**Gépi látás (GKNB_INTM038)**
**Útjelző tábla felismerés**
**(Habán András-Zh5uo8)**

## Feladatismertetés:

A Gépi látás nevű tantárgy keretein belül, ahol igazából képfeldolgozás alapjaival ismerkedtünk meg, az én feladatom volt egy Útjelző tábla felismerő program megvalósítása volt. Előre meg kellett határozni, hogy melyik táblákat tudja a program beazonosítani. A választásom 6 kör alakú táblára esett, mivel Pythonban a körök felismerésére nagyszerű beépített függvények állnak rendelkezésre. A következő táblákat képes a program felismerni: „Behajtani tilos!” (lásd 1. ábra), „Kötelező haladási irány egyenesen és balra!” (lásd 4. ábra), „Kötelező haladási irány egyenesen és jobbra!” (lásd 6. ábra), „Kötelező haladási irány egyenesen!” (lásd 2. ábra), „Kötelező haladási irány balra!” (lásd 3. ábra), „Kötelező haladási irány jobbra!” (lásd 5. ábra).

*1. ábra Behajtani tilos!*

*2. ábra Egyenesen!*

*3. ábra Balra!*

*4. ábra Egyenesen és balra!*

*5. ábra Jobbra!*

*6. ábra Egyenesen és jobbra!*

## Elméleti háttér:

**Hough-transzformáció:** Megadja a kép egy pontján átmenő összes egyenes egyenletét. Akkor alkalmazzuk, ha ismert alakzatokat keresünk egy képen, még ha zajosak és részben takartak is. Ez a körök felismerésénél is használható.

**Éldetektálás:** Ott található él, ahol az intenzitás hirtelen megváltozik. Lehetnek élek egy objektumon belül is, tehát körültekintően kell eljárni ilyenkor. Az élek detektálásának leggyakoribb módszere a Canny-detektor, illetve amikor a képek deriváltját Sobel és Prewitt kernelekkel közelítjük.

**Szürkeárnyalati konverzió:** Számos alacsony szintű képfeldolgozó művelet szürkeárnyalatos intenzitásképeken kerül definiálásra. A fotóink, képeink rendszerint színesek, ezeket át kell alakítanunk szürkeárnyalatosra a műveletek végrehajtása előtt.

**Színfelismerés:** A számítógépek képernyőjén megjelenő színek a vörös, zöld és kék színkomponensek intenzitásának elegyítéséből állnak elő. A fotók betöltés után jellemzően ezen a három színcsatornán tárolódnak. Ezen adatok ismeretében konkrétan beazonosíthatók az egyes színek.

**Elméleti megoldás:** Az alapelképzelés az, hogy keressünk köröket egy statikus képen. Ha megtaláltuk a köröket valahogy el kell dönteni, hogyan hozzon döntést a program. A legegyszerűbb megoldás, ha a BGR színskálán meg tudnánk határozni, hogy az adott képszeleten melyik szín milyen értékkel szerepel 0 és 255 között. Ez alapján már egy piros és egy kék táblát könnyedén meg lehet különböztetni. Ha viszont képesek vagyunk a kört egy négyzet beírt körévé tenni, akkor a képszeletet még további képszeletekre lehet osztani, ami alapján tudunk következtetni, hogy melyik tábláról van is szó. Ezen az úton indultam el a megvalósítás felé.

## Megvalósítás:

Első lépésben importáljuk a szükséges könyvtárakat (lásd 7. ábra), amiket előzetesen telepítettünk. Mivel egy általunk választott mappából szeretnénk képeket kinyerni, szükségünk lesz a „glob”-ra is, ami globális változókkal való műveleteket tesz lehetővé.

```
1  import cv2
2  import numpy as np
3  from scipy.stats import itemfreq
4  import glob
```

*7. ábra Szükséges könyvtárak és csomagok importálása*

**Színfelismerő függvény:**

```
def get_dominant_color(image, n_colors):
    pixels = np.float32(image).reshape((-1, 3))
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 200, .1)
    flags = cv2.KMEANS_RANDOM_CENTERS
    flags, labels, centroids = cv2.kmeans(
        pixels, n_colors, None, criteria, 10, flags)
    palette = np.uint8(centroids)
    return palette[np.argmax(itemfreq(labels)[:, -1])]
```

*8. ábra Színfelismerő függvény*

Egy függvényt (lásd 8. ábra) fogunk alkalmazni, amikor el kell döntenünk egy adott képről, hogy melyik szín dominál a kép egy adott szeletén. Eredménye egy tömb lesz, ami a BGR színskálán fogja megmondani (ebben a sorrendben), hogy milyen érték tartozik hozzá. Először át kell konvertálni a képet 32 bites, lebegőpontos számmá és átméretezni. Következő sorokban végig iterálunk a képpontokon, majd csoportosítjuk őket. Végül pedig visszaalakítjuk, visszaadjuk a domináns színt.

**Bemeneti adatok megadása:**

```
15
16 bh=0
17 e=0
18 j=0
19 b=0
20 ej=0
21 eb=0
22
23 cv_img = []
24 for i in glob.glob("Jobb/*.jpg"):
25     n= cv2.imread(i)
26     cv_img.append(n)
27
28 kepek=len(cv_img)
29 for i in range(kepek):
30     img=cv_img[i]
```

*9. ábra A képek programba történő beolvasása*

Deklarálunk minden táblára egy-egy változót, amikben majd el fogjuk tárolni, hogy hányat ismert fel a program. Bemeneti képeink egy-egy mappában vannak, innen szeretnénk egy tömbbe eltárolni. Hogy ezt megtehessük, ezért kellett importálni a „globot". Innentől append utasítással feltöltjük a tömböt. Érdemes eltárolni, hogy hány kép van a tömbben a len függvény segítségével. Ezek után könnyedén végig tudunk menni a képeken egyesével.

**Körök érzékelése:**

```
meret = 500
height, width,channels = img.shape

arany = height/width

meretarany = int(round(meret*arany))
meretk = cv2.resize(img, (meret,meretarany), interpolation = cv2.INTER_AREA)

frame = meretk

 #hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
 #lower_red = np.array([50,50,50])
 #upper_red = np.array([130,255,255])
 #mask = cv2.inRange(hsv, lower_red, upper_red)
 #res = cv2.bitwise_and(frame,frame, mask= mask)



szurke = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
homaly= cv2.medianBlur(szurke,11)
korok = cv2.HoughCircles(homaly, cv2.HOUGH_GRADIENT,
                         1, 50, param1=190, param2=100)
```

*10. ábra A kép előkészítése és a körök érzékelése*

Szerencsére a cv2 HoughCircles függvénye képes érzékelni a köröket, mindazonáltal a jobb működés érdekében érdemes előtte pár dolgot megtenni. Először is a függvény fix paraméterekkel dolgozik és a kép szélességét érdemes 500-ra állítani és a magasságát a megfelelő arány szerint változtatni.
Ha ezzel megvagyunk, az átméretezett képen elvégezzük a szürkeárnyalati konverziót és elhomályosítjuk. A HoughCircles függvény paramétereivel tudjuk beállítani, hogy mennyire legyen érzékeny a program a kör objektumokra. Ezt érdemes a tesztadatoknak megfelelően beállítani.
param1-ezen paraméter növelésével egyre több kört fog érzékelni a program
param2-ezen paraméter növelésével elhagyja a felesleges köröket.

**Körök rendezése:**

```
if not korok is None:
    korok = np.uint16(np.around(korok))
    max_r, max_i = 0, 0
    for i in range(len(korok[:, :, 2][0])):
        if korok[:, :, 2][0][i] > 50 and korok[:, :, 2][0][i] > max_r:
            max_i = i
            max_r = korok[:, :, 2][0][i]
    x, y, r = korok[:, :, :][0][max_i]
    if y > r and x > r:
        negyzet = frame[y-r:y+r, x-r:x+r]
```

*11. ábra A körök "max" rendezése*

Rendezzük a köröket méretük szerint (lásd 11. ábra), mivel nekünk nagy valószínűséggel a legnagyobb, azaz a hozzánk legközelebbi táblát kell megvizsgálnunk. Ha ezzel megvagyunk, ezen a ponton érdemes egy négyzetbe szorítani a talált köröket. Ezt a négyzetet fogjuk átadni majd a programnak további elemzésre.

**Piros-kék döntés:**

```
szin = get_dominant_color(negyzet, 2)
if szin[2] > 100:
    szoveg="Behajtani tilos!"
    bh=bh+1
elif szin[0] > 80:
    balnyil= negyzet[negyzet.shape[0]*3//8:negyzet.shape[0]
                     * 5//8, negyzet.shape[1]*1//8:negyzet.shape[1]*3//8]
    balnyil_szin = get_dominant_color(balnyil, 1)

    felnyil = negyzet[(negyzet.shape[0]*1//8)-30:(negyzet.shape[0]
                     * 3//8)-30, (negyzet.shape[1]*3//8):(negyzet.shape[1]*5//8)]
    felnyil_szin = get_dominant_color(felnyil, 1)

    jobbnyil= negyzet[negyzet.shape[0]*3//8:negyzet.shape[0]
                     * 5//8, negyzet.shape[1]*5//8:negyzet.shape[1]*7//8]
    jobbnyil_szin = get_dominant_color(jobbnyil, 1)
```

*12. ábra Piros és kék tábla elkülönülése*

Megkeressük ezen a képen a domináns színt az előre definiált függvénnyel és eltároljuk egy változóba. Ha a piros szín van rajta, akkor a szin[2] értéke nagyobb száznál és egyértelmű, hogy a „Behajtani tilos!” tábla van a képen. Ellenkező esetben a kék táblák között kell keresgélnünk.
Itt érdemes végig gondolni, hogyan lehet ezeket a táblákat felismerni. Meg kell tudnunk, hogy a jobbra, balra és fenn található-e nyíl. További képeket kell kivágnunk a négyzetből a nyilak helyén (lásd 12. ábra), ami alapján már nem lesz nehéz döntést hozni. Ezután el kell tárolni 3 változóba ezen képszeletek színét.

**Végső döntés:**

```
        if felnyil_szin[2] < 60:
            if sum(balnyil_szin) > sum(jobbnyil_szin):
                szoveg = "Balra!"
                b=b+1
                cv2.imshow('Bal',balnyil)
            else:
                szoveg = "Jobbra!"
                j=j+1
                cv2.imshow('Jobb',jobbnyil)
        else:
            if sum(felnyil_szin) > sum(balnyil_szin) and sum(felnyil_szin) > sum(jobbnyil_szin):
                szoveg = "Egyenesen!"
                e=e+1
                cv2.imshow('Egyenes',felnyil)
            elif sum(balnyil_szin) > sum(jobbnyil_szin):
                szoveg = "Egyenesen es balra!"
                eb=eb+1
                cv2.imshow('Bal',balnyil)
                cv2.imshow('Egyenes',felnyil)
            else:
                szoveg = "Egyenesen es jobbra!"
                ej=ej+1
                cv2.imshow('Jobb',jobbnyil)
                cv2.imshow('Egyenes',felnyil)
    else:
        szoveg = "Nem tabla!"
```

*13. ábra Az elágazások felépítése*

Alapelvünk az, hogy ahol nyíl található, ott nagyobb lesz az intenzitás, mint ahol nincs (BGR (255,255,255) =fehér). Először megnézzük, hogy a felső nyíl helyén kék szín van-e, ezután azt vizsgáljuk, hogy jobb vagy bal oldalon nagyobb-e az intenzitás és döntést hozunk. A megfelelő képszeletet megjelenítjük, a darabszám változót növeljük és a szöveget, amit majd rá akarunk illeszteni a képre, beállítjuk.
A logikát alkalmazzuk végig, összehasonlítjuk a képszeletetek intenzitását és döntünk (lásd 13. ábra).

## Tesztelés dokumentálása és felhasználói leírás:

**Bemenet:** Bemeneti adatként egy mappát vár a program egyforma típusú képfájl tartalommal. Én az egyszerűség kedvéért külön mappában tároltam a táblák tesztadatait.

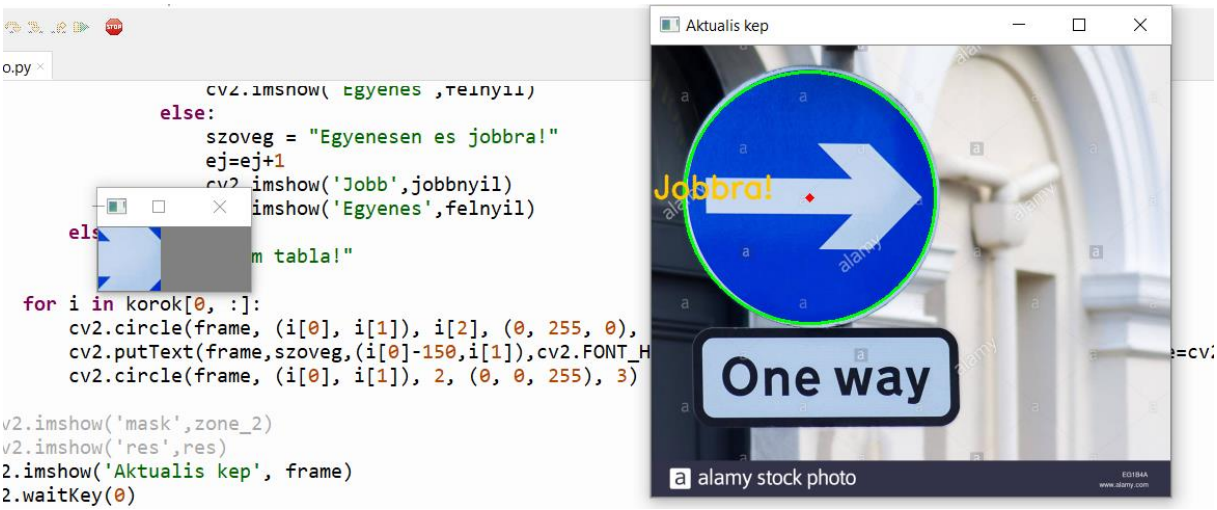
**Kimenet:**

```
    for i in korok[0, :]:
        cv2.circle(frame, (i[0], i[1]), i[2], (0, 255, 0), 2)
        cv2.putText(frame,szoveg,(i[0]-150,i[1]),cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0, 200, 255),2, lineType=cv2.LINE_AA)
        cv2.circle(frame, (i[0], i[1]), 2, (0, 0, 255), 3)

#cv2.imshow('mask',zone_2)
#cv2.imshow('res',res)
cv2.imshow('Aktualis kep', frame)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

*14. ábra Kimenetek megjelenítése és az ablakok bezárása*

Megjelenítjük az eredeti képet, rárajzolunk egy zöld kört, hozzáadjuk a szöveget és egy piros ponttal megjelöljük a kör középpontját (lásd 14. ábra). Itt billentyűlenyomással tudunk végig menni a képeken. Minden lenyomásnál bezáródik az összes ablak és újak nyílnak ki. A program kimenetén megjelenik az a képszelet is, ami alapján a program meghozta végső döntését (lásd 15. ábra).




*15. ábra Egy példa a kimenetre*

**Hatékonyságvizsgálat:**

Mivel nem konkrét dataset-et használtam, hanem az interneten kerestem minél változatosabb képeket, (szám szerint 101 darabot) egyedi kiírást csináltam minden táblára bele véve a hamis adatokat is. Ezért a felhasználónak kell „kikommentezni”, melyik tábláról készüljön a statisztika. Egyszerűen el kell osztani a talált képek számát a mappában lévő képek számával, szorozni 100-zal és kerekíteni 2 tizedesjegyre(lásd 16. ábra).

```
# print("A behajtani tilos tablakat a program ekkora százalékban ismeri fel:",round(bh/(kepek-2)*100,2),"%")
print("A kötelező haladási irány jobbra tablakat a program ekkora százalékban ismeri fel:",round(j/kepek*100,2),"%")
# print("A kötelező haladási irány balra tablakat a program ekkora százalékban ismeri fel:",round(b/(kepek)*100,2),"%")
# print("A kötelező haladási irány egyenesen tablakat a program ekkora százalékban ismeri fel:",round(e/kepek*100,2),"%")
#print("A kötelező haladási irány jobbra és egyenesen tablakat a program ekkora százalékban ismeri fel:",round(ej/kepek*100,2),"%")
# print("A kötelező haladási irány balra és egyenesen tablakat a program ekkora százalékban ismeri fel:",round(eb/kepek*100,2),"%")
```

*16. ábra A kimenet felhasználó általi hangolása*

**Eredmény táblázatosan:**

| Táblák | Tesztadatok (db) | Helyes adatok (db) | Hamis adatok (db) | Hatékonyság (%) |
|---|---|---|---|---|
| Behajtani tilos! | **17** | **15** | **2** | **66,67** |
| Jobbra! | **16** | **15** | **1** | **73,33** |
| Balra! | **19** | **17** | **2** | **77,78** |
| Egyenesen! | **18** | **18** | **0** | **83,33** |
| Egyenesen és balra! | **11** | **11** | **0** | **63,64** |
| Egyenesen és jobbra! | **18** | **18** | **0** | **77,78** |

**Értékelés:** A program ideális körülmények között, ahol a tábla fókuszban van és jól elkülönül a háttértől, ott jól működik. Ha sötétebb kép van vagy messze van a tábla, nehezen találja meg a kört. Mindenesetre ez a program könnyen tovább bővíthető további táblákkal, ami nagy pozitívum**.**

## Felhasznált források:


1. https://www.programcreek.com/python/example/89394/cv2.kmeans
2. https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_ml/py_kmeans/py_kmeans_opencv/py_kmeans_opencv.html
3. https://stackoverflow.com/questions/30230592/loading-all-images-using-imread-from-a-given-folder
4. https://stackoverflow.com/questions/273946/how-do-i-resize-an-image-using-pil-and-maintain-its-aspect-ratio
5. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghcircles/py_houghcircles.html