR kódu ve vstupním obraze. Dále je vhodné před pořízením obrazu zkontrolovat, že QR není zasypaný mělí a zajistit co možná nejvíce rovnoměrné osvětlení snímané plochy. Měl by měla být na podložce dostatečně rozhrnuta, aby se eliminoval počet zakrytých kleštíků, které je obtížnější detekovat.

Načtení vstupních dat:

Třída __init__ obsahuje kód pro získání dat v požadovaném 4D array formátu.

```
22  class VarroaDetector():
23      def __init__(self):
24          pth = "d:\images\*.jpg"
25          fotky = glob.glob(pth)
26          ll = len(fotky)
27          data = np.zeros((ll, 3024, 4032, 3), dtype = int)
28
29          for i in range(ll):
30              img_fotky = plt.imread(fotky[i])
31              image = img_fotky[np.newaxis, ...]
32              f = np.asarray(image)
33              data[i,:,:,:] = f
34          pass
```

Třída predikt obsahuje kód pro detekci kleštíků.

Popis kódu:

Definice 3D pole pro výstup (ubrání dimenze pro počet kanálů):

```
output = np.zeros_like(data[...,0])

for m in range(data.shape[0]):
    img = data[m,:,:,:]
```

Pro každý vstupní obrázek se provedou následující operace:

1) **převod z RGB do HSV**
   Dále pracujeme se 3. kanálem HSV (Value). Zkoušely jsme i převod do grayscale, ale pomocí Value získáváme lepší výsledky.
   ```
   umg_hsv = skimage.color.rgb2hsv(img)
   obr_v = umg_hsv[:,:,2]
   ```
2) **Filtrace a prahování**
   Provedeme filtraci pomocí Gaussova filtru pro „rozmazání“ nežádoucích odlišných jasů (měl, odlesk) na kleštíkách.
   Dále prahováním získáme dva binární obrazy. Obraz img_tr_v získáme z vyfiltrovaného obrazu gaussian_img_v a použijeme ho později pro vyhledání kleštíků. Obraz imgQR_tr získáme z obrazu obr_v (Value) a použijeme ho pro vyhledání prvků QR kodu.

```
img_vyrez_v = obr_v
gaussian_img_v = ndimage.gaussian_filter(img_vyrez_v, sigma=2)
img_tr_v = (gaussian_img_v < 0.32 * 10 ** -7)
imgQR_tr = (img_vyrez_v < 0.45 * 10 ** -7)
```

3) Vyhledání prvků QR kódu

Abychom byli schopni určit předpokládanou maximální velikost kleštíka v konkrétním obraze, použijeme předpoklad, že kleštík v „normální poloze“ zabírá ¼ plochy čtverečku na hranici QR kódu. Proto nejprve vyhledáme tyto čtverečky a zjistíme jejich velikost v konkrétném obraze. Použijeme morfologickou operaci otevření pro odstranění případných drobných nečistot na hranách čtverečků. Poté odstraníme objekty <80 pixelů (předpokládáme, že vstupní obraz bude focen z dostatečné blízkosti a čtvereček bude vždy >80 pixelů).
Provedeme barvení (labeling) a uložíme si prostorové vlastnosti (regionprops).

```
kernel_img3 = skimage.morphology.square(5).astype(np.uint8)
imgQR_tr_op = skimage.morphology.binary_opening(imgQR_tr, kernel_img3)
imgQR_tr_rso = skimage.morphology.remove_small_objects(imgQR_tr_op, 80)

imgQR_label = skimage.morphology.label(imgQR_tr_rso > 0)
propsQR = skimage.measure.regionprops(imgQR_label+1)
```

Následuje for-cyklus pro nalezení čtverečků QR kódu. Z obarvených objekt určíme jako čtverce ty, které splňují podmínky: mají vysokou hodnotu pravoúhlosti a jejich hlavní a vedlejší osa jsou si skoro rovny.

```
for i in range(len(propsQR)):
    a = propsQR[i].major_axis_length
    b = propsQR[i].minor_axis_length
    area = propsQR[i].area
    pravouhlost = area/(a*b)

    if pravouhlost > 0.7:
        if b > a - 2:
            if ctverec_max1 < a:
```

Z takto detekovaných objektů vybíráme 3 největší. Předpokládáme, že se na obrázku nenachází jiný objekt čtvercového tvaru, který by byl větší než čtverečky na QR kódu.

```
if pravouhlost > 0.7:
    if b > a - 2:
        if ctverec_max1 < a:
            #print ("id: ", i)
            ctverec_max_id3 = ctverec_max_id2
            ctverec_max3 = ctverec_max2
            ctverec_max2 = ctverec_max1
            ctverec_max_id2 = ctverec_max_id1
            ctverec_max1 = a
            ctverec_max_id1 = i
        elif ctverec_max2 < a:
            ctverec_max_id3 = ctverec_max_id2
            ctverec_max3 = ctverec_max2
            ctverec_max2 = a
            ctverec_max_id2 = i
        elif ctverec_max3 < a:
            ctverec_max3 = a
            ctverec_max_id3 = i
```

Z nalezených čtverečků vypočteme jejich průměrnou velikost. Může nastat situace, že některý ze čtverečků nenalezneme (kvůli zakrytí mělí nebo odlesku fólie). Pro tento případ testujeme, jestli jsou si tyto 3 největší nalezené čtverce velikostí podobné. Poměr velikostí dvou z nich musí být větší než 0.8 (ideálně by měl být = 1). Pokud zjistíme, že je jeden nebo dva čtverce výrazně menší než největší nalezený, potom je z výpočtu průměrné velikosti vyřadíme.

```
a1 = propsQR[ctverec_max_id1].area
a2 = propsQR[ctverec_max_id2].area
a3 = propsQR[ctverec_max_id3].area

# test nalezených čtverců
if a2/a1 < 0.8:
    #print("jen 1")
    prum_a = a1
elif a2/a1 > 0.8 and a3/a2 <0.8:
    #print("jen 2")
    prum_a = (a1 + a2) / 2
else:
    #print("všechny 3")
    prum_a = (a1 + a2 + a3) / 3
```

4) Vyhledání kleštíků

Pro naprahovaný obraz img_tr_v provedeme morfologické operace eroze a dilatace (otevření) a odebrání malých objektů. Provedeme barvení (labeling) a uložíme si prostorové vlastnosti (regionprops).

```
small = (prum_a/4)/2.5

kernel_img2 = skimage.morphology.square(3).astype(np.uint8)
img_tr_er = skimage.morphology.binary_erosion(img_tr_v, kernel_img2)
img_tr_rso = skimage.morphology.remove_small_objects(img_tr_er, small)
img_tr_di = skimage.morphology.binary_dilation(img_tr_rso, kernel_img2)

img_label = skimage.morphology.label(img_tr_rso > 0)
props = skimage.measure.regionprops(img_label+1)
```

Dále si definujeme maximální a minimální předpokládanou velikost kleštíka v daném obraze a prázdné pole pro uložení výsledného binárního obrazu (masky s nalezenými kleštíky).

```
# POKUS O NALEZENÍ KLEŠTÍKŮ ------------------------------
max_v_kl = prum_a/4
min_v_kl = max_v_kl/2

vysledek = np.zeros_like(img_vyrez_v)
```

V následujícím for-cyklu provedeme detekci kleštíků podle velikosti a nekompaktnosti oblasti. Předpokládáme, že kleštík má nekompaktnost blízkou nekompaktnosti kruhu. Hodnoty minimální a maximální hodnoty pro nekompaktnost byly určeny empiricky. Pokud je oblast určena jako kleštík, vloží se do výsledného obrazu s hodnotou 1.

```
for i in range(len(props)):
    nekomp = props[i].perimeter**2 / props[i].area
    #a = props[i].major_axis_length
    #b = props[i].minor_axis_length
    #obvod = props[i].perimeter
    area = props[i].area

    if min_v_kl < area < max_v_kl:
        if 12 < nekomp < 13.8:
            hezky_klestik = hezky_klestik + 1
            id_hezky.append(i)
            vysledek[img_label==i] = 1
            #vys[img_label==i] = [255,0,0]
```

Další část větvení if-else detekuje oblasti, které splňují jednu z podmínek dobře a druhou o něco hůře. Původně byly tyto oblasti přidány do výsledného obrazu s hodnotou 0.5 značící menší jistotu správnosti určení kleštíka. Tuto funkcionalitu je možno opět spustit odkomentováním příslušných řádků.

```
        if min_v_kl < area < max_v_kl:
            if 12 < nekomp < 13.8:
                hezky_klestik = hezky_klestik + 1
                id_hezky.append(i)
                vysledek[img_label==i] = 1
                #vys[img_label==i] = [255,0,0]
            elif 11 < nekomp < 15:
                fuj_klestik = fuj_klestik + 1
                id_fuj.append(i)
                #vysledek[img_label==i] = 0.5
                #vys[img_label==i] = [0,255,0]
        elif min_v_kl_f < area < max_v_kl:
            if 11 < nekomp < 15:
                fuj_klestik = fuj_klestik + 1
                id_fuj.append(i)
                #vysledek[img_label==i] = 0.5
                #vys[img_label==i] = [0,0,255]
```

Nakonec je výsledný binární obraz uložen do pole výstupu.

```
output[m,:,:] = vysledek
```