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 a képen 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 megírásakor kihasználtam, hogy a képek leírhatók mátrixok segítségével és matematikai számítások végezhetők el rajta, így módosíthatunk, szűrhetünk az említett képeken.

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

Következő lépésként bizonyos szűrőkkel javítok a kép minőségén (BillateralFilter, 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.

$$Edge\_Gradient\ (G) = \sqrt{G^2x + G^2y}$$

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

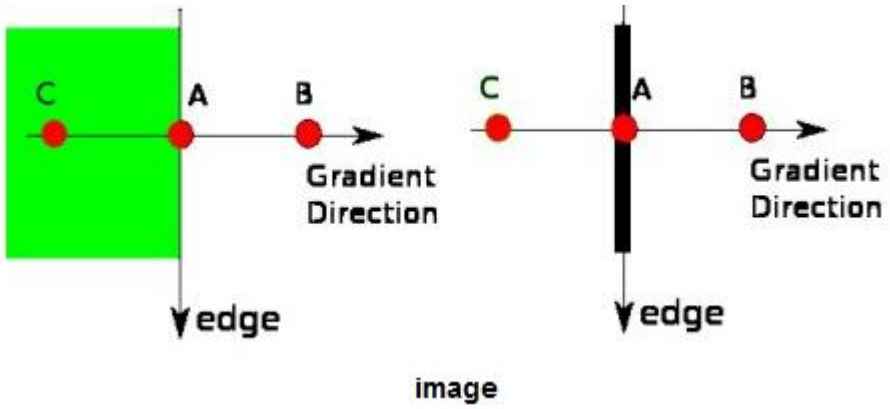
A szög iránya mindig merőleges az élekre. A függőleges, vízszintes és két átlós irányt képviselő négy szög egyikére van kerekítve.

Non-maximum Suppression

A gradiens nagyságának és irányának megadása után a kép teljes beolvasása megtörténik, a nem kívánt pixelek eltávolításra kerülnek, amelyek nem képezhetnek kontúrt.

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".



1. ábra: éldetektálás

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. [1]

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. [2]

A dilatációs művelet a következő.

$$dst(x,y) = max\,(x',y'): elem\,(x',y') \neq 0\, src(x + x', y + y')$$

3 Kivitelezés

A kivitelezésben nagy szerepet játszik az OpenCV, illetve a Tesseract OCR engine (Optical Character Recognition). Az OpenCV képes a preprocessing feladatok elvégzésére, aminek az eredménye jelen esetben egy „előkészített" kép, amit a Tesseract feldolgoz a paraméter beállításai szerint. Mivel a Tesseract alapvetően egy szövegfelismerő engine, alkalmas lesz a rendszámtábla kiértékelésére és a megfelelő eredmények szolgáltatására. 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

1. Kódrészlet: külső 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

2. Kódrészlet: Windows fájlkezelő

```
import tkinter as tk
from tkinter import filedialog
root = tk.Tk()
root.withdraw()
file_path = filedialog.askopenfilename()
tess.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'
```

Ebben a szekcióban a Windows fájlkezeléshez szükséges kódsorok találhatók, illetve a Tesseract modul elérésnek útvonala, ami azért szükséges, hogy a program megtalálja az enginet a futtatáshoz.

3.2 Az előfeldolgozási fázis

3. Kódrészlet: előfeldolgozás

```
kernel = np.ones((1,1),np.uint8)
kep=cv2.imread(file_path)
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 függvényhez használok. Erre azért van szükség, hogy a világos részek területe bővüljön. Beolvasom a képet, amiből generálok egy segéd képet, hogy a végén a kiírásnál az eredetit tudjam megjeleníteni. Átméretezem a képet, hogy csökkentsem a hibák lehetőségét. Történik egy átalakítás színesről "fekete-fehérre", így megnövekedik azon műveletek száma, amely értelmezhető OpenCV-vel a képeken. BilateralFilter segítségével zajcsökkentés történik a képen. Canny éldetektálás (min, max) paraméterek beállítása. Minimumnak 30 maximumnak 200-as értéket adtam meg, ami megfelelő kompromisszumnak tűnik. A tesztek alapján megfelelő számú él marad meg, hogy a további feladatok végrehajthatók legyenek, de mégis kiszűri a zavaró élek nagy részét, ami a végső kimenetre negatív hatással lehetne. Ezen értékek módosításával finomhangolható a függvény, ha egy olyan kimenetet kapok, ahol a függvény nem talált kontúrt, akkor az említett (min, max) limitek kitolásával lehet még javítani egy-egy kimeneti eredményen.
Az élek detektálása után letárolom a kontúrokat.

A lépések a teljesség igénye nélkül.

2.ábra: az előfeldolgozás eredménye

3.3 Rendszám megkeresése a képen

4. Kódrészlet: téglalap alak keresése 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,)
(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, megvizsgálja az adott él hosszát és a hossz segítségével pixelről pixelre végigmegy a kontúron, és megpróbál sokszöget keresni a képen, ami jelen esetben egy négyszög, pontosabban egy téglalap lenne. A kontúrok pixelein történő közelítés megkezdődik, és ha a függvény görbére kezd hasonlítani, akkor el is vethetjük, hisz ha minimális eltéréssel nem egyenes vonal mentén halad, akkor vélhetően nem téglalap alak.
Ha talál egy ilyen téglalapot akkor kirajzolja cián színnel, ha nem talál visszaad egy hibaüzenetet. (No contour detected) A bitwise_and javít a képre ráilleszteni a négyzetet.

3. ábra: lokalizált rendszám

Ezek után a program kiszámolja a rendszáma koordinátáit, amiből legenerálódik egy új kép amin már csak a rendszám van. A következő látható. (C:\Python\rendszam.jpg)

4. ábra: rendszám létrejötte a fájlrendszerben

3.4 A rendszám feldolgozása, karakterfelismerés

5. Kódrészlet: 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)
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 ')
```

5. ábra: rendszám előfeldolgozás előtt és után

Ismét átméretezem a kivágott rendszám képét. Ahogy korábban említettem, az eljárással csökkenthető a hibafaktor, így elengedhetetlen az alkalmazása.
Átméretezés, utána lementem egy mappába, hogy vissza tudja olvasni a program OpenCV segítségével és a korábbihoz hasonlóan elvégzem az előfeldolgozást. Végeredményként egy bináris képet kapok, amit Gauss szűrővel, küszöböléssel és átalakításokkal érek el. [3] A Tesseract megfelelően előkészített képen jobb eredményt tud elérni, így kulcsfontosságú ez a fázis. Beolvasom a bináris képet a Tesseractba aminek az eredményét később a text változóban tárolom.

A tesseract paramétereiről. Alább egy lista arról, hogy milyen módszer alapján van lehetőség a karakterek szegmentálását végrehajtani. [4]

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*

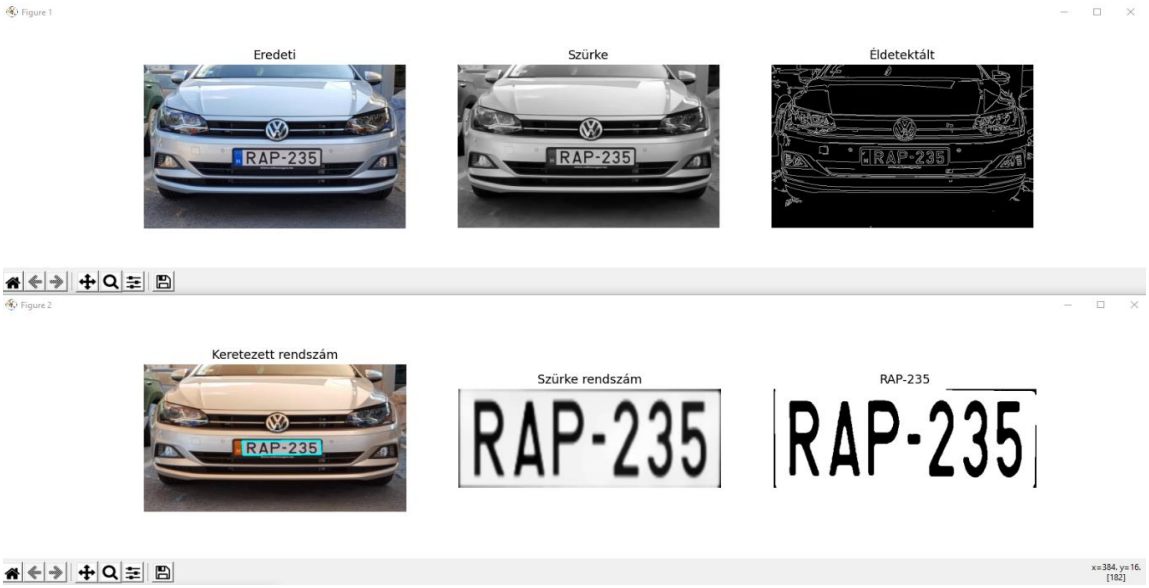
6. Kódrészlet: kimenetek kiíratása

```
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 eltüntetek, hogy a kimenet a lehető legpontosabb legyen. A továbbiakban pedig a plotolások kódjai láthatók. Mivel két külön ábrán jelenítem meg a képeket így kétszer három darab subplotot használok fix méretekkel.

3.6 *A program kimenete*

A program kimenetére egy példa. Az utolsó kép címe maga a rendszám, amit kinyertem a képről.



6. ábra: végeredmény

4 Tesztelés

A tesztelést 40 db képen végeztem. A teszt1 a repoban a „Teszt_kepek” alatt az 1-es képnek felel meg és így tovább.
Szigorúan véve a 40 db 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 hozzáfűzése vagy karaktertévesztés történt.

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, ezen problémák megoldásához kutatásra és még több munkára lenne szükségem, hogy az OK rátát növelni tudjam. Az eredményeket egy excel táblában tároltam.

1. táblázat: Teszteredmények

Teszt azonosító	Tesztelt kép neve a repoban	Minősítés	Komment
teszt1	1.jpg	ok	
teszt2	2.jpg	ok	
teszt3	3.jpg	nok	plusz egy szóköz
teszt4	4.jpg	nok	hozzátesz egy I betűt, ez a rendszám forgatásával megoldható
teszt5	5.jpg	nok	egy betű hiányzik
teszt6	6.jpg	ok	
teszt7	7.jpg	nok	ha egyéb négyszög forma van a képen
teszt8	8.jpg	nok	újra a rendszám forgatása megoldhatja
teszt9	9.jpg	ok	
teszt10	10.jpg	ok	
teszt11	11.jpg	nok	több szűrő kell
teszt12	12.jpg	nok	néhány karaktert nem megfelelően ismer fel
teszt13	13.jpg	nok	rendszámot felismeri, a karakterek detektálása problémás
teszt14	14.jpg	ok	
teszt15	15.jpg	ok	
teszt16	16.jpg	nok	plusz karakterek
teszt17	17.jpg	nok	plusz karakterek
teszt18	18.jpg	ok	
teszt19	19.jpg	nok	plusz karakterek
teszt20	20.jpg	ok	
teszt21	21.jpg	nok	plusz karakter
teszt22	22.jpg	nok	plusz karakterek, rossz kontúr
teszt23	23.jpg	ok	
teszt24	24.jpg	nok	karaktertévesztés
teszt25	25.jpg	nok	karaktertévesztés
teszt26	26.jpg	nok	plusz karakter

teszt27	27.jpg	ok	
teszt28	28.jpg	ok	
teszt29	29.jpg	nok	karaktertévesztés
teszt30	30.jpg	nok	rendszám megvan, karakter nem jó
teszt31	31.jpg	ok	
teszt32	32.jpg	ok	
teszt33	33.jpg	ok	
teszt34	34.jpg	ok	
teszt35	35.jpg	nok	plusz karakter
teszt36	36.jpg	nok	rendszám ok, karakter nok
teszt37	37.jpg	nok	rendszám ok, karaktertévesztés
teszt38	38.jpg	nok	rendszám ok, karaktertévesztés
teszt39	39.jpg	nok	rendszám ok, karaktertévesztés
teszt40	40.jpg	ok	
OK:	17	42,50%	
NOK:	23	57,50%	

5 Következtetés

Jellemző hibák amikkel a tesztelés során találkoztam.

1. A bemeneti képen nem talált éleket
2. Más alakzatot titulált rendszámnak, tévesen
3. A rendszámhoz plusz karaktert fűzött vagy 1-1 karakter lemaradt

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

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 az előfeldolgozó műveleteket sokkal speciálisabbra lehetne szabni. Korábban említettem a canny detektálás limitjeit, finomhangolással javítható lenne az olvasási eredményesség.

Az elhelyezett kamera betekintési szögének, távolságának és magasságának fixálása további segítség lehetne a program kedvező kimenetelére, hiszen ha a kamera bemenetként kevesebb, egyéb, számunkra haszontalan információt rögzít, akkor a kimenetet is nagyobb valószínűséggel lesz helyes.

A rendszám lokalizálását a függvény, a kontúr pixeleinek egymáshoz való közelítésével vizsgálja és a cél az, hogy a képen négy pontot találjon, ami egy téglalapot határoz meg. A tesztek során több esetben is problémába ütköztem azoknál a képeknél, ahol a rendszámon kívül megtalálható volt egyéb téglalap alakzatra hasonlító forma.
A kutatás során találtam olyan megoldást, ahol a függvény előre definiált formát keresett a képen (másik kép tartalmazza a formát). Ígéretes megoldásnak tűnik, esetleg továbbfejleszthető a program ebbe az irányba. [5]

A plusz karakterek abból adódhatnak, hogy túl sok zaj maradt a képen. Ezeket a zajokat további függvények használatával csökkenteni lehet.

6 Felhasználói leírás

A program jelen verziója csak Windows operációs rendszerrel kompatibilis a kódban használt fájl elérési utak miatt.

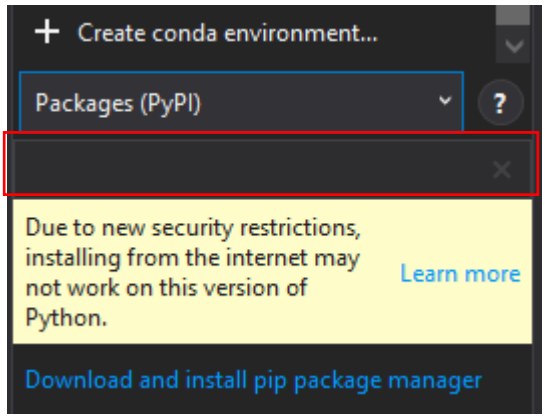
6.1 Szükséges eszközök

1. Bármilyen Python fordító, ahol lehetőség van külső könyvtárak telepítésére
2. Új Python projekt létrehozása és forráskód bemásolása
3. Külső könyvtárak telepítése
4. Tesseract telepítése
5. C:\Python mappa létrehozása, ahova van írási, módosítási joga a programnak

A program fejlesztését Visual Studioban - továbbiakban VS - végeztem, így a következőkben ezzel a fejlesztő környezettel foglalkozok, ebben tudok konkrét tapasztaltokról beszámolni. VS-ben kreálni kell egy új Python projectet és bemásolni a forráskódot.

6.2 Külső könyvtárak telepítése

VS-ben a 7. ábrán pirossal keretezett részben kell megkeresni a könyvtárakat és telepíteni. - Megvizsgáltam egy másik platformot, ami a www.repl.it , ahol a könyvtárak telepítése abban kimerül, hogy beírjuk (pl. „import cv2”), ebben az esetben automatikusan telepítésre kerül a könyvtár. – Viszont a VS-ben külön telepíteni kell az alábbi panel segítségével.



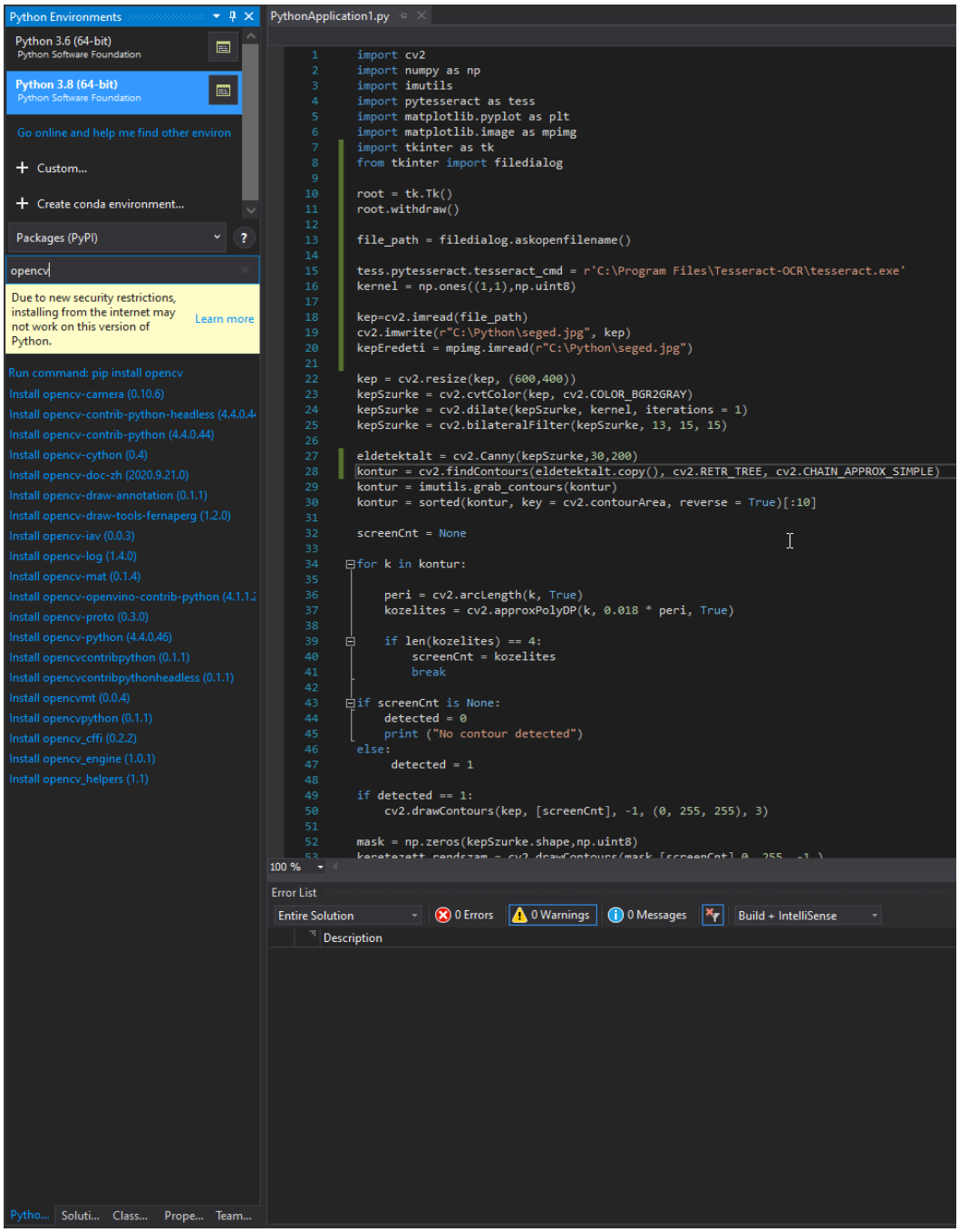
7. ábra: külső könyvtárak telepítése

Elsőként az „install pip package manager” telepítése szükséges, ezek után a többi könyvtár telepítése egyesével. A fenti ábra helye a 8. képről beazonosítható.

A következő módon találhatók meg a telepítendő könyvtárak.

- opencv-python
- matplotlib
- numpy
- imutils
- pytesseract

A következő ábra azért került bele a leírásba, hogy könnyebb legyen a tájékozódás a felületen. A 7. ábra egy kiemelt kép a kezelőfelületből.



8. ábra: telepítő felület a VS-ben

6.3 *Tesseract telepítése*

Az alábbi linken található a Tesseract OCR telepítője.

https://github.com/UB-Mannheim/tesseract/wiki

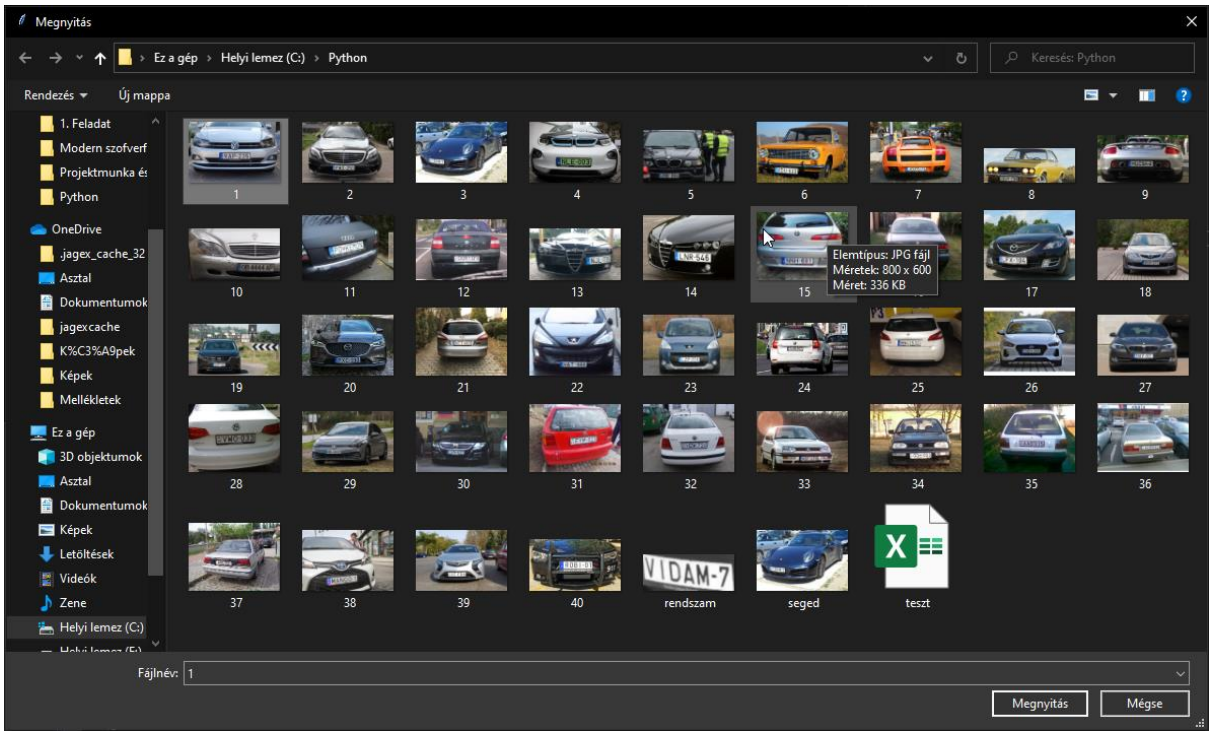
Használt verzió: tesseract-ocr-w64-setup-v5.0.0-alpha.20200328.

6.4 *Mappa létrehozása*

C:\Python mappa létrehozása, ahova van írási, módosítási joga a programnak.
Ebben a mappában segéd fájlok is generálódnak. Ezeket a fájlokat minden alkalommal legenerálja a program, így nem okoz gondot, ha futás után kitöröljük.
A kimenet ugyanide történik, de minden elérési utat meg lehet változtatni a forráskódban.

6.5 *Program futtatása*

Ctrl + F5-tel futtatható az állomány. Felugrik a Windows fájlkezelő, ahol ki kell választani a képet, amin szeretnénk lefuttatni a programot.



9. ábra: Input kép megadása

Megnyitás után lefut a program és megkapjuk a korábban bemutatott kimenetet.

7 Felhasznált irodalom

[1] https://docs.opencv.org/master/da/d22/tutorial_py_canny.html
[2] https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html
[3] https://docs.opencv.org/master/d4/d13/tutorial_py_filtering.html
[4] https://nanonets.com/blog/ocr-with-tesseract/
[5] https://www.pyimagesearch.com/2014/10/20/finding-shapes-images-using-python-opencv/