



SZÉCHENYI
ISTVÁN
EGYETEM

Gépi látás

GKLB_INTM038

Dobókocka számláló alkalmazás

Antal Krisztián – KYCNDX – 2021/2022 1. félév

Tartalomjegyzék

1. Téma bemutatása	3
2. Felhasznált eszközök ismertetése	3
3. Elméleti háttér ismertetése	3
4. Képek bemutatása, azok elemzése	5
5. Képfeldolgozás megkezdése	7
6. Pontok keresése a képen	8
7. Kockák keresése a képen	10
8. Eredmény megjelenítése	12
9. Tesztelés	15
10. A program korlátai	20
11. Fejlesztési lehetőségek	23
12. Felhasznált irodalom	24

1. Téma bemutatása:

Féléves feladatomban egy kockadobások értékét számláló alkalmazást készítetek python programnyelven. Az elkészített programnak képesnek kell lennie megszámolni az adott kockákkal dobott értéket, valamint azt is megállapítani, hogy azt az értéket hányszámos 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

Az alapfeladatom készítése során a kockákról készített fényképek háttérül egy tölgy mintás lapú íróasztal szolgált. A szükséges fényt 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 későbbi tesztelési fázisban, mind a háttéröt, mind a fényforrás módosítottam, de ezekhez is olyan eszközöket használtam fel, ami többnyire minden háztartásban megtalálható. Pl: fekete ülögarnitúra, fehér lap, egérpad, szőnyeg, társasjáték fedlapja.

A forráskód létrehozásához és fordításához a Thonny nevű alkalmazást használtam fel.

3. Elméleti háttér ismertetése

A feladatban ismertetett problémák megoldásához több a képfeldolgozás kapcsán használatos elméleti háttértudással is tisztában kell lenni.

Egyrészt geometria transzformációt hajtok végre a képen, amikor kicsinyítem azt. Ezzel annak értelmezési tartománya módosulni fog, ahogy az eredeti felbontását változtatom egy általam választottra, amivel a továbbiakban a program dolgozni fog.

Továbbá az árnyékok elhalványításánál egymás után alkalmazok dilatációt és medián szűrőt is. A dilatáció egy morfológiai művelet. Ezekre jellemző, hogy egy „csúsztatott” strukturáló elemmel hasonlítjuk össze a képeket. Dilatáció esetén a strukturáló elem egybeesik a vizsgált kép adott helyén vett képpontjával. Ilyenkor a képpontok kiegészítésre kerülnek a strukturáló elem képpontjaival. A medián szűrő a zajszűrés egyik formája, ahol a vizsgált méretű képrészlet alapján képzett számsorozat mediánja adja az új intenzitási értéket. Ennek azért lesz jelentősége, mert a dilatációs „feltöltés” és a medián szűrést követően az

árnyékok már elhalványulnak annyira, hogy azok már ne olvadjanak egybe kockákkal a szürkeárnyalatos konverziót követően.

Ha a képen már halványak az árnyékok, a feladat szempontjából már jóval eredményesebben hajtható végre a szürkeárnyalatos konverzió. Az értékkészlet transzformáció ezen módjára azért van szükség, mert egy szürkeárnyalatos képen könnyebben lehet az alakzatokat detektálni, mint egy színes képen.

A dobókockák oldalán található pontok érzékeléséhez és megszámlálásához az opencv függvénykönyvtár által biztosított SimpleBlobDetectort használtam fel. Ez az algoritmus a pontok, foltok képről való kinyerésére szolgál. Maga az algoritmus a következő lépéseket használja fel:

1. A forrás képet bináris képekké alakítja a paraméterben megadott küszöbértékek felhasználásával.
2. A létrejött bináris képekből kinyeri az összeköttetésben álló komponenseket és a körvonalaiat alapul véve kiszámítja a középpontjukat.
3. Csoportosítja a középpontjait több bináris képnek a koordinátáik alapján. Az egymáshoz közelí középpontokat egy csoportba foglalja, amit aztán egy foltnak fog kezelní.
4. A csoportokból kiszámítja a végső középpontját és sugarát a foltoknak, amit visszaad, mint hely és méret értéket a megtalált, úgynevezett kulcspontrnak.

Ez a kereső algoritmus jól paraméterezőhető szín, terület, körkörösségek, konvexitás és ellipszis forma hasonlóság alapján.

A kockák, mint objektumok megtalálásához feltétlen szükséges éldetektálást végrehajtani. Ehhez azonban tisztában kell lenni az éldetektálással kapcsolatos néhány fontos részlettel. Egy képen az él ott található, ahol az intenzitási érték nagy mértékben, hirtelen megváltozik. Az éldetektálás célja egy, vagy több objektum határvonalainak felismerése. Az élek meghatározását azonban nehezíti több tényező is: árnyék, túl erős vagy rossz szögből érkező fény, alakzatok az objektumon belül, valamint a törésvonal az objektum felületén.

A konverziók közben hamar szembetűnhet, hogy pont a nehezítő körülményeknek és a háttér változékonyságának köszönhetően a kockák éleit elég nehézkes lenne megkülönböztetni a háttér és az árnyák által vetett zajtól. Emiatt célravezetőbb az éldetektálást már a BlobDetector által megjelölt dobókocka pontokon végrehajtani, majd azokat, mint objektumokat kezelní.

Az éldetektálást megelőzően a pontosabb és a későbbiekben jobban felhasználható eredmény érdekében zajszűrést alkalmazok. A zajszűrés egy konvolúciós eljárás, ahol az adott képpont értéke megváltozik a szomszédos pixelek és egy úgynevezett szűrő (kernel) lineáris kombinációja által. Ezzel valójában a magas frekvenciájú tartalom kerül szűrésre, ami sok esetben a zaj mellett éleket is érinti.

A megfelelően zajcsökkentett képen már eredményesebben lehet az éldetektálást végrehajtani. Erre a feladatra a Canny éldetektálást használtam, ami talán az egyik legelterjedtebb éldetektálási eljárás. Ez a keresési algoritmus több lépésből áll:

- szürkeárnyalatos konverzió
- Gauss simítás
- differencia számítás
- nem-maximum vágás
- kettős küszöbölés
- hiszterézis küszöbölés

4. Képek bemutatása, azok elemzése

Az alapfeladatom 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 a 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(norm_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 tulajdonságúval 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 mi ténylegesen is annak szeretnénk érzékelni.

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

```
#pont kereso paramtereinek megadása
params = cv2.SimpleBlobDetector_Params()

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

# terulet 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 hiányozhat 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 érzékelő 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 lépései az imént kapott, a pontokkal megjelölt képet újfent szürkeárnyalatosra konvertáltam, majd a zavarok eltüntetése céljából az opencv blur (elmosás) metódusát hívtem meg.

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

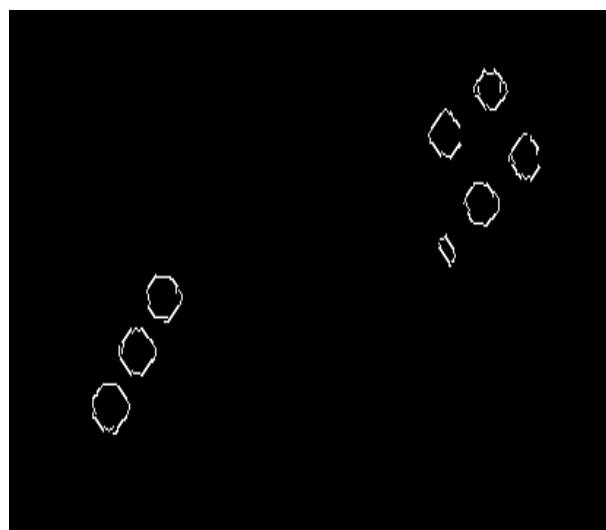
```
| zavar = cv2.blur(keypoints_ff,(7,7))
```



Az elmosódott kép

A kapott képen egy viszonylag magas kezdőparaméterrel beállított Canny éldetektálást hajtak 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 elköpzelhető, 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 betűtípuszt, annak méretét, illetve az adott sorok pozícióját.

```
font          = cv2.FONT_HERSHEY_SIMPLEX
bottomLeftCornerOfText = (10,50)
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)
```



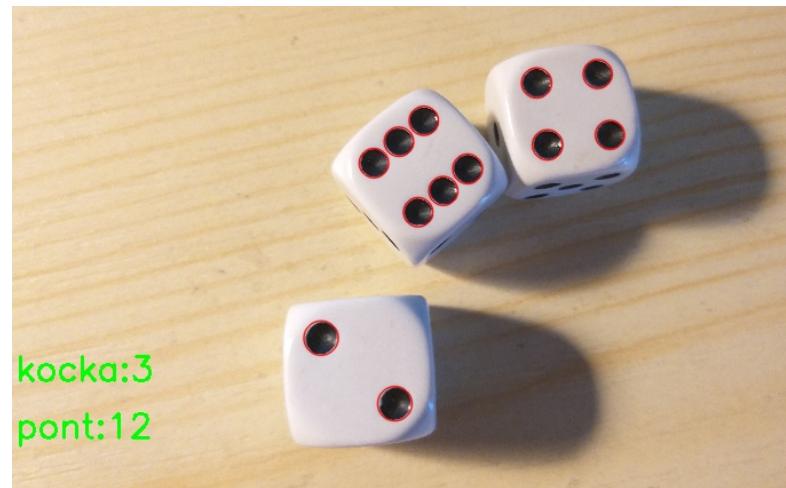
2 kocka esetén



kocka:1

pont:5

1 kocka esetén



kocka:3

pont:12

3 kocka esetén

Ez elkészült képet a program a „kepek” mappán belül az „eredmeny” alkönyvtárba menti egy megadott néven.

```
#eredmeny fajlba mentese  
cv2.imwrite("./kepek/eredmeny/eredmeny_20220111_101152.jpg", eredeti)
```

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

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

9. Tesztelés

Mivel a program a kitűzött célt megvalósítja és az alapfeladatban megjelölt környezetben az elvártak szerint működik, így mindenképpen érdemes tesztelni az ideálistól eltérő körülmények között. Három körülményt változtattam a tesztelési fázik folyamán:

- fényviszony
- háttér
- kamera távolsága

A továbbiakban néhány eredmény képet mutatnék be, ahol az imént megnevezett tényezők módosultak.



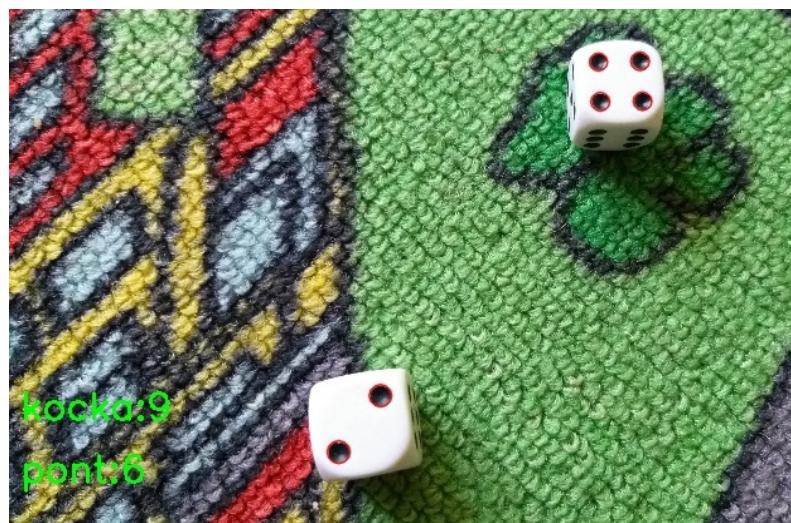
Fehér háttér előtt



kocka:3

pont:14

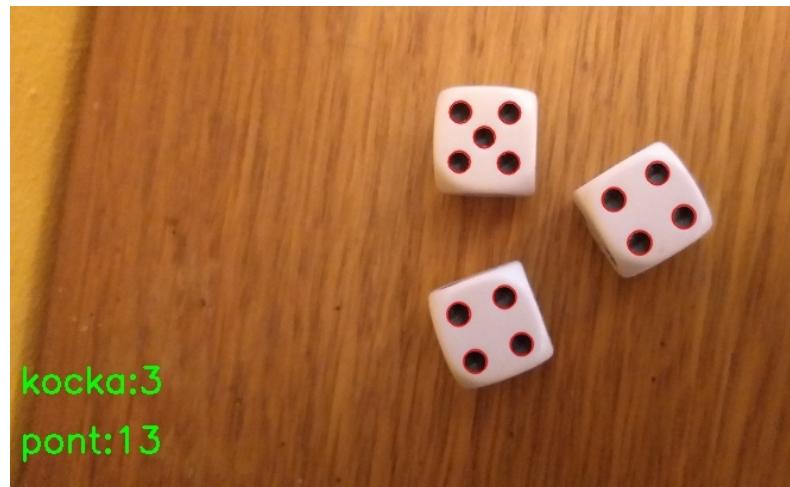
Erős oldalfény



kocka:9

pont:6

Színes, változatos mintázatú szőnyeg



kocka:3
pont:13

Erősebb, sötétebb fa mintázatú háttér

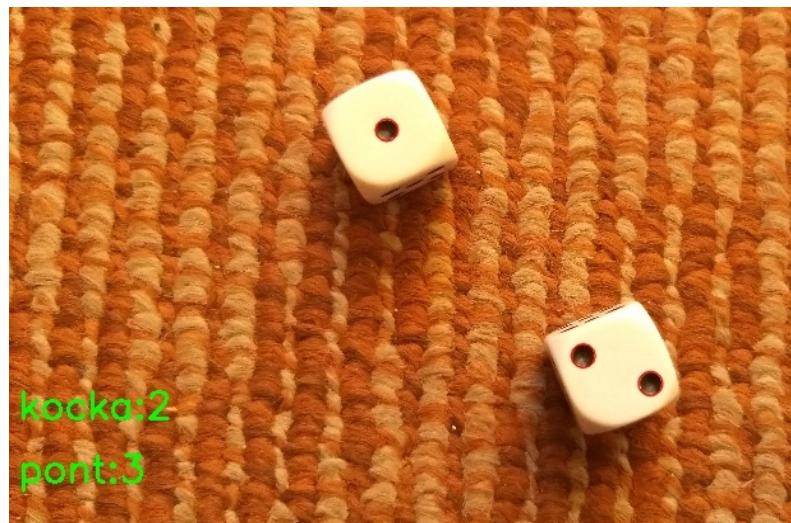


kocka:3
pont:7

Fekete műbőr háttérrel



Színes minta, de a kocka egyszínű háttérrel



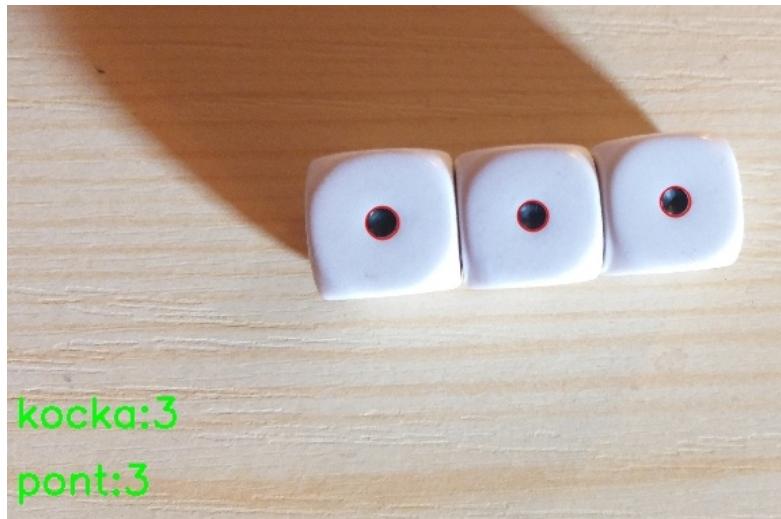
Ismétlődő mintázatú kárpit



kocka:3

pont:9

Két oldalról érkező fény



kocka:3

pont:3

Egy oldalról érkező erős fény, kockák egymás mellett

10. A program korlátai

A tesztelés során kiderült, hogy bizonyos kritériumok módosításával a program pontossága kisebb vagy nagyobb mértékben romlik. A következőkben felsorolok pár esetet, amikor a kód nem pontos eredményt ad vissza eredményül. Ezekhez egyesével magyarázatot is fűzök, hogy mi az eltérés valószínűsíthető oka.



Jól látható, hogy a fekete, rajzos mintájú szönyeg esetén ugyan a megfelelő darabszámú pontot észleli a program, de jóval több objektumot ismer fel, mint ami ténylegesen látható a képen. Ennek oka valószínűleg abban rejlik, hogy vastag vonalak a zajszűrés során nem tűnnek el és az éldetectálást követő dilatáció már megvastagítja azok kontúrját annyira, hogy aztán egész, különálló objektumként kerülnek megszámolásra.



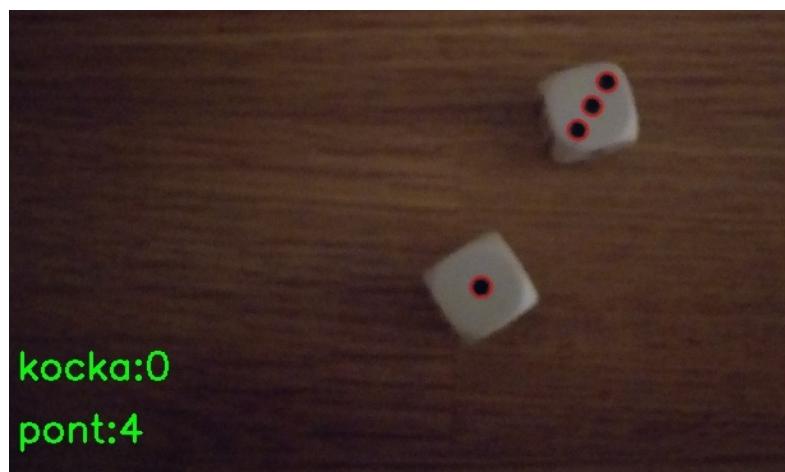
Az erős, a dobókockákra megközelítőleg 45° -os szögben érkező fény esetében a program a dobókockák számát ugyan jól érzékeli, de a lakkozott felületű fekete mélyedésekből (ami maga a pont a kockán) fény annyira visszaverődik, hogy megtöri a fekete folt körkörösségét, amit a BlobDetector észlelne.



Ezen a képen ugyan a háttér mesés, de maga az eredmény korántsem az. A dobókockán szereplő pontok felismerésre kerülnek, de jól láthatóan más olyan fekete foltot is annak számít a kód, amit eltérő (jelen esetben fehér) színű szegély határol. Ezek mellett a különböző területen elhelyezkedő fekete alakzatok a rajzos szőnyeghez hasonlóan a zajszűrés során nem teljes mértékben kerülnek szűrés, így az éldetektálást követően alakzatként kerülnek értelmezésre.



A maximális értékükkel közvetlen egymás mellék helyezett és erős 45°-os szögben megvilágított dobókockák esetén sem kapunk pontos eredményt. Több pont nem kerül észlelésre a fénynek köszönhető körkörösség megszakadása miatt, valamint részben az erős fénynek, részben a pontok közelégeinek köszönhetően egy objektumként kerül észlelésre a három kocka.



De a fényforrás hiánya is pontatlan észlelést eredményez. Ilyenkor a BlobDetector ugyan felismeri és megszámolja a valamennyire még kirajzolódó kockán szereplő foltokat, de a zajszűrés során azok annyira elhalványulnak, hogy mint objektumokat már nem tudja a program megszámolni.



Megállapítható még, hogy túlzott fókuszálás esetén is pontatlan az észlelés. Ebben az esetben a dobókockák darabszáma pontosan kerül észlelésre, viszont a pontok (főleg a kamerához közelebbiek) már több esetben kívül esnek a BlobDetector paraméter beállításain.

Összegezve az állapítható meg, hogy a fény erőssége és beesési szöge, valamint a háttérben szerepelő erőteljes fekete alakzatok, sok esetben befolyásolják a program működésének hatásfokát.

A pöttyök felismerésében és megszámlálásában a fényerő játszik erősebb szerepet, míg a kockák darabszámának észlelésében a háttér.

Megállapítható továbbá, hogy homogén, vagy a feketétől eltérő mintázatú háttér esetén a kódsor jól detektálja a dobókockák számát, valamint normál fényerő mellett a kockákon szereplő értékek is nagyrészt pontosan észlelhetőek.

11. Fejlesztési lehetőségek

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

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 neve automatikusan módosuljon a bemenetnek megadott név alapján, egy előre megadott előtagot illesztve csak a fájlnév elé.

12. Felhasznált irodalom

- Hollósi János - Gépi látás (GKLB_INTM038) levelező elméleti diasorozat
https://drive.google.com/drive/folders/10HUJswKEpn2K5rADj4_wDx68u5TUCo13
- Gary Bradski - Dr. Dobbs Journal, 2000 - The OpenCV Library
<https://www.drdobbs.com/open-source/the-opencv-library/184404319>
- Kari Pulli, Anatoly Baksheev, Kirill Konyakov, Victor Eruhimov - Communications of the ACM, June 2012 - Real-Time Computer Vision with OpenCV -
<https://cacm.acm.org/magazines/2012/6/149789-real-time-computer-vision-with-opencv/fulltext>
- The OpenCV team - OpenCV hivatalos weboldala - <https://opencv.org/>
- Satya Mallick – Learning OpenCV course - <https://learnopencv.com/>