



GKLB INTM038

Gépi látás

Rendszám-tábla felismerés
beadandó

2021/22 2. félév

Készítette:

Gacs Péter (BSBMQW)

Tartalomjegyzék

1. Felépítés:.....	3
file-ok:	3
A működéshez szükséges minták:.....	3
Rendszer felépítése:	3
2. Folyamat	4
2.1 Előfeldolgozás:	4
2.1.1 Kép megnyitása és méretezése	4
2.1.2 Szürke árnylat	4
2.1.3 Zajcsökkentés.....	5
2.1.4 Élek detektálása	5
2.2 Rendszámtábla detektálása	5
2.2.1 Kontúrok keresése	5
2.2.2 Maszk készítése	6
2.2.3 Sarokpontok detektálása.....	6
2.3 Rendszámtábla transzformálása	7
2.3.1 Sarokpontok meghatározása.....	7
2.3.2 Transzformáció számítása	7
2.3.3 Transzformáció	7
2.3.4 Treshold	8
2.4 Szövegfelismerés	8
2.4.1 Pytesseract.....	8
2.4.2 Hibajavítás.....	8
3. Teszt és értékelés	9
3.1 Teszt	9
3.1.1 Összegzés	9
3.1.2 Előnyök.....	9
3.1.3 Hibák összegzése	10
3.1.4 Statisztika	11

1. Felépítés:

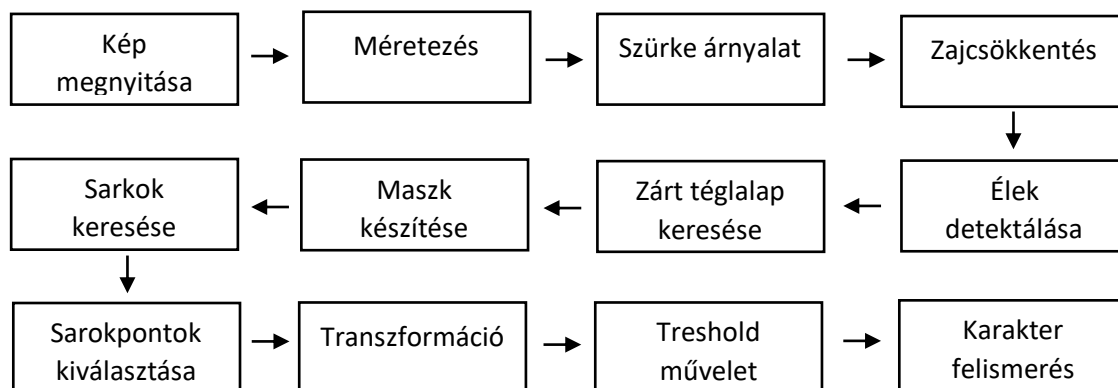
1.1 file-ok:

- licence_plate_detecting.py
- licence_plate_transform.py
- tesseract.py
- main.py
- tester.py
- test.txt
- dokumentation.pdf

1.2 A működéshez szükséges minták:

- fail.jpg
- rszimg mappa:
 - 66 db kép file

1.3 Rendszer felépítése:



2.Folyamat

2.1 Előfeldolgozás:

Az előfeldolgozás során a képen olyan manipulációkat hajtunk végre, aminek végeredményeként megkapjuk a kép éleit. Az előfeldolgozás egészét a „licence_plate_detecting.py” file végzi. A file-ban található LicenceDet objektumban vannak beágyazva a megfelelő metódusok.

2.1.1 Kép megnyitása és méretezése

A *LicenceDet* objektum hívása:

- argumentumként várja a rendszám-táblát tartalmazó kép nevét és annak méretarányait. (default size: fx=0.15, fy=0.15)

A konstruktor automatikusan megnyitja és méretre szabja a képet, továbbá beállítja az átalakításokhoz szükséges default értékeket.

Használt függvények: *cv2.imread()*, *cv2.resize()*



2.1.2 Szürke árnylat

A méretezett képet szürkeárnyaltos formában adja vissza. *gray()* metódussal hívható.

Használt függvények: *cv2.cvtColor()*, *cv2.COLOR_BGR2GRAY*



2.1.3 Zajcsökkentés

A szürkeárnyaltos képen sor kerül a zajcsökkentésre, kiszűri a haszontalan részeket. Enyhe elmosódás hatást tesz a képre. A *bilfil()* metódussal hívható.

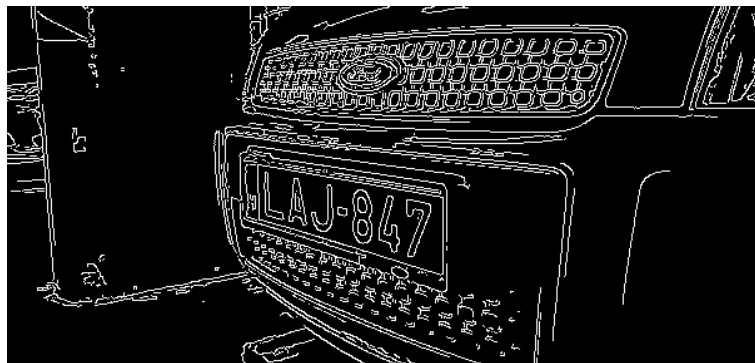
Használt függvények: *cv2.bilateralFilter()*



2.1.4 Élek detektálása

A képen az úgynevezett Canny éldetektálást hajtjuk végre, aminek eredményeként egy binális képet kapunk vissza. Amit élek érzékel: 255 intenzitást kap, minden más: 0. *canny()* metódussal hívható.

Használt függvények: *cv2.canny()*



2.2 Rendszámtábla detektálása

Az éleket tartalmazó képet alapul véve meghatározza a rendszámtábla helyét, leválasztja azt az eredeti képről, majd meghatározza a sarokpontjait.

2.2.1 Kontúrok keresése

A következő lépésben a program az éleket tartalmazó képen zárt alakzatokat keres. Amennyiben az előfeldolgozás során jól állítottuk be az értékeket, a legnagyobb területtel

rendelkező négyszög lesz a rendszámtábla. A `contour_detect()` metódus ezen alakzat pontjait adja vissza.

Használt függvények: `cv2.findContours()`, `cv2.contourArea()`

2.2.2 Maszk készítése

A kontúr detektálása után a program létrehoz egy új, teljesen fekete az éldetektált kép méreteivel megegyező képet. Ezek utána a képen a kontúr által határolt területen az intenzitást 255 értékre állítja, ezáltal létrehozva a maszkot, ahol elkülönül a rendszámtábla. Az így kapott maszkon finomítási célokkal végrehajt egy morfológiai nyitást is. Ezáltal a kisebb hibák eltűnnek.

Ennek segítségével már könnyedén ki tudjuk jelölni az eredeti képen a rendszámtábla helyét. A `mask()` és a `cut_the_mask()` metódusok által hívható.

Használt függvények: `numpy.ndarray()`, `numpy.ones()`, `cv2.drawContours()`, `cv2.morphologyEx()`



2.2.3 Sarokpontok detektálása

A későbbiekben szükség lesz a rendszámtábla 4 sarkának meghatározására. Hogy a képen található sarkok ne zavarjanak, a sarkok keresését a program a maszkon hajtja végre. A keresést Harris sarokdetektorral végzi. A `corners()` metódussal hívható.

Használt függvények: `cv2.cornerHarris()`

A sarokpontok detektálása a *LicenceDet* objektum utolsó lépése. Eredményeként lekérhető a sarokpontok helyzete és a maszkolt szürkeárnyaltos kép.

- `get_img()` metódus visszaadja a képet
- `get_cornes()` metódus visszaadja a sarokpontokat

A `licence_plate_detecting.py` összes lépését a beállított értékekkel a `do_it()` metódus végzi.

2.3 Rendszámtábla transzformálása

A képeken jellemzően a perspektíva miatt torzítva jelennek meg a rendszámtáblák. Ezt a torzítást a szövegfelismerő szoftverek nem tudják jól kezelni, ezért előbb normalizálni kell a kapott rendszámtáblát.

A transzformáláshoz tartozó folyamatokat a ***licence_plate_transform.py*** nevű file-ban található ***LicenceTrans*** objektum tartalmazza.

2.3.1 Sarokpontok meghatározása

Ahhoz, hogy végre lehessen hajtani a transzformálást, meg kell határozni a rendszámtábla 4 sarkának pontját. A program ezt a következő folyamatokon keresztül teszi meg.

- A *get_corners()* metódus rengeteg pontot ad eredményül, attól függően hogy a Harris sarokdetektort milyen paraméterekkel használtuk.
- A sarokpontokat csoportosítjuk aszerint, hogy milyen távolságra vannak egymástól.
- Majd minden csoportból egy átlag segítségével meghatározunk 1 konkrét pontot, amit innentől kezdve 1 sarokpontnak tekintünk *group_corners()*
- Ezután hozzápárosítjuk az így kapott pontokat a négyszög megfelelő sarkához. *order_corners()*

*Amennyiben a folyamat során nem pontosan 4 sarokpont kerül megállapításra, a teljes folyamat végén a **fail.png** képpel tér vissza.*

2.3.2 Transzformáció számítása

A transzformáció végrehajtása előtt meg kell határozni, hogy mik legyenek a rendszámtábla sarkainak új, normalizált pozíciói. A program ezt a következő folyamatokon keresztül teszi meg.

- A leghosszabb szélesség és a legnagyobb magasság megállapítása
- Az új pontok koordinátái megfelelnek egy téglalap sarkainak, amelynek a szélessége és magassága a meghatározott legnagyobb értékekkel egyezik meg. *new_points()*

„A tesztek során kiderült, hogy a szövegfelismerő szoftver lényegesen jobban teljesít, ha a transzformáláskor hagyok egy kis méretű keretet. A példa esetében ez 5 pixel ”

2.3.3 Transzformáció

Az eredeti és az új sarokpontok felhasználásával a transzformáció végrehajtása. *transform()*

Használt függvények: `cv2.getPerspectiveTransform()`, `cv2.warpPerspective()`



2.3.4 Treshold

Az így kapott képet a jobb felismerhetőség érdekében érdemes alávetni treshold műveleteknek, hogy a felmerülő fényesebb vagy sötétebb területek által okozott torzulásokat ellensúlyozzuk.

A jobb felismerhetőség érdekében az így kapott képen egy morfológiai zárást majd nyitást is végrehajtok. `treshold()`

Használt függvények: `cv2.adaptiveThreshold()`, `cv2.morphologyEx()`



A `treshold` művelet a `LicenceTrans` objektum utolsó lépése. Eredményként a `get_img()` módszerrel kérhetjük le.

A `licence_plate_transform.py` összes lépését a beállított értékekkel a `do_it()` metódus végzi.

2.4 Szövegfelismerés

A karakterek felismeréshez tartozó folyamatokat a **`tesseract.py`** nevű file tartalmazza

2.4.1 Pytesseract

A `pytesseract` egy a pythonhoz telepíthető külső könyvtár, amely a karakterfelismeréshez tartalmaz függvényeket. A `tesseract.image_to_string()` függvény egy képről leolvassa, és egy sztring értékben visszaadja a képen látható szöveget.

2.4.2 Hibajavítás

A beolvasott szövegek hatékonysága érdekében egy szűrést végzünk. Ha egy olyan karakter beolvasása történik, ami nem szerepelhet egy rendszámtábla karakterei között (ékezetes betűk, kis betűk, mondatjelek stb.), abban az esetben az nem íródik az eredményhez.

A karakterek meghatározásával az alkalmazás elvégezte a feladatát, ezt a *main.py* nevű file-ban összegezi, és ezt egy lépésben végrehajtja *get_licence_plate()* nevű függvény.

3. Teszt és értékelés







3.1 Teszt




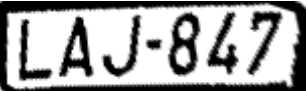

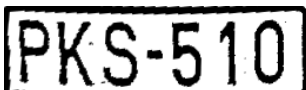
A programot 66 eltérő képen teszteltem, amik a *rszimg* nevű mappában találhatók. A tesztet a *tester.py* nevű file tartalmazza. A tesztek eredményeit a *teszt.txt* file tartalmazza.

3.1.1 Összegzés










- **56 esetben:** működött hibátlan eredménnyel (100%)
- **5 esetben:** több karaktert talált (T) (10,14,31,44,63)
- **2 esetben:** kevesebb karaktert talált (H) (54,55)
- **1 esetben:** 1 karakter tévedés (83%) (7)
- **1 esetben:** a karakterek felét találta el (50%) (5)
- **1 esetben:** teljes mértékben hibás a találat (0%) (6)




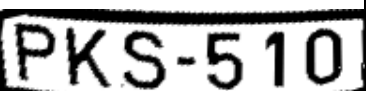





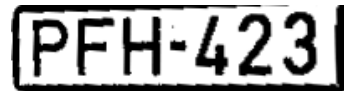
3.1.2 Előnyök

Eredeti Kép	Normalizált kép	Sztring	Következtetés
		LAJ-847	Éles szög
		LAJ-847	Ferde rendszám
		PLL-761	Vakuval készült kép

		SFC-455	Nem a kép közepén található a rendszám
		LAJ-847	Magas beesési szög
		PKS-510	Homályos kép

3.1.3 Hibák összegzése

Eredeti Kép	Normalizált kép	Sztring	Következtetés
		IEN-847	Rossz fényviszonyok
		ANB	Túl éles szög
		LAJ-G47	OCR tévedés
		AJBD-361	Felségjelzés kivágásra került
		ARXM-535	Felségjelzés kivágásra került

		IPKS-510	OCR tévedés
		IPKS-510	OCR tévedés
		VF-868	Felségjelzés kivágásra került + ORC tévedés
		VF-868	Felségjelzés kivágásra került + ORC tévedés
		IPFH-423	OCR tévedés

3.1.4 Statisztika

- **84,8 %** -> 100% - os pontosság
- **7,6 %** -> 1 karakterrel többet észlelt
- **3 %** -> 1 karakterrel kevesebbet észlelt
- **1,5 %** -> 83% - os pontosság
- **1,5 %** -> 50 % - os pontosság
- **1,5 %** -> 0 % - os pontosság