

# Gépi látás beadandó

Az én választott témám a rendszám felismerés volt, a kiírtak közül. Ez egy képről történő leolvasást jelent úgy ,hogy a fotón életszerűek a körülmények tehát nem biztos hogy a rendszám úgy látszik ahogy azt terveznénk. Mindenféle szögből feltűnet nyilván a legoptimálisabb ha szemből van és nincs más objektum körülötte.

A feladat kivitelezéséhez szükséges ismerni a menetét egy ilyen folyamatnak , hogy milyen lépésekből érdemes összerakni illetve nyilván programozói tudást is igényel, én pythonban készítettem el. Érdemes ezen felül ismerni mindenféle kép konverziót amely segíti a kép zajmentesítését illetve amely javít az olvashatóságon , minőségen. De ennek interneten is utána lehet nézni esetleg másoktól ötletet meríteni. Az első probléma amibe ütköztem a „bővítmények” feltelepítése. Windowsra a pytesseract elég körülményesen beszerezhető elidőztem vele amíg működőképpessé tudtam varázsolni. De amint fent vannak a könyvtárak:

- cv2
- numpy
- pytesseract

kezdődhet a móka.

A lépések amelyek mentén sikerült elindulnom a következők:

- 1.A kép modifikációk a jobb detektálás érdekében
2. A képről a rendszám felismerése
- 3.A rendszám kivágása a képről
- 4.Karakter felismerés a levágott képről

Az alábbiakban majd látható a forráskód némi magyarázattal, amit kibővítenék végig menve az egyes sorokon. Így kifejtve a kódolás menetét, miértjét, illetve az előző 4 pontot.

*Az 1 lépés(kép modifikációk):*

A kép beolvasása az első ezt a cv2 segíti az cv2.imread funkcióval, itt megadjuk a pontos helyét, elérhetőségét a fájlnak és beolvassa onnan. Figyelni kell hogy kiterjesztés, minden stimmeljen , én meg is jelenítettem utána hogy jobban lássam mi történik pontosan ez főleg az első lépéseknél fontos hogy tudjuk merre haladunk.

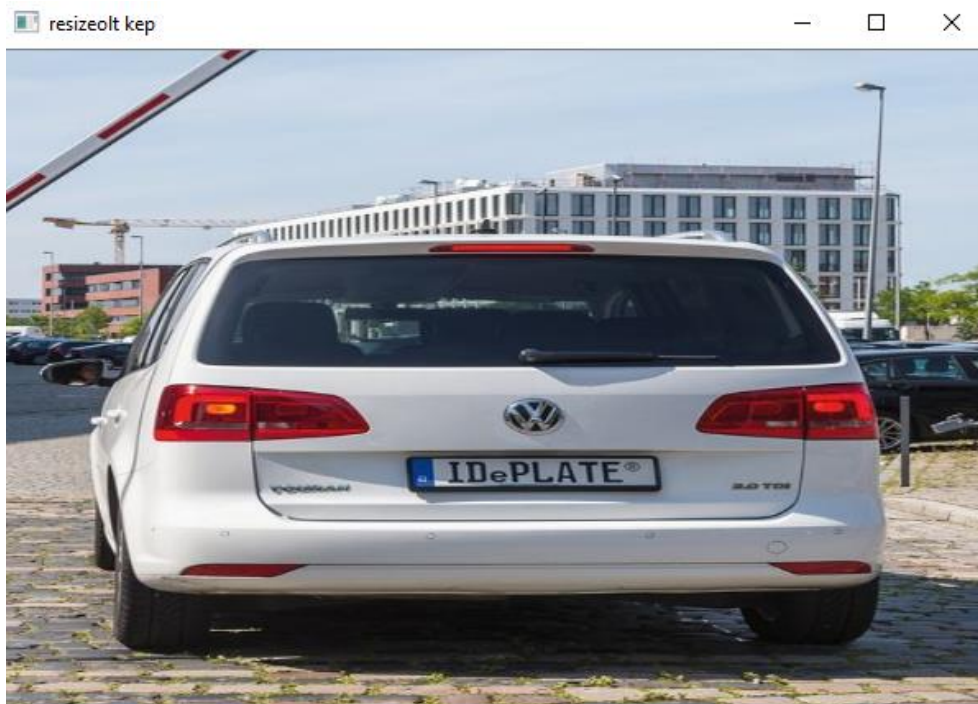
```
img = cv2.imread('D://tthibas.jpg') #kép beolvasás  
cv2.imshow('A sima kep',img)
```

A sima kep



Aztán a képet átméreteztem mivel ez segít a különböző felbontású képeket úgymond általánossá tenni illetve nem lesz több gondunk a nagy méretű képekkel. Itt is megjelenítettem hogy lássam a lépés eredményét, minden ilyen debugot segítő képmegjelenítésnek külön nevet is adtam hogy jobban felismerhető legyen a lépések sorozata.

```
img = cv2.resize(img, (600,410))#kep atmeretezes  
cv2.imshow('resizeolt kep',img)
```



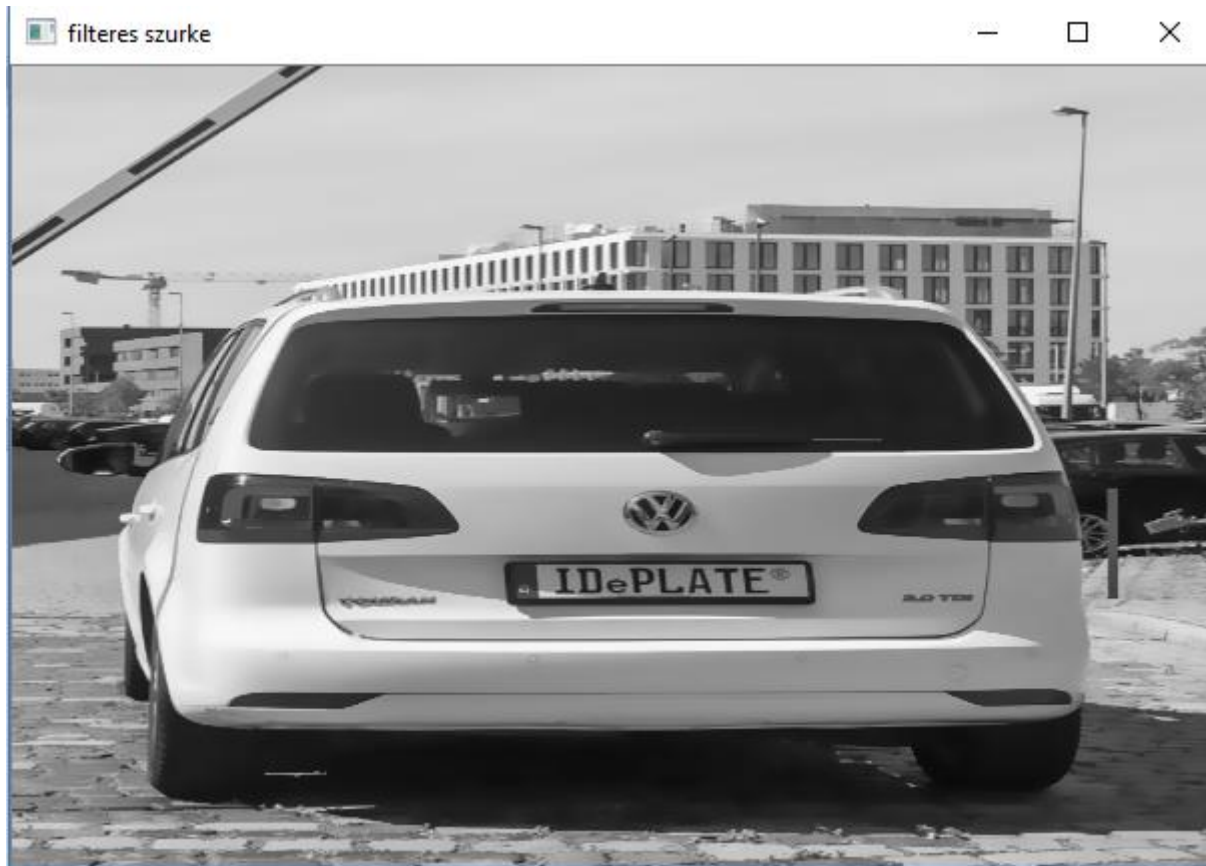
Ezután egy általános dolog a kép szürkeárnyalatosba konvertálása így nem kell többé foglalkoznunk a színekkel amelyek csak rontanák a kép felismeretőséget illetve kevesebb információval kell dolgoznunk ami gyorsítja a folyamatokat illetve könnyíti a munkánkat. Látszik hogy a konverzió után új nevet is adtam a képnek itt fontos hogy a név árulkodó is legyen arról amit csináltunk a követhetőség miatt.

```
szurke = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #a színek atkonvertalasa RGBből szürkébe  
cv2.imshow('sima szurke',szurke)
```



A következő lépés elhagyható lehetne de javítja a felismeretőséget kicsit szóval jobb ha mindent bevetünk hogy minél sikeresebb eredményt kaphassunk. Ez a filter a zajok eltüntetésére való az élek megtartásával, a ártteret elmossa így a nem kívánatos kis élek jobban eltűnnek a képről. A változóknak 12,14,14 környékén kellene maradniuk lehet emelni az értékükön viszont lehet a képünk rovasára megy ha túlzásba vesszük.

```
szurke = cv2.bilateralFilter(szurke, 11, 16, 16)#filter a zajok eltüntetéséhez  
cv2.imshow('filteres szurke',szurke)
```



Végül utolsó sorban a szélek detektálásához értünk amely az utolsó módosításunk a képen ebbe a fejezetben. Itt a cv2 könyvtárból a cannyt használtam mert elég populáris. A paraméterei a következők: `destination_image = cv2.Canny(source_image, thresholdValue 1, thresholdValue 2)`. Azok az élek lesznek kirajzolva amelyek a minimum és a maximum közé esnek.

#### #szélek detektálása

```
edged = cv2.Canny(szurke, 10, 210)# Csak azok az élek amelyek intenzitási értéke a minimum és maximum értékek közé esik lesz kirajzolva  
cv2.imshow('sarkositott',edged)
```



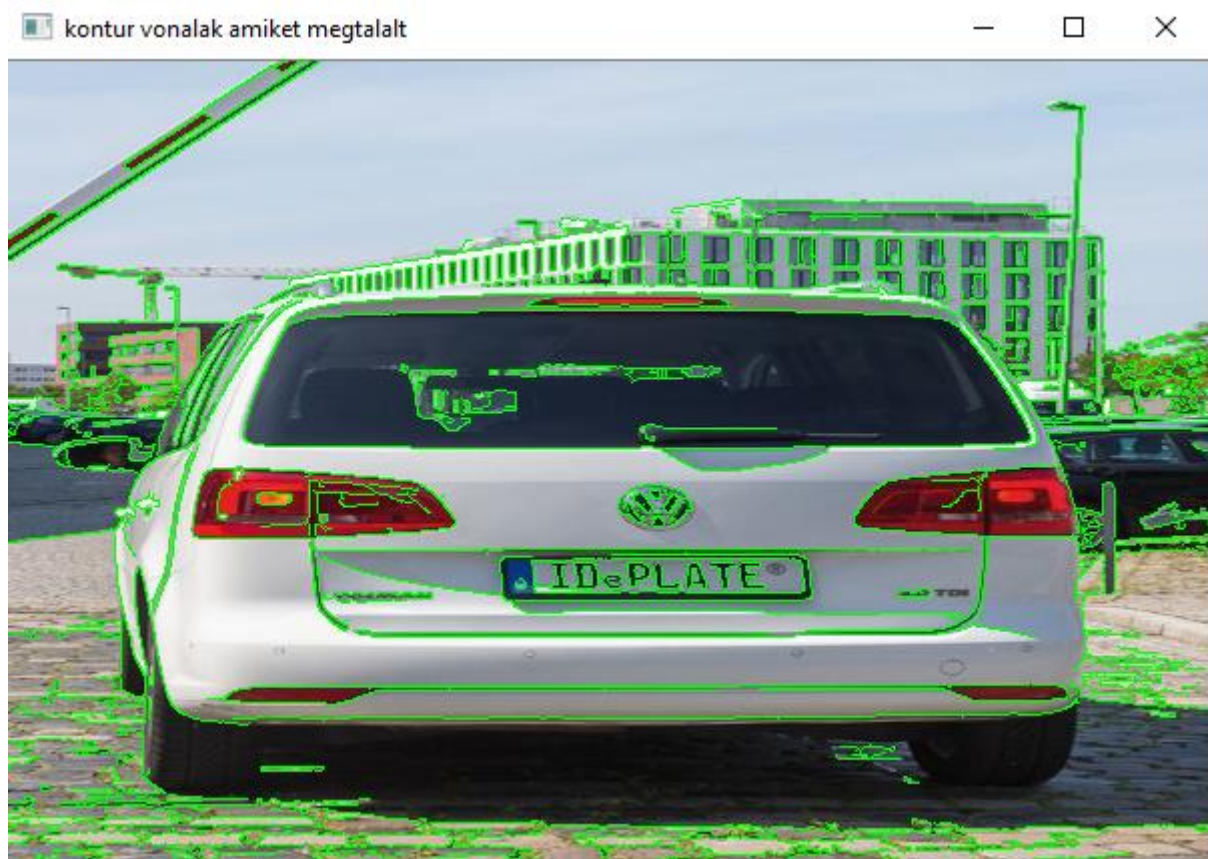
## 2.lépés (a képről a rendszám felismerése):

A kép másolatát `edged.copy()` használjuk mert a `findContours()` megváltoztatja magát a képet. `cv2.findContours` argumentumai: első a kép, amin dolgozni szeretnénk, második a körvonal kinyerési mód, harmadik a körvonal közelítési metódus. A hierarchia a kimenetnél megadja melyik körvonal melyikben található. A nekünk fontos változó az a `contours` néven elmentett numpy tömb, amelyekben raktározódnak a lehetséges rendszám jelöltek.

A `cv2.CHAIN_APPROX_SIMPLE` módszer pedig memóriát takarít meg hogy nem a határvonalakat rajzolja be hanem csak a sarokpontokat egy objektumnál. Én itt ki is írtam hogy mennyit talált érdekes képben. A kontúr vonalak szemléltetéséhez viszont nem elég egy `mezei print`, ezt `cv2.drawContours` metódus tudja nekünk megoldani amelynél szintén a kép másolatát kell alkalmaznunk amelyen alkalmazzuk, így az eredeti példány is megmarad sértetlenül. A `drawContours` paraméterei: a kép, a mentett kontúrok (körvonalak), -1 tehát az összeset kirajzolja amúgy azt a számot írjuk ide amelyiket szeretnénk hogy kirajzolja mínuszra az összeset kifogja.

```
contours,hierarchy= cv2.findContours(edged.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
print("Körvonalak száma:"+str(len(contours)))
kont1=cv2.drawContours(img.copy(), contours, -1, (0,255,0), 1)
cv2.imshow('kontur vonalak amiket megtalalt',kont1)
```





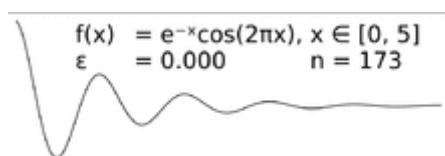
A körvonalakat ezek után rendezzük a sorteddel. Argumentumai a következők: első a körvonalakat tartalmazó tömb, második a rendezési szempont, harmadik hogy csökkenő vagy növekvőbe szeretnénk-e rendezni. Én nagytól kicsikig rendeztem és a 7 legnagyobb tartalmazza. A végén található [:7] pedig a maximális számát jelöli körvonalakból ami a tömbben lehet.

```
contours = sorted(contours, key = cv2.contourArea, reverse = True)[:7]#körvonalrendezés nagytól  
kicsikig és csak az első 7et nézzük
```

Először is mivel for ciklusunk lesz és van egy bool változónk azt nullára állítjuk.  
found = None # 0 találattal kezdünk

Ahhoz hogy kiszűrjük a rendszámot végig kell iterálni a találatok között és leellenőrizni melyiknek van téglalap alakja (négy körvonalból áll és zárt). Mivel ezek a rendszám jellemzői. Ebben segít még a len ami a tömbben lévő elemek számát megmondja. Ha négy elemből áll akkor valószínűleg megtaláltuk amit kerestünk.

A cv2.arcLength kerület kiszámítására való. A cv2.approxPolyDP pedig a körvonalak alakját közelíti egy másik alakhoz az általunk megadott pontossággal. Ez a Douglas-Peucker algoritmus megvalósítása. Az alábbi módon működik:



100 % No issues found

Locals

Search (Ctrl+E) Search Depth:

Name	Value	Type
c	array([[[[282, 248]], [[281, 249]], [[278, 249]], [[277, 250]], [[272, 250]], [[271, 249]], [[261, 249]]...])	ndarray
[0:48]	[array([[[[282, 248]], ...ype=int32], array([[[[281, 249]], ...ype=int32], array([[[[278, 249]], ...ype=int32], array([[[[277, 250]]...	list
dtype	dtype('int32')	dtype
max	397	intc
min	248	intc
shape	(48, 1, 2)	tuple
size	96	int
__internals__	{'T': array([[[[282, 281, 2...ype=int32]], 'base': None, 'ctypes': <numpy.core._interna...x050EBAC0>, 'data': <mem...	dict
contours	[array([[[[114, 202]],...ype=int32], array([[[[113, 203]],...ype=int32], array([[[[282, 248]],...ype=int32], array([[[[281, 24...	list
cv2	<module 'cv2.cv2' from 'C:\Users\xy\AppData\Local\Programs\Python\Python38-32\lib\site-package...	module
edged	array([[0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], ..., [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, ...,	ndarray
found	None	NoneType
hierarchy	array([[[[ 1, -1, -1, -1], [ 3, 0, 2, -1], [-1, -1, -1, 1], ..., [429, 422, -1, -1], [-1, 428, 430, ...	ndarray
img	array([[[[230, 210, 192], [230, 210, 192], [229, 209, 191], ..., [230, 210, 192], [230, 210, 192], [...	ndarray
kerulet	309.3553384542465	float
kont1	array([[[[230, 210, 192], [230, 210, 192], [229, 209, 191], ..., [230, 210, 192], [230, 210, 192], [...	ndarray
kozelit	array([[[[261, 249]], [[263, 265]], [[397, 264]], [[392, 248]]], dtype=int32)	ndarray
[0:4]	[array([[[[261, 249]], ...ype=int32], array([[[[263, 265]], ...ype=int32], array([[[[397, 264]], ...ype=int32], array([[[[392, 248]]...	list
dtype	dtype('int32')	dtype
max	397	intc
min	248	intc
shape	(4, 1, 2)	tuple
size	8	int

Autos Locals Threads Modules Watch 1

A fenti futtatás közbeni változó elemzésben látható hogy a c tömb 48 elemet tartalmaz. A közelít viszont csak 4et. Ebből leszűrhető hogy a sok mutatóból álló c tömb nem ad egy letisztult találatot, inkább több vonalból áll és követi a képen található objektum vonalát:



Míg az `cv2.approxPolyDP` után kapunk egy szemnek jobban tetsző letisztult megoldást:



for c in contours:

`kerulet = cv2.arcLength(c, True)`#kontúr kerület kiszámítás , masodik argument true tehat zart a körvonal

`kozelit = cv2.approxPolyDP(c, 0.02*kerulet, True)`#0.02 a pontosság mértéke (2%) minel nagyobb ez a szám annal biztosabb hogy eltérő objektumunk van mint amire számítunk

#ha 4 sarka van a korvonalunknak megtaláltuk valószínűleg a rendszámot



```

    if len(kozelit) == 4: #azok kiválasztása aminek 4 sarka van, len az elemben talalat darabszámmal tér
        vissza
        found= kozelit
        break

```

Ha nincs találat kilépünk.

```

if found is None: # ha nincs talalat
    detected = 0
    print ("A korvonal nem felismereto.")
else:

```

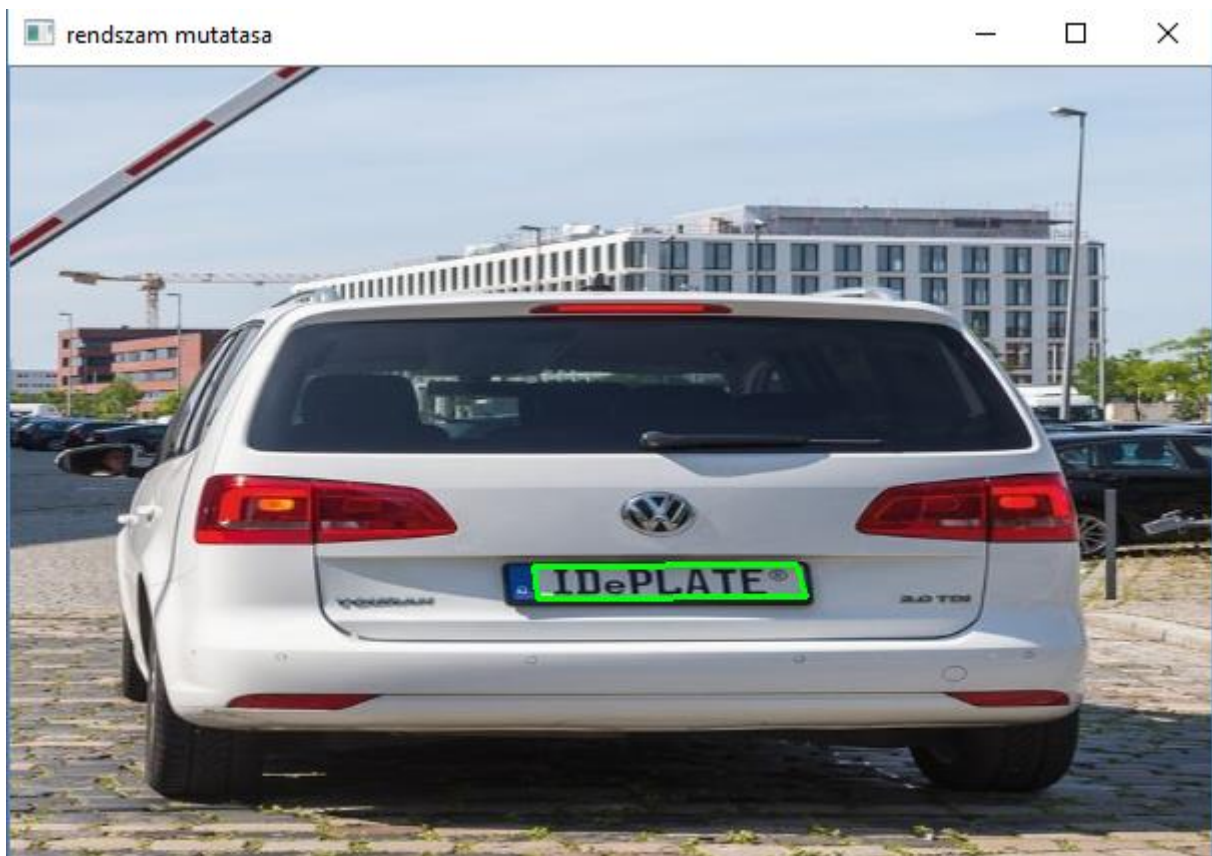
*3.lépés (a rendszám levágása a képről):*

Körbe rajzoljuk a talált objektumot:

```

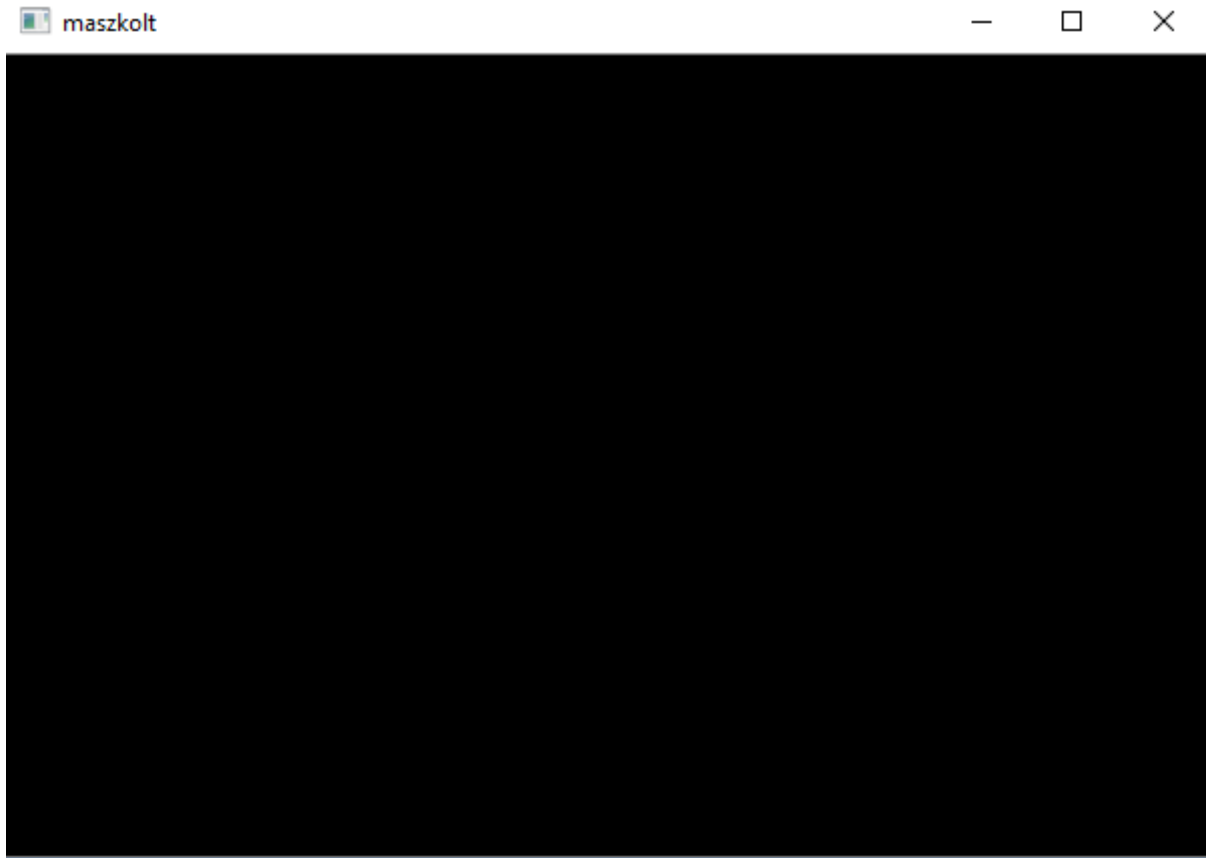
imgcpy=cv2.drawContours(img.copy(),[found],-1, (0, 255,0),2) #rendzam korberajzolása:-1 az összes
kontúr megrajzolását jelenti, zárójelben a szín , utolsó a vastagsága
cv2.imshow('rendszám mutatása',imgcpy)

```



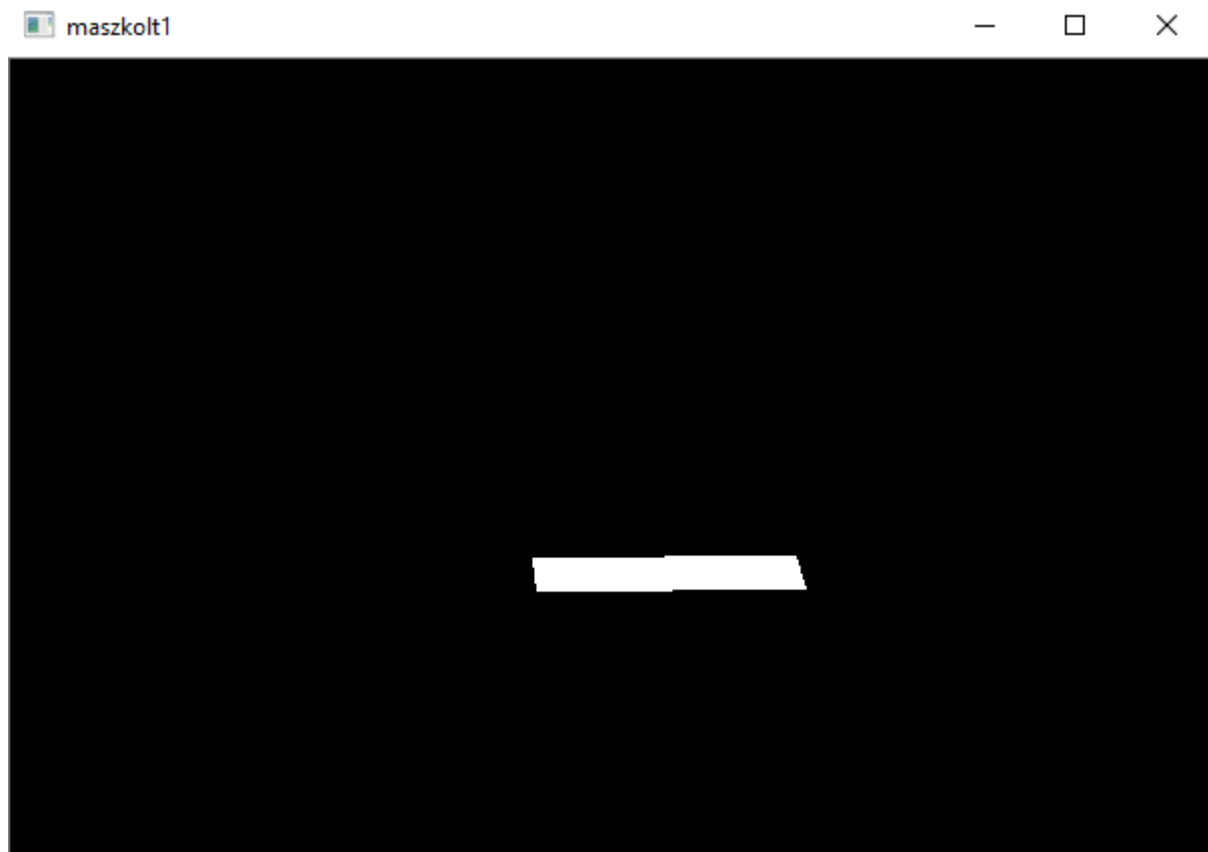
Ezután kezdődhet a maszkolás. Ez azért fontos mert a karakterfelismerőnek jobb csak azt adni amivel dolgozni kell különben hatástalan de még a tökéletes inputot is kap majd látunk rá példát hogy akkor sem garantált a sikeres végeredmény. Első lépésben maszkot feltöltjük nullákkal tehát egy fekete képet kapunk.

```
#maszkolása mindennek ami nem a rendszám  
mask = np.zeros(szurke.shape,np.uint8) #nullakkal feltoltes  
cv2.imshow('maszkolt',mask)
```

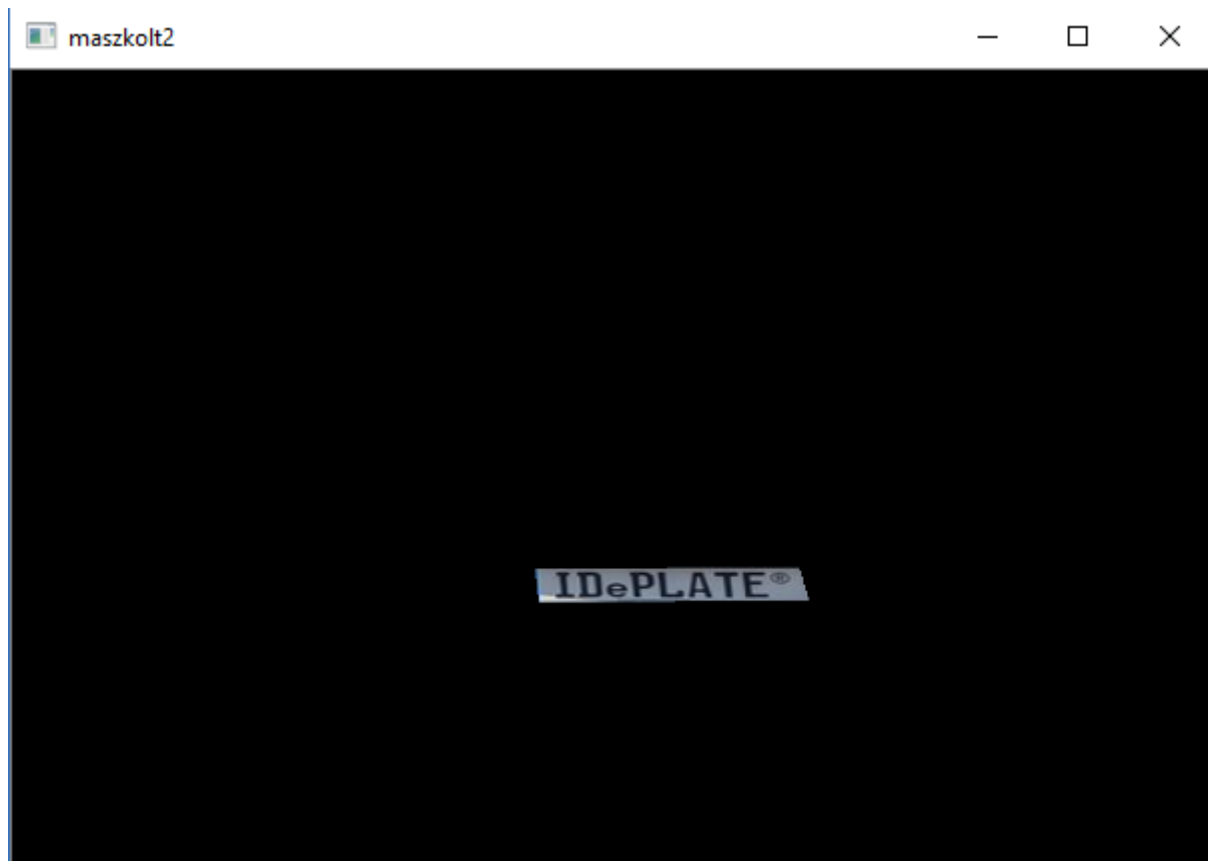


Majd fehérrel feltöltjük a rendszám helyét. Így a következő lépésben a `cv2.bitwise_and` egy pixelenkénti és művelet a fehér rendszám helyén maszkolt képet „éseli” össze az originál képpel aminek következtében a fekete 0 értékű pixelek nem kapják meg a rendes kép értékét mert 0-al szorozva értékük 0 marad.

```
new_image = cv2.drawContours(mask,[found],0,255,-1)#fehérrel a rendszámot maszkolja  
cv2.imshow('maszkolt+',mask)  
cv2.imshow('maszkolt1',new_image)
```



```
new_image = cv2.bitwise_and(img,img,mask)#és művelet pixelenként, tehát a fekete rész 0 a fehér 1  
,ezért a fehér részt tölti ki a rendszámmal  
cv2.imshow('maszkolt2',new_image)
```



Itt pedig kimentjük a rendszám pozícióját egy két dimenziós tömbbe. Szintén a maszkot és annak fehér színét kihasználva. Majd a minimum és maximum pozíciókat lekérjük és kirajzoljuk a levágott rendszámot `tól-ig` segítségével.

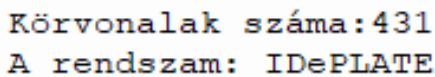
```
(x, y) = np.where(mask == 255) # mivel a maszkot módosítja a drawcontour ezért benne marad a
# két x, y numpy tömbbel elmenti a rendszám pozícióját
(minx, miny) = (np.min(x), np.min(y)) # min max pozíciók mentése
(maxx, maxy) = (np.max(x), np.max(y))
Cropped = szurke[minx+1:maxx+1, miny+1:maxy+1] # mintol maxig, így rajzoltat ki
cv2.imshow('cropped', Cropped)
```



Utolsó lépésben pedig következik a rendszám felismerés a levágott képről. Ezt a pytesseract bővítmény segítségével tettem meg. Itt megadható többféle konfiguráció is neki, én a 11est használtam ami a legtöbb szöveget keresi meg. Majd a legvégén felnagyítom a levágott rendszámot a jobb látatóság érdekében és megjelenítem. A `cv2.waitKey(0)` illetve `cv2.destroyAllWindows()` metódusok a képek megtekinthetőségét segítik.

```
#karakter leolvasas
text = pytesseract.image_to_string(Cropped, config='--psm 11')#11es config ->Sparse text. Find as
much text as possible in no particular order.
print("A rendszam:",text)
Cropped = cv2.resize(Cropped,(300,150))
cv2.imshow('auto',img)
cv2.imshow('vagott kep',Cropped)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

A végeredmény: 

A tesztelés eredményei random képekkel:

- 100db kép
- Valamiféle felismerés (hibás is és jó is): 50db
- Jól felismert rendszám, tökéletes működés (Valamiféle felismerésen belül): 21db
- A körvonal nem felismerhető: 50db

Következtetés levonás:

Magán a változókon, paramétereken lehetne még változtatni így némi próbálgatással, finomhangolással elérhető lenne a jobb hatásfok. Próbálgatásaim során sikerült rájönnöm hogy a kép méretezésével való „játszadozás” segít a rosszul felismert vagy nem felismerhetők javára. Viszont ronthat a jól felismerhetőkön így nem biztos hogy minden esetben jól járunk ha egy képnél optimálissá tesszük a lefutást a többi rovására.

## Felhasználói Útmutató:

`img = cv2.imread('D:/tthibas.jpg')` #kep beolvasas A kódsor ezen sorában az argumentumnál lehetséges az input megadása, ide a kép elérési útvonalát szükséges bemásolni, pontosan. A képnek nagyobbnak kell lennie mint 600\*400. Fullhd minőségű már elég jó. Majd ezután futtatni kell a kódot és a rengeteg visszajelző képnek köszönhetően végig követhető maga a folyamat is ahogy a kép feldolgozásra kerül, mindegyik képnek az egyedi címe biztosítja a követhetőséget. Szöveges



kimenetben ( a konzolon) pedig látható a Körvonalak száma, ami a talált összes körvonalat jelenti és a várt eredmény, a rendszám maga. Itt érdemes ellenőrizni hogy jó eredményt kaptunk-e.

```
Körvonalak száma:431  
pl: A rendszam: IDePLATE
```

Ha azt az üzenetet kapjuk a konzolban hogy „A korvonal nem felismereto” a rendszernek nem sikerült detektálni a megfelelő objektumot. Ezután a program kilép. Általában minél több részlet van a képen a rendszámon kívül annál biztosabb hogy nem találja meg , a szerkesztett vagy egyéb szövegek a képen nagyban ronthatják a felismerhetőséget. Egy gombnyomással bezárja a képeket ha végeztünk a nézelődéssel.