



GKLB INTM038

Gépi látás

Rendszámtábla felismerés
beadandó

2021/22 2. félév

Készítette:

Gacs Péter (BSBMQW)

Tartalomjegyzék

1. Bevezetés	4
2. Elméleti háttér	4
2.1 Használt technológiák	4
2.1.1 Python	4
2.1.2 OpenCV	4
2.1.3 Python-Tesseract	5
2.2 Rendszer felépítése	5
2.3 Használt technikák	6
2.3.1 Geometriai transzformációk	6
2.3.2 Canny éldetektor	6
2.3.3 Harris sarokdetektor	7
2.3.4 Morfológiai szűrés	7
3. Megvalósítás	8
3.1 Előfeldolgozás:	8
3.1.1 Kép megnyitása és méretezése	8
3.1.2 Szürke árnyalat	9
3.1.3 Zajcsökkentés	9
3.1.4 Élek detektálása	9
3.2 Rendszámtábla detektálása	10
3.2.1 Kontúrok keresése	10
3.2.2 Maszk készítése	10
3.2.3 Sarokpontok detektálása	11
3.3 Rendszámtábla transzformálása	11
3.3.1 Sarokpontok meghatározása	12
3.3.3 Transzformáció	12
3.3.4 Treshold	13
3.4 Szövegfelismerés	13
3.4.1 Pytesseract	13
4. Teszt és értékelés	14
4.1 Teszt	14
4.2 Előnyök	14
4.3 Hibák összegzése	15
5. Használat	16
6. Irodalom	17

1. Bevezetés

A rohamosan fejlődő világunkban a személygépjárművek a hétköznapiak részévé váltak. Robbanásszerű növekedésükkel párhuzamosan megnövekedett az igény a gépjárművek azonosítására is. Minden gépjármű rendelkezik egy egyedi azonosítóval, amely gyakran egy alfanumerikus karaktersorozat, amit a forgalmi rendszám-tábla jelenít meg. A rendszám-tábla leolvasása a legegyszerűbb mód a járművek azonosítására, így meglehetősen nagy az igény a különféle rendszám-tábla leolvasó rendszerekre.

Napjainkban számos helyen használnak rendszám-tábla felismerő rendszereket, gondoljunk a rendőrök által használt traffipaxokra, vagy a parkolóházakra, esetleg az okosotthonokra. A rendszám-tábla felismerés témakörében fontos tisztázni, hogy a detektálás és az azonosítás nem egyenértékű fogalmak. A felismerés az a folyamat, amikor meghatározzuk a tábla pontos helyét, az azonosítás pedig amikor a rendszám-tábláról a számítógép által is értelmezhető karaktereket (sztring formátumban) nyerünk ki. A felismerés egy fontos momentuma a tábla detektálása, lokalizálása.

Összefoglalva a rendszám-tábla egy olyan számítógépes látástechnika, amellyel a járművek rendszáma kinyerhető egy digitális képből. A következőkben ezt a folyamatot mutatom be.

2. Elméleti háttér

Több technika is létezik a rendszám-tábla felismerésére. Az általam megvalósított rendszerben a következő technikák kerültek megvalósításra.

2.1 Használt technológiák

2.1.1 Python

A Python egy magas szintű, általános célú programozási nyelv, mely rendkívül dinamikus. A széleskörű elterjedését többek között köszönheti annak, hogy bővíthető, ingyenes, valamint egyszerű szintaxissal rendelkezik. Előnyei között élen szerepel a platformfüggetlensége, tehát használhatjuk különböző Unix változatokon, MacOS-en és valamennyi Windows változatokon is. A nyelv a korábban említett nyílt forráskódú technológiák táborát bővíti, ennek köszönhetően folyamatosan fejlődik a lelkes fejlesztők és felhasználók által.

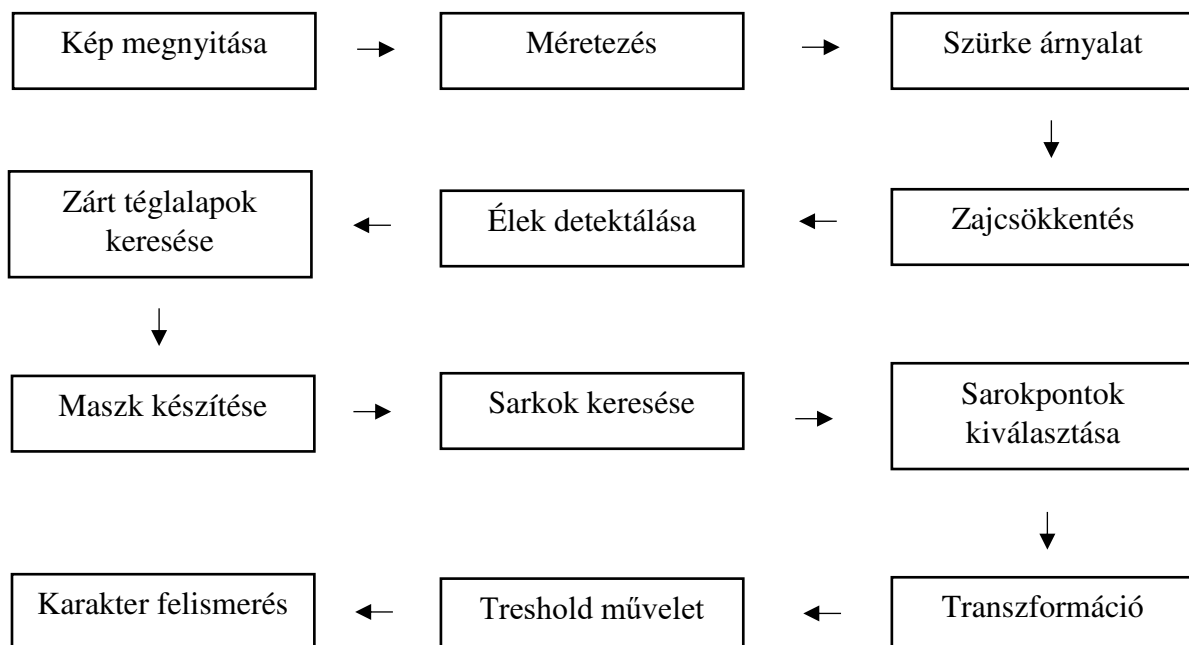
2.1.2 OpenCV

Az Open source Computer Vision röviden csak OpenCV, egy olyan nyílt forráskódú függvénykönyvtár, ami a számítógépes látásra lett kifejlesztve. A könyvtár C és C++ programnyelven íródott, és többek között Windows, Linux és Mac OS X operációsrendszerekkel kompatibilis. Elsődleges interfésze a C++, de fejlesztik több különféle nyelvekre, köztük Python-ra, Ruby-ra, valamint Matlab-ra is. Elsődleges céljai között szerepel, hogy a használói számára biztosítson egy egyszerűen használható infrastruktúrát a számítógépes látás témakörében, amely segít az alkalmazások gyors fejlesztésében. Az OpenCV könyvtár több mint 500 függvénnyel rendelkezik, melyek számos területet lefednek a számítógépes látás területén, ideértve az orvosi képalkotást, a robotikát és többek között a termékvizsgálatot a különféle gyárakban.

2.1.3 Python-Tesseract

Python-Tesseract egy optikai karakterfelismerő (OCR) eszköz a pythonhoz, vagyis képes felismerni és kiexportálni karakterek formájában a képeken lévő szöveget. A Python-Tesseract egy úgynevezett csomagolás (wrapper) a Google által fejlesztett Tesseract-OCR-hez. Több bemeneti formátummal is képes dolgozni, mint például JPEG, PNG, GIF, BMP file-ok:

2.2 Rendszer felépítése



2.3 Használt technikák

2.3.1 Geometriai transzformációk

A képek geometriai tulajdonságait mátrix műveletekkel könnyedén manipulálhatjuk.

- **Eltolás:** $T = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix}$

- **Forgatás:** $T = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

- **Nyújtás:** $T = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$

A Transzformációk szabadon kombinálhatók, a mátrixok szorzatát kell használni.

$$T = \begin{bmatrix} \cos\theta \cdot s_x & \sin\theta \cdot s_y & \cos\theta \cdot s_x \cdot x_0 + \sin\theta \cdot s_y \cdot y_0 \\ -\sin\theta \cdot s_x & \cos\theta \cdot s_y & -\sin\theta \cdot s_x \cdot x_0 + \cos\theta \cdot s_y \cdot y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

2.3.2 Canny éldetektor

A Canny az egyik legelterjedtebb élkereső módszer, leghatékonyabban szürkeárnyaltos képen. 5 lépésből áll:

- Gauss simítás: Cél: A képen fellelhető zajokat kiszűrjük. A módszer után a kép enyhén elmosódik.
A legáltalánosabb megoldási lehetőség a **konvolúció** használata, ami a képpontok egy környezetében elvégzett súlyozott átlag számítását jelenti.

Képlet: $J(x, y) = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} * I(x, y)$

- Differenciálszámítás: Cél: Megtalálni azokat a pontokat, ahol az egymás melletti pixeleknél éles intenzitás változások vannak.

Képlet:
$$\nabla I = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial I}{\partial x} & \frac{\partial I}{\partial y} \end{bmatrix}^T$$

- Nem-maximum vágás: Cél: nem maximális élek elnyomása. Egy lokális környezetben csak a legnagyobb gradiens magnitúdó érték marad meg.

$$J(x, y) = \begin{cases} I(x, y), & I(x, y) \geq I(x, y - 1) \wedge I(x, y) \geq I(x, y + 1) \\ 0, & \text{különben} \end{cases}$$

- Hiszterézis küszöbölés: Cél: élek kiválasztása. 2 küszöbérték kerül megjelölésre.
 - Ha magasabb az intenzitás mint a felső küszöb -> él
 - Ha alacsonyabb mint az alsó küszöb -> nem él
 - Ha a kettő között van az intenzitás -> akkor él, ha van a szomszédjában él.

A Canny éldetektor mindig egy binális képet ad vissza, az élek intenzitása 255, a többi pixel intenzitása 0

2.3.3 Harris sarokdetektor

A sarokdetektorok általában olyan pontokat keresnek a képen, ahol a gradiens minden irányban nagy. A Harris sarokdetektor egy "sarkosság" értéket jelez a képen, mivel nem minden pontról lehet eldönteni, hogy valóban sarokpont-e.

Csúszó ablak módszerrel térképezi fel a képet, Sarok pont ott lesz ahol az ablak bármilyen irányú elmozdulása esetén nagy képfüggvény változás áll elő.

$$R = \det(M) - \text{atr}(M)^2$$

- A vizsgált pont sarok, amennyiben $R > 0$
- A vizsgált pont él, amennyiben $R < 0$
- Amennyiben $|R| = 0$, a pont se nem él se nem sarok.

2.3.4 Morfológiai szűrés

Binális képeken alkalmazandó művelet, használatával alakzatok apró hibáit lehet korrigálni. Egy un. strukturáló elem (kernel) alapelve, annak folyamatos csúsztatásával hasonlítja össze a képpel. két alpművelete az **erózió** és a **dilatáció**.

- Erózió: A strukturáló elem minden hasznos pontja egybeesik a vizsgált kép adott helyén vett képpontjával.

- Dilatáció: A strukturáló elem legalább egy hasznos pontja egybeesik a vizsgált kép adott helyén vett képpontjaival.

Gyakorlatban az eróziót és a dilataciót együttesen alkalmazzuk.

- Morfológiai **nyitás**: binális zajok eltávolítása, vágása.
- Morfológiai **zárás**: hézagok, kis csatornák kitöltése.

3. Megvalósítás

3.1 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.

3.1.1 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()



1. ábra. Az optimális méretre való méretezés után

3.1.2 Szürke árnylat

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

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



2. ábra. Szürkeárnyaltossá alakítás után

3.1.3 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()*

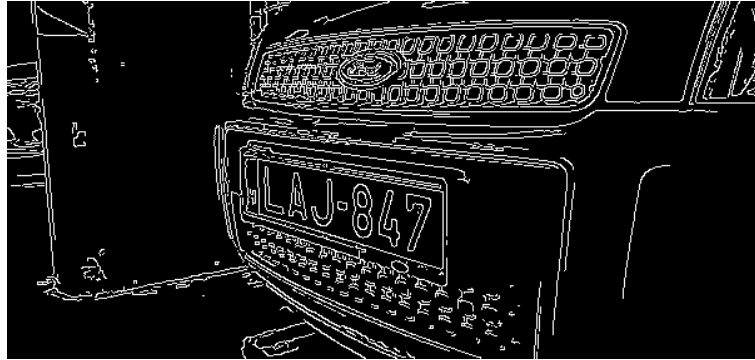


3. ábra. Zajcsökkentés után

3.1.4 Élek detektálása

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

Használt függvények: *cv2.canny()*



4. ábra. Éldetektálás után

3.2 Rendszámtábla detektálása

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

3.2.1 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()* módszer ezen alakzat pontjait adja vissza.

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

3.2.2 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()`



5. ábra. Maszk kivágása után

3.2.3 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.

3.3 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.

3.3.1 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.

3.3.2 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 ”

3.3.3 Transzformáció

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

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



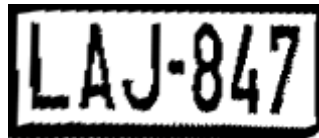
6. ábra A rendszám transzformálás után

3.3.4 Threshold

Az így kapott képet a jobb felismerhetőség érdekében érdemes alávetni threshold 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. *threshold()*

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



7. ábra. A rendszám binarizálás után

A **threshold** 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()* módszer végzi.

3.4 Szövegfelismerés

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

3.4.1 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.

3.4.2 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.

4. Teszt és értékelés

4.1 Teszt


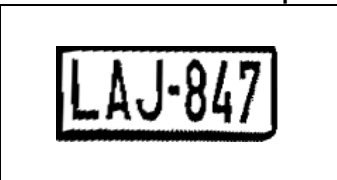

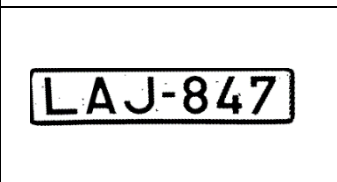

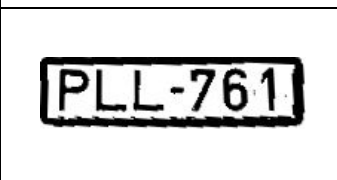

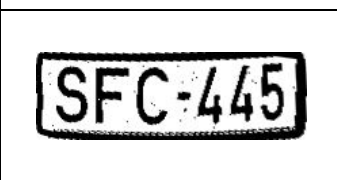
A programot 66 eltérő képen teszteltem, amik a *rszing* nevű mappában találhatók. A tesztet a *tester.py* nevű file tartalmazza. A tesztek eredményeit a *teszt.txt* file tartalmazza.


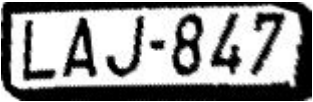

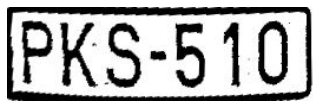
Statisztika:

- **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)

4.2 Előnyök

1. táblázat. Sikeres eredmények


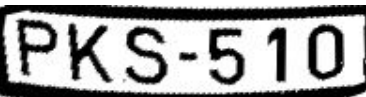





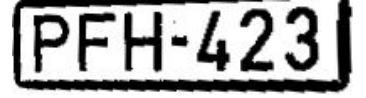
Eredeti Kép	Normalizált kép	Sztring	Következtetés
		LAJ-847	Éles szög
		LAJ-847	Ferde rendszám
		PLL-761	Vakuval készült kép
		SFC-455	Nem a kép közepén található a rendszám

		LAJ-847	Magas beesési szög
		PKS-510	Homályos kép

4.3 Hibák összegzése

2. táblázat. Hibás eredmények

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 + OCR tévedés
		VF-868	Felségjelzés kivágásra került + OCR tévedés
		IPFH-423	OCR tévedés

5. Használat

5.1 Szükséges file-ok:

- `licence_plate_detecting.py`
- `licence_plate_transform.py`
- `tesseract.py`
- `main.py`
- `tester.py`
- `test.txt`
- `dokumentation.pdf`
- `fail.jpg`
- rszimg mappa: 66 db kép file

5.2. Működés

A programot a **main.py** nevezetű file indítja.

A program egy sorszámot kér inputnak 1 és 66 között, amely a rszimg mappában található teszt képek sorszámára utal. Az adatok bekérése után visszaadja rendszám-tábla karaktereit sztringben, majd megnyitja az eredeti kép kicsinyített változatát ellenőrzés céljából. A kép bezárását követően a program befejezi a működését.

5.2.1. Működés feltétele

- Telepített python interpreter (3.7)
- OpenCV - python könyvtár (Numpy)
- Pytesseract könyvtár

6. Irodalom

- Swinnen, G.: Tanuljunk meg programozni Python nyelven. 2005.
<https://mek.oszk.hu/08400/08435/08435.pdf>
- Bradski, G., Kaehler, A.: Learning OpenCV: Computer vision with the OpenCV library, O'Reilly Media, Inc., 2008.
<https://www.bogotobogo.com/cplusplus/files/OReilly%20Learning%20OpenCV.pdf>
- https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html
- Széchenyi István egyetem, Gépi látás kurzus, elméleti diáor: Hollósi János
<https://drive.google.com/drive/folders/1CPEfeE7YhP7P6qwLKhcyfmK3XiNLRsCI>