Gépilátás beadandó

2019/2020/2

Feladat: Rubik kocka állapotának felismerése

Készítette: Gvárdián Levente

Neptunkód:U7A44W

Bevezetés

Rubik kocka felismerő alkalmazást választottam féléves feladatnak. Ez az alkalmazás 6 fényképet olvas be, amin a rubik kocka minden egyes oldala külön-külön található meg, és ezután átkonvertálja a képet HSV színtérbe, ez alapján készít maszkot, és minden maszkra külön kontúrkereséssel, majd megrajzolással készít az eredetileg beadott képre kontúrokat, azok köré virtuálisan egy kör alakú kontúrt képez, és ennek a körnek a középpontját olvassa be. Ekkor egy koordinátát kapunk, illetve egyből a színt is elmenti egy kétdimenziós tömbbe, majd ezután ezt az x,y,szín koordinátát sorbarendezzük y majd x szerint, és így megkapjuk a kockákat sorrendben balról jobbra haladva. Ezt már csak kiíratással a végén kiírattam a konzolba.

Elméleti háttér

Elméleti háttere a feladatomnak, hogy először is ugye a képeket az opencv segítségével beolvastatom:

cv2.imread(“kép.jpg”)

A beolvasott kép formátumát átalakítom szintén az opencv segítségével:

cv2.resize(kep,(1920,1080))

Mivel a képet RGB, színrendszerben kapom meg(Ez a programban BGR),ezért az egyeztetett első konzultáció alapján, másik színrendszert kerestem, és kis internetes kutatás után HSV színrendszert találtam a legmegfelelőbbnek.

hsv = cv2.cvtColor(kep,cv2.COLOR_BGR2HSV)

Matematikai háttere:

Szükséges először is, hogy az értékeket elosztjuk 255-vel minden színcsatornánál (piros=R’, zöld=G’, kék=B’), majd ezeket eltároljuk. Szükségünk lesz ezeknek az új értékeknek a maximumára(Cmax) és a minimumára(Cmin), illetve egy különbségre: $\Delta = Cmax - Cmin$.

Hue kiszámolása:

$$H = \begin{cases} 0^\circ & \Delta = 0 \\ 60^\circ \times \left(\frac{G'-B'}{\Delta} mod 6\right) & , C_{max} = R' \\ 60^\circ \times \left(\frac{B'-R'}{\Delta} + 2\right) & , C_{max} = G' \\ 60^\circ \times \left(\frac{R'-G'}{\Delta} + 4\right) & , C_{max} = B' \end{cases}$$

Szaturáció kiszámolása:

$$S = \begin{cases} 0 & , C_{max} = 0 \\ \frac{\Delta}{C_{max}} & , C_{max} \neq 0 \end{cases}$$

Value, azaz érték kiszámítása:

$$V = Cmax$$

Ez sikerült, és átkonvertálta a képet a program a számunkra megfelelő színrendszerbe, ekkor következett a maszkolás problémája. Hiszen szükséges tudni

a kívánt szín maszkolásához a szín HSV-ben megadott határértékeit, amit lower, illetve upper előtaggal adtam meg. Ehhez egy kis interneten talált program általi segítséget kellett alkalmaznom, hogy ezt a lower,és upper határokat megtaláljam minden színhez HSV szerint.(A dokumentáció legvégén írom le, hogy hol található ez a program)

```
lowup =np.array ([[42,55,60],
                  [93, 255, 255],
                  [100,151,200],
                  [106,255,255],
                  [171,130,0],
                  [255,255,255],
                  [21,121,144],
                  [255,255,196],
                  [0,86,169],
                  [63,255,255],
                  [62,0,130],
                  [255,255,255]])
```

-Ez két soronként értelmezendő. Két soronként változik, hogy melyik színt nézi, például az első két sor az a zöld szín lower és upper értékei, a következő két sor a kék lower és upper értékei, stb. stb. egészen a fehér színnel bezárólag.

Matemaikai háttere ennek:

Vesszük a [H-10,100,100] lower értéknek, illetve [H+10,255,255] upper értéknek

Maszkolás pedig színenként történik ezzel a kódsorral:

mask = cv2.inRange(hsv,lowup[i],lowup[i+1])

Ezt követően jött a következő probléma, hogy hogyan határozzam meg hol van melyik kocka, illetve milyen színű az a kocka. Ehhez ismételten a konzultáció alapján használtam a kontúrozást.

contours, _ = cv2.findContours(mask, cv2.RETR_LIST, cv2.CHAIN_APPROX_NONE)

A koordinátákat pedig úgy kapom meg, hogy mivel a kontúrrajzolás az abban merül ki, hogy a képen látható 9 kockára a maszk alapján rajzolja a négyzet kontúrokat.

x, y, w, h = cv2.boundingRect(cnt)

cv2.rectangle(kep, (x, y), (x + w, y + h), (0, 255, 0), 2)

Eköré egy függvény segítségével a lehető legkisebb területű kört “rajzoltattam” virtuálisan:

circles = [cv2.minEnclosingCircle(cnt)]

Ennek a körnek megnézem a középpontját, és az a középpont visszaadott értéke lesz az x és y koordináta. Ehhez tömböt használtam, ami egy 9x3-mas mátrix, hiszen mind a 9 kockának van x,y koordinátája, és egy színe, amit szintén eltároltam az alapján, hogy a maszkolás for ciklusa hányszor futott le, hiszen ahányszor lefutott már a for ciklus, annyiadik szín kerül maszkolásra, ami alapján a kontúrt rajzolja a program.

```
141                 M2 = cv2.moments(cnt)
142                 #print("Momment2:",M2)
143                 center2 = (int(M2["m10"] / M2["m00"]), int(M2["m01"] / M2["m00"]))
144                 #print("circlecenter:", center2)
145                 cx= int(center2[0])
146                 cy = int(center2[1])
147                 cxcyj=np.empty(3)
148                 cxcyj[0] = cx
149                 cxcyj[1] = cy
150                 cxcyj[2] = j
151                 #print("cxcyj:", cxcyj)
152                 cszintomb = np.append(cszintomb,cxcyj)
153                 print("cszintomb:", cszintomb)
```

Matematikai háttere ennek:

$$Cx=M10/M00$$

$$Cy=M01/M00$$

Amint ez megvan, és ezeket eltudom tárolni, akkor szükségem van egy sorrendbe rendező algoritmusra, amely a tömb elemeit az x,y koordináta alapján rendezi el, úgy hogy a kockákat balról jobbra kapjuk meg.

```
    print("cszintomb a vegen:", cszintomb)
    #segedtömb segítségével csinálok ebből egy 9x3 mátrixot
    segedt = np.ndarray(shape=(9),dtype=[('x','f4'),('y','f4'),('c','f4')])
    print("segedt", segedt)
    l = 0
    for t in range(0,9):
        for u in range(0,3):
            segedt[t][u] = cszintomb[l]
            l = l+1

    print("segedt:", segedt)
    #y koordináta szerint sorbarendezem (axis=0)
    asd = sorted(segedt, key=lambda segedt_entry: (segedt_entry[1],segedt_entry[0]))
    print("sorted:", asd)
```

Ha ezt is megkaptuk, akkor már csak kiíratjuk sorrendben a színeket, és kész a program.

```
    o=1
    for z in range(9):
        for q in range(3):
            if (szin(asd[z][q])=="G"):
                print('G',end='')
            elif (szin(asd[z][q])=="B"):
                print('B',end='')
            elif (szin(asd[z][q])=="R"):
                print('R',end='')
            elif (szin(asd[z][q])=="Y"):
                print('Y',end='')
            elif (szin(asd[z][q])=="O"):
                print('O',end='')
            elif (szin(asd[z][q])=="W"):
                print('W',end='')
        if (o%3)==0:
            print("\n")
        o=o+1
```

Megvalósítás terve és kivitelezése

Megvalósítás első része a kép beolvasása. Először is ugye szükségem volt a numpy és opencv plugin segítségére. Ezeket beimportáltam a program elején. A képeket beolvastatom a program elején, majd HSV-be konvertálom a cv2.cvtColor segítségével. Ha ezzel kész van, akkor következik egy forciklus ami azt fogja jelenteni, hogy mind a hat maszkra (mivel hat szín van a rubikkockán, és színenként szeretnénk maszkolni) a cv2.inRange segítségével maskot készítünk, ami után egyből következik a kontúrokkal való foglalkozásnak a része. Itt előszöris a kontúrokat megkeresi a program, és szépen végig rajzolja, de egy if segítségével a kis területű kontúrokat ki tudjuk szűrni, tehát aminek területe (pixelben) nagyobb mint 10000, akkor megrajzolja nekünk a kontúrt. Eköré a kontúr köré készítünk egy kör alakú kontúr, ami pontosan ráilleszkedik a rajzolt kontúr négyzetre, és ennek a körnek a közepe fogja nekünk megadni a koordinátákat, tehát a kör középpontja a alapján határozom meg a kockákat. Ez azért is fontos, mivel különben nem tudnánk a végén a kiíratásnál, hogy melyik kockát szeretnénk kiíratni. Ezt követően egy kétdimenziós tömbbe tárolom el, aminek 9 sora és 3 oszlopa van, hiszen minden kockának van egy koordiánátája, amit a kör alapján határoztunk meg, illetve minden kockának van egy színe, amit egy “j” változó segítségével határozok meg. Ehhez szükséges tudni, hogy hányadik for ciklusban járunk a maszkolásnak, így a “j” változót minden maszk forciklus végén növeltem eggyel. Tehát ezalapján tudjuk, mivel a színek így vannak sorrendben: zöld, kék, piros, sárga, narancs, fehér. Hiszen ezeket én adtam meg a lowup tömbnél, aminek a célja ugye a színek kiszűrése HSV segítségével, ezt használja a maszk is. Szóval ez is eltárolódik a cszintomb-ben tehát 9x3 mátrix szükséges, hogy mind a megtalált kilenc kockánknak x,y koordinátáját, és színének értékét(ami egy int, de mivel bekerül a tömbbe ezáltal float típusú lesz a koordinátákkal együtt). Ezt követően jön a segéd tömb bevezetése, ami ahhoz kell, hogy a cszintomb-ből külön szedjem a float típusokat külön értékekké, hiszen a cszintomb nem teljesen 9x3-mas mátrix, hanem csak egy float tömb ami 1x27, hiszen az np.append segítségével adtam hozzá a dolgokat. Visszatérve, ebből egy mátrix lesz a segéd tömb, azaz a programomban segedt alapján. Viszont ez így nem maradhat, hiszen koordináták alapján szeretnénk ezt sort-olni, tehát rendezni, és ezt is úgy, hogy először az y, majd x koordináták alapján. Így megkapva mind a 9 kockát balról jobbra haladva, fentről lefelé. Ezt követően már csak annyi a feladat, hogy ezt konzolba kiírjuk, tehát megadja, hogy melyik szín, hol van. Ezt leegyszerűsítettem: G,B,R,Y,O,W betűrkre, hiszen így is tökéletesen érzékelhető, hogy a programnak mi az eredménye. Sajnos nem sikerült megoldani, hogy mind a 6 képet amit beadunk neki, azt automatikusan magától egyesével végig menjen rajtuk, de még utólag próbálkozom vele, hátha azt is sikerül, és akkor majd feltöltöm külön kódfájlként.

Tesztelés működő képekkel

Tesztelésem során megadtam a következő képeket:

1.ábra: zold.jpg, 2.ábra: kek.jpg, 3.ábra: piros.jpg, 4.ábra: sarga.jpg, 5.ábra: narancs.jpg, 6.ábra: feher.jpg

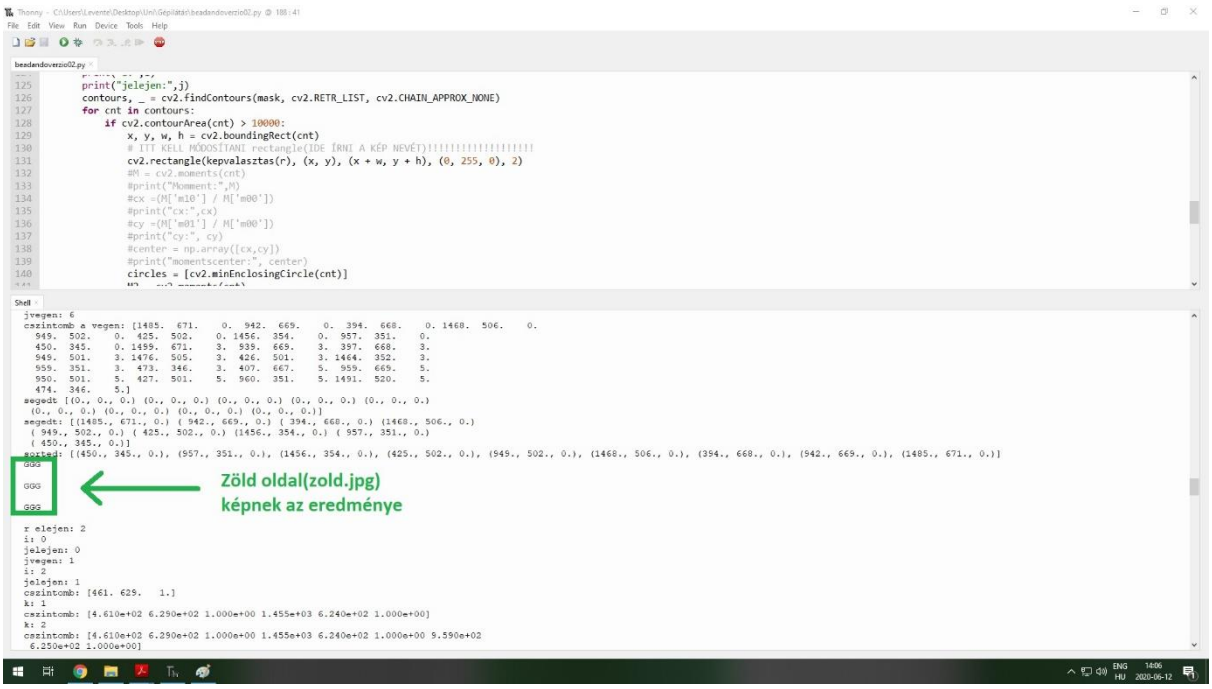
Eredménye a tesztelésnek:

Jelentése az eredményen látható betűknek:

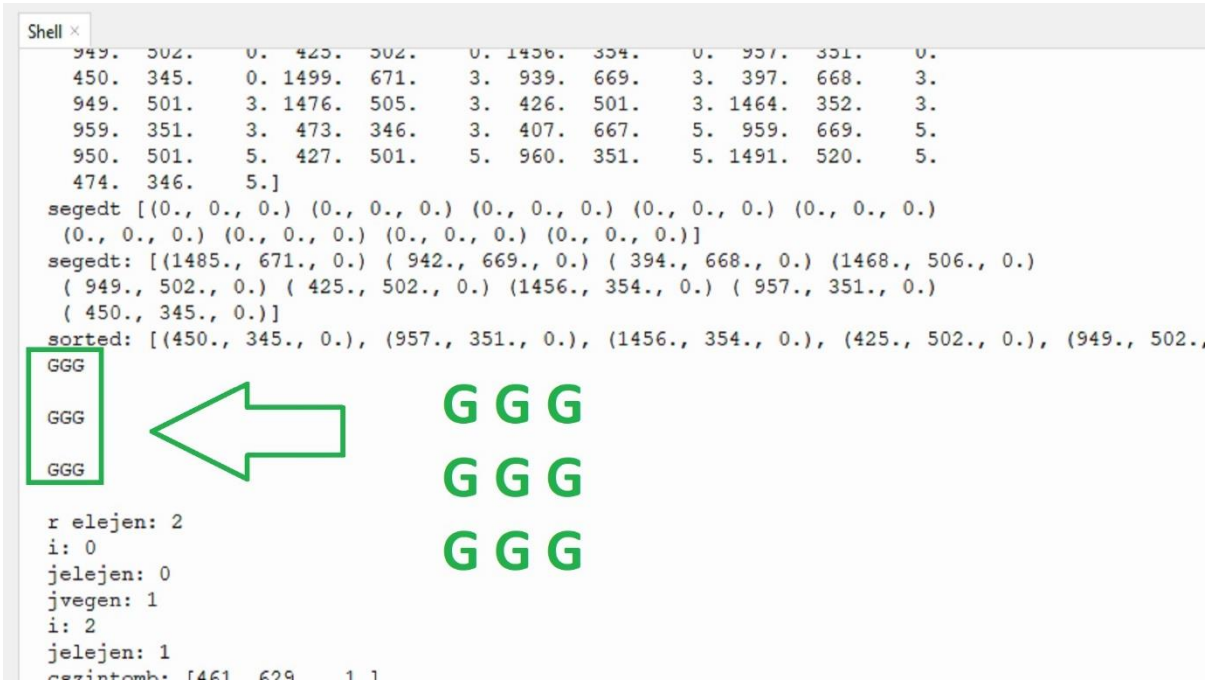
G->Green, ami magyarul a zöld színt jelenti
B->Blue, ami magyarul a kék színt jelenti
R->Red, ami magyarul a piros színt jelenti
Y->Yellow, ami magyarul a sárga színt jelenti
O->Orange, ami magyarul a narancs színt jelenti
W->White, ami magyarul a fehér színt jelenti

Fontos, hogy minden eredmény egy 3x3-mas mátrix, amiben 9db betűjelet fogunk látni, attól függően természetesen, hogy mi az eredmény.

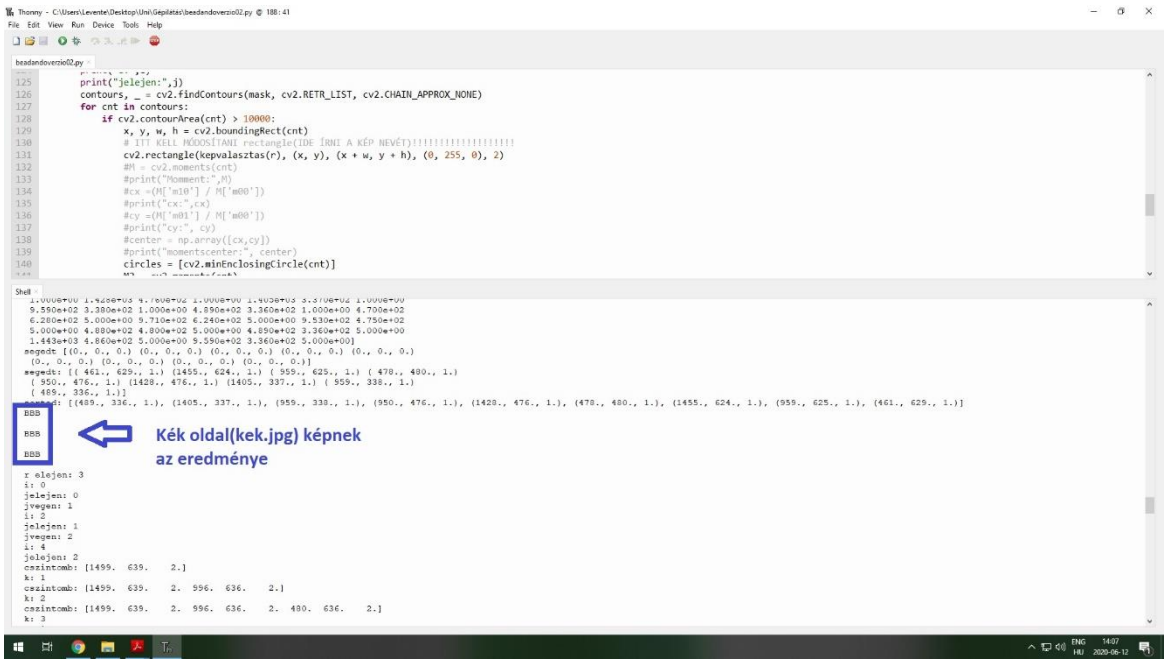
1. **Kép tesztelése: (zold.jpg)**



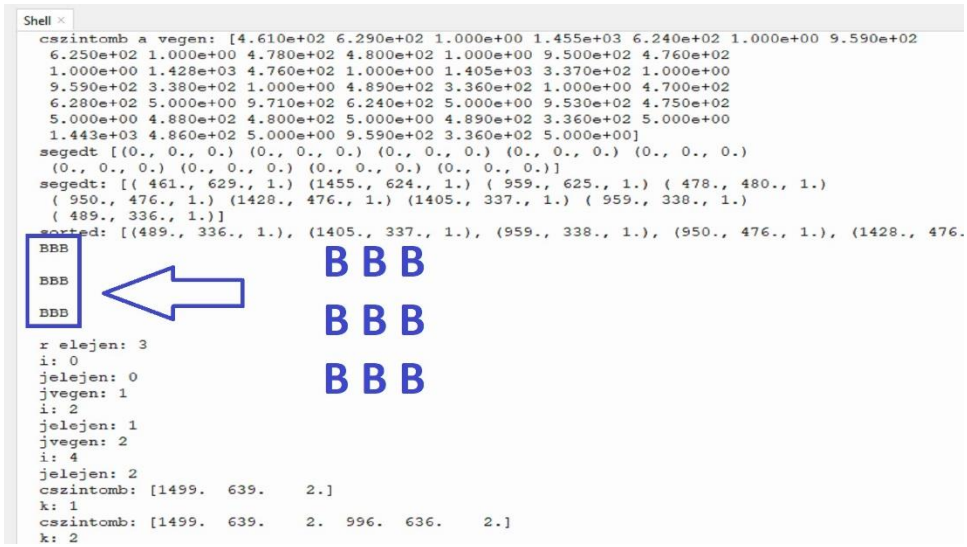
Nagyítva az eredmény:



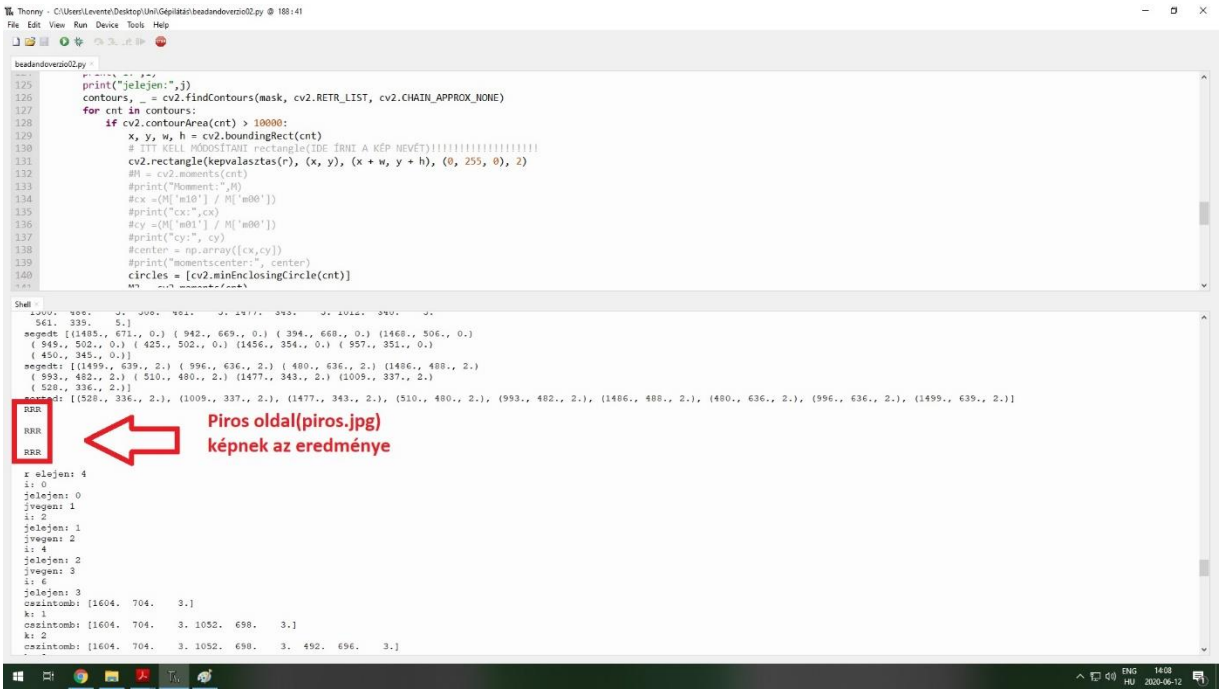
2. Kép tesztelése: (kek.jpg)



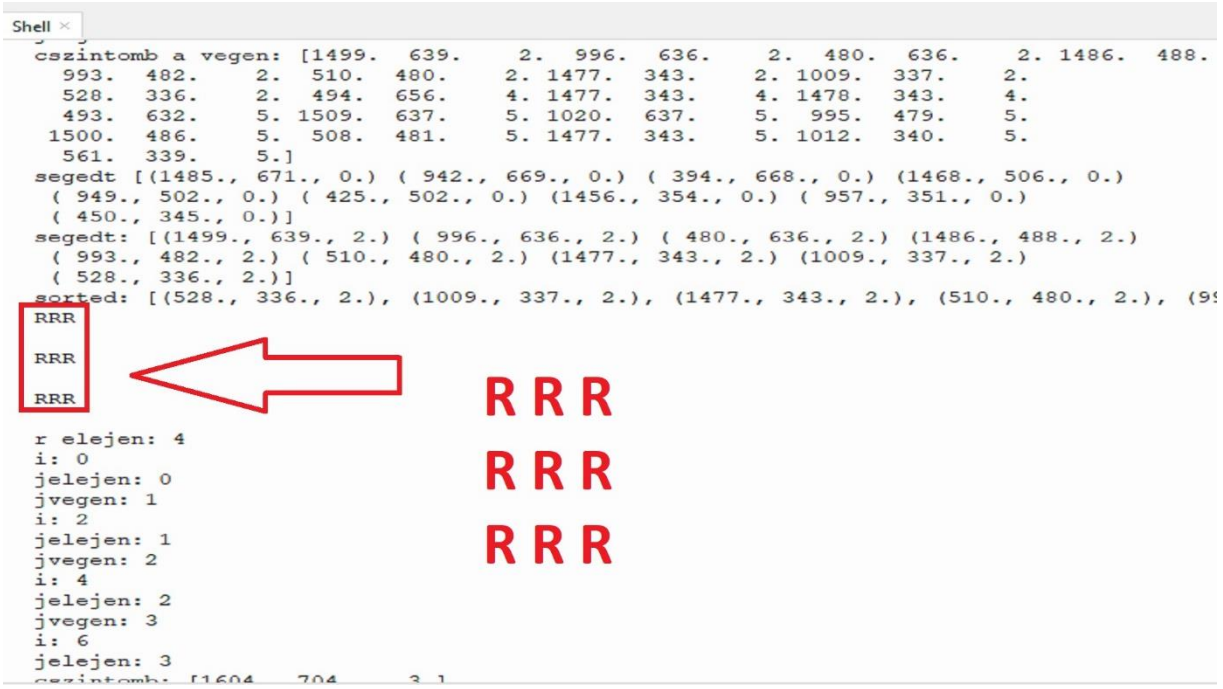
Nagyítva az eredmény:



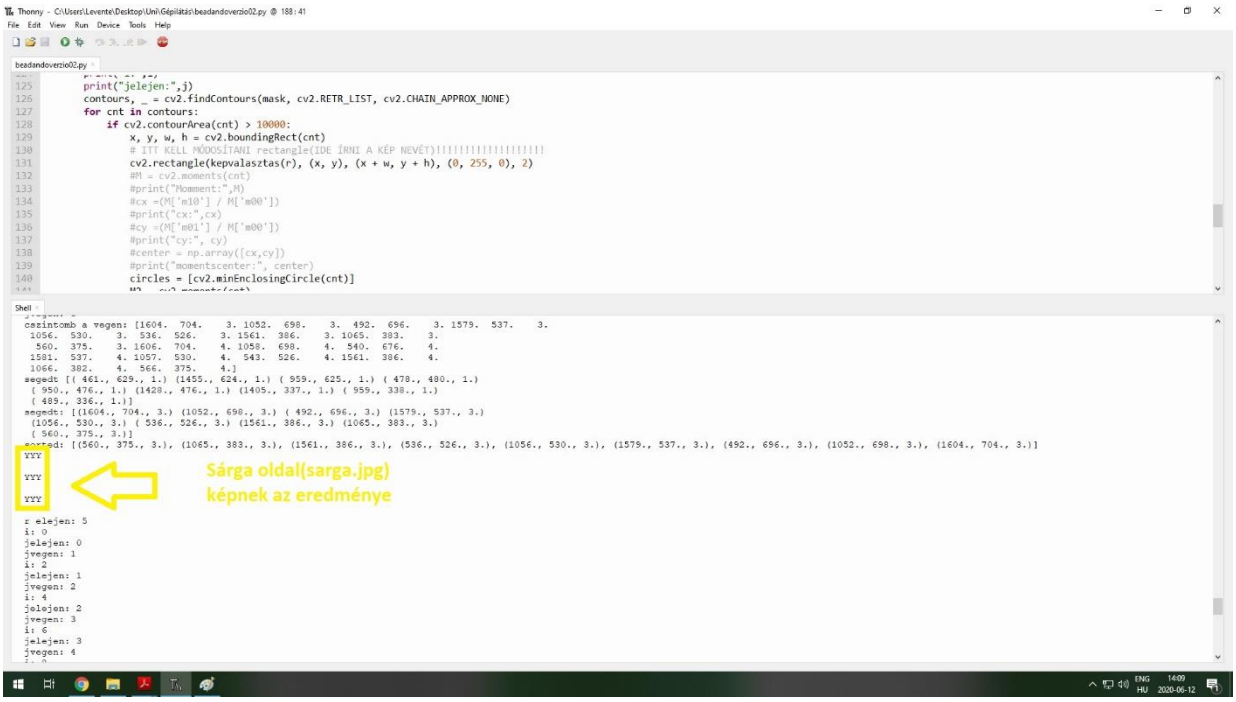
3. Kép tesztelése: (piros.jpg)



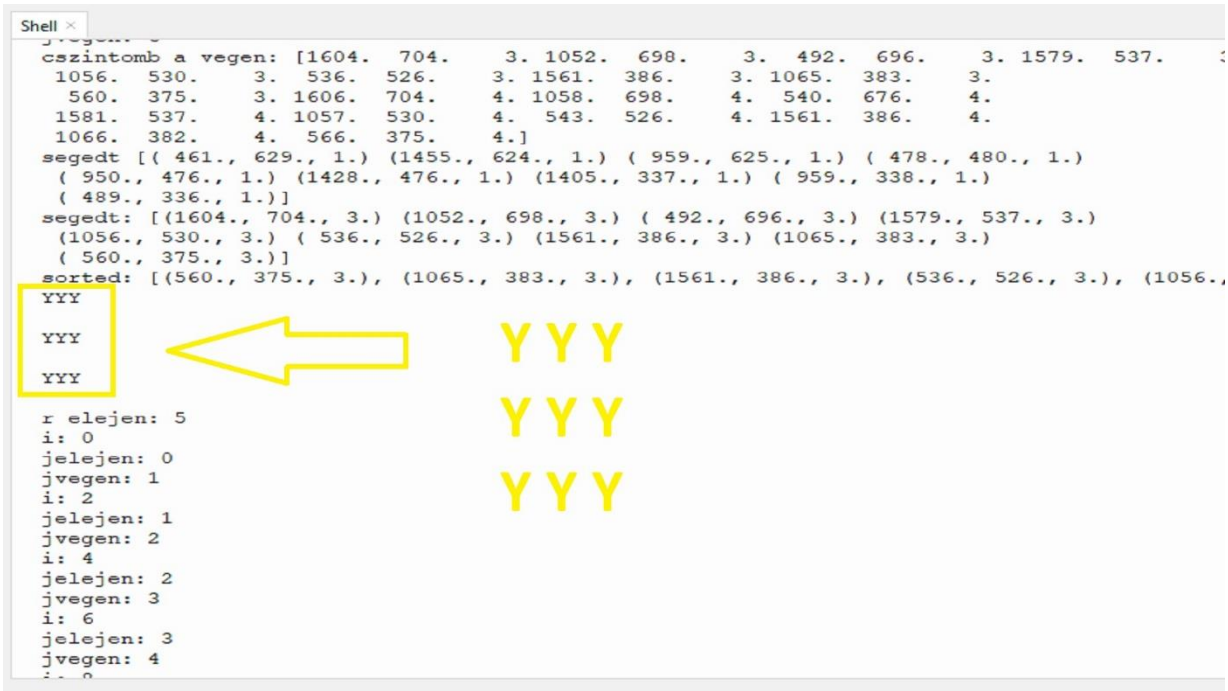
Nagyítva az eredmény:



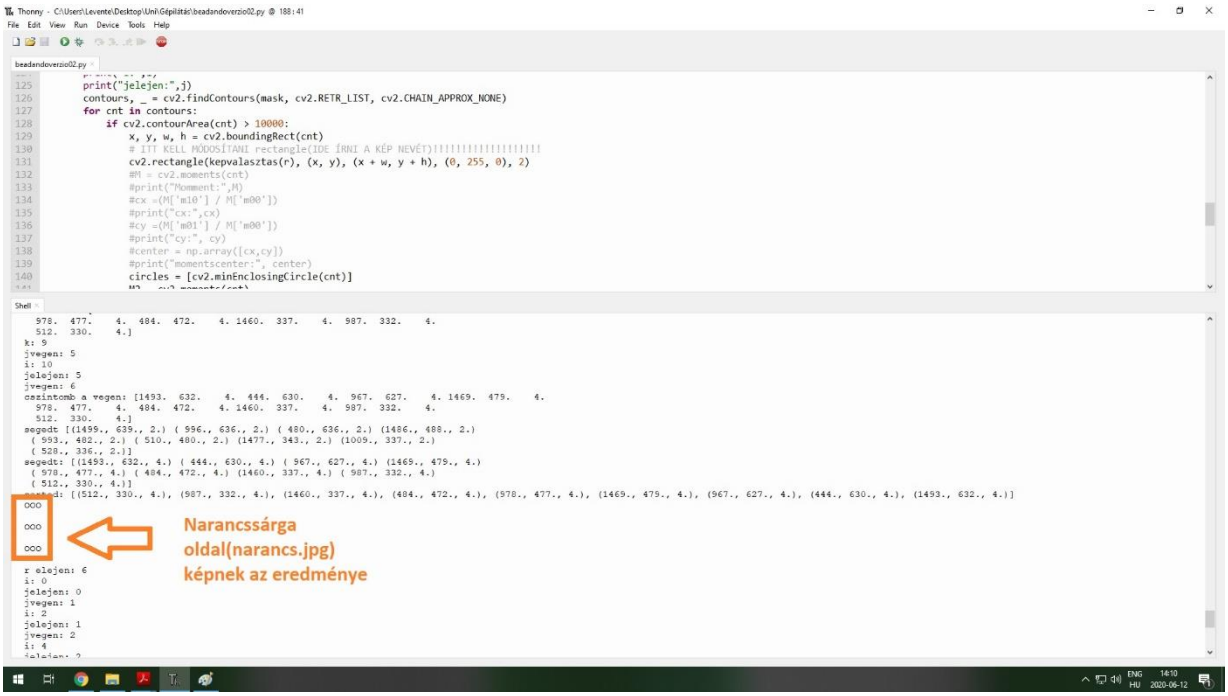
4. Kép tesztelése: (sarga.jpg)



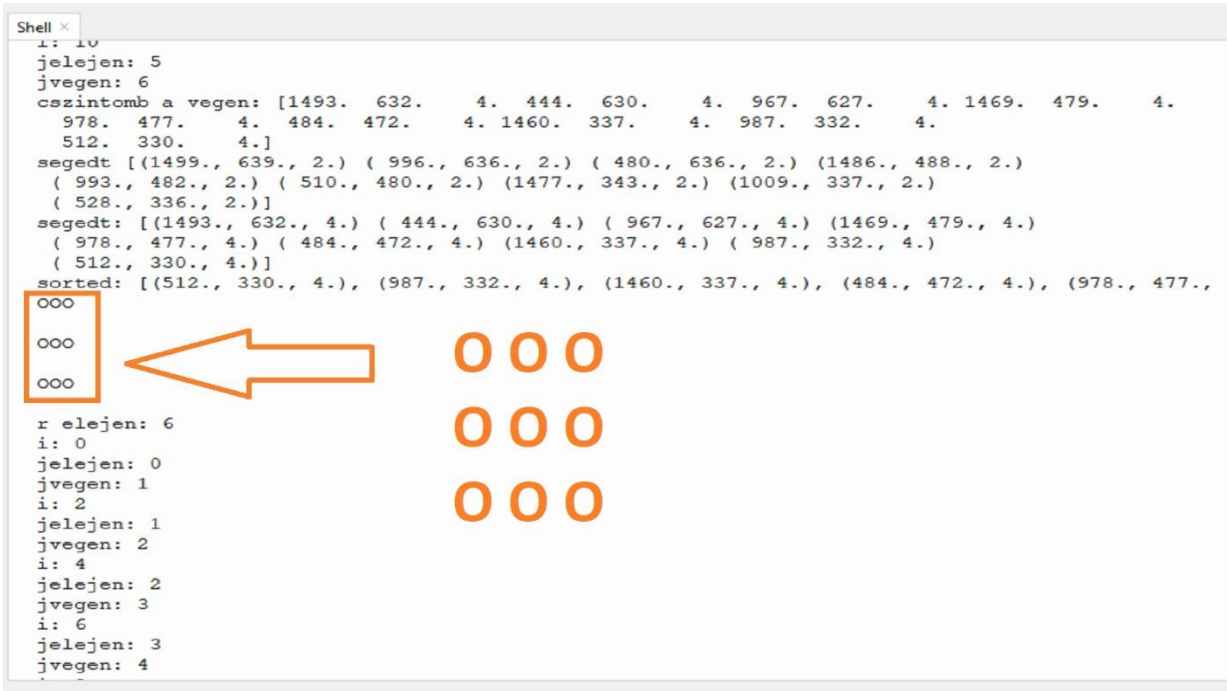
Nagyítva az eredmény:



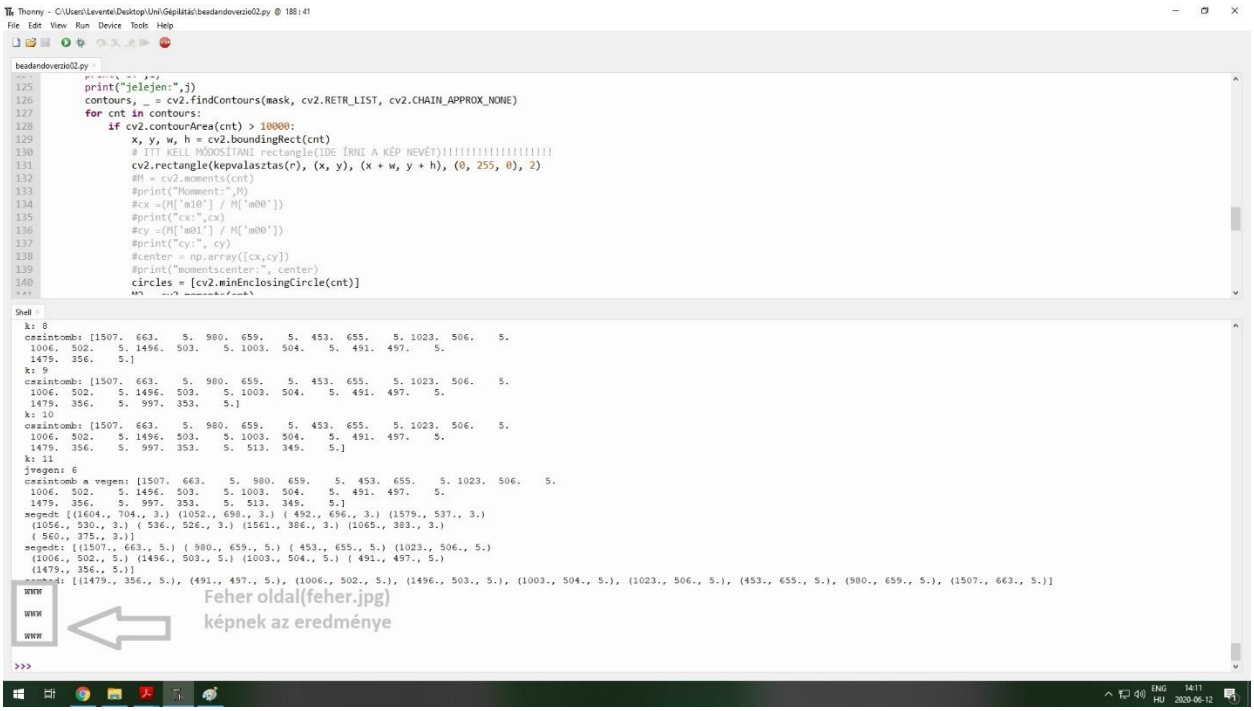
5. Kép tesztelése: (narancs.jpg)



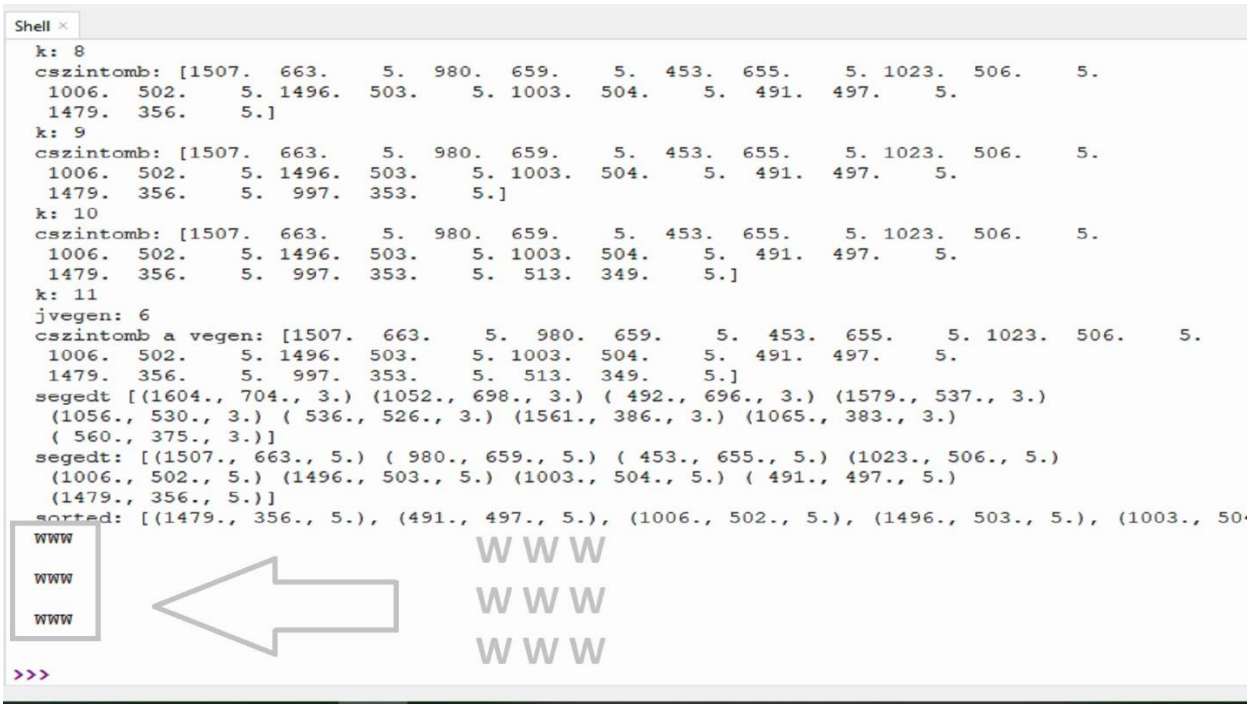
Nagyítva az eredmény:



6. Kép tesztelése: (feher.jpg)



Nagyítva az eredmény:



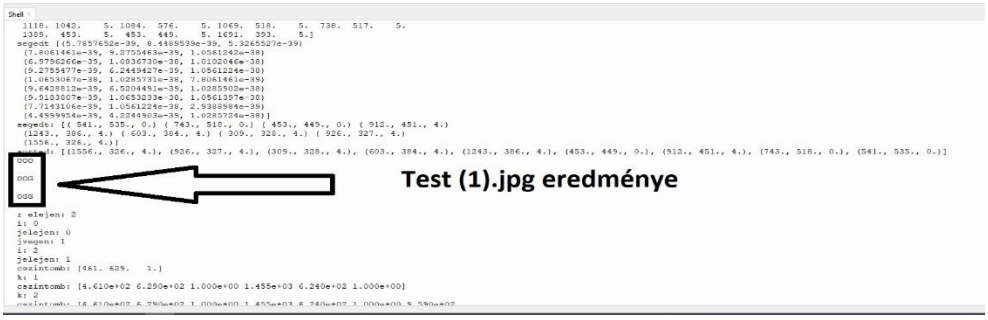
Tesztelés zavaró tényezős képpel

Ezen az oldalon fogom bemutatni azokat a teszteket, amelyek tartalmaznak kirakott, és összekevert kockát egyaránt, de olyan környezeti háttérrel, és olyan zavaró tényezőkkel, vagy esetleg helytelen fényviszonyok között, hogy azt a program már nem tudja rendesen feldolgozni.

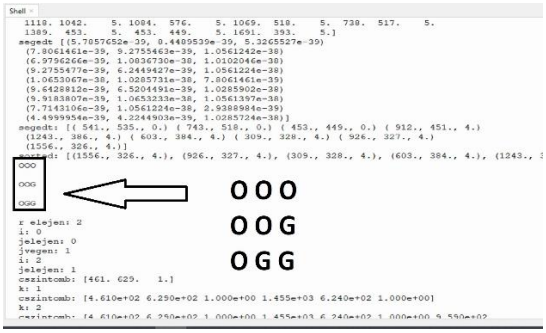
Jelentése az eredményen látható betűknek:
G->Green, ami magyarul a zöld színt jelenti
B->Blue, ami magyarul a kék színt jelenti
R->Red, ami magyarul a piros színt jelenti
Y->Yellow, ami magyarul a sárga színt jelenti
O->Orange, ami magyarul a narancs színt jelenti
W->White, ami magyarul a fehér színt jelenti

1. **Tesztelés(Test (1).jpg):** Ez a kép egy kirakott kockát tartalmaz, de az benne a zavaró tényező, hogy több oldalát is látjuk, illetve a fényviszonyok sem megfelelőek, hiszen árnyék tűnik fel mögötte:

Eredménye a tesztnek:



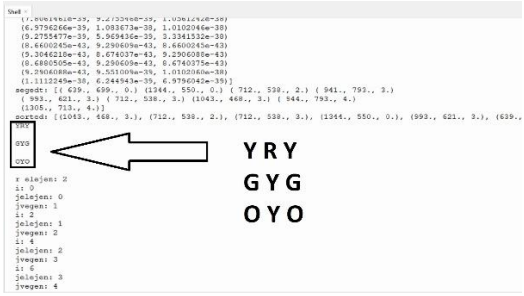
Nagyítva:



2. **Tesztelés (Test (15).jpg): Ezen a képen egy kevert kockát láthatunk elforgatva, illetve egy zavaró tényezőt. A hátrány, amiért a program hibát fog kidobni az az, hogy zavaró tényezőt raktam a képre, és a fényviszonyok is rosszak (árnyékok a képen).**

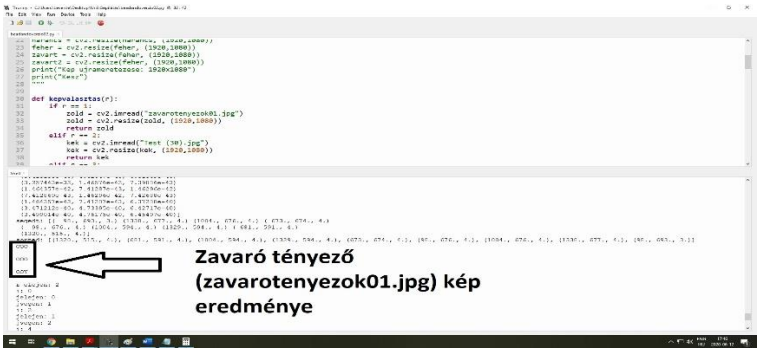
Eredménye a tesztnek:

Nagyítva:

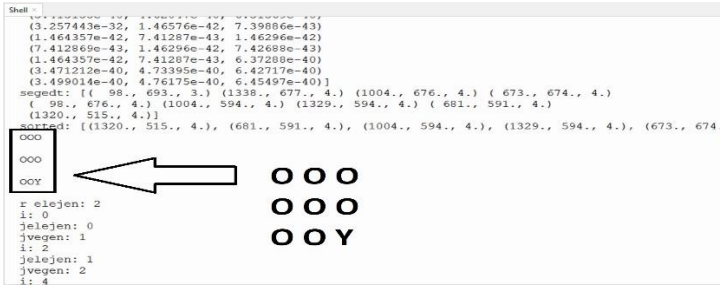


3. **Tesztelés (zavarotenyezok01.jpg): Ezen a képen egy kirakott kocka piros oldalát láthatjuk. Több zavaró tényező tárgy is látható a képen. Fényviszony is más mint az előző képeken.**

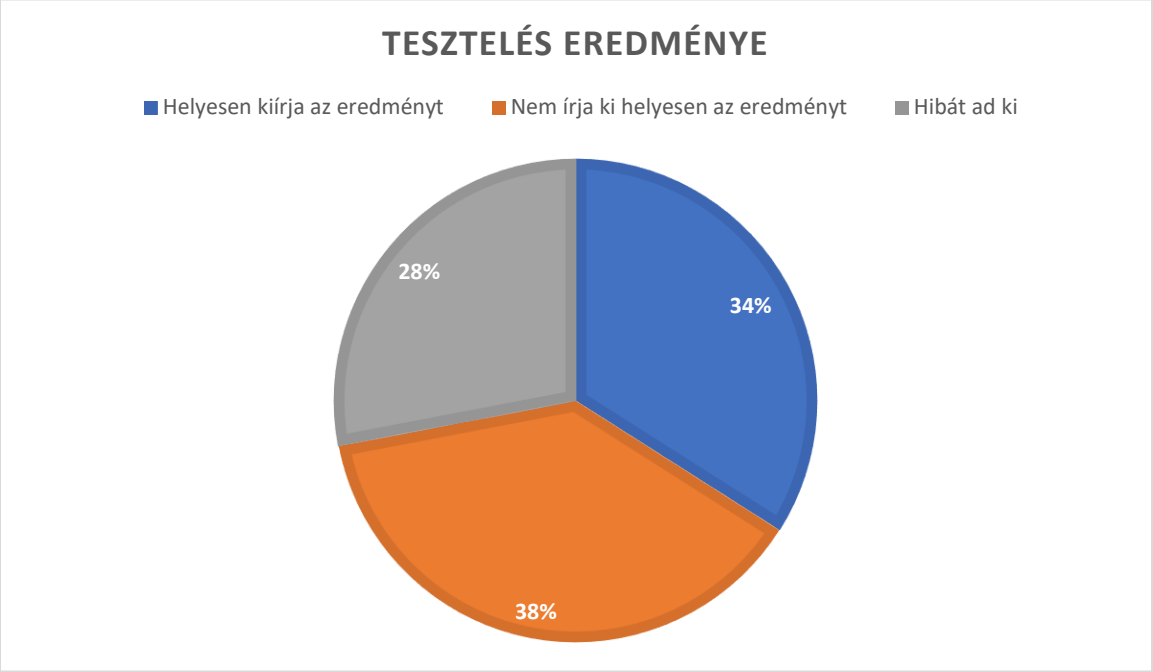
Eredménye a tesztnek:



Nagyítva:



Teszteltem mind a 32 teszt képet, és százalékosan ez az eredmény született:



<u>Ezek között a teszt képek között található:</u>

- -Kirakott állapotú kocka
- -Kevert kocka
- -Homogén háttérrel rendelkező kocka
- -Zavaró tényezővel rendelkező háttér
- -Zavaró tárgyak, zavaró színekkel
- -Fényviszonyok különböző formája
- -Elforgatott kocka
- -Mind a három oldalt látható kocka pozíció

Felhasználói leírás

Először is 6 darab képre van szükségünk a program helyes futásához(Ezeket úgy kell elkészíteni, hogy a kocka mind a 6 oldalát lefotózzuk). Ezt a hat képet ugyan abba a mappába kell tenni, ahol a program kódja található. Olyan néven nevezzük el, ahogy a képen látható kocka oldalának a közepén lévő szín van. Tehát például ha lefotóztuk 6 külön oldalát a kockának, és az adott képen a fehér van középen, akkor azt érdemes feher.jpg névre átnevezni, és így tovább. Ezt a 6 képet ha bemásoltuk, akkor további teendők nincs. A programnak működnie kell.

Felhasznált program:

Ide inkább egy kódot szeretnék megadni, amit feltöltöttem a githubra, ami a hsv detektálásában segített, hogy a színeket jól tudja maszkolni a program, és ennek a fájlnak a neve:hsv_detection.py

Természetesen ez nem az én munkám, de nem találtam meg sajnos a linket, amelyről letöltöttem.