Gépi látás

GKLB_INTM038

Rendszámtábla felismerő

Szalados Gábor

EP3IXC

Győr, 5. félév

Tartalomjegyzék

1 Megoldandó feladat bemutatása

A felvázolt témák közül a rendszámtábla felismerő programot választottam saját „projektnek”. A feladat, amit kitűztem, hogy egy elkészített digitális fotón képes legyen a program felismerni és kiírni egy jármű rendszámát.

Tehát a feladatom az volt, hogy beolvassak egy képet és valamilyen technikával előkészítsem arra, hogy megtaláljam rajta a keresett objektumot, ami jelen esetben a rendszámtábla.

A lokalizálás után ennek az objektumnak az éleit illetve a benne lévő egyelőre pixelek halmazát szöveggé alakítsam.

2 Elméleti háttér

A program kihasználom, hogy a képek leírhatók mátrixok segítségével és matematikai számítások végezhetők el rajta, így módosíthatunk, szűrhetünk az említett képeken.

Az első lépésben átalakítjuk a színes képet „fekete-fehérre”, amit szürke-árnyalat konverziónak nevezünk.

Következő lépésként bizonyos szűrőkkel javíthatjuk a képünk minőségét (Gauss szűrőt használtam)

Következő lépés az élek kiemelése és detektálása, amihez a Canny módszert használom.

A továbbiakban lokalizálom a rendszámot mint négyszög alakú objektum és ezt az objektumot plotolom egy új képként.

Az új képen használom a már korábban említett technikákat, mint szürke-árnyalat konverzió, Gauss szűrő, Canny élkiemelés és detektálás, így a képem készen áll, hogy fogadja a Tesseract engine.

Canny

Több lépcsős algoritmus.

Zajcsökkentés

Mivel az élfelismerés érzékeny a kép zajára, első lépésként 5x5 Gauss-szűrővel el kell távolítani a kép zaját.

A kép intenzitási gradiensének megkeresése

A kisimított képet ezután Sobel-kernel vízszintes és függőleges irányban is szűrjük, hogy az első deriváltat vízszintes (Gx) és függőleges (Gy) irányban kapjuk. Ebből a két képből az egyes pixelek élátmenetét és irányát a következőképpen találhatjuk meg.

$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2}$$

$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

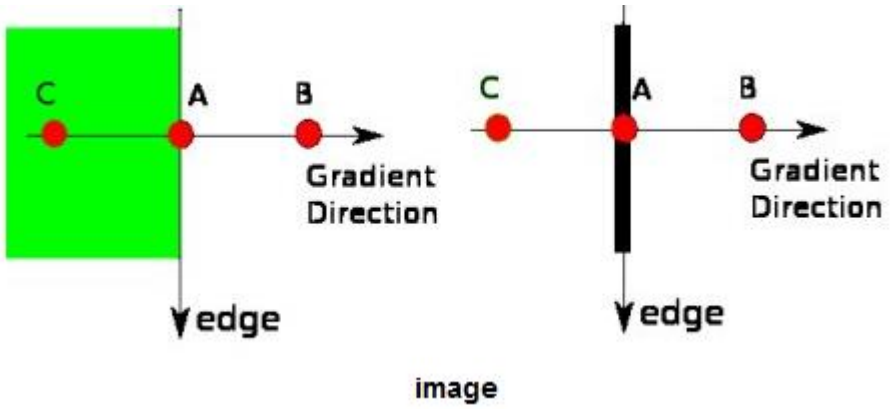
A szög iránya mindig merőleges az élekre. A függőleges, vízszintes és két átlós irányt képviselő négy szög egyikére van kerekítve.

Non-maximum Suppression

A gradiens nagyságának és irányának megadása után a kép teljes beolvasása megtörténik, a nem kívánt pixelek eltávolítására kerülnek, amelyek nem képezhetnek kontúrt. Ehhez minden pixelnél ellenőrizzük a pixelt, ha a szomszédságában a gradiens irányában lokális maximum van.

Az A pont a szélén van (függőleges irányban). A színátmenet iránya normális az élhez képest. A B és C pont gradiens irányban vannak. Tehát az A pontot a B és a C ponttal ellenőrizzük, hogy nem alkot-e helyi maximumot. Ha igen, akkor a következő szakaszra lépünk, ellenkező esetben elnyomják (nullára teszik).

Röviden, az eredmény egy bináris kép, "vékony élekkel".



Hysteresis Thresholding

Ez a szakasz eldönti, hogy melyik élek valóban élek és melyek nem. Ehhez két küszöbértékre van szükségünk, a minVal és a maxVal. Minden olyan él, amelynek intenzitási gradiense meghaladja a maxVal értéket, biztosan él, és a minVal alatti élek biztosan nem élek, ezért el kell dobni. Azok, akik e két küszöb között helyezkednek el, összekapcsolhatóságuk alapján osztályozott élek vagy nem élek. Ha "biztos élű" képpontokhoz vannak kapcsolva, akkor az élek részének tekintendők. Ellenkező esetben őket is eldobjuk.

Dilate

Ez a művelet az A kép összekeveréséből áll néhány kernellel (B), amelynek bármilyen alakja vagy mérete lehet, általában négyzet vagy kör.

A B magnak van egy meghatározott rögzítési pontja, amely általában a mag középpontja.

Amint a B kernelt a kép fölött beolvassa, kiszámoljuk a B-vel átfedésben lévő maximális pixelértéket, és a horgonypontban lévő képpontot ezzel a maximális értékre cseréljük. Amint arra következtethet, ez a maximalizáló művelet a képen belüli világos területek "növekedését" okozza.

A dilatációs művelet a következő:

dst(x, y) = max (x ′, y ′): elem (x ′, y ′) ≠ 0 src(x + x ′, y + y ′)

3 Kivitelezés

A kivitelezésben nagy szerepet játszik az OpenCV, illetve a Tesseract OCR engine (Optical Character Recognition). A két modul kifejezetten jól tud együtt működni, hiszen az OpenCV képes a preprocessing feladatok elvégzésére, aminek az eredménye jelen esetben egy „előkészített” kép amit a Tesseract egész jó arányban fel tud dolgozni az igényeink szerint. Mivel a Tesseract alapvetően egy szövegfelismerő engine, alkalmas lesz a rendszámtáblánk kiértékelésére és a megfelelő eredmények szolgáltatására. Természetesen nem minden esetben lesz kiváló kimenet, de ezt a teszt részben kifejtem bővebben, illetve az észrevételeimet és lehetőségeket is összefoglalom a későbbiekben.

3.1 Felhasznált python könyvtárak:

```
import cv2
import numpy as np
import imutils
import pytesseract as tess
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

1. sor: képfeldolgozáshoz használatos könyvtár, ami magában foglalja a szükséges függvényeket
2. sor: különböző matematikai műveletek elvégzéséhez, kernel előállításához (mátrix műveletek)
3. sor: a körvonalak, élek megkeresése
4. sor: karakterfelismerés és feldolgozás
5. sor: az eredmények kiíratása
6. sor: ezt a libraryt ahhoz használtam, hogy a plotnál az eredeti színében tudjam kimutatni a képeket

```
tess.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'
```

A Tesseract modul elérésének útvonala és deklarálása

3.2 Az előfeldolgozási fázis

```
kep=cv2.imread(r"C:\Python\40.jpg") #---<------------------------input

cv2.imwrite(r"C:\Python\seged.jpg", kep)
kepEredeti = mpimg.imread(r"C:\Python\seged.jpg")

kep = cv2.resize(kep, (600,400))
kepSzurke = cv2.cvtColor(kep, cv2.COLOR_BGR2GRAY)
kepSzurke = cv2.dilate(kepSzurke, kernel, iterations = 1)
kepSzurke = cv2.bilateralFilter(kepSzurke, 13, 15, 15)

eldetektalt = cv2.Canny(kepSzurke,30,200)
kontur = cv2.findContours(eldetektalt.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
kontur = imutils.grab_contours(kontur)
kontur = sorted(kontur, key = cv2.contourArea, reverse = True)[:10]
```

Deklaráltam egy kernelt, amit a dilate függvénnyel fogok használni. Erre azért van szükség, hogy a fehér részek területe kicsit bővüljön.

Beolvasom a képet, amiből csinálok egy segéd képet, hogy a végén a kiírásnál az eredetit szépen meg tudjam jeleníteni. Amit OpenCV-vel olvastamba azzal fogok dolgozni.

Átméretezem a képet, hogy csökkentsem a hibák lehetőségét.

Történik egy átalakítás színesről "fekete-fehérre", így sokkal több művelet lesz értelmezhető OpenCV-vel, mint színes képeken.

A dilate, amit már a kernelnél említettem.

BilateralFilter érdekes, mert enélkül volt olyan kép aminél az összekötő "-" helyett "+" olvasott le az algoritmus, így mint további képfeldolgozó bent hagytam.

Canny éldetektálás (min, max) paraméterek beállítása, ezzel is sokat teszteltem, hogy végül kijöjjön a 30,200. Megfelelő kompromisszumnak tűnik, még elég él marad meg, de nagyjából a zavarókat kiiktatja. Ezekkel az értékekkel lehet játszani, ha egy olyan kimenetet kapok, hogy nincs kontúr, ezen tresholdok kitolásával lehet még javítani egy egy kimenet eredményén.

Az élek detektálása után letárolom a kontúrokat amihez közelítést végez a program.
A lépések a teljesség igénye nélkül:

3.3 *Rendszám megkeresése a képen*

```
screenCnt = None

for k in kontur:

    peri = cv2.arcLength(k, True)
    kozelites = cv2.approxPolyDP(k, 0.018 * peri, True)

    if len(kozelites) == 4:
        screenCnt = kozelites
        break

if screenCnt is None:
    detected = 0
    print ("No contour detected")
else:
     detected = 1

if detected == 1:
    cv2.drawContours(kep, [screenCnt], -1, (0, 255, 255), 3)

mask = np.zeros(kepSzurke.shape,np.uint8)
keretezett_rendszam = cv2.drawContours(mask,[screenCnt],0,255,-1,)
keretezett_rendszam = cv2.bitwise_and(kep,kep,mask=mask)

(x, y) = np.where(mask == 255)
(topx, topy) = (np.min(x), np.min(y))
(bottomx, bottomy) = (np.max(x), np.max(y))
rendszam = kepSzurke[topx:bottomx+1, topy:bottomy+1]
```

A program végig iterál a kontúrokon és közelítést végez, és megpróbál sokszöget keresni a képen ami jelen esetben egy 4 szög, téglalap lenne. Elkezd futni a közelítés és ha a függvény görbére kezd hasonlítani akkor el is vethetjük, hisz ha minimális eltéréssel nem egyenes a vonal mentén halad akkor vélhetően nem sokszög forma.

Ha talál egy ilyen sokszöget akkor kirajzolja cián színnel, ha nem talál visszaad egy hibaüzenetet.

A bitwise_and javít a képre ráilleszteni a négyzetet.

Keretezett rendszám

Utána a program begyűjti az új képhez a koordinátákat, amiből legenerálódik egy új kép amin már csak a rendszám van. Ez itt látszik: (C:\Python\rendszam.jpg)

3.4 *A rendszám feldolgozása, karakterfelismerés*

```
rendszam = cv2.resize(rendszam,(500,200))

cv2.imwrite(r"C:\Python\rendszam.jpg", rendszam)

rendszam = cv2.imread(r"C:\Python\rendszam.jpg")
szurke_rendszam = cv2.cvtColor (rendszam, cv2.COLOR_BGR2GRAY)
szurke_rendszam = cv2.GaussianBlur(szurke_rendszam,(5,5),0)                    #a blur miatt pontosabb eredmenyt lehet kapni, az
igaz, hogy minel magasabb a blur annal pontosabb, de elveszhetnek fontos infok
binary_rendszam = cv2.threshold(szurke_rendszam, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]

kernel2 = cv2.getStructuringElement(cv2.MORPH_RECT, (3,3))
binary_rendszam = cv2.morphologyEx(binary_rendszam, cv2.MORPH_OPEN, kernel2, iterations=1)
binary_rendszam = 255 - binary_rendszam

text = tess.image_to_string(binary_rendszam, lang ='eng', config=' --psm 6 ')
```

Ismét átméretezem a kivágott rendszám képét, amit le is teszteltem és amikor kihagytam a lépést az output hibás volt.

Átméretezés, utána mentem egy mappába, hogy vissza tudja olvasni OpenCV-vel és a korábbihoz hasonlóan elvégezzük az előfeldolgozást. Annyi különbséggel, hogy itt már csak a rendszámra értelmezem. Végeredményként egy csodás bináris képet kapunk, amit Gauss szűrővel egy kis küszöböléssel és átalakításokkal érek el. Nagyon fontos, ugyanis a tesseract ilyen típusú képen a leghatékonyabb.

Beküldöm a bináris képet a tesseractba amit kiraktam a text változóba és ha minden összejött megvan a rendszám.

A tesseract paraméterezéséről kicsit, mivel ezzel is sokat kellett tesztelni, hogy mi lehet a megfelelő. Találtam egy nem hivatalos leírást arról, hogy milyen módszer alapján szeretnénk a karakterek szegmentálását végrehajtani.

0 Orientation and script detection (OSD) only.
1 Automatic page segmentation with OSD.
2 Automatic page segmentation, but no OSD, or OCR.
3 Fully automatic page segmentation, but no OSD. (Default)
4 Assume a single column of text of variable sizes.
5 Assume a single uniform block of vertically aligned text.
6 Assume a single uniform block of text.
7 Treat the image as a single text line.
8 Treat the image as a single word.
9 Treat the image as a single word in a circle.
10 Treat the image as a single character.
11 Sparse text. Find as much text as possible in no particular order.
12 Sparse text with OSD.
13 raw line. Treat the image as a single text line, bypassing hacks that are Tesseract-specific.

3.5 *Kimenetek kiíratása*

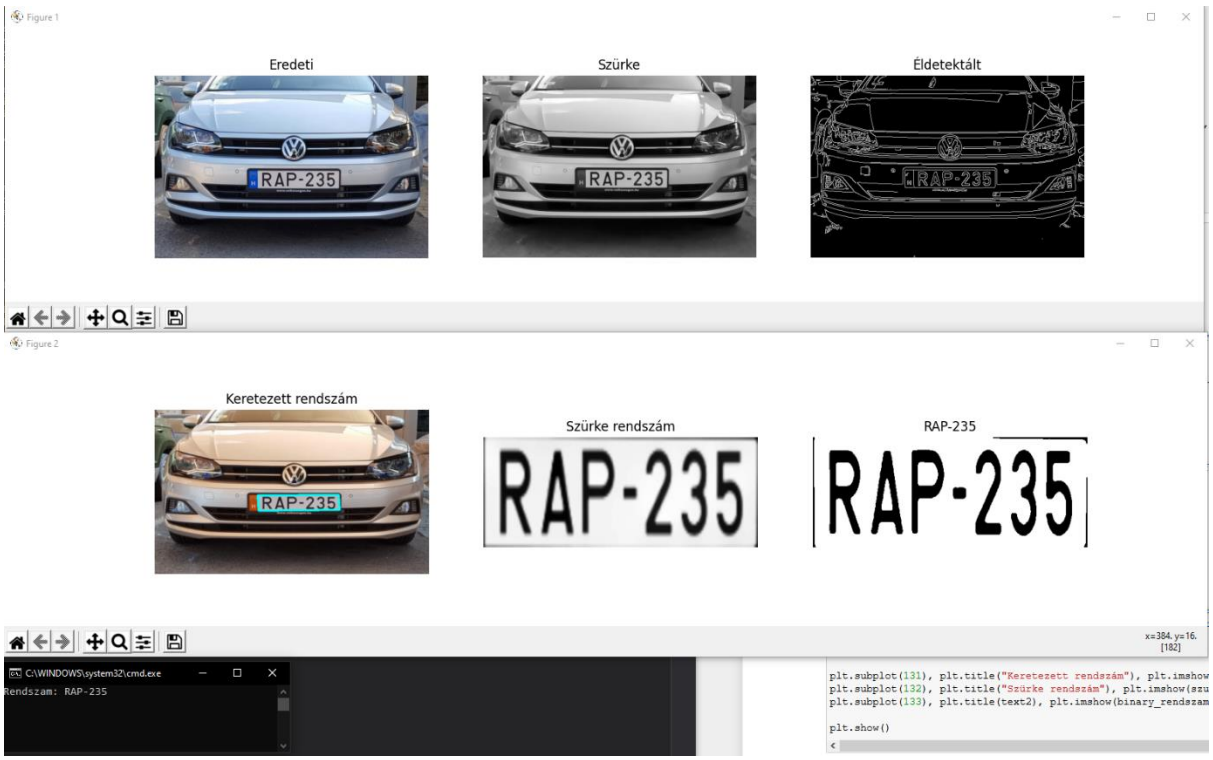
```
utolsoChar = len(text)
text2 = ""
for x in range (utolsoChar-2):
    text2 = text2 + text[x]

print("Rendszam:",text2)

plt.figure(1, figsize=(20, 8))

plt.subplot(131), plt.title("Eredeti"), plt.imshow(kepEredeti), plt.axis("off")
plt.subplot(132), plt.title("Szürke"), plt.imshow(kepSzurke, cmap="gray"), plt.axis("off")
plt.subplot(133), plt.title("Éldetektált"), plt.imshow(eldetektalt, cmap="gray"), plt.axis("off")

plt.figure(2, figsize=(16, 6))

plt.subplot(131), plt.title("Keretezett rendszám"), plt.imshow(kep), plt.axis("off")
plt.subplot(132), plt.title("Szürke rendszám"), plt.imshow(szurke_rendszam, cmap="gray"), plt.axis("off")
plt.subplot(133), plt.title(text2), plt.imshow(binary_rendszam, cmap="gray"), plt.axis("off")

plt.show()
```

A tesseract az eredmények végére hozzárak egy szóközt, illetve egy karaktert, amit a for ciklussal leszedek, hogy a kimenet szebb legyen.

A továbbiakban pedig a plotolások kódjai láthatók, ebben nem sok varázslat van.

3.6 *A program kimenete*

Megpróbáltam nagyjából az összes fontos lépést képek formájában is kiemelni, hogy átlátható legyen az elvégzett munka.
Az utolsó képen illetve a consoleba is kiíratom a rendszámot.



4 Tesztelés

A tesztelést 40 db képen végeztem.

A teszt1 a repoban az 1-es képnek felel meg és így tovább.

Szigorúan véve a 40 kép 42,5 %-a volt OK, illetve 57,5 %-a NOK, de ha megvizsgáljuk a jobb oldali oszlopot, ahol kommenteket fűztem az eredményhez látszik, hogy több olyan eset is van, ahol plusz karakter vagy tévesztésről beszélünk.

A plusz karakterek általában abból adódtak, hogy mivel eltérő képminőséggel dolgoztam így nem lehet minden esetet ugyanolyan szűrővel lefedni, úgy gondolom, hogy ha minden képre speciálisan állítom be a cannyt, dilate és egyéb filtereket akkor lehet tökéletes eredmény.

Természetesen ez nem életszerű, így valószínűleg jóval több munkába kerülne, hogy az OK rátát növelni tudjam.

Az eredményeket egy excelben tároltam íme:

teszt1	ok		
teszt2	ok		
teszt3	nok	plusz egy szőköz	
teszt4	nok	hozzátesz egy I betűt, ez a rendszám forgatásával megoldható	
teszt5	nok	egy betű hiányzik	
teszt6	ok		
teszt7	nok	ha nagyon szögletes objektum van a képen, tipikusan téglalap vagy határozottan 4 pont határolja	
teszt8	nok	újra a rendszám forgatása megoldhatja, de amúgy csak egy karaktert tesz hozzá a végére, szóval kijön a rendszám	
teszt9	ok		
teszt10	ok		
teszt11	nok	több szűrő kell	
teszt12	nok	néhány karaktert nem megfelelően imser fel	
teszt13	nok	rendszámot felismeri, a karakterek detektálása problémás	
teszt14	ok		
teszt15	ok		
teszt16	nok	plusz karakterek	
teszt17	nok	plusz karakterek	
teszt18	ok		
teszt19	nok	plusz karakterek	
teszt20	ok		
teszt21	nok	plusz karakter	
teszt22	nok	plusz karakterek, rossz kontúr	
teszt23	ok		
teszt24	nok	karaktertévesztés	
teszt25	nok	karaktertévesztés	
teszt26	nok	plusz karakter	
teszt27	ok		
teszt28	ok		
teszt29	nok	karaktertévesztés	
teszt30	nok	rendszám megvan, karakter nem jó	
teszt31	ok		
teszt32	ok		
teszt33	ok		
teszt34	ok		
teszt35	nok	plusz karakter	
teszt36	nok	rendszám ok, karakter nok	
teszt37	nok	rendszám ok, karaktertévesztés	
teszt38	nok	rendszám ok, karaktertévesztés	
teszt39	nok	rendszám ok, karaktertévesztés	
teszt40	ok		
OK:	17	42,50%	
NOK:	23	57,50%	

Következtetés

Jellemző hibák amikkel a tesztelés során találkoztam.

1. A bemeneti képen nem talált éleket
2. Más alakzatot titulált rendszámnak, ami sajnos nem az volt
3. A rendszámhoz esetleg plusz karaktert rakott vagy 1-1 karakter lemaradt

Az eredmények alapján több következtetést is levontam, amit a következőkben szeretnék kifejteni.

1. Amit mindenképp fontos kiemelni, hogy a képek minősége, háttere, beállítása teljesen eltérő.

   Ebből következik, hogy ha a program konstans módon ugyanolyan képeket kapna bemenetként - *gondolok itt egy fixre telepített kamerára ami egy sorompónál azt a célt szolgálja, hogy mozgásérzékelő jelére egy képet készít a járművekről amit azután kiértékel és ha az engedély jelen van akkor engedje át a sorompón a járművet* - akkor a preprocessing műveleteket sokkal jobban speciálisra lehetne szabni.

   Gondolok itt a betekintési szögre, nagyjából fix szögről és távolságról beszélünk, nem beszélve az elhelyezett kamera magasságáról, amivel további segítséget adhatunk a program kedvező kimenetelére, hiszen ha a kamera bemenetként kevesebb egyéb számunkra haszontalan információt rögzít, akkor a kimenetet is nagyobb valószínűséggel ad helyes kimenetet.

   Pl. volt egy kép, ami egy rendőr autót ábrázolt és a program nem a táblát vette alapul, hanem a rendőrség feliratot. Ez kiszűrhető, ha a kamera megfelelő magasságban van elhelyezve, ami készíti számunkra a bemeneti képet.

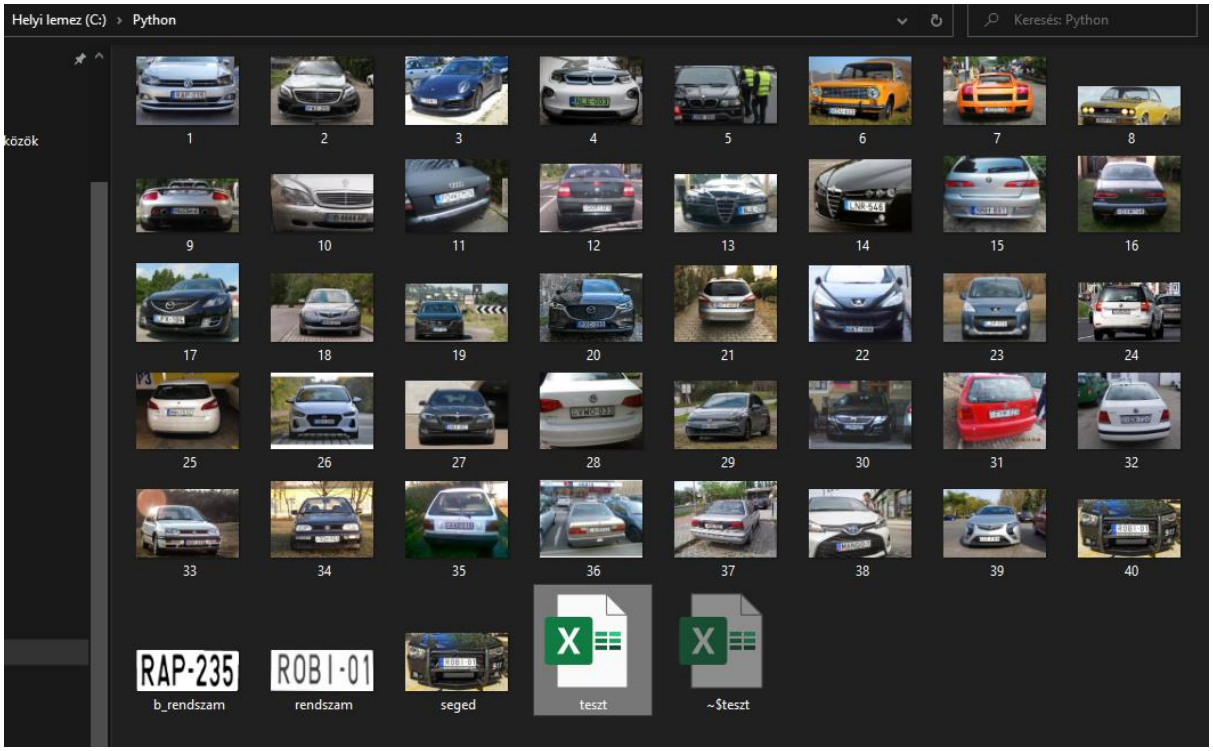
   2. A rendszám megtalálását a függvény úgy oldja meg, hogy a képen 4 pontot keres, ami nagyon hasonlít egy téglalapra, sajnos volt olyan amikor más pontokat vélt a rendszám sarkainak így félrevezette az algoritmust.

   3. A plusz karaktereket korábban kifejtettem.

5 Felhasználói leírás

A program működtetéséhez Visual Studiot használok. Illetve telepíteni kell a számítógépre a Tesseractot.

Szükség van egy könyvtárra amiben a programnak van jogosultsága írni, módosítani. Az egyszerűség kedvéért itt tárolom a bemeneti képeket is.



A program futásához a pirossal bekeretezett sorban át kell írni a kép nevét és lefuttatni a programot.

```
1   import cv2
2   import numpy as np
3   import imutils
4   import pytesseract as tess
5   import matplotlib.pyplot as plt
6   import matplotlib.image as mpimg
7
8   tess.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'
9
10  kernel = np.ones((1,1),np.uint8)
11
12  kep=cv2.imread(r"C:\Python\40.jpg") #---<-----------------------input
13
14  cv2.imwrite(r"C:\Python\seged.jpg", kep)
15  kepEredeti = mpimg.imread(r"C:\Python\seged.jpg")
16
17  kep = cv2.resize(kep, (600,400))
18  kepSzurke = cv2.cvtColor(kep, cv2.COLOR_BGR2GRAY)
19  kepSzurke = cv2.dilate(kepSzurke, kernel, iterations = 1)
20  kepSzurke = cv2.bilateralFilter(kepSzurke, 13, 15, 15)
```

A korábban említett mappában részeredményeket fog tárolni a program, amik szükségesek a végeredmény eléréséhez. Ezeket a fájlokat minden alkalommal legenerálja a program, így nem okoz gondot, ha a futás után kitöröljük.

A kimenet ugyanide történik, de természetesen minden elérési utat meg lehet változtatni kedvünk szerint.

Ahogy a kimeneteknél látszik, 6 kép fog kiíratásra kerülni a végeredményként, ahol az utolsó kép neve maga a rendszám, feltéve, ha megfelelően megtalálta a program, illetve a consoleban is kiírásra kerül az eredmény.

És zárszóként egy újabb sikeres futás: