# Gépi látás

## Dokumentáció

Készítette: Káldy Kristóf (R9ZHPM)

A tárgy hallgatói beadandójának egy rendszámfelismerő programot választottam. Ennek feladata ideális esetben egy tetszőleges bemeneti képen a fellelhető rendszám pozíciójának meghatározása, és a rajta található karakterek felismerése. Kimenetként a talált rendszámot kapjuk meg szöveges adatként és az eredeti képet, amelyen egy színes téglalapot helyezkedik el a rendszám körül.

A projekt Python-ban készült, a Standard Library-n(os, glob, timeit) túl szükséges még az OpenCV és a NumPy nevű könyvtárak telepítése.

- `pip3 install opencv-python numpy`

A NumPy segítségével a képeket többdimenziós tömbökként lehet kezelni, az OpenCV feladata pedig a képfeldolgozás mellett például a rendszámfelismerés template matching-gel.

A kép előkészítésének menete:

```
    # beolvassa a lehetséges képet az alfa csatorna eldobásával
➢   img_original = cv2.imread(image, 1)
    # szürkeárnyalatossá teszi a képet
➢   img_gray = cv2.cvtColor(img_original, cv2.COLOR_BGR2GRAY)
    # megnöveli a kontrasztot
➢   img_contrast = cv2.convertScaleAbs(img_gray, alpha = 1.25, beta = 0)
    # küszöbértékelés
➢   thresh = cv2.threshold(img_contrast, 0, 255, cv2.THRESH_OTSU)
    # kontúrvonalak keresése
➢   contours = cv2.findContours(thresh, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
```

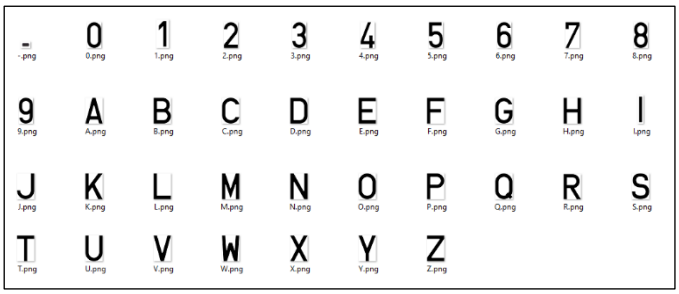

Balról-jobbra, fentről-lefelé: eredeti kép, szürkeárnyalat, megnövelt kontraszt, küszöbértékelés, kontúrozás

A megtalált kontúrokat először megszűri (eldobja aminek területe túl kicsi/túl nagy vagy nem eléggé téglalap alakú), így már viszonylag nagy pontossággal megtalálható a rendszámtábla.

```
for contour in contours:
    (x, y, w, h) = cv2.boundingRect(contour)
    area = w * h
    if ((2000 < area < 50000) and (w >= h * 2) and (w <= h * 6)):
```

Ezt követően a megmaradt ROI-kat (Region Of Interest) fix méretűre átméretezi és template matching-gel karakterfelismerést végez rajtuk.

A magyar rendszámtáblák betűtípusa HUN-DIN 1451, így telepítettem ezt a betűtípust és a kötőjelről, minden számról illetve nagybetűről készítettem egy képet.



A kód elején a `glob` modul segítségével betöltődnek a karakterek képei egy listába, amelyen később egy for ciklussal megyünk végig, hogy egy más után mindegyikkel elvégezhető legyen a template matching.

```
for character in characters:
    char = cv2.imread(character)
    res = cv2.matchTemplate(possible_plate, char, cv2.TM_CCOEFF_NORMED)
    threshold = 0.9
    loc = np.where(res > threshold)
```

A fenti kódrészlet működése részletezve:

1. beolvas egy karaktert (képként)
2. végig csúsztatja a karaktert a lehetséges rendszámtáblán
3. hasonlóságot keres, minél jobban egyezik annál nagyobb számot rendel hozzá (0-1 között)
4. ha a hasonlóság értéke megüti a megadott küszöbértéket elmentjük az adott képszakaszt

Valójában azonban nem egy kép, hanem csak a bal felső pixelének koordinátái kerülnek elmentésre a `loc` változóba, ezért ahhoz, hogy kirajzolhassuk a képre a talált karakterek helyét, további számításokra lesz szükség:

```
(width, height) = char.shape[::-1]
for point in zip(*loc[::-1]):
    top_left = point
    bottom_right = (point[0] + width, point[1] + height)
    cv2.rectangle(possible_plate, top_left, bottom_right, (0,0,255), 1)
```

Megfelelő találat esetén egy számláló, ami a talált karakterek számát tárolja megnő 1-el, továbbá a talált karakter neve és az oszlopának sorszáma is eltárolódik a `found_characters` nevű listába. A név megállapításához az `os` modul `path` függvényét használom:

```
for point in zip(*loc[::-1]):
    name = str(path.basename(character)).split(".")[0]
    col = point[0]
    found_characters.append((name, col))
```

Az elmentett karakterek azonban így ABC sorrendben vannak, még oszlop szerint sorba kell őket rendezni. Ecélból létrehoztam egy algoritmust ami mindig visszaadja a kapott változó második elemét, ami ebben az esetben az oszlop sorszáma. Ezt a Python saját lista rendezőjében kulcsként megadva növekvő sorba rendezhetjük a karaktereket:

```
def sortBySecond(element):
    return element[1]
found_characters.sort(key = sortBySecond)
```

Ekkor már csak az összehasonlítás van hátra, hogy a teljes rendszámot felismerte-e a program. Ezt úgy automatizáltam, hogy a bemeneti kép neve a rendszám, ami a képen található és a karakterek nevének beolvasásához hasonlóan itt is az `os.path.basename` függvényt használom.

## Tesztelés:


```
Várt rendszám:
NLE-003

Talált rendszám:
NLE-003

Rendszám sikeresen felismerve!

Számítási idő: 0.35345 másodperc
```

A program sikeresen felismerte az NLE-003 rendszámot


```
Várt rendszám:
EZVAN-9

Talált rendszám:
EZVA-

A rendszám felismerése sikertelen!

Számítási idő: 0.26926 másodperc
```

Az N és a 9-es karaktereket nem ismerte fel

Az egyes rendszámtáblák megtalálásához szükséges időt is méri a program. Ehhez a `timeit` modul `default_timer` függvényét használom, a végeredmény másodpercben van, 5 tizedes jegyre kerekítve:

```
start_time = timeit.default_timer()
[…]
stop_time = timeit.default_timer()
runtime = round(stop_time - start_time, 5)
```

## Felhasználói leírás:

A program futtatása után rögtön megjelenik az első beolvasott kép és a rajta talált rendszám. Ez után bármelyik billentyűt lenyomva automatikusan továbblép a következő fájlra, és így tovább amíg végig nem ér az összesen. A használt tesztképek a forráskód mellett lévő images nevű mappában találhatóak, ide igény szerint helyezhetünk el további képeket, de csak JPG kiterjesztéssel.

## Felhasznált weboldalak:


- https://pyimagesearch.com/
- https://stackoverflow.com/
- https://docs.opencv.org/3.4/index.html
- https://answers.opencv.org/questions/