Gépi látás

GKNB_INTM038

Automatikus rendszámtábla felismerés

Lengyel Márk

LNXQYO

Győr, 2021. január 16.

Tartalomjegyzék

1. Bevezetés

A dokumentumban szereplő projekt célja egysoros magyar rendszámtáblák beolvasása, megfelelő fénykép biztosításával. Ennek a megvalósításához a programot Python programozási nyelven készítem el, a gépi látási eszközöket pedig az OpenCV könyvtár használatával érem el.

A program működése során a biztosítot képen felkeresi a lehetséges rendszámtáblákat, majd azokat kiszűrve, szöveges módon közli a felhasználóval a megtalált egy vagy több rendszámot, amelyet a kimenetből könnyedén kiolvasva későbbi azonosításra, vagy egyéb felhasználásra lehet fordítani.

A következő részekben ennek a programnak a működéséről, elkészítéséről, és használatáról fogok beszélni, továbbá beszélek a tesztelés menetéről, és a felismerő program pontosságáról.

2. A használt gépi látási eszközök, elméleti háttér

Mielőtt belevágnák a valós program működésébe, ismertetni szeretném dióhéjban az egyes folyamatok során használt eszközök működését.

2.1. Előkészítő értékkészlet transzformációs folyamatok

Ezek a digitális képfeldolgozás kategóriájába sorolható folyamatok a képek 'átalakítását' végzik különböző módokon, bizonyos esetekben azért, hogy valamilyen másik feldolgozási folyamat minőségét növeljék, máskor pedig elengedhetetlenek a további folyamatok lefutásához.

Ilyen folyamatok lehetnek a zajcsökkentés, homályosítás és szürkeárnyalatolás a bináris képek előállításához. Ezek a folyamatok általában lokálisan, a szomszédos intenzitásokat figyelembe véve transzformálják a képet, ezzel egy új képet előállítva, más esetekben viszont a pixeleket egyesével transzformálják.

A szürkeárnyalatolás közben a pixelek piros, zöld és kék intenzitásainak átlagai kerülnek felhasználásra, ahhoz hogy egy képet a szürke szín árnyalataival jelenítsünk meg. Ez a számítási igényeket csökkenti, és bizonyos folyamatokat (pl.: bináris képpé transzformálás) tesz lehetővé.

2.2. Az éldetektálás folyamata

Az éldetektálás fontos szerepet játszik a felismerésben, hiszen a kontúrok meghatározásához szükség van az élekre. Egy képen az élek oylan helyen jelentkeznek, ahol az intenzitásban hirtelen változás történik. A program által használt Canny éldetektálás öt különböző lépésre bontható.

Első sorban egy Gauss szűrő kerül alkalmazásra, annak érdekében, hogy elsimítsa a képet. Ez a lépés az előzőekben hallott zajszűrés, homályosítás folyamata egy bizonyos kernellel.

A következő lépésben a Canny detektor felkeresi a különböző irányokban megjelenő éleket azoknak megfelelő szűrők segítségével. Ennek a lépésnek eredményeként az irányok deriváltjait kapjuk, amelyből megállapítható az élek meredeksége és iránya.

A deriválás után egy un. nem-maximum vágás művelet következik. Ennek a célja a kontúrok vékonyítása, amellyel kiemelhetők a legnagyobb intenzitásbeli változások.

A két utolsó lépés küszöbölés, avagy 'thresholding'. Az első küszöbölési folyamat egy kétlépéses küszöbölés, ilyenkor a kép háromféle intenzitásra bomlik. A második küszöbölés egy hiszterézises küszöbölés, amelynek végeredményeként kapjuk a bináris képet. A fent említett küszöbölések során a gyenge élek kerülnek szűrésre.

2.3. Kontúrok keresése

A kontúrok keresésekor, az éldetektáláskor kapott bináris képen, a vonalakat kétdimenziós koorinákkal jellemzett alakzatokkal közelítjük. Ezzel az egyes alakzatok szűrését, kivágását tesszük lehetővé. Az alakzatok területe is könnyen megállapítható ez alapján, továbbá ha szükséges, adott alakzat közelíthető télalappal. Ezen algorimus[1] implementációja része az OpenCV könyvtárnak.

2.4. Mintakeresés

A mintakeresés segítségével tudjuk megállapítani, hogy egyes rendszámtáblákon milyen karakterek szerepelnek. Az algoritmus a mintaként megadott kép intenzitásai hasonlítja össze a rendszámtáblából kinyert karakter képével. A hasonlóság mértékét egy függvénnyel méri, amely jelen esetben az alul látható OpenCV TM_SQDIFF függvénye[2],

$$R(x,y) = \sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2 \tag{1}$$

ahol R az eredményül kapott képet, T a mintául tekintett képet, I a bementei képet x, y, x' és y' pedig az adott pixelek pozícióját jelöli.

A mintakeresés lefuttatva az összes minta karakterre egy tömböt eredményez, amelyek közül ezen esetben a minimális értékkel rendelkező jelzi a legjobb egyezést.

3. A program megvalósítása

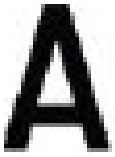
Most, hogy megismertük a felismerő programot felépítő egyes elemeket, ismertetem a program felépítését, annak működését az egyes működési lépések alapján.

3.1. Előkészítés a működéshez

Ahhoz, hogy a program fel tudja ismerni a rendszámtábla karaktereit mintakereséssel, először el kell készíteni a hasonlításhoz használt mintákat, és azokat megfelelő nevükkel el kell látni. Ehhez az 1. ábrán látható, a magyar jogszabályban[3] megtalálható képre lesz szükség.

1. ábra. Magyarországon lehetséges rendszámtábla karakterek.
Forrás: 326/2011. (XII. 28.) Korm. rendelet a közúti közlekedési igazgatási feladatokról, a közúti közlekedési okmányok kiadásáról és visszavonásáról

Ez a kép azonban önmagában nem megfelelő, hiszen fel kell bontani különálló karakterekre. Ehhez a fetch_chars.py szkript használható, amely az előzőekben említett küszöbölés és kontúrekeresési módszerekkel felbontja a képet külön karakterekre. Ennek a működése nagymértékben hasonlít a felismerő program rendszámtábla felbontó kódjához, ezért ennek a működését a későbbiekben részletezem. A végeredményül kapott karakterek egyike látható a 2. ábrán.



2. ábra. Felbontás után keletkezett *'A'* karakter.

3.2. Átméretezés

Ez a lépés nem létfontosságú, de túl nagy képek esetén a feldolgozás nagy időket vehet igénybe, ezért a képek átméretezésre kerülnek úgy, hogy azok beleférjenek egy 1920x1080 méretű téglalapba. Az átméretezés mértéke, amennyiben a szélesség vagy magasság átlépi a korlátot $r_1 = W_{max}/W$ és $r_2 = H_{max}/H$ közül a kisebbik, ahol r_x az adott mértéket, W és H a szélességet és magasságot, W_{max} és H_{max} pedig a maximális megfelelőket jelöli.

1. kódrészlet. Az átméretezéshez használt kód.

```
1 max_width = 1920
2 max_height = 1080
3 if (w > max_width) or (h > max_height):
4     ratio = min(max_width / w, max_height / h)
5     new_size = (round(w * ratio), round(h * ratio))
6     img = cv2.resize(img, new_size, interpolation=cv2.INTER_AREA)
```

Az átméretezés után kapott kép a 3. ábrán látható.

3. ábra. Az átméretezett kép.
Forrás: www.azauto.hu

3.3. Zajszűrés

Ebben a részben az átméretezett képen különféle zajszűrések mennek végbe, annak érdekében, hogy a küszöbölés és éldetektálás során jobb eredményt kapjunk. A zajszűrés kétféle módon megy végbe, először egy specifikus OpenCV zajszűrővel[4] történik meg a kép szűrése.

2. kódrészlet. A zajszűréshez használt függvény.

```
1 img_denoised = cv2.fastNlMeansDenoisingColored(img_start,None,15,10,7,21)
```

A 2. kódrészletben az *img_denoised* a kimeneti képet jelöli, az *img_start* pedig a bemeneti átméretezett képet.

A második zajszűrési lépés elsimítja, homályosítja a képet, amennyiben a homályossága nem ér el egy bizonyos szintet. Ezt a szintet a felhasználó felülírhatja argumentum megadásával. A homályosság mérésére[5] és a homályosításra[6] szintén OpenCV függvények biztosítanak megoldást.

3. kódrészlet. A homályosság méréséhez használt függvény.

```
1 blur_value = cv2.Laplacian(
2         img_blurred,
3         cv2.CV_64F).var() * 100000 / (img.shape[0] * img.shape[1])
```

A fenti 3. kódrészletben a *blur_value* adja meg a kiszámított homályosság értéket, az *img_blurred* a bemenetként adott, homályosított képet, továbbá az *img.shape[0]* és *img.shape[1]* pedig a kép szélességét és magasságát megfelelő sorrendben.

4. kódrészlet. A homályosításhoz használt függvény.

```
1 img_blurred = cv2.GaussianBlur(img_denoised, (3, 3), 0)
```

A 4. kórdészletben látható homályosításnál az *img_blurred* adja meg a kimeneti elsimított képet, az *img_denoised* pedig a bemeneti zajszűrésen átesett képet. A *(3, 3)* paraméter azt jelzi, hogy a Gauss elsimításnál[6] mekkora kernel legyen haszálva, jelen esetben ez egy 3x3 méretű mátrix.

A fentiekben említett zajszűrési lépések az előzőekben megbeszélt elmélet alapján működnek. A zajszűrés elvégzése után kapott kép megtekinthető az 4. ábrán.

4. ábra. Kép a zajszűrés után.

3.4. Éldetektálás

Az éldetektálás nélkülözhetetlen ahhoz hogy felismerjük az egyes kontúrokat a képen, amelyekről később megállapítjuk hogy rendszámtáblák vagy sem. Az előző lépésben kiszűrtük a zajt a képről, aminek köszönhetően az éldetektálás jobb eredményeket fog adni.

Ahhoz hogy éleket detektáljunk, először venni kell a képünk szürkeárnyalatolt verzióját. Ezt az 5. kódrészletben látható módon[7] tehetjük meg.

5. kódrészlet. A szürkeárnyalatoláshoz használt függvény.

```
1 img_grayscale = cv2.cvtColor(img_blurred, cv2.COLOR_BGR2GRAY)
```

A fenti 5. kódrészletben az *img_grayscale* a kapott szürkeárnyalati képet, az *img_blurred* pedig a bemeneti képet jelöli, továbbá a *COLOR_BGR2GRAY* megadja az átalakítás módját, amely ezesetben színesről szürkeárnyalatosra történik.

A szürkeárnyalatos képből a canny éldetektáláshoz ki kell deríteni a megfelelő küszöbértékeket. Ezt a következő küszöbölő metódussal[8] tudjuk elvégezni.

6. kódrészlet. A küszöbérték megállapítása.

```
1 thr = cv2.threshold(
2         img_grayscale,
3         0,
4         255,
5         cv2.THRESH_BINARY | cv2.THRESH_OTSU)[0]
```

Ebben a kódrészletben a *thr* a küszöbértéket, az *img_grayscale* a bemeneti képet jelöli. A szürkeárnyalati kép és küszöbérték megállapítása után megvan minden ahhoz, hogy lefuttassuk a canny éldetektálást. Ez az előző részekben megismert módon a következő kóddal[9] tehető meg.

7. kódrészlet. Canny éldetektálás.

```
1 img_edges = cv2.Canny(img_grayscale, thr, 0.5 * thr)
```

Itt *img_edges* jelöli a kimeneti bináris képet, *img_grayscale* a bemeneti képet, *thr* pedig az előzőleg kiszámolt küszöbértéket. A Canny függvény utolsó két paramétere a hiszterézis küszöbölés során használódik fel, a kódban látható elosztással általánosan jó eredmény érhető el.

A lefutás végeztével keletkezett bináris képet szemlélteti az 5. ábra.

5. ábra. Kép a zajszűrés után.

3.5. Rendszámtáblák keresése és szűrése

Az élek érzékelése után a következő folyamat a rendszámtáblák keresése. Ahhoz, hogy megtaláljuk a lehetséges rendszámtáblákat, szükség van az élek tárolására, hogy azt a program használni tudja. Az első lépés tehát, hogy kinyerjük a kontúrokat a bináris képből. Ezt az előzőleg megismert módon az OpenCV egyik függvényével[10] tehetjük meg.

8. kódrészlet. Kontúrok keresése.

```
1 cnt = cv2.findContours(img_edges, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)[0]
```

A fenti 8. kódrészletben a *cnt* változóba kerül a kontúrok listája, amelyek kétdimenziós koordinátákból állnak és az *img_edges* adja meg a bináris képet, amelyből a kontúr adatokat gyűjtjük.

Mituán kigyűjtöttük a kontúrokat, azokat több lépésben szűrni kell, hogy valóban rendszámtáblákat jelölnek vagy sem. A következőkben erről beszélek részletesebben.

3.5.1. Szűrés méret alapján

Ez a szűrés elsősorban összeveti az egyes kontúrok méretét a kép méretével. A kontúrok mérete a ***cv2.contourArea(x[0])*** függvénnyel[11] állapítható meg. Ilenykor a helyesség megállapításához használt módszer a következő.

$$A_{cnt} <= (W_{img} * H_{img})/100 \tag{1}$$

$$A_{cnt} >= (W_{img} * H_{img})/10000 \tag{2}$$

Itt A_{cnt} jelöli a kontúr területét, W_{img} és H_{img} pedig a kép szélességét és magasságát. A 100 és 10000 konstansok változtathatók a rendszámtábla képen való méretének függvényében.

Az előző szűrést követően van mégegy méret alapján való szűrés, amely egyszerűen sorbarendezi a kontúrokat méretük alapján, majd kiválassza belőlük az 50 legnagyobbat. Ez a lépés elhanyagolható, azonban segít a teljesítmény javításában.

3.5.2. Belső kontúrok vizsgálata

Ennél a lépésnél a kontúrok belsejében elhelyezkedő kontúrok megszámlálása alapján szűrjük a jelölteket. Ahhoz, hogy megállapítsuk, hogy egy kontúr benne van egy másikban, a következő metódusra lesz szükség.

9. kódrészlet. Egymásban elhelyezkedő kontúrok ellenőrzése.

```
1  def is_inside(inside, outside, limit_val=-1):
2      point_limit = limit_val * len(inside)
3      if limit_val < 0:
4          point_limit = 1
5      in_point = 0;
6      for i in inside:
7          is_in = cv2.pointPolygonTest(outside,tuple(i[0]),False)
8          if is_in >= 0:
9              in_point += 1
10             if in_point >= point_limit:
11                 return True
12     return False
```

A fenti függvény használatához szükséges *inside* jelöli a feltételesen belső kontúrt, az *outside* pedig a külsőt. Ha az *inside* kontúr az *outside* kontúrban helyezkedik el, a függvény egy logikai igaz értékkel tér vissza, más esetben hamissal. A *limit_val* paraméterrel megadható egy bizonyos limit, hogy mennyi pont szerepelhet az *outside* kontúrban mielőtt a függvény igaz értéket dob.

Az algoritmus múködése során az *inside* kontúr pontjain végig iterálva nézi meg, hogy azok benne vannak e a másik kontúrban. Ezt az ellenőrzést az OpenCV ***cv2.pointPolygonTest*** függvényével[12] lehet elvégezni.

A fenti folyamatok lefutása után minen kontúrhoz kapunk egy számot, hogy abban mennyi kisebb kontúr szerepel. Azon kontúrok, amelyekben kevesebb, mint 3 kisebb kontúr szerepel kidobásra kerülnek. Ideális esetben a belső kontúrok száma 6 vagy több lenne, ám egybefolyó kontúrok miatt itt 3 kerül felhasználásra.

3.5.3. Méretarányok szerinti szűrés

Itt a fennmaradt jelöltek szélességének és magasságának arányát vizsgálja a program. Ahhoz hogy ezt meg lehessen tenni, azonban téglalapokkal kell számolni, viszont a kontúrokban előfordulhatnak különféle alakzatok. A köré írható téglalapot a ***cv2.boxPoints*** függvénnyel[13] kaphatjuk meg.

A kontúrok közül kidobásra kerülnek azok, amelyek téglalpjaihoz tartozó szélesség és magasság arányai egy bizonyos mértéken túl eltérnek a jogszabályban[3] meghatározott magyar egysoros táblák méretarányaitól (510mm x 110mm).

3.5.4. Duplikált táblák szűrése

Ez az utolsó szűrési lépés, amely a belső kontrúrok vizsgálatával foglalkozó résznél használt, kontúrok egymásba vágását vizsgáló függvényt használja. Az előzővel ellentétben azonban itt most a lehetséges jelöltek egymással való viszonyát vizsgálja a program. Amennyiben két kontúr egybevág, azok közül egyik törlésre kerül. Ezzel megelőzhetjük azt, hogy ugyanazt a rendszámtáblát többször ellenőrizzük.

3.5.5. A szűrés eredménye

A szűrési folyamatok végeztével, a megmaradt jelöltek nagy valószínűséggel jelölnek valós rendszámtáblákat. A folyamat során kiszűrt jelölteket szemlélteti a 6. ábra.

6. ábra. A szűrés közben kiválogatott kontúrok.
Narancs: belső kontúr vizsgálat, Piros: méretarány vizsgálat, Türkíz: duplikáció vizsgálat

4. Karakterek felismerése

A karakterek felismerése az utolsó lépés ahhoz, hogy a képünkön lévő rendszámot szöveges formára hozzuk.

4.1. Karakterek elkülönítése

Ahhoz, hogy a karaktereket külön tudjuk összehasonlítani, először a táblákat ki kell vágni, majd feldarabolni külön karakterekre. A kivágás könnyen megoldható, hiszen az előző lépésekben kapott kontúrokat kell kivágni. Egy ilyen kivágott táblát szemléltet a 7. ábra.

7. ábra. Kivágott rendszámtábla.

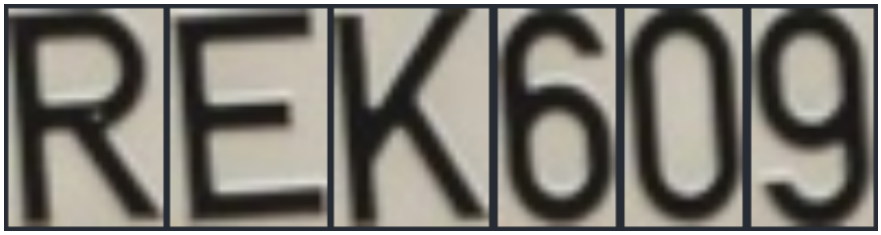
A kivágást követően szintén az előzőekben már használt módszerek segítségével bontja fel a program a rendszámtáblát.

Ezek a lépések a következők:

- Szürkeárnyalatolás
- Canny éldetektálás
- Kontúrok keresése
- Kontúrok szűrése
- Jelöltek kivágása

Az egyetlen lényeges különbség a kontúrok szűrésében van, ugyanis a karakterek szűrésénél csak az egymásba ágyazott kontúrok kerülnek kiszűrésre, úgy, hogy a külső kontúrok maradjanak.

A 7. ábrán látható kép karakterei a folyamat után a 8. ábrán láthatóak.



8. ábra. A kivágott karakterek egymás után.

4.2. Mintaillesztés

A mintaillesztés során eldöntjük a különválasztott karakterekről, hogy melyik mintának felelnek meg. A karakterek sorrendjét a képen való pozíciójuk határozza meg, balról jobbra haladva. Ebben a sorrendben a karakterek összehasonlításra kerülnek minden mintával, amelyeket az előkészületi részben állítottunk elő.

Az egyes karakterek képei és a minták szürkeárnyalatolva[7] és küszöbölve[14] lesznek, majd a tábláról kivágott karakterek a hasonlításhoz használt minta méretére lesznek átméretezve. A minták illesztését[2] a korábban megismert elmélet alapján végzi a program, az OpenCV függvényeit használva.

10. kódrészlet. A karakter képek előkészítése.

```
1 gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
2
3 thr = cv2.adaptiveThreshold(
4         gray,
5         255,
6         cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
7         cv2.THRESH_BINARY_INV,
8         15,
9         0)
```

A fenti kódrészletben a *gray* adja meg a szürkeárnyalatolás eredményéül kapott képet, az *img* a bemeneti képet, a *thr* pedig a küszöbölés eredményeként kapott bináris képet.

A képek előkészítése után már csak az összehasonlítás[2] van hátra, amelyet a következő programrészlet hajt végre.

11. kódrészlet. Mintával való hasonlítás.

```
1 cv2.matchTemplate(template, image, cv2.TM_SQDIFF)
```

Itt a *template* paraméter a mintául használt képet, az *image* pedig a tábláról kivágott képet jelöli.

Miután minden mintát összehasonlítottunk a kivágott képpel, kapunk egy listát az egyes minták egyezésének mértékéről. Ezek közül ***cv2.TM_SQDIFF*** esetén a minimális értéket kiválasztva megkapjuk a legnagyobb egyezést mutató mintát, így a mintát reprezentáló karaktert.

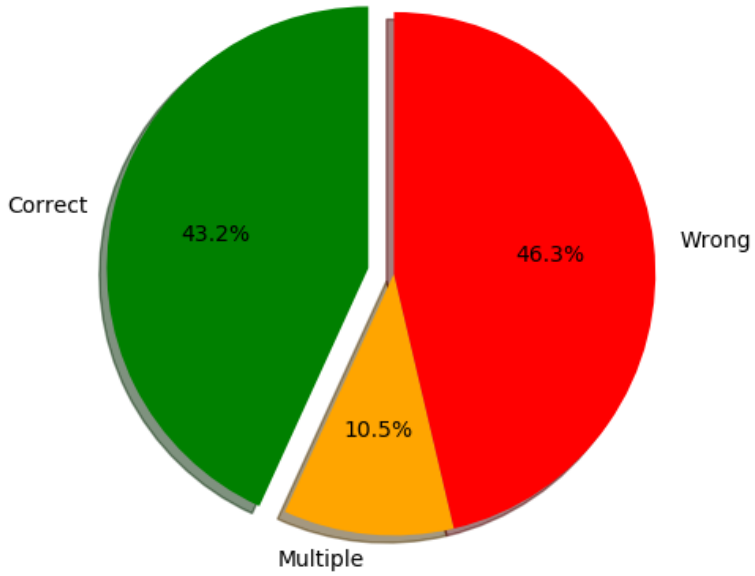
A mintákat lefuttatva az összes képen, megkapjuk azok felismert karaktereit, amelyeket összefűzve egy rendszámtáblát kapunk szöveges formában. A 9. ábrán egy sikeresen felismert rendszámtábla jelölése látható.

9. ábra. Sikeresen felismert rendszámtábla.
Zöld: felismert tábla, Kék: felismert karakterek

5. Tesztelés

A teszteléshez 95 kép került felhasználásra, ezeket a tesztelő szkript a felsimerő programmal lefuttatta és a végeredményüket összehasonlította a várt eredménnyel. A képeken[15] egy rendszám szerepel, minden renszám leolvasható.

Az összehasonlítás során helyes találatnak számít, ha a program egy rendszámtáblát talál megfelelő karakterekkel, félig sikeres, ha több renszámtáblát talál a vártnál, de közülük egyik helyesen megadja a várt eredményt, és helytelen ha a szám nem egyezik meg. A 10. ábrán a teszt végeredménye látható.



10. ábra. A teszt eredménye.
Zöld: helyes, Narancs: félig helyes, Piros: helytelen

1. táblázat. A találatok sikeressége.

	Találatok száma (Összes: 95db)
Helyes	41 db
Félig helyes	10 db
Helytelen	44 db

Az 1. táblázatban látható, hogy a 95 kép közül 41 képen sikeres volt a leolvasás, 10 képen jó volt a leolvasás, de nem létező táblákat talált a program, és 44 képen sikertelen a leolvasás.

```
NOZ-822 --- HOZ-822 Score: 5 Reads: 1
PFC-808 --- PFG-008 Score: 5 Reads: 1
NVW-388 --- NVW-300 Score: 4 Reads: 1
NUX-659 --- NUX-659 Score: 6 Reads: 1
PPE-525 --- PPE-525 Score: 6 Reads: 1
Test done in 236.819 s
Total: 95
Correct: 41
Multiple: 10
Wrong: 44
Average score: 4.315789473684211
```

11. ábra. A tesztelés megjelenése a konzolablakban.

A tesztelés végén látható, hogy mennyi időt vett igénybe a teszt, továbbá, hogy átlagosan hány karaktert talált el a maximális 6 karakterből (Average Score). A teszteléshez használt képek elérhetőek a ***személyes Dropbox tárhelyemen*** .

6. Használat

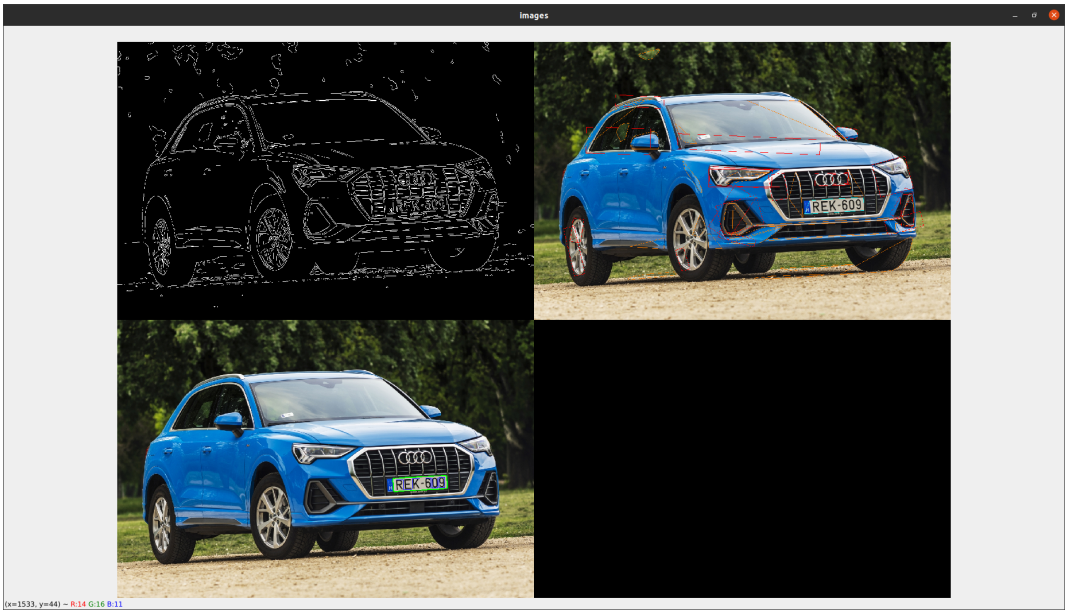
A program használata konzolablakból tehető meg, a ***read_plate.py*** fájl futtatásával. A program működéséhez és testreszabásához parancssori argumentumokat kell használni, ilyenek:

- ***-input=útvonal*** - a leolvasáshoz használni kívánt képfájl elérési útvonala
- ***-blur-limit=x*** - a homályosítási/zajszűrési lépés során használt homályosítási limit
- ***-no-image*** - a program befejezéskor ne jelenítsen meg állapotképeket
- ***-silent*** - a program befejezéskor csak a leolvasott rendszámokat listázza ki

Az argumentumok közül az ***-input*** kötelezően megadandó, a többi tetszés szerint, bármilyen sorrendben használható. A program működéséhez szükség van a ***read_plate.py*** programmal megegyező helyen egy ***templates*** mappára, ami a karakterek felismeréséhez használt karakterek képeit tartalmazza.

```
862    ms| Blur value: 9.01
872    ms| Found 209 contours.
1038   ms| 1 plates found.
Plate 1 number: REK-609
Executed in 1050 ms
```

12. ábra. A program normál kimenete a konzolban.



13. ábra. A program befejeztekor megjelenő állapotképek.

7. Felhasznált irodalom

[1] Satoshi Suzuki and KeiichiA be. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32 – 46, 1985.

[2] OpenCV documentation of 'matchTemplate' function. https://docs.opencv.org/master/df/dfb/group__imgproc__object.html#ga586ebfb0a7fb604b35a23d85391329be.

[3] 326/2011. (xii. 28.) korm. rendelet a közúti közlekedési igazgatási feladatokról, a közúti közlekedési okmányok kiadásáról és visszavonásáról, 2011. http://njt.hu/cgi_bin/njt_doc.cgi?docid=140326.418311.

[4] OpenCV documentation of 'fastNlMeansDenoisingColored' function. https://docs.opencv.org/master/d1/d79/group__photo__denoise.html#ga03aa4189fc3e31dafd638d90de335617.

[5] OpenCV documentation of 'Laplacian' function. https://docs.opencv.org/master/d4/d86/group__imgproc__filter.html#gad78703e4c8fe703d479c1860d76429e6.

[6] OpenCV documentation of 'GaussianBlur' function. https://docs.opencv.org/master/d4/d86/group__imgproc__filter.html#gaabe8c836e97159a9193fb0b11ac52cf1.

[7] OpenCV documentation of 'cvtColor' function. https://docs.opencv.org/master/d8/d01/group__imgproc__color__conversions.html#ga397ae87e1288a81d2363b61574eb8cab.

[8] OpenCV documentation of 'threshold' function. https://docs.opencv.org/master/d7/d1b/group__imgproc__misc.html#gae8a4a146d1ca78c626a53577199e9c57.

[9] OpenCV documentation of 'Canny' function. https://docs.opencv.org/master/dd/d1a/group__imgproc__feature.html#ga04723e007ed888ddf11d9ba04e2232de.

[10] OpenCV documentation of 'findContours' function. https://docs.opencv.org/master/d3/dc0/group__imgproc__shape.html#gadf1ad6a0b82947fa1fe3c3d497f260e0.

[11] OpenCV documentation of 'contourArea' function. https://docs.opencv.org/master/d3/dc0/group__imgproc__shape.html#ga2c759ed9f497d4a618048a2f56dc97f1.

[12] OpenCV documentation of 'pointPolygonTest' function. https://docs.opencv.org/master/d3/dc0/group__imgproc__shape.html#ga1a539e8db2135af2566103705d7a5722.

[13] OpenCV documentation of 'boxPoints' function. https://docs.opencv.org/master/d3/dc0/group__imgproc__shape.html#gaf78d467e024b4d7936cf9397185d2f5c.

[14] OpenCV documentation of 'adaptiveThreshold' function. https://docs.opencv.org/master/d7/d1b/group__imgproc__misc.html#ga72b913f352e4a1b1b397736707afcde3.

[15] Az Autó weboldal. http://www.azauto.hu/.

8. Mellékletek

A. read_plate.py

B. test_images.zip

C. templates