

Gépi látás beadandó

Kocsis Dávid – WS0LM1

Téma: Rendszám tábla felismerés: Rendszám tábla detektálása és leolvasása fényképről.

Tartalomjegyzék

Bevezetés	1
Technológia.....	2
Algoritmus lépései	2
Fejlesztői környezet meghatározása és használt külső könyvtárak.....	2
Python	Hiba! A könyvjelző nem létezik.
OpenCV	Hiba! A könyvjelző nem létezik.
Tesseract-OCR.....	3
Nehézségek	4
Algoritmus	4
Teszttek	10
Összegzés.....	12

Bevezetés

Az automatikus rendszámtábla felismerő (ANPR) egy tömeges ellenőrzésre szolgáló módszer, amely optikai karakterfelismerést alkalmaz a képeken, így ismerve fel a járművek rendszámait. Felhasználása nem csak a közutakon, de még az egyetemünkön is elterjedt. Tipikus felhasználási célja a forgalom ellenőrzése autóúton, autópályán. Lehetőséget biztosít a szabálysértők (pl.: gyorsajtók) vagy körözött gépjárművek felkutatására. Egyetemünk belépési jogosultságokra

használja, rendszám tábla leolvasása után, illetve az adatbázisban történő egyeztetés után nyílik fel a sorompó és hajthatunk be a megadott helyekre.

Technológia

Az ANPR alapja a képmanipuláció. Első és talán legfontosabb lépése a rendszám tábla pozíciójának helyes detektálása. A rendszernek képesnek kell lennie egy képen meghatározni a rendszám tábla pontos helyét, ehhez egy állandót kell meghatároznunk, ami alapján tudjuk tervezni az algoritmust. Ez az állandó pedig nem más, mint a rendszám tábla alakja. Országoktól eltérően lehet más-más a rendszám táblán lévő karaktersorozat, tehát erre nem hagyatkozhatunk. Különböző autómárkákon más-más helyen fordulhatnak elő a rendszám táblák. Különböző célra használt járművek, vagy elektromos árammal hajtott autóknak pedig más színű a rendszám táblája (pl.: taxi – sárga, 25 vagy 50 km-t megtételére képes tisztán elektromosan hajtott gépjárműveknek – zöld, diplomata gépjárművek - kék). Az egyedüli állandó, amit érdemes a képen az algoritmusnak keresni, az a rendszám tábla alakja, ami téglalap alakú poligon. Amennyiben a képen talál olyan poligont, ami lehetségesen egy rendszám tábla, akkor végrehajtja rajta a karakterfelismerést és megkapjuk a rajta lévő szöveget.

Algoritmus lépései

1. Kezdeti kép beolvasása a fájlhoz vezető elérési út megadásával
2. Szürkeárnyaltos kép előállítás, majd zajcsökkentés és éldetektálás
3. Az éldetektálással kapott képen kontúr keresése és eltárolása
4. Lehetséges rendszám 4 sarokpontjának meghatározása
5. Maszkolás és rendszám tábla kivágása
6. Frontális kép meghatározása a kép átnyújtásával
7. Karakterfelismerés
8. Felismert karakterek visszaadása a felhasználó számára

Fejlesztői környezet meghatározása és használt külső könyvtárak

A projekt fejlesztéséhez az órán megismert Thonny nevű programot használtam. Ez az IDE képek módosítására, illetve ablakokban történő folyamatos megjelenítésére kiválóan alkalmas.

Programozási nyelvnek a már tanulmányaim alatt korábban megismert és alkalmazott Python nyelvet használtam a kifejezetten gépi látás projektek számára létrehozott külső könyvtárakkal.

Használt könyvtárak:

- Opencv-python
- Numpy
- Imutils
- pytesseract
- Matplotlib
- difflib

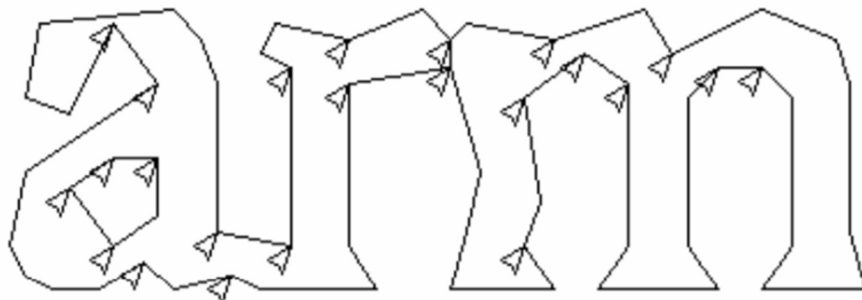
A könyvtárakból használt függvényeket a dokumentációmban később fogom kifejteni.

Tesseract-OCR

Vonalkereséssel kezd, a vonal részének gondolt pixeleket eltárolja, majd vonalhoz rendeli, így megkapjuk a karakterek pozícióját a képen. Ezután vízszintes, „párhuzamos” vonalakkal körül határolja a szöveget. Ez nem teljesen igaz, csak a képen látható fekete vonal egyenes, a többi csak közelít hozzá, enyhén görbítettek. Itt a különböző magasságú karaktereknél húz vonalat, így tudja megkülönböztetni számokat, nagybetűket és különböző karaktereket.



Ezután szétszedi karakterenként a már vízszintesen behatárolt szöveget. Következő lépés a lehetséges szavak megkeresése. Itt nagyobb „kihagyásokat”, vagyis szóközöket keres, így tudja behatárolni a szavak elejét és végét.



Töréspontok mentén az egybeérő karaktereket szétszedi. Ezek prioritási sorrendben vannak egy rekurzív függvénnyel folyamatosan meghívva, ha egy vágás nem éri el megfelelő eredményt, akkor visszavonja majd megnézi a prioritás szerinti következőt. A hiányos karaktereket ezután megpróbálja következtetni. Mivel már megvannak a karakterek is, ezért összeveti az adatbázisban lévő adatokkal, majd visszaadja a karakterláncot, amit leolvasott a képről.

Nehézségek

Számos nehézség lehetséges, amivel a szoftvernek meg kell tudnia birkózni. Ezek a következők:

- Gyenge képfelbontás nagy távolság miatt, vagy csak szimplán gyenge minőségű kamera miatt
- Elmosódott kép a mozgó autóról
- Kedvezőtlen fényviszonyok, árnyékok, tükröződések
- Több potenciális rendszám alakú poligon a képen
- Kicsi betekintési (rálátási) szög a rendszámra

Algoritmus

Projektem nulladik lépése a külső könyvtárak telepítése volt. Thonny IDE feltelepítése után a Tools menüpont open system shell funkcióját használva a következő parancsokat futtattam le a megfelelő környezet kialakításához:

```
pip install easyocr-python
```

```
pip install numpy
```

```
pip install imutils
```

```
pip install pytesseract
```

```
pip install matplotlib
```

Következő lépés ezeknek az importálása volt. Ehhez az import parancsot használtam. A pytesseract telepítése kicsit nehezkesebb lett a kelleténél, utólag kellett hozzáadni a PATH-hoz.



Kiinduló kép

A kód nagy részét egy nagy függvénybe ágyaztam be, ami paraméternek megkapja a képet és visszaadja a kinyert szöveget, ami remélhetőleg egy rendszámot tartalmaz. A kapott képen az első művelet, amit végrehajtottam, az a szürkeárnyaltosra alakítás volt. Ehhez a korábban importált opencv könyvtár cvtColor függvényét használtam, majd paraméterként megadtam a képet, illetve a COLOR_BGR2RGB paramétert, ami így sikeresen elvégezte a műveletet.



Szürkeárnyaltos előállított kép

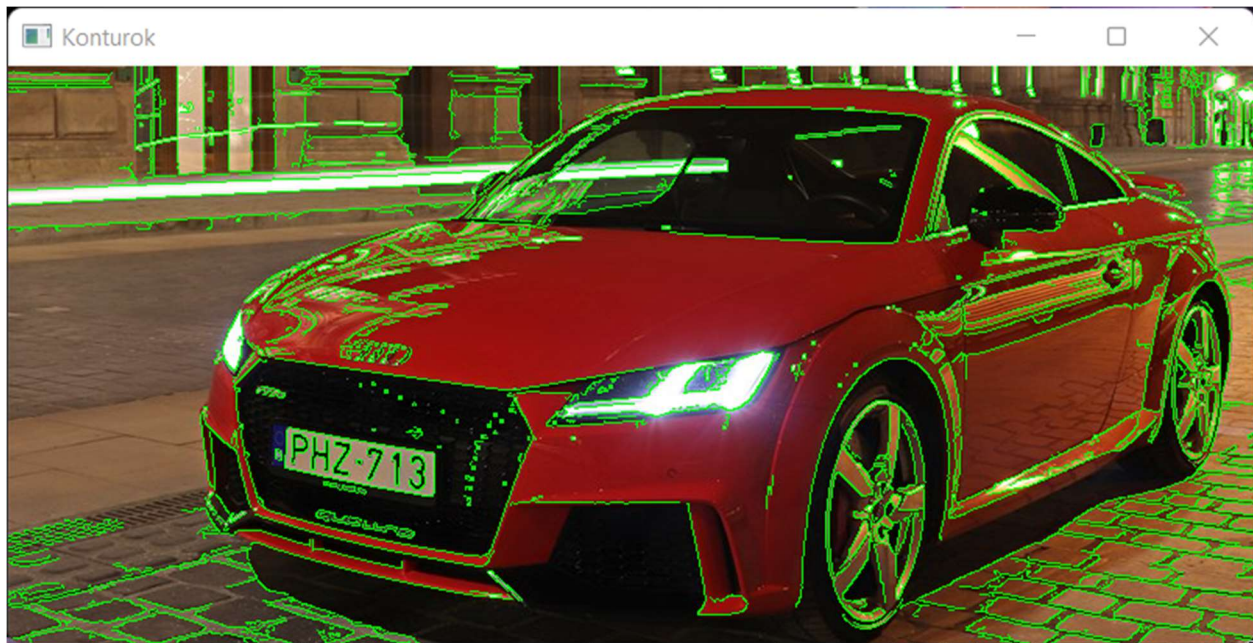
Ezután egy zajszűrést hajtottam végre a már szürkeárnyaltos képen, ehhez szintén az opencv könyvtárat használtam, azon belül is a bilateralFilter függvényt. Ennek paraméterként a képet kellett megadni, illetve három szám értéket, az én algoritmusomhoz a 11, 17, 17 kombináció passzolt a legjobban, de teszteltem még két-három másik kombinációt is. Ezeket különböző zajcsökkentéses projektekből vettem át.

Az órán tanult Canny módszerrel éldetektálást hajtottam végre a képen, ez gyönyörűen kirajzolta a fő éleket a képen.



Canny féle éldetektálás

Itt jutottam el ahhoz a ponthoz, hogy már nem kell a képen semmiféle változtatást elvégeznem, csupán meg kell keresnem a rendszámtábla pozícióját. Kontúrkeresés következett az edged képen. Ehhez a findContours függvényt használtam, aminek meg kellett adni a képet, de mivel ez módosítaná, ezért először le kellett másolnom. Első paraméterének ezt adtam meg (edged.copy()). Második paraméter a kinyerési mód, harmadik pedig a körvonal közelítési metódus. A CHAIN_APPROX_SIMPLE memóriamegtakarítási célokra szolgál, mivel nem az egész éleket, hanem csupán az élek végén lévő csúcspontokat rajzolja be.



Detektált kontúrok a kiinduló képre helyezve

A körvonalakat ezek után rendezzük a sorted függvény segítségével, ennek először megadjuk a kontúrokat, utána a rendezési szempontot a key funkcióval, majd a csökkenő sorrendet. Ebből pedig kiválasztjuk az első 10-et.

Létrehoztam egy location változót, aminek kezdeti értékét none-ra állítottam. For ciklus segítségével végig megyek a kontúrokat tartalmazó tömbön, majd, ha megvan a 4 sarokpont akkor a break kulcsszóval leállítom. Ezt a négy sarokpontot eltárolom a location változóban. Az approxPolyPD függvényt használtam a cikluson belül, ami közelíti a poligont a kontúrból, minél nagyobbra állítjuk a második argumentum-ot, annál jobban fogja egyenesnek kezelni a „recés” részeket.

Amennyiben kilépett sikeresen a for ciklusból a break paranccsal a program, akkor sikeresen eltárolt egy olyan körvonalat, aminek 4 sarokpontja van. Ahogy a dokumentáció elején is említettem, ez az állandó a különböző rendszám táblákban, ezért kerestem egy ilyen alakzatot. Ha egy rendszám tábla pl.: 6 szögű lenne, akkor olyan egybefüggő körvonalat kellene keresni, aminek 6 sarokpontja van, majd, ha megtalálta mind a 6-ot a program akkor szintén el kell tárolni és kilépni a ciklusból.

Mivel nekünk most megvan a 4 sarokpont, nincs más dolgunk, mint létrehozni egy képet, amin ez a 4 sarokpont által közbezárt terület ki van emelve valamilyen módon. Ehhez egy maszkot használtam, mivel megvannak a sarokpont koordinátái, ezért könnyedén létre tudtam hozni a rendszámtábla kiemelését. A numpy könyvtár segítségével létrehoztam egy nullákkal feltöltött maszkot. Létrehoztam egy `new_image` változót, aminek a `drawContours()` függvény segítségével megadtam a nullákkal feltöltött maszkot és a korábban meghatározott `location` változóban eltárolt koordinátákat.



Detektált poligon kiemelése a kiinduló képen

A numpy `bitwise_and()` függvényével a képre ráillesztette a maszkot, így a kiinduló képünkből most csak a rendszámtábla látszódik, de ez még nincs körbevágva, ezért a kép megtartotta az eredeti nagyságát, de a kép nagy része így fekete. Pár soros kóddal orvosoltam ezt a problémát, csupán a maszk legmagasabb, legalacsonyabb és két irányban legszélesebb pontját kellett venni, így minimalizálni tudtam a felesleges részt.



Maszkolt rendszámtábla

Ahhoz, hogy növeljem a hatékonyságát a rendszernek, a korábban kapott, kivágott, rendszámtáblát visszanyújtottam egy téglalappá, így a különböző éles szögből fotózott táblákon jobban érzékeli a

karaktereket a pytesseract. Ehhez két függvényt írtam, az egyik az order_points, másik a four_point_transform.

Először az order_points függvényt kell meghívni (ez a four_point_transformon belül kerül meghívásra), majd a location változóban eltárolt sarokpontok koordinátáit átadni neki. Ehhez először egy pts változót hoztam létre, amely egy np.array típusú változó és float32 értékeket tárol. Ennek a függvénynek a lényege, hogy egy rect tömbbe visszaadja a megfelelő sorrendben a koordinátákat. A sorrend a következő: bal felső sarok, jobb felső sarok, jobb alsó sarok és végül a bal alsó sarok. Miután ezeket a pontokat már jó sorrendben visszaadta a four_point_transform függvénynek, a következő képlet segítségével kiszámolja az új téglalap szélességét és magasságát:

Leghosszabb szélesség meghatározása:

Hasonló képlettel a megfelelő koordináták beillesztésével ki tudjuk számolni a magasságot is,

$$AB = d = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2}$$



majd, ha ez megtörtént, akkor megvan a „keretünk”, már csak az opencv segítségével be kell illesztenünk a kivágott képünket.

Szemből nézetes rendszámtábla

Mivel most már a legélesebb, de még detektálható szögből készített képünket is át tudtuk alakítani frontális nézetre, így már nincs más hátra, mint leolvasni a rendszámtábláról a karaktereket. Ezt



megelőzően érdemes volt még egyszer alkalmazni a zajcsökkentést egy magasabb értékkel, hogy még sikeresebben járjunk. A pyesseract külső könyvtárát használtam a leolvasáshoz, annak is az `image_to_string` függvényét. Ezt a `-psm 11` konfigurációval használtam, aminek a lényege, hogy minden karaktert leolvas a kivágott képről. Vannak konfigurációk, amik egybefüggő karakterláncokat keresnek, de ezek nem adtak jó eredményt számomra. Miután leolvasta a karaktereket, eltároltam egy text változóban. Mivel minimalizálni akartam a leolvasás utáni hibákat, ezért a véletlenszerűen leolvasott karaktereket megpróbáltam törölni. Ezt a következőképp hajtottam végre. Meghívtam a `strip()` függvényt, ami törölte a szóközöket. Mivel minden rendszám csakis alfanumerikus karaktereket tartalmazhat, ezért egy `for` ciklussal végig iteráltam a kapott sztring karakterein és amelyik nem volt alfanumerikus, azt töröltem. Ezzel el lehetett kerülni a külföldi rendszámokban lévő kör alakra hasonlító karakterek leolvasását.



Példa külföldi rendszám táblára, amely félrevezetheti a leolvasást

Előfordult olyan hiba is, mint például kisbetű lett leolvasva a nagy helyett. Mivel minden rendszám tábla csak nagybetűt tartalmazhat, ezért az `upper()` függvénnyel a kicsiket nagygyá alakítottam. Ezután nem volt más hátra, mint visszaadni a leolvasott karakterláncot.

Tesztek

A korábban említett műveletek nagy része egy nagy függvényben voltak megírva, ami csupán a kinyert stringet adja vissza. Célszerű volt függvényben csinálni a teszt miatt, így ezután egy `for` ciklussal folyamatosan meg tudjuk hívni különböző képekre, amelyekből ki tudja nyerni a rendszámokat és összehasonlítani az általam leolvasott ténylegesen helyes rendszámmal. A gyűjtött rendszámaim fájl neveit átírtam a tényleges rendszámra. Ezek lesznek az elvárt értékek.

Első teszttemhez egy `test_script_different_cars()` függvényt írtam, ami az elején deklarált 3 listát, az egyik a `predicted_list`, itt lesznek sorban a leolvasott rendszámok, második a `known_plates`, ami a fájlok neveit tárolja és ezek egyben a helyes értékek is. Harmadik a `statistic` lista, ami pedig a két különböző lista elemeit veti össze és eltárolja százalékos egyezést. Itt lehetett volna más adattípust használni, például egy szótárt vagy tuple-t, ami szorosabb kapcsolatot biztosít a listák között, de a teszt szempontjából a listák is tökéletesen megfeleltek. Az automatizálás szempontjából egy txt fájlban tároltam el a képek elérési útját, így egy for ciklussal folyamatosan be tudtam olvasni soronként. A beolvasás során `egynumber_plate_text` változóban eltároltam a `numberplate_to_text` függvény által visszaadott leolvasott rendszámot, az `actual_plate` változóban pedig a helyes értéket. Mielőtt viszont ezt eltároltam volna, ezen végre kellett hajtanom egy darabolást, mert ez az elérési utat és a kiterjesztést tárolta, nem pedig csakis a rendszámot string formában. Ehhez írtam egy függvényt, ami paraméterként megkapta az `inputpath` változó értékét (ez a korábban txt fájlból beolvasott i. iteráció sora). Itt a `replace` függvényt használtam (kicseréli a régi szöveget, ami a karakterláncban megtalálható, az új szövegre), mivel egy mappában van az összes kép és az összesnek ugyan az a kiterjesztése, ezért könnyedén megadtam paraméternek a régi szöveget, ami a „rendszám kepek\\” volt, új szövegnek pedig egy üres sztringet: „”. Ugyanígy a „.jpg”-re is végrehajtottam majd visszaadtam a nyers rendszámot. Ezt eltároltam az `actual_plate` változóban, majd az `append` módszerrel hozzáadtam a `known_plates` listához, mint következő listaelem. A leolvasott rendszámot hozzáadtam a `predicted_list`-hez, így biztosítva, hogy egy indexen legyenek az egymással kapcsolatban lévő értékek, csak különböző listákban. Utolsó függvény amit írtam, a `similar()` függvény egy külső könyvtárat használ, nem mást mint a `diff`-et. Erre a két sztring összehasonlításához volt szükség. Meghívása után a `similar()` visszaadott egy 0 és 1 közti értéket, ezt felszoroztam 100-al majd kerekítettem 2 tizedesre, hogy szép százalékos értéket kapjak. Ezt az értéket hozzáadtam a `statistic` listához. Szemléltetés kedvéért minden egyes sorra kiírtam a kívánt rendszám táblát, a leolvasott rendszám táblát, majd a százalékos egyezést.

```
Exact plate number:  SGP414    Predicted plate number:  SGP414    Match:  100.0 %  
Exact plate number:  SIA909    Predicted plate number:  S1A909    Match:  83.33 %
```

A mainben csupán egy `statistic` változóra van meghívva a `test_script_different_cars` függvény, ez kiírja statisztikám átlagát.

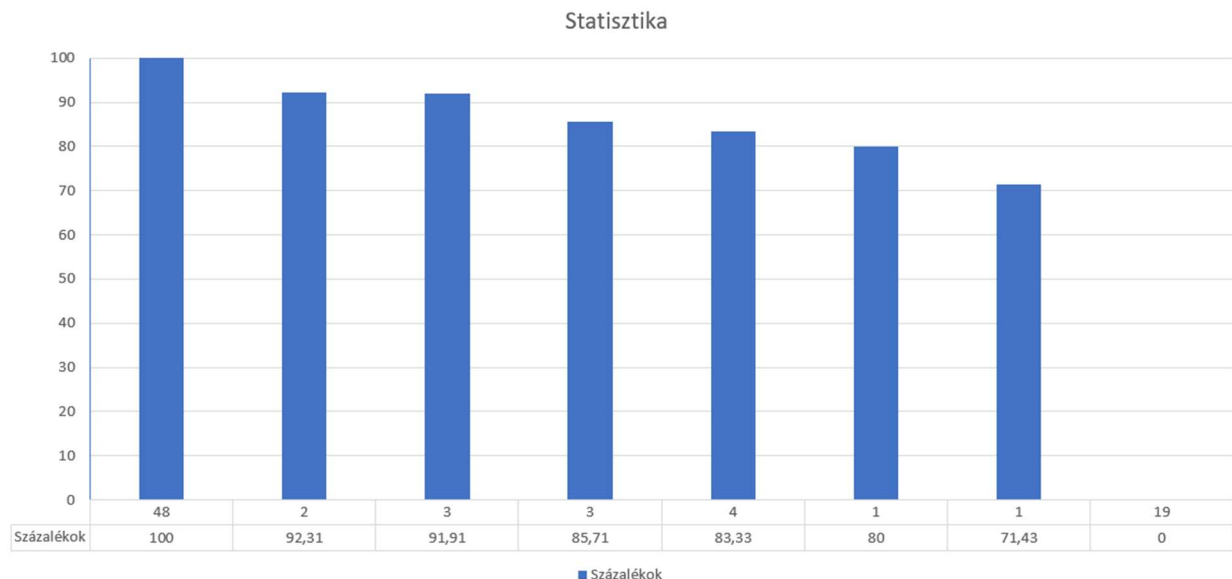
Percentage of 81 detected plates: 74.06 %.

Ez a teszt különböző gépjárművek leolvasására szolgált, de írtam egy másik jellegű tesztet is, ami egy és ugyan azon jármű rendszámát olvassa le, csak különböző szögekből. Ehhez egy hasonló függvényt írtam, mint a korábbi, csak mivel az összes rendszám ugyan az, ezért a fájlok nevét nem a rendszám adja, hanem egy növekvő számsorozat, így nem volt szükségem még egy txt-re. Egy for ciklussal könnyedén tudtam generálni sztringeket, amelyek az elérési utat tárolják egy listába.



Ezen a két képen látható igazán, hogy elengedhetetlen volt a rendszámtábla téglalappá való kinyújtása. Éles szögből is könnyedén jól látható rendszámtáblát sikerül létrehoznia a programnak. 16 különböző szögből 85%-os teljesítménnyel végzett a program.

Összegzés



A diagramról leolvasható a projekt eredményessége. Kiderül belőle, hogy a 81 leolvasott rendszámtáblából 48-at tudott 100%-osan leolvasni és ténylegesen helyes eredményt visszaadni.

Volt 19 eset, amikor 0%-os egyezést adott vissza, ez akkor fordul elő, hogyha nem tud semmit sem leolvasni a kivágott képről. Ebből arra lehet következtetni, hogy nem találta meg a képen a rendszámtáblát. Ha az egész adathalmazt nézzük, akkor 74,06% volt a hibátlanul leolvasott rendszámtábla. Érdekes vizsgálni azt az esetet, amikor nem vesszük figyelembe a 0 %-os értékeket. Ha ezeket mellőzzük és újra számoljuk az átlagot, akkor 96,76%-kal olvasta le hibátlanul a rendszámtáblát. Itt kiderül, hogy a programunk „gyenge” része a rendszám körvonalának megtalálása. Ez fejleszthető a későbbiekben úgy, hogy preseteket állítunk be, majd azokon a képeken, ahol nem talál rendszámtáblát, ott a különböző preseteket használjuk és újra hívjuk a függvényt. Ez a preset lehet a zajcsökkentő függvény argumentuma, Canny detektor paramétere, vagy csak nem az első 10 kontúrt nézzük (jelenleg így van beállítva), hanem többet.