Gépi látás

GKLB_INTM038

Dobókocka számláló alkalmazás

Antal Krisztián – KYCNDX – 2021/2022 1. félév

Tartalomjegyzék

1. Téma bemutatása:

Féléves feladatomban egy kockadobások értékét számláló alkalmazást készítek python programnyelven. Az elkészített programnak képesnek kell lennie megszámolni az adott kockákkal dobott értéket, valamint azt is megállapítani, hogy azt az értéket hány darab kockával dobták.
Fontos részlete a feladatnak, hogy figyelemmel kell lenni a képeken uralkodó fényviszonyra, illetve amennyiben a fényképen látszik a dobókocka oldalsó számlapja, az ne számítson bele a megszámlált értékbe.

2. Felhasznált eszközök ismertetése

A feladatom készítése során a kockákról készített fényképek hátteréül egy tölgy mintás lapú íróasztal szolgált. A szükséges árnyékot egy asztali lámpa biztosította, valamint a dobásokat három, fehér alapon fekete pontokkal jelölt dobókockákkal hajtottam végre. A fényképeket mobiltelefonommal készítettem, ami 4608x3456 pixel méretű képeket készít.
A forráskód létrehozásához és fordításához a Thonny nevű alkalmazást használtam fel.

3. Elméleti háttér ismertetése

A feladatban ismertetett problémák megoldásához az egyik legfontosabb elméleti ismeret, amiben elmélyültem az élek detektálása volt. Az éldetektálásnak nagy jelentősége van a képfeldolgozás és gépi látás során, hiszen az emberi látás is nagy mértékben támaszkodik az élek felismerésére.
Egy képen az él ott található, ahol az intenzitási érték nagy mértékben, hirtelen megváltozik. Az éldetektálás célja egy, vagy több objektum határvonalainak meghatározása. Az élek meghatározását azonban nehezíti több tényező is: árnyék, alakzatok az objektumon belül, valamint a törésvonal az objektum felületén.
Munkám során többek között Canny detektort is használni fogom a szürkeárnyalatos képeken való éldetektálásra.
Az élek és alakzatok pontos meghatározásához nélkülözhetetlen a szürkeárnyalatos transzformációja egy képnek, ugyanis az ilyen módon átalakított képen felismerhetőbbek lesznek az azonos tulajdonsággal rendelkező pontok halmazai.
Mindezek mellett fontos megemlíteni még zajszűrést, mint alkalmazandó technikát. Ennek szerepe is elengedhetetlen, hogy az objektumok a háttér és az árnyékok zavaró hatásai nélkül kerüljenek értelmezésre.

4. Képek bemutatása, azok elemzése

A feladatom során három fajta képet készítettem. Olyanokat, amin 1, 2 és 3 db kockával dobott értéket kell a programnak érzékelnie és megszámolnia. Ezek típusonként a következőképpen néznek ki:

1 kocka

2 kocka

3 kocka

Jól látható mindegyik képen az oldalról érkező fénynek köszönhető, a kockák által vetett árnyék, valamint az asztal mintája, ami a képfeldolgozáskor külön zavart fog eredményezni a háttérben, amit majd szűrni kell az alakzatok megszámlálásakor.

5. Képfeldolgozás megkezdése

Első lépésben a képfeldolgozáshoz szükséges könyvtárakat importáltam be a keretprogramba.

```
import cv2
import numpy as np
```

Ezután a cv2 imread funkcióját meghívva beolvastam a képet annak eredeti formájában. Erre azért van szükség, mert kép tényleges feldolgozása előtt szükségszerű a dobókockák által vetett árnyékok megszüntetése, még mielőtt a fényképet szürkeárnyalatossá alakítanánk át.

```
img = cv2.imread("./kepek/20220108_152049.jpg", -1 )
```

Az eredetileg készített fényképek méretét és részletgazdagságát túl nagynak találtam, aminek köszönhetően azok feldolgozás közben szemmel átláthatatlanok lettek, valamint a kód futtatása közben a számítógép erőforrás kihasználása is drasztikusan megnőtt, beleértve a kód fordítására felhasznált időt is.
Emiatt az eredeti 4608x3456 felbontású fényképeket 800x600 pixelesre méreteztem át.

```
down_width = 800
down_height = 600
down_points = (down_width, down_height)
resized_down = cv2.resize(img, down_points, interpolation= cv2.INTER_LINEAR)
```

Az átméretezett képen szükségszerű az árnyékot elhalványítani, hogy az a későbbi konverzió után már csak, mint zaj jelenjen meg és ne mint egy objektumként is feldolgozható alakzat.

```
rgb_planes = cv2.split(resized_down)

result_planes = []
for plane in rgb_planes:
    dilated_img = cv2.dilate(plane, np.ones((7,7), np.uint8))
    bg_img = cv2.medianBlur(dilated_img, 21)
    diff_img = 255 - cv2.absdiff(plane, bg_img)
    norm_img = cv2.normalize(diff_img,None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8UC1)
    result_planes.append(diff_img)

result = cv2.merge(result_planes)
```

Árnyék nélküli kép

Az elkészült képet már szürkeárnyalatossá lehet alakítani a további feldolgozás érdekében.

```
arnyek_gray = cv2.cvtColor(result, cv2.COLOR_BGR2GRAY)
```

Szürkeárnyalatossá alakítva

6. Pontok keresése a képen

A kockák oldalán lévő pontok megkereséséhez talán az egyik legkézenfekvőbb megoldás az opencv függvénykönyvtár által biztosított SimpleBlobDetector használata. Blob alatt olyan pontok halmazát értjük, amik a környezettől eltérő azonos tulajdonsággal rendelkeznek és ezzel létrehoznak egy kerek, pontszerű formát.

A helyes érzékelés érdekében, használat előtt a Detector paramétereit célszerű definiálni, hogy csak azokat az elemeket vegye pontnak, amit ténylegesen is az.

Az alábbi kép paraméterezi a BlobDetector-t és a megadott paraméterekkel meghívja annak működését.

```
#pont kereso parametereinek megadasa
params = cv2.SimpleBlobDetector_Params()

#treshold beallitasok
params.minThreshold = 127
params.maxThreshold = 255

# teruleti szuro - minel kisebb meretet ne vegye pontnak
params.filterByArea = True
params.minArea = 150

# kereksegi szuro - minimalis kerekseg, ami pontnak szamit
params.filterByCircularity = True
params.minCircularity = 0.65

# forma kereksegi szuro - mekkora resz hianyozhat a korbol, amit pontnak vesz
params.filterByConvexity = True
params.minConvexity = 0.57

# elipszis forma szuro - mennyire lehet elipszis formaja a kornek
params.filterByInertia = True
params.minInertiaRatio = 0.2

# erzekelo letrehozasa a megadott parameterekkel
detector = cv2.SimpleBlobDetector_create(params)
```

A meghívást követően a szürkeárnyalatos képen a Detector elvégzi a keresést.

```
keypoints = detector.detect(arnyek_gray)
```

Az elemzés eredményét aztán egy tetszőleges színnel jelölve visszarajzolom az elemzett képre.

```
im_with_keypoints = cv2.drawKeypoints(arnyek_gray, keypoints, np.array([]), (0,0,255), cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
```

A kép a berajzolt pontokkal

Ezek után már csak annyi a feladat, hogy az érzékelt pontok értékét leolvassa és azt eltárolja az alkalmazás egy szöveges változóba, amit majd a feladat végén ráír az eredmény képre.

```
olvasas_int = len(keypoints)
olvasas_str = "pont:"+str(olvasas_int)
```

7. Kockák keresése a képen

Lévén, hogy a dobókockák háttérszíne fehér és még szürkeárnyalatos kép esetében is nehéz megkülönböztetni azok éleit a háttérzajtól, így közvetlen éldetektálással nem célszerű próbálkozni, mert meglehetősen csekély rá az esély, hogy találunk olyan beállítást, ahol a zaj kiszűrésre kerül, de az élek felismerhetőek maradnak a képen.

Emiatt célszerűbbnek vélem az észlelt pontokat objektumba foglalni és azok számát visszaadni, mint értéket.

Ennek első lépésében az imént kapott, a pontokkal megjelölt képet újfent szürkeárnyalatosra konvertáltam, majd a zavarok eltüntetése céljából az opencv blur (elmosás) metódusát hívtam meg.

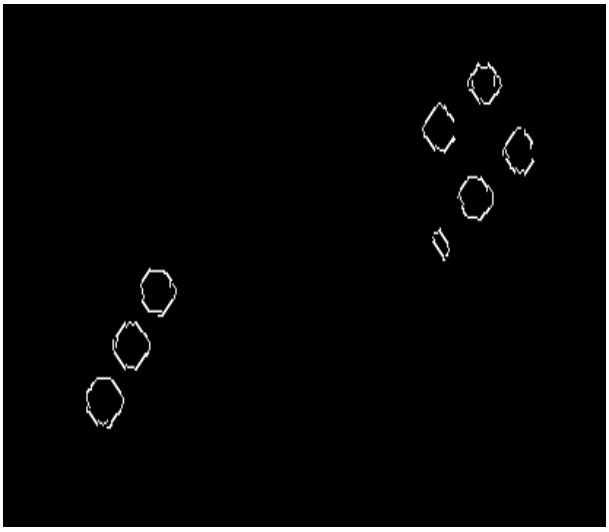
```
keypoints_ff = cv2.cvtColor(im_with_keypoints, cv2.COLOR_BGR2GRAY)
```

```
zavar = cv2.blur(keypoints_ff,(7,7))
```

Az elmosódott kép

A kapott képen egy viszonylag magas kezdőparaméterrel beállított Canny éldetektálást hajtok végre, aminek eredményeképpen már csak a pontok maradnak a feldolgozás alatt álló képen.

```
hatarok = cv2.Canny(zavar,200,255)
```



Az észlelt élek részletei

Feltűnhet, hogy ez az éldetektálási módszer elképzelhető, hogy felismeri, más a kocka oldalán lévő pontok éleit is. De mivel most az alakzatokat fogjuk egybevonni és azokat megszámolni, így annak nincs számottevő jelentősége.

A következő lépésben az éleket megvastagítom (dilation) annyira, hogy azok összeérjenek és egy objektumot formáljanak.

```
kernal = np.ones((2, 2), np.uint8)
dilation = cv2.dilate(hatarok, kernal, iterations=41)

contours, hierarchy = cv2.findContours(
    dilation, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

A létrehozott alakzatok

A kapott alakzatokat megszámolása a következő módon történik.

```
objects = str(len(contours))
```

8. Eredmény megjelenítése

Mivel a program már eltárolta, mind az objektumok számát, mind a dobások értékét, így most már csupán annyi tennivaló maradt, hogy a két szöveges formátumban eltárolt szám az eredeti képre kerüljön.
Ehhez elengedhetetlen definiálni a betűtípust, annak méretét, illetve az adott sorok pozícióját.

```
font                   = cv2.FONT_HERSHEY_SIMPLEX
bottomLeftCornerOfText = (10,500)
bottomLeftCornerOfTXT  = (10,450)
fontScale              = 1
fontColor              = (0,255,0)
thickness              = 2
lineType               = 2
```

Ezen beállítások alapján a szöveg elhelyezése a képre:

```
#szam kepre irasa
cv2.putText(eredeti,olvasas_str,
    bottomLeftCornerOfText,
    font,
    fontScale,
    fontColor,
    thickness,
    lineType)

#kockaszam kepre irasa
cv2.putText(eredeti,text,
    bottomLeftCornerOfTXT,
    font,
    fontScale,
    fontColor,
    thickness,
    lineType)
```

Az elkészült kép megjelenítése:

```
cv2.imshow("Eredmeny", eredeti)
```

2 kocka esetén

1 kocka esetén

3 kocka esetén

Végezetül a program várakozik az ablak bezárásával egy tetszőleges billentyű lenyomásáig.

```
#varkozas billentyu lenyomasra
cv2.waitKey(0)
#ablak bezarasa
cv2.destroyAllWindows()
```

9. Fejlesztés lehetőségek

Mint minden program esetében, úgy ennél is lehetőség nyílik annak további fejlesztésére. Véleményem szerint az általam készített kockadobás számláló kódsornál két fejlesztési opciót lehetne megemlíteni.

1. Lehetőség nyíljon egy adott mappába csak behelyezni egy vagy több fényképet, amit aztán a program feldolgoz és elemez.

2. Az elemzések végén az elkészült kép kerüljön mentésre egy adott kimeneti mappába.

10. Felhasznált irodalom

- Gépi látás – Hollósi János, Széchenyi István Egyetem

- OpenCV hivatalos weboldala - https://opencv.org/

- Satya Mallick OpenCV blog - https://learnopencv.com/