SZÉCHENYI EGYETEM
UNIVERSITY OF GYŐR

INFORMATIKA TANSZÉK
DEPARTMENT OF INFORMATICS

Gépi Látás Projektmunka

Rendszámtábla felismerő rendszer

Győri Bence - HZWHYM

2022

Tartalomjegyzék

Ábrajegyzék

Bevezetés

Napjainkban a használatban lévő gépjárművek száma megtöbbszörözte önmagát az elmúlt évekhez képest. Az egymástól való megkülönböztetésük érdekében olyan egyedi azonosítókat alkalmaznak, mint pl. a rendszámtáblák. Minden forgalomban lévő személygépjármű rendelkezik egy egyedi, olyan alfanumerikus sorozattal, melyet a köznyelv rendszámként kezel. A rendszámok leolvasásának legegyszerűbb és leghatékonyabb módja a járművek azonosítása, így az elmúlt években az erre a célra alkalmas rendszerek száma megnövekedett. Napjainkban számos helyen alkalmaznak ilyen szoftvereket olyan különböző célokra mint pl. parkolórendszerek, autópálya matrica ellenőrző kapuk, de a rendőrség által alkalmazott sebességmérő berendezések is a rendszám alapján azonosítják az esetleges gyorshajtókat.

Amikor a rendszámfelismerés témaköre felvetődik, fontos tisztázni, hogy a rendszámtábla felismerése és leolvasása eltér egymástól. Míg az előbbinél elég csak magát a rendszámtáblát detektálni, addig az utóbbinál ezen felül le is kell olvasni annak tartalmát. A karakterek felismerésekor fontos, hogy a rendszer pontossága magas mértékű legyen. A karakterek leolvasása olyan számítógépes látástechnika, mely során a járművek rendszáma kinyerhető egy digitális képből.

A detektálás módszertana az ANPR (Automatic Number Plate Recogniton). Az ANPR-t 1976-ban találták fel a rendőrség tudományos fejlesztési részlegénél Nagy-Britanniában. 1979-ben már működtek a prototípus rendszerek, és szerződést kötöttek ipari rendszerek gyártására, először az EMI Electronics, majd a Computer Recognition Systems (CRS, ma a Jenoptik része) vállalatnál, Wokinghamban, az Egyesült Királyságban. A korai kísérleti rendszereket az A1-es autópályán és a Dartford-alagútban vetették be. Az első letartóztatás egy lopott autó felismerése révén 1981-ben történt. Az ANPR azonban nem terjedt el széles körben, amíg az 1990-es években az olcsóbb és könnyebben használható szoftverek új fejlesztései úttörő szerepet nem játszottak. Az ANPR-adatok gyűjtését későbbi felhasználásra (azaz az akkor még ismeretlen bűncselekmények felderítésére) a 2000-es évek elején dokumentálták. 2005 novemberében történt az első dokumentált eset, amikor az ANPR-t gyilkosság felderítésére használták, az Egyesült Királyságban, Bradfordban, ahol az ANPR fontos szerepet játszott Sharon Beshenivsky gyilkosainak felkutatásában és későbbi elítélésében. Az ANPR-nek 2005 novemberében volt az első olyan dokumentált esete, amikor egy gyilkosság felderítésében segített. [1]

A rendszámfelismerés elméleti háttere

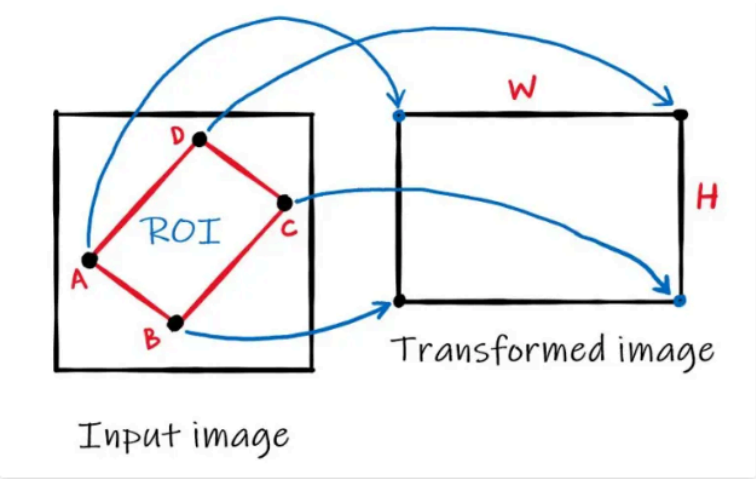
A felismerés folyamata általában két lépésből áll: az első, hogy a beolvasott képen először detektálni kell az azonosítót tartalmazó táblát, majd ezután a következik a karakterfelismerés, mely során a program leolvassa a képen látható alfanumerikus karaktereket.

A rendszámfelismerés során a rendszám helyzetének meghatározása és normalizálása a legnehezebb feladat. A nehézséget a táblatípusok méretbeli különbségei is befolyásolják, ezen felül formájukban és színükben sem egységesek. A nehézség kiküszöbölésére a Canny Edge Detector algoritmus megfelelő megoldás. A projektmunka is ezt az algoritmust használja, ez a későbbiekben illusztrálásra kerül. Az éldetektor főbb lépései:

- Simítás Gauss függvénnyel
    - Nagyobb szigma esetén nagyobb méretű éleket detektálhatunk
    - Kisebb szigma a finomabb részleteket is detektálja
- Gradiens operátor (Sobel)
    - Választható környezetmérettel
- Nem-maximális élek elnyomása
    - Egy lokális környezetben csak a legnagyobb gradiens magnitúdó érték marad meg
    - Vékony él-kezdemények maradnak
- Hiszterézis küszöbölés (alsó és felső küszöbértékkel)
    - Ha nagyobb az él erőssége, mint a felső küszöbérték, akkor megtartjuk
    - Ha kisebb az él erőssége, mint az alsó küszöbérték, akkor elvetjük
    - Kettő közötti értékek esetén akkor tartjuk meg él pontnak, ha van olyan szomszédja, ami él
    - Javasolt arány a küszöbökértékekre 2:1 és 3:1 között [2]

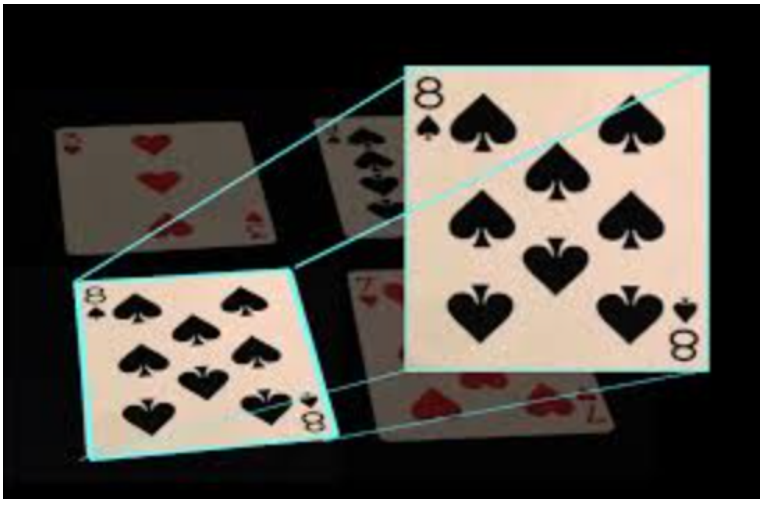
A rossz szögben befotózott rendszám sem könnyíti meg a feladatot, ennek megoldásaképp különféle algoritmusokat kell alkalmazni. A projektmunka során az OpenCV „Perspective Warping” funkciója került alkalmazásra. Az OpenCV (Open source computer vision) egy programozási funkciókat tartalmazó könyvtár, amely elsősorban a valós idejű számítógépes látást szolgálja. Az OpenCV pythonban segít egy kép feldolgozásában és különböző funkciók alkalmazásában, mint például a kép átméretezése, pixel manipulációk, objektum detektálás stb. A Perspective Warp algoritmus segítségével megváltoztathatjuk egy adott kép vagy videó perspektíváját, hogy jobb betekintést nyerjünk a kívánt fényképbe. A perspektíva-átalakítás

során meg kell adnunk a képen azokat a pontokat, amelyekből a perspektíva megváltoztatásával információt szeretnénk gyűjteni. Meg kell adnunk azokat a pontokat is, amelyeken belül meg akarjuk jeleníteni a képet. Ezután megkapjuk a transzformációt a két megadott pontkészletből, és összehasonlítjuk az eredeti képpel.



1. ábra Algoritmus működése (Forrás: https://theailearner.com/tag/cv2-warpperspective/)

Az előre megadott, vagy automatikusan kiválasztott sarokpontok alapján az algoritmus kiszámolja a transzformációs mátrixot és úgy alakítja át a rossz szögben lévő fotót, hogy az a későbbiekben alkalmas legyen arra, hogy az esetleges karaktereket helyesen tudja leolvasni a program.



2. ábra Az algoritmus működése (Forrás: https://answers.opencv.org/question/181902/warpperspective-advice-with-correct-bbox-pixels/)

A karakterek leolvasása az OpenCV OCR funkciójával történik. Az EasyOCR, ahogy a neve is mutatja, egy Python-csomag, amely lehetővé teszi a számítógépes látás fejlesztőinek, hogy

könnyedén végezzenek optikai karakterfelismerést. Ez a bővítmény 58 különböző nyelven képes karaktereket keresni, így az előre betöltött képeket gyorsan és pontosan fogja elemezni.

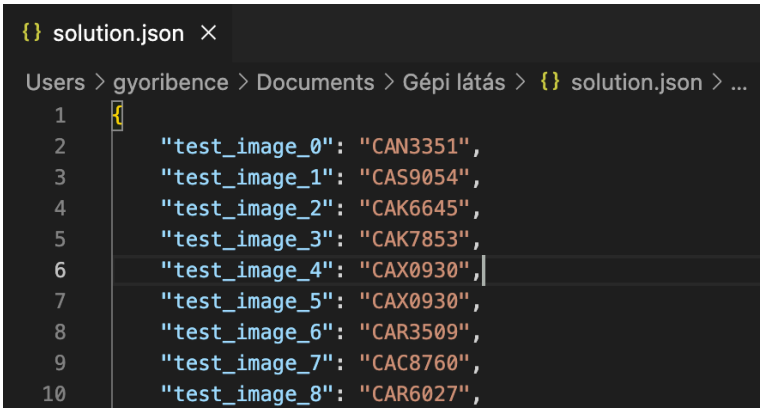
3. ábra OCR működése (Forrás: https://pyimagesearch.com/2020/09/14/getting-started-with-easyocr-for-optical-character-recognition/)

A program működése

A rendszámfelismerés 100 db., előre megadott képen történik. Ezek a fotók internetről letöltött, kifejezetten gépi tanulásra készített tartalmak. A képek tartalmaznak olyan gépjárműveket, melyek oldalról kerültek lefotózásra, illetve különböző országok rendszámtáblái jelennek meg.

4. ábra Későbbiekben felhasznált kép

A későbbi tesztelések sikerességének mérése érdekében a képekről először manuálisan kell leolvasni a rendszámot, hogy azok egy külön file-ban kerüljenek mentésre. Ez egy „solotuion.json” file lesz, amit az algoritmus a tesztelés fázisába lépve fog használni. Ennek segítségével a program képes kiszámolni, hogy hány %-os egyezéssel olvasta le a karaktereket, illetve hány olyan eset volt, amely során nem talált rendszámtáblát. A végeredmény, ahogy az a későbbiekben bemutatásra kerül, tartalmazni fogja azt is, hogy hány rendszámtáblát ellenőrzött a program, illetve azt is, hogy azok közül hány esetben nem talált rendszámot.



5. ábra Manuálisan leolvasott rendszámok (Forrás: saját kép)

A program első lépésként beolvassa a képet és mivel a mai kamerákkal készített képek elég részletesek, ezért érdemes a számítógép számára ezeket lecsökkenteni, hiszen amúgy is csak számok ezreit látja belőle. A választott méret 600x600 pixel így a kód is lényegesen gyorsabb ekkora méreten.

```
loaded_image = cv2.imread(filename)
resized_image = cv2.resize(loaded_image, (600, 600),
interpolation=cv2.INTER_LINEAR)
```

Szürkeárnyalatossá alakítjuk a képet, így a számítógép is könnyebben kezeli. Hagyományos képen 3 színcsatorna van (R - red, G - green, B - blue), ezekből tevődik össze a színe az adott pixelnek. A számítógépnek erre nincs szüksége, ezért szürkeárnyalatossá alakítjuk és ilyenkor csak egy színcsatorna van, ami az intenzitás. Ez jelenti, hogy "mennyire" fehér vagy "mennyire" fekete egy adott pixel.

```
gray_image = cv2.cvtColor(resized_image, cv2.COLOR_BGR2GRAY)
```

6. ábra Szürkeárnyalatos, méretezett kép (Forrás: saját kép)

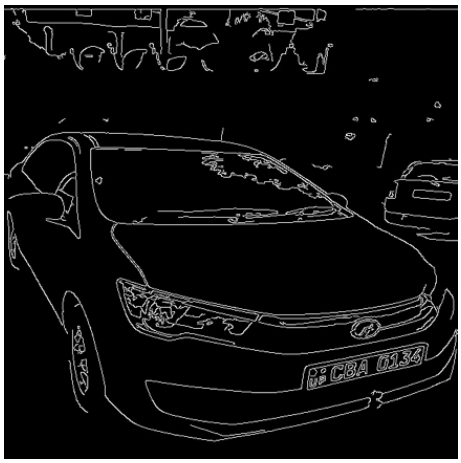
A következő sorban történik meg a kép elhomályosítása a bilateral filter segítségével. Éldetektálás előtt ajánlott ezt a módszert használni, mivel így pontosabb lesz az eredmény

```
bfiltered_image = cv2.bilateralFilter(gray_image, 11, 17, 17)
```

7. ábra Homályosított kép (Forrás: saját kép)

Miután a kép előkészítésre került, következhet az éldetektálás a fentebb említett Canny Edge Detector segítségével.

```
edged_image = cv2.Canny(bfiltered_image, 100, 300)
```



8. ábra Éldetektálás a képen (Forrás: saját kép)

Mivel megvannak az élek, ezért keypoints-okat hozunk létre. Első paraméter az eredeti kép másolata lesz, mert nem akarjuk módosítani az eredetit. Második paraméter felel a kontúrok visszaadásáért. A CHAIN_APPROX_SIMPLE felel azért, hogy a kontúrban szereplő vonalak végpontjait tárolja csak el. Ez rendkívül hasznos mert egyértelműen megtudjuk határozni a

kontúrokat alkotó vonalakat a végpontok alapján, de mégis kevesebb adatot kell tárolnunk. Ezután következik a kontúrok megjelenítése az eredeti képen, hogy tudjuk, miről is van szó. Utolsó lépésként területük alapján rendezzük őket és az első 30-at kiválasztjuk.

```
contours, _ = cv2.findContours(edged_image.copy(), cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    tmp_image = resized_image.copy()
    cv2.drawContours(tmp_image, contours, -1, (0, 255, 0), 1)
    contours = sorted(contours, key=cv2.contourArea,
reverse=True)[:30]
```

9. ábra Kontúrok a képen (Forrás: saját kép)

A következőkben egy for ciklus kerül alkalmazásra végig a listán ami tartalmazza a kontúrokat. Poligont próbál illeszteni a kontúrokra és eltárolja az approx változóban. Ha 4 oldala van akkor break-el, mert megvan a 4 oldalú poligon. Ez az egész program kulcsa, hogy olyan alakzatot keres aminek 4 oldala van, hiszen a rendszámtábla mindig minden esetben 4 oldalú. Színe lehet más, kitöltése is lehet más, de az alak állandó.

```
 location = None
    for contour in contours:
        approx = cv2.approxPolyDP(contour, 10, True)
        if len(approx) == 4:
            location = approx
            break
```

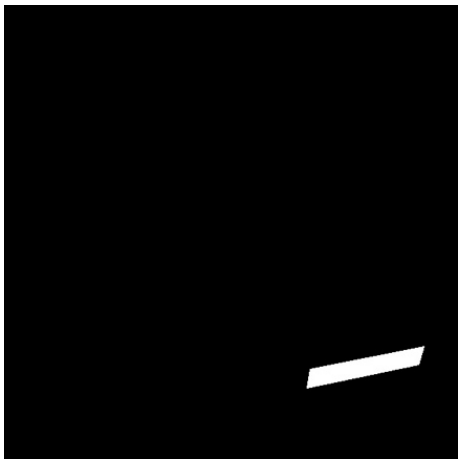
Amennyiben az algoritmus nem találta meg a rendszámtáblát, a függvény visszatér egy hibaüzenettel, vagyis a függvényhívás befejeződik.

```
if (location is None):
        return "No plate detected"
```

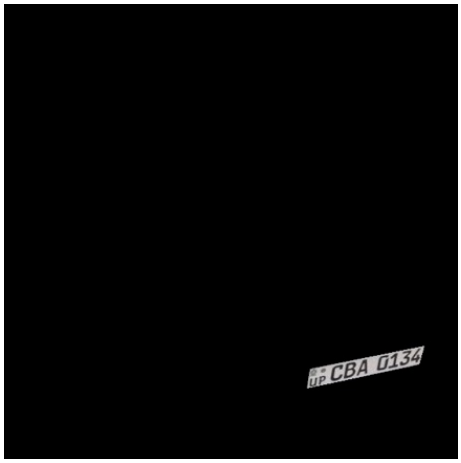
Mindezek után következik a maszk, feltöltve nullákkal (ami alapesetben fekete) a resized alakban (600x600). A kiválasztott poligon oldalai által közbezárt területen lévő értékeket 255-re állítjuk, így a fekete maszkon kirajzolódik a fehér sokszög, amely a rendszámtábla helyét jelöli.

```
mask = np.zeros(gray_image.shape, np.uint8)
tmp_image = cv2.drawContours(mask, [location], 0, 255, -1)
```

10. ábra Maszk körvonala a képen (Forrás: saját kép)



11. ábra Maszkolás (Forrás: saját kép)



10. ábra Maszkolt rendszámtábla (Forrás: saját kép)

Következő lépés megszabadulni az értéktelen fekete területektől. A program kiválasztja az x és y koordinátákat, ahol 255 van, vagyis fehér. Ezentúl ezekkel fog csak foglalkozni az algoritmus, a többi nem tartalmaz értéket számunkra. Ezután kiválasztja a legkisebb és legnagyobb x és y koordinátát.

```
(x, y) = np.where(mask == 255)
(x_min_value, y_min_value) = (np.min(x), np.min(y))
(x_max_value, y_max_value) = (np.max(x), np.max(y))
cropped = gray_image[x_min_value: x_max_value + 1, y_min_value:
y_max_value + 1]
```

11. ábra Kivágott rendszámtábla (Forrás: saját kép)

12. ábra Szembeforgatás (Forrás: saját kép)

A következőkben a szürke képből kivágja a kiszámolt minimum és maximum koordináták közé eső pixeleket. A get_warp_perspective függvényem elvégzi a szembeforgatást. Ez a függvény egy saját függvény, amelynek az algoritmusa a dolgozat elején említett forrásból származik.

```
cropped = get_warp_perspective(gray_image, location.reshape(4, 2))
```

A korábban bemutatott homályosítás újra alkalmazásra kerül a kivágott és szembeforgatott képen. Az OCR library segítségével a homályos, szürkeárnyalatos képről megpróbálja leolvasni a szöveget. Utolsó lépésként a leolvasott szövegből törli a nem alfanumerikus karaktereket, valamint nagybetűssé alakítja.

```
 result = reader.readtext(cropped)[0][1]
 result_formatted = ''.join(ch.upper() for ch in result if
ch.isalnum())
```

Az egész egy try-except blokkban van benne, mert ha nem sikerült megtalálni jól a rendszámtáblát, akkor is lefut ez a kódrészlet és hibával térne vissza az OCR, mivel nem tartalmaz szöveget a kivágott kép. Amennyiben ez történik, akkor szimplán kiírja, hogy nem sikerült és folytatja a következő iterációval.

Tesztelés

A tesztelés során a már bevezetőben említett „solution.json” file alapján működik. Egy külön erre a célra létrehozott python file végzi a rendszámok felismerését, illetve a karakterek leolvasását.

Ebben a fájlban beolvasásra kerül az elvárt eredményeket tartalmazó json fájl. A program megnyitja, majd a fájlnevek alapján beolvassa a fájlokat. A korábban részletesen bemutatott rendszámtábla leolvasó függvényt felhívja, majd az visszatér a rendszámtábla szövegével. Ezt a szöveget ezután összehasonlítja a json fájlban tárolt elvárt kimenettel, majd egy százalékos egyezést számol rá. A program figyelembe veszi azt is, ha nem sikerül a leolvasás. Eredményként a terminálba kiírja a fájlnevet, az elvárt kimenetet, a leolvasott rendszámtáblát majd az egyezés mértékét. Ezt minden egyes fájlra elvégzi, ami a json fájlban meghatározásra került. A program végén átlagolja a az egyezéseket, így megkapva a program eredményességét.

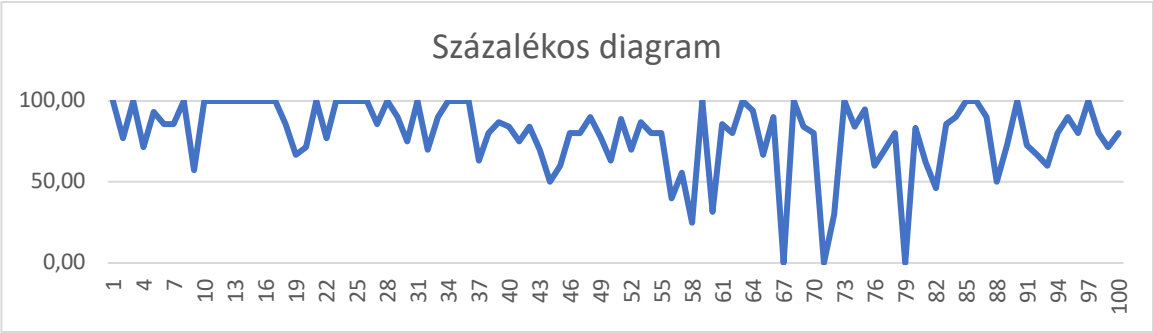
```
Filename: test_image_91 Expected output: GL395X Calculated output: 6LR395       Matching ratio: 66.67%.
Filename: test_image_92 Expected output: DL6CAB123X     Calculated output: DLCCABYZ3U   Matching ratio: 60.0%.
Filename: test_image_93 Expected output: KA51MJ8156     Calculated output: KAS10J8156   Matching ratio: 80.0%.
Filename: test_image_94 Expected output: HR26DK6475     Calculated output: HR2GDK6475   Matching ratio: 90.0%.
Filename: test_image_95 Expected output: MH14DT8831     Calculated output: HHI4DT8831   Matching ratio: 80.0%.
Filename: test_image_96 Expected output: MH01BU5207     Calculated output: MH01BU5207   Matching ratio: 100.0%.
Filename: test_image_97 Expected output: KL10AW2814     Calculated output: KL104M2814   Matching ratio: 80.0%.
Filename: test_image_98 Expected output: KL49H5270      Calculated output: KL49H        Matching ratio: 71.43%.
Filename: test_image_99 Expected output: TN07BU5427     Calculated output: T8078U5427   Matching ratio: 80.0%.

Length of dataset: 100  Average matching ratio: 79.46%.

Plate not found or couldn't read: 3     Average matching ratio with only found plates: 81.92%.
```

13. ábra Eredmények listája (Forrás: saját kép)

Ahogy az 15. ábrán is látható, a program 100 db. rendszámon végezte el a leolvasást, az egyezési pontosság 79.4%. 3 esetben nem talált rendszámot és ha ezeket nem veszi számításba, a pontosság 81.92%.

Összesítve:

- 80-100% közötti pontosság: 65 db.
- 60-80% közötti pontosság: 23 db.
- 0-60% közötti pontosság: 12 db.



14. ábra Eredmények diagramon való bemutatása (Forrás: saját kép)

A tesztek alapján kijelenthető, hogy a program megfelelően működik, a 79.46%-os pontosság elfogadható.

Irodalomjegyzék

[1] https://reglocker.co.uk/uses-benefits-of-anpr

[2] http://www.anpr-international.com/history-of-anpr/

[3] https://www.inf.u-szeged.hu/~tanacs/pyocv/canny_ldetektor.html