Gépi látás

Márton Arnold  
xdt7oo

Széchenyi István Egyetem

Tartalomjegyzék

[Bevezetés 3](#_Toc167914300)

[Eszközök és technológiák ismertetése 3](#_Toc167914301)

[Python 3](#_Toc167914302)

[OpenCV 3](#_Toc167914303)

[NumPy 3](#_Toc167914304)

[Openpyxl 3](#_Toc167914305)

[Műszaki dokumentáció 4](#_Toc167914306)

[Az elkészült verzió elérhetősége 4](#_Toc167914307)

[Felhasznált könyvtárak 4](#_Toc167914308)

[Definiált segédfüggvények 4](#_Toc167914309)

[A script lépései 5](#_Toc167914310)

[Gamma érték beállítása 5](#_Toc167914311)

[Kép beolvasása és előfeldolgozása 5](#_Toc167914312)

[A kivágott Rubik-kocka képének feldolgozása 8](#_Toc167914313)

[Színek felismerése 9](#_Toc167914314)

[Log fájl készítése 10](#_Toc167914315)

[Eredmény prezentálása Excel táblázat használatával 10](#_Toc167914316)

[A script használata 11](#_Toc167914317)

[A Rubik-kocka fényképezése 11](#_Toc167914318)

[Képfájlok elhelyezése 12](#_Toc167914319)

[Script futtatása 12](#_Toc167914320)

[A színfelismerés kiértékelése 13](#_Toc167914321)

[Táblázat a felismerési arányról 13](#_Toc167914322)

[Felismerés értékelése 13](#_Toc167914323)

[További fejlesztési lehetőségek 14](#_Toc167914324)

[Összefoglaló 14](#_Toc167914325)

# Bevezetés

A projekt célja a Rubik-kocka állapotának felismerése fényképek alapján, és a felismert állapot prezentálása. A script képes feldolgozni a Rubik-kockáról készített fényképeket, azonosítani a színeket, majd az eredményeket egy Excel táblázatban megjeleníteni. A dokumentáció célja bemutatni az elkészült python script működését és használatát.

# Eszközök és technológiák ismertetése

A projekt megvalósításához az alábbi eszközöket és technológiákat használtam:

Python: A projekt fő programozási nyelve.

A Python egy magas szintű, általános célú programozási nyelv, amelyet Guido van Rossum alkotott meg, és először 1991-ben jelent meg. A Python tervezésénél nagy hangsúlyt fektettek a kód olvashatóságára és egyszerű szintaxisára, amely lehetővé teszi a programozók számára, hogy kevesebb kódsorral fejezzék ki az ötleteiket. Ennek köszönhetően a Python egy rendkívül hatékony és könnyen tanulható nyelv, amely széles körben elterjedt mind az iparban, mind az oktatásban.

OpenCV: A képfeldolgozáshoz és színazonosításhoz használt könyvtár.

Az OpenCV (Open Source Computer Vision Library) egy nyílt forráskódú számítógépes látás könyvtár, amelyet az Intel hozott létre 1999-ben, és ma széles körben használják különféle képfeldolgozási és számítógépes látási feladatok megoldására. Az OpenCV C++, Python és Java nyelveken is elérhető, de a Python interfész az egyik legnépszerűbb a könnyű használhatóságának és rugalmasságának köszönhetően.

NumPy: A numerikus számításokhoz és tömbkezeléshez használt python könyvtár.

Openpyxl: Az eredmények Excel táblázatba történő írásához használt python könyvtár.

# Műszaki dokumentáció

## Az elkészült verzió elérhetősége

repository: <https://github.com/Arnold0802/gepilatas24>

python fájl: rubik9.py

A githubra feltöltött repository tartalmazza a működő scriptet, a párhuzamos próbálkozásokat, a fejlesztés folyamán használt segéd állományokat, tesztek eredményeit, valamint a 23 mappát melyek a script működésének kiértékelésénél voltak mintaként felhasználva.

## Felhasznált könyvtárak

* OpenCV (cv2):
  + Képfeldolgozás, gamma korrekció, kontúrok keresése, színek megállapítása/összehasonlítása.
* NumPy:
  + Tömbműveletek, színek átlagolása.
* openpyxl:
  + Excel fájl létrehozása, Rubik-kocka ábrázolása, megállapított értékek megjelenítése

## Definiált segédfüggvények

**adjust\_gamma(image, gamma):**

Ez a függvény egy kép gamma-korrekcióját végzi el. A gamma-korrekció egy képfeldolgozási művelet, amelynek célja a kép fényességének módosítása anélkül, hogy a színeket aránytalanul befolyásolná. A gamma értékének megváltoztatásával sötétíthetjük vagy világosíthatjuk a képet.

Paraméterek:

image: A bemeneti kép, amelyen a gamma-korrekciót végre kell hajtani.

gamma: A gamma korrekciós értéke. Alapértelmezés szerint 1,5. Ha gamma < 1, a kép sötétebb lesz, ha gamma > 1, a kép világosabb lesz.

**find\_closest\_color(color, colors):**  
Megkeresi a legközelebbi színt az előre definiált színek közül.

**rgb\_to\_hex(rgb):**  
RGB színkódot hexadecimális formátumba konvertál.

# A script lépései

## Gamma érték beállítása

A képfeldolgozás során nagyon fontos a megfelelő színek használata, így a scriptet kiegészítettem egy külső ciklussal.

Ez a ciklus egy nagyon magas (2,5) gamma értéket állít be kezdőértékként, és a Rubik-kocka oldalairól készült képeket ezzel a gamma-korrekciós értékkel dolgozza fel. A cél az, hogy minden színből 9 darabot találjon. Ha nem jár sikerrel, csökkenti a gamma értékét 0,1-el, majd újra feldolgozza a Rubik-kockáról készült képeket az új gamma értékkel. Mindezt addig ismétli, amíg nem lesz sikeres a színek felismerése, vagy el nem ér egy túl alacsony, 0,1 alatti gamma értéket. Túl alacsony érték esetén egy „flag” beállításával jelzi, hogy a script képfeldolgozó része egy utolsó alkalommal fog lefutni, az alapértelmezett 1,0 gamma-érték használatával. A kimeneti fájl az 1,0 gamma érték használatával megállapított színek alapján készül el (1. forráskód részlet).

    if all\_nines or flag:

        go=0

    else:

        if gammavalue < 0.1:

            gammavalue = 1

            flag = 1

        else:

            gammavalue = gammavalue - 0.1

1. forráskód részlet: Gamma korrekciós ciklus feltételei

## Kép beolvasása és előfeldolgozása

A beolvasott képet a script először olyan képpé alakítja, ami csak fekete vagy fehér képpontokból áll (1. ábra). Ez a lépés azért szükséges, mert így sokkal könnyebb megtalálni a fényképen a Rubik-kocka kontúrjait.

A képen tér, tervezés, rács, fekete-fehér látható

Automatikusan generált leírás

1. ábra: fekete-fehér Rubik-kocka.

A bináris kép az alábbi módon készült (2. forráskód részlet):

A bemeneti képet először szürkeárnyalatos képpé alakítja a script, majd a szürkeárnyalatos képen a megadott küszöbérték alapján minden pixelről eldönti, hogy fehér vagy fekete színű lesz a visszaadott képen. Ehhez az OpenCV küszöbérték függvényét használtam (cv2.threshold) mely az alábbi módon működik:

A cv2.threshold függvény két értéket ad vissza: a küszöbértéket, amit nem használtam, ezért aláhúzással helyettesítem (’\_’), és a küszöbértékelt képet (’thresholded\_image’).

Azok a pixelek a gray\_image képen, amelyeknek értéke 50 vagy annál kisebb, 255 értéket kapják (fehérek lesznek), valamint azok a pixelek, amelyeknek az értéke 50-nél nagyobb, 0 értéket kapnak (feketék lesznek).

Ez a küszöbértékelt kép (thresholded\_image) egy bináris kép lesz, ahol a fehér (255) és a fekete (0) pixelek képviselik az eredeti kép alapján a küszöbértékelt területeket.

Példa:

Tegyük fel, hogy van egy szürkeárnyalatos képünk, amelynek néhány pixelértéke 30, 70, és 120.  
A küszöbértékelés eredményeként:

* A 30-as értékű pixelek 255-öt kapnak (fehér).
* A 70-es és 120-as értékű pixelek 0-át kapnak (fekete).

Ez az eljárás hasznos lehet olyan képfeldolgozási feladatokhoz, ahol a képből ki szeretnénk emelni bizonyos területeket, például objektumokat vagy alakzatokat, amelyek megkülönböztethetők a környezetüktől.

  gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

  \_, thresholded\_image = cv2.threshold(gray\_image, 50, 255, cv2.THRESH\_BINARY\_INV)

2. forráskód részlet: bináris kép elkészítése

A megtalált kontúrok segítségével egy bounding box megrajzolása következik, ami pontosan meghatározza a képen a Rubik-kocka helyzetét (2. ábra), majd ezeknek az értékeknek a felhasználásával az eredeti képből kivágja a hasznos területet, így a kép többi részével nem kell a későbbiekben foglalkozni.

A képen Mechanikus kirakójáték, játék, Rubik-kocka, puzzle látható

Automatikusan generált leírás

2. ábra: Kontúrok és bounding Box.

A bounding box meghatározásához az OpenCV **boundingRect** függvényét használtam  
(3. forráskód részlet).

A függvény megkeresi a legkisebb téglalapot, amely teljesen körülzárja a megadott kontúrt, így a Rubik-kocka köré rajzolt szabálytalan alakzat alapján egy szabályos téglalapot kaptam.  
A függvény által visszaadott értékek:

* bal felső sarok x és y koordinátái
* téglalap szélessége és magassága

  # Bounding box meghatározása

  x, y, w, h = cv2.boundingRect(largest\_contour)

3. forráskód részlet: boundingRect használata

## A kivágott Rubik-kocka képének feldolgozása

Miután elkészült a kivágott kép, a script alkalmazza erre a képre a gamma-korrekciós függvényt.  
A gamma-korrekció után a képet 2 vízszintes és 2 függőleges vonal segítségével a script 9 egyenlő részre osztja (3. ábra), majd ezeket a képrészleteket egyesével, külön képekként dolgozza fel   
(4. ábra).

A képen Színesség, színek, paletta, művészet látható

Automatikusan generált leírás

3. ábra: 9 egyenlő területre osztás.

A képen Színesség, Téglalap, színek, művészet látható

Automatikusan generált leírás

4. ábra: a 9 különálló cella.

A Rubik-kocka fehér oldalán középen található a termék logója. Ez a színekkel teli logó vezetett ahhoz a megoldáshoz, hogy mind a 9 képből egy kisebb részletet vág ki a script a bal felső részből, a cella színének megállapításához. (És ahhoz is, hogy kötött a kockáról készült képek elkészítésének módja.)

## Színek felismerése

Miután megvan a kilenc színminta, a mintán található színek átlagolása történik, a kimenete pedig egy RGB színkód.

A kapott RGB értéket a script összehasonlítja a Rubik-kocka színeit tartalmazó előre definiált színtáblázattal (4. forráskód részlet), majd megállapítja, hogy melyik színkódhoz áll a legközelebb. A megtalált színt eltárolja egy numpy tömbben.

# Definiáljuk a 6 szín RGB kódját

colors = {

    'red': np.array([173, 44, 72]),

    'green': np.array([38, 145, 92]),

    'blue': np.array([16, 103, 168]),

    'yellow': np.array([204, 193, 44]),

    'orange': np.array([197, 129, 18]),

    'white': np.array([216, 204, 200])

}

4. forráskód részlet: színkódok

A színkódok meghatározásához a Rubik-kockáról készült első felvételsorozatot használtam. A színek 9 példányának RGB értékeit egy Excel táblázatban rögzítettem (5. ábra), majd az értékek átlagait használtam fel a „colors” értékeinek megadásához. (repo\ színértékek.xlsx)

A képen szöveg, képernyőkép, szám, sor látható

Automatikusan generált leírás

5. ábra: Excel tábla a színkódok megállapításához.

Miután mind a hat kép feldolgozása megtörtént, minden színből kilenc darabnak kell lennie, valamint a hat középső cellának mind egyedi színnel kell rendelkeznie, itt nem lehet színismétlődés.

## Log fájl készítése

Kiegészítettem a scriptet egy szöveges dokumentum elkészítésével, amiben megtalálható a felismert színek adatstruktúrája két verzióban. Az első a színek RGB kódjait tartalmazza, míg a második a színek neveit.

A log fájl továbbá tartalmazza a megtalált színek darabszámát, a középső cellák színeit és egyediségét, valamint a színfelismerés során használt gamma értéket.

## Eredmény prezentálása Excel táblázat használatával

A script az utolsó fázisában készít egy Excel fájlt, ahol a teljes Rubik-kocka ábrázolása történik 2 dimenzióban.

Az eredmény ellenőrzésének, hibák keresésének megkönnyítése érdekében három módon jeleníti meg a megállapított értékeket.

Az első részben a cellákat az RGB színkódnak megfelelő színnel tölti fel a script, a második rész a színek neveit tartalmazza, a harmadik rész pedig az RGB színkódot.

A „kiterített kocka” alatt a színösszesítés és a feldolgozás során használt gamma érték is megtalálható.

Az Excel fájl elkészítése után a script a futtatásának végéhez ér.

# A script használata

## A Rubik-kocka fényképezése

A script használatához szükség van a Rubik-kocka hat oldaláról készült fényképekre.  
A fényképeknek egységes színű háttérrel kell rendelkeznie, a nagyon erős árnyékok, színátmenetek, mintázatok hatására a kontúrok megtalálása nehézkessé válhat.

A „kiterített kocka” ábrázolásmódnak megfelelően kell elkészíteni a képeket a Rubik-kockáról, viszont az első képnek a logóval ellátott fehér oldalnak kell lennie.

A zöld, narancssárga, piros és kék oldalak csak a teljesen kirakott esetben vannak a 7. ábrán látható pozíciókban, a kocka más állapotában más pozícióban is lehetnek.

Fehér oldal helyes tájolása (6. ábra):

A képen szöveg, képernyőkép, tér, sor látható

Automatikusan generált leírásA képen rajzfilm látható

Automatikusan generált leírás

7. ábra: 2D-s ábrázolás, "kiterített kocka"

6. ábra: Rubik's logó tájolása.

A fényképek elkészítésének folyamata:

1. kép: a fehér oldal, a Rubik’s logó a 8. ábrán látható tájolásban van.

2. kép: a fehér oldal fölötti rész, a kockán egyet kell fordítani magunk felé.

3. kép: a fehér oldal a kiinduló pont, a balra eső oldalról kell képet készíteni, ehhez a kockán jobbra kell fordítani egyet.

4. kép: a fehér oldal ismét a kiinduló pont, a jobbra eső oldalról kell képet készíteni, ehhez a kockán balra kell fordítani egyet.

5. kép: a fehér oldal alatti oldal, a kockán a fehér oldaltól számítva egyet kell fordítani fölfelé.

6. kép: a sárga oldal, a kocka alja (ha a fehér oldalt tekintjük a tetejének). A fehér oldaltól számítva kettőt kell fordítani a kockán fölfelé.

## Képfájlok elhelyezése

A képek számára létre kell hozni egy mappát, majd a mappában a kép sorszámának megfelelően kell a képet elnevezni. (pl: 1.jpg, 2.jpg, 3.jpg, 4.jpg, 5.jpg, 6.jpg)

A scriptben meg kell adni a mappa nevét a 69. sorban található mappa változóban (3. forráskód részlet).

mappa = 23

3. forráskód részlet: mappa változó értékének megadása

## Script futtatása

A script futtatásához szükség van a python telepítésére, valamint pip segítségével telepíteni kell az opencv-python könyvtárat (ez telepíti a numpy-t is) és az openpyxl könyvtárat.

Ehhez Command prompt-ban az alábbi sorokat kell beírni:

pip install openpyxl

pip install opencv-python

Eltérő telepítés esetén a futtatás eltérhet, Windows-os környezetben a Command prompt-ból a futtatás történhet az alábbi módon:

python rubik9.py

A script futtatásának végén a Command prompt ablakban a „kész” felirat jelenik meg.

A log fájl és az eredményt tartalmazó Excel fájl a képeket tartalmazó mappába kerül.

# A színfelismerés kiértékelése

## Táblázat a felismerési arányról

Rubik-kocka felismerésének eredménye

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Vizsgálat** | **Minták száma** | **Sikeres felismerés** | **Sikertelen felismerés** | **Felismerési arány** |
| teljes kocka | 23 | 20 | 3 | 87% |
| piros szín | 207 | 198 | 9 | 96% |
| zöld szín | 207 | 198 | 9 | 96% |
| kék szín | 207 | 198 | 9 | 96% |
| narancssárga szín | 207 | 197 | 10 | 95% |
| citromsárga szín | 207 | 196 | 11 | 95% |
| fehér szín | 207 | 198 | 9 | 96% |

## Felismerés értékelése

A teljes kocka felismerése a 23 mintából mindössze 3 alkalommal nem volt sikeres, ez annak köszönhető, hogy tudatosan próbáltam keresni a határait a képfeldolgozó scriptnek.

A hármas számú mintánál egy érdekes hiba jött elő, itt ugyanis a színek darabszáma megfelelt az elvárásoknak, mivel 2 hibás szín is volt, egy citromsárga helyett narancssárga és egy narancssárga helyett citromsárga. Az eredmények vizsgálatánál kiderült, hogy nagyon sötét volt a fényképeken a Rubik-kocka árnyéka, és olyan pozícióban helyezkedett el a kockához képest, hogy szabályos négyszöget sikerült felismerni a képeken úgy, hogy a kocka az árnyékával együtt szerepelt a kivágott képen. Emiatt a cellákra bontás is elcsúszott, így a színfelismeréshez rossz helyről történt a mintavételezés.

A hetes számú minta rossz fényviszonyok között készült, itt a fehér színek is elég kékes árnyalatúak voltak, de a legközelebbi szín továbbra is a fehér volt, így a felismerés sikeres volt. Itt egy hiba volt, egy sárga színt fehérnek ismert fel a script.

A nyolcas számú minta teljesen értékelhetetlen lett sajnos. Bekapcsolt, sötét hátterű képernyő előtt fényképeztem a Rubik-kockát, meglepő lett volna, ha így is sikeres lesz a felismerés. Sajnos a Rubik-kockát nem sikerült megtalálni a képeken. Ennek a mintának köszönhető, hogy semelyik színfelismerés nem érte el a 100%-os értéket.

# További fejlesztési lehetőségek

A scriptet további funkciókkal lehetne kiegészíteni a jövőben, az egyik hasznos kiegészítés a grafikus felület lenne, ami szerencsére a python programnyelv esetén egyszerűen kivitelezhető.

Egy másik hasznos dolog lenne a script kiegészítése argumentumokkal. Meg lehetne adni argumentumokkal a mintaképeket tartalmazó mappa nevét, valamint argumentumok alapján lehetne módosítani a script futásán. Ötletek a módosított futtatásra:

* nem teljes Rubik-kockát próbál meg felismeri, csupán egy megadott oldalt vizsgál,
* futás közben minden lépésnél egy ablakban megmutatja, épp mi történik a képpel,
* fix gamma-értékkel fut le.

Két bonyolultabb funkció kiegészítési lehetőség:

1. A felismert képek elemzése, ami alapján a script el tudná dönteni, hogy melyik oldal hogyan csatlakozik a többihez, így nem lenne szükség a fényképek kötött módon való elkészítésének. Valamint hibás/lehetetlen állapot felismerése esetén a felhasználó figyelmeztetése.
2. A felismert Rubik-kocka állapot alapján elkészíteni a kirakáshoz szükséges lépések listáját.

# Összefoglaló

A fejlesztési folyamat során sok funkciót kipróbáltam az opencv lehetőségei közül, viszont az újratervezések során sok lépés feleslegessé vált. A repoban továbbra is megtalálható a korábbi próbálkozásaim több verziója is. Kísérleteztem a színek élénkítésével, használtam maszkolást a nem használt részek elfedésére, ez a szintén használaton kívül esett leggyakoribb szín megtalálása függvényhez volt hasznos.

Én elégedett vagyok a script teljesítményével, a korai verziókhoz képest sokkal magabiztosabb lett a felismerés, ami két fontos résznek köszönhető: a Rubik-kocka valódi színei alapján elkészített színérték táblázat, valamint a scriptben alkalmazott gamma-érték léptetés, a felismert színek darabszáma alapján. Ennek segítségével lehetett a különböző fényviszonyok között készült képeken a sárga területeket megkülönböztetni a narancssárga színű területektől.

Források

[1.] <https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html>

[2.] <https://hu.wikipedia.org/wiki/Rubik-kocka>

[3.] <https://openpyxl.readthedocs.io/en/stable/>