

Проектна задача

“Деткција на регистарски таблички и препознавање на карактери”

по предметот “Дигитално процесирање на слика”.

Изработил:

Андреј Влаховски 211136

Ментор:

Проф. д-р Ивица Димитровски

ФИНКИ, септември 2023



Универзитет „Св. Кирил и Методиј“ во Скопје
**ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И
КОМПЈУТЕРСКО ИНЖЕНЕРСТВО**

Содржина:

1. Вовед.....	3
2. Користени библиотеки.....	4
3. Вчитување на слика.....	6
4. Noise reduction.....	6
5. Edge Detection.....	9
6. Барање на правоаголник.....	11
7. Креирање маска.....	13
8. OCR.....	15
9. Можни грешки.....	17
10. Целосен код	18
11. Користена литература.....	19

1. Вовед

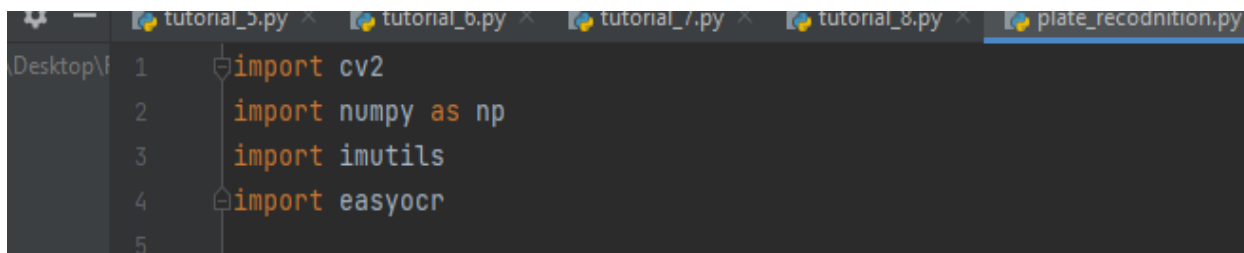
Детекција на регистарски таблички или “License Plate Recognition (LPR)” е напредна технологија која наоѓа разновидни примени во областа на управувањето со сообраќајот, безбедноста и автоматизацијата. Во светот што се поврзува се побрзо, способноста автоматски да се фотографира и интерпретира информација од регистарските таблички од слики станува неценливо богатство. Во овој контекст, претставената Python апликација ги прикажува сложените, но изразито ефикасни чекори за детекција на регистарски таблички.

Во оваа проектна задача ќе ги истражime основните компоненти на оваа апликација за детекција на регистарски таблички во детали, почнувајќи од земањето и предпроцесирањето на слики, потоа ќе ги разгледаме намалувањето на шумот, детекцијата на рабовите, анализата на контурите и техниките за препознавање на знаци, откривајќи го сложениот механизам на работење на секој чекор.

2. Користени библиотеки

Овој проект е изработен во програмскиот јазик Python, но не би можеле ништо да постигнеме без користење на надворешни готови кодови, односно библиотеки.

Нашиот код започнува со вчитување (import) на следните библиотеки: OpenCV, NumPy, Imutils, EasyOCR.

A screenshot of a Python IDE with a dark theme. The top of the window shows several tabs: 'tutorial_5.py', 'tutorial_6.py', 'tutorial_7.py', 'tutorial_8.py', and 'plate_recognition.py'. The 'plate_recognition.py' tab is active. The editor shows the following code:

```
1 import cv2
2 import numpy as np
3 import imutils
4 import easyocr
5
```

Cv2 / OpenCV

Оваа библиотека е многу популарна за компјутерска визија и процесирање на слики. Вклучува функции за израмнување на слики (image smoothing), пронаоѓање на рабови (edge detection), филтрирање на слики (image filtering) и други. Ова им овозможува на програмерите да манипулираат и подобруваат сликите за разни примени и потреби.

NumPy

NumPy (Numerical Python), е библиотека за програмски јазик Python која обезбедува многу функции и операции за манипулација со низи (arrays) и матрици, како и математички функции за обработка на податоци. Оваа високо ефикасна библиотека е оптимизирана за брзина при обработка на големи количини податоци. Бидејќи сликите ги претставуваме како матрици оваа библиотека ни е многу битна и корисна во оваа програма.

Imutils

Оваа библиотека е создадена за да ја упрости работа со OpenCV. Таа обезбедува збир на употребливи функции со цел да биде полесно да се изведуваат различни задачи за обработка на слики во областа на компјутерска визија, односно ја прави OpenCV полесна за работа на корисниците, односно програмерите.

EasyOCR

EasyOCR е библиотека за Python која служи за препознавање на карактери од слика. Лесна и едноставна за користење, а истовремено е брза и ефикасна. Повеќе и подетално за оваа библиотека ќе видиме понатаму.

3. Вчитување на слика

Најпрво вчитуваме една слика во случајов “car_5.png” преку функцијата `cv2.imread()`. Потоа мора да ја направиме сликата црно-бела бидејќи сите слики се многу полесни така за било какви обработки и манипулации. Тоа го правиме со методот `cv2.cvtColor()` и одбираме `cv2.COLOR_BGR2GRAY`, односно да се направи промена на пикселите од BGR (Blue, Green, Red) во GRAY, односно црно-бели.

```
7
8  img = cv2.imread('car_5.png')
9  gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
10
11
```

4. Noise Reduction (Намалување на шумот)

Намалувањето на шумот во обработка на слики се однесува на процесот на минимизирање или отстранување на случајни варијации во вредностите на пикселите или боите кои не претставуваат важна информација на сликата. Шумот може да ја отежни задачата за извлекување на делови од сликата или при изведување на точни анализи. Затоа, техниките за намалување на шум се суштински за подобрување на квалитетот на сликата и подобрување на резултатите на следните обработки на сликите.

За намалување на шумот се користат следниве техники:

- Smoothing Filters,
- Bilateral Filterin,
- Wavelet Denoising,
- Image Averaging,
- Non-local Means (NLM) Filtering,
- Deep Learning-Based Denoising,
- Adaptive Filters

За Деткција на регистарски таблички ќе ја користиме техниката на Bilateral Filtering (Билатерално филтрирање). Ова е посоефистицирана техника за намалување на шумот која ги задржува рабовите додека ги замаглува областите со слични бои. Таа ја зема во предвид и близината и сличноста на боите кога го пресметува просекот за секој пиксел. Ова е многу корисно кога е важно да се задржат деталите на сликата.

Билатерално филтрирање користиме преку функцијата `cv2.bilateralFilter()` од библиотеката OpenCV.

Првиот параметар е дијаметарот на пикселот во околината. Тоа го дефинира големината на регионот околу секој пиксел кој ќе се земе при применувањето на филтерот. Поголема вредност означува поголема околина. Во овој случај, 11 покажува дека ќе се земе предвид околина од 11x11 пиксели околу секој пиксел.

```
13
14     bfilter = cv2.bilateralFilter(gray, 11, 17, 17)
15
```

Вториот параметар е вредноста за спектар на бои. Тоа одредува како различните бои во околината се влезени во функција при филтрирањето. Поголема вредност означува поголем опсег на бои кои се сметаат за слични. Во овој случај, 17 е вредноста за бојата.

```
13
14     bfilter = cv2.bilateralFilter(gray, 11, 17, 17)
15
```

Третиот параметар е вредноста за позициониот простор (position space), односно близината на пикселите. Таа ја контролира распределбата на функција во просторниот домен. Слично на претходната вредност, поголема вредност за овој параметар резултира во поголема околина која се зема во предвид. Во овој случај, 17 е вредноста за овој параметар.

```
13
14     bfilter = cv2.bilateralFilter(gray, 11, 17, 17)
15
```

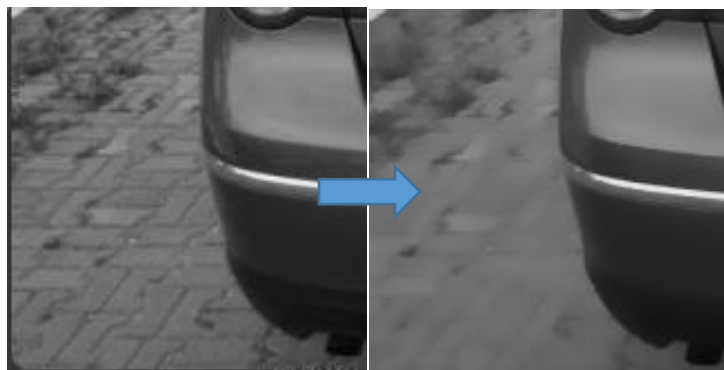
Билатералниот филтер го пресметува тежинскиот просек на пикселите во околината, земајќи ги предвид и просторната близина и сличноста на боите. Пикселите со слични бои и блиску едни до други ќе допринесуваат повеќе на просечната вредност, додека пикселите кои се различни во боја или се далеку оддалечени ќе имаат помало влијание. Ова резултира во намалување на шумот додека се задржуваат важните детали и рабови во сликата

Пример од слика пред и после аплицирање на методот за Билатерално филтрирање:



Пред намалување на шумот

После намалување на шумот



Пред намалување

После намалување



Пред намалување

После намалување

5. Edge Detection (Наоѓање на рабови)

Детекцијата на рабови во обработката на слики е фундаментална техника која се користи за идентификација на граници во слика каде се случуваат значителни промени во интензитетот на боите. Овие граници често одговараат на објекти или целини во сликата. Главната цел на детекцијата на рабови е да ги истакне или извлече тие рабови, додека се намалува влијанието на другите детали. Најкористени алгоритми за детекција на рабови се:

- Собел Оператор,
- Canny Edge Detector,
- Prewitt Operator,
- Laplacian of Gaussian (LoG).

Во оваа ситуација најдобро одговара Canny Edge Detector. Тој е познат по отпорноста на шум, со што се откриваат само значајните рабови. Работи на принцип на прво изгладнување (Smoothing), после бара нагли промени во градиентот на боите, се отргнуваат слабите рабови и за крај со “Hysteresis” се поврзуваат пикселите на рабовите со предвидување. Ова прави непрекинати рабови и ги елиминира изолираните пиксели.

Овој алгоритам го користме со повик на функцијата `cv2.Canny()`.

```
18  
19 edged = cv2.Canny(bfilter, 30, 200)  
20
```

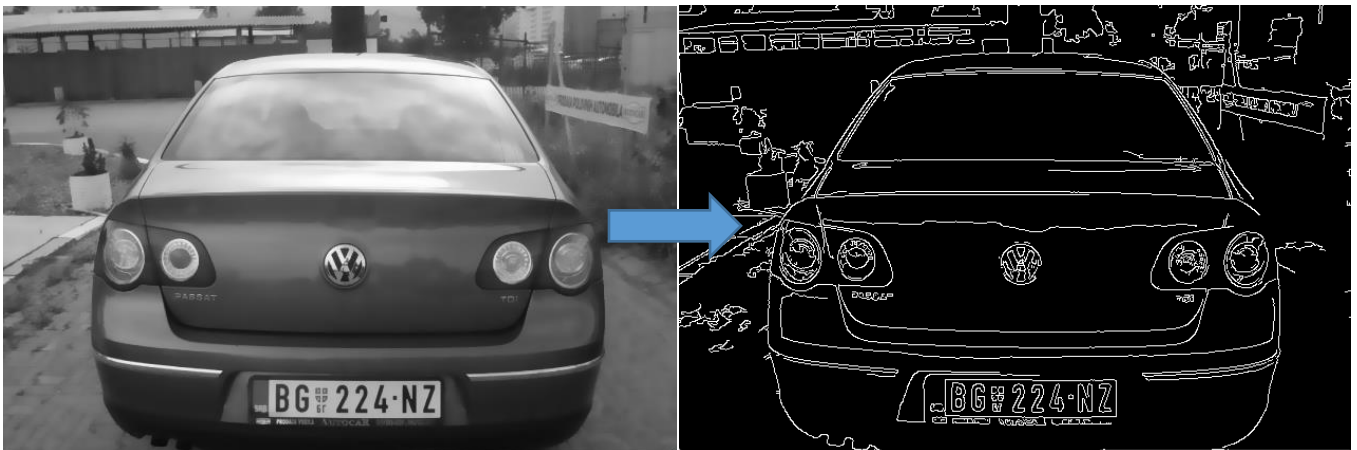
Првиот аргумент е ниската вредност на прагот за детекција на рабови. Алгоритмот Canny работи со детекција на рабови врз основа на промените во интензитетот или боите. Пикселите со промени во интензитет поголеми од овој праг се сметаат за потенцијални пиксели на рабови. Во нашиот случај тоа е 30.

```
18  
19 edged = cv2.Canny(bfilter, 30, 200)  
20
```


Вториот аргумент е високата вредност на прагот за детекција на рабови. Во Canny алгоритмот за детекција на ивици, пикселите со промени во интензитет поголеми од овој праг се сметаат за силни пиксели на рабови. Пикселите со промени во интензитет помеѓу ниската и високата вредност на прагот се сметаат за слаби пиксели на рабови.

```
18  
19 edged = cv2.Canny(bfilter, 30, 200)  
20
```

Пример од слика пред и после аплицирање на алгоритмот “Canny” за наоѓање на рабови:



Пред “edge detection”

После “edge detection”

6. Барање на правоаголник

Следен чекор за најдеме таблицата е да најдеме перфектен четириаголник на сликата. Тоа ќе направиме со неколку чекори. Најпрво треба да ги најдеме контурите во сликата со функцијата `cv2.findContours()`. Ова ни враќа листа од сите контури пронајдени на сликата според “keypoints”.

```
keypoints = cv2.findContours(edged.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

За од овие “keypoints” да ги добиеме вистинските контурни линии ја користиме функцијата од `imutils`: `imutils.grab_contours(keypoints)`

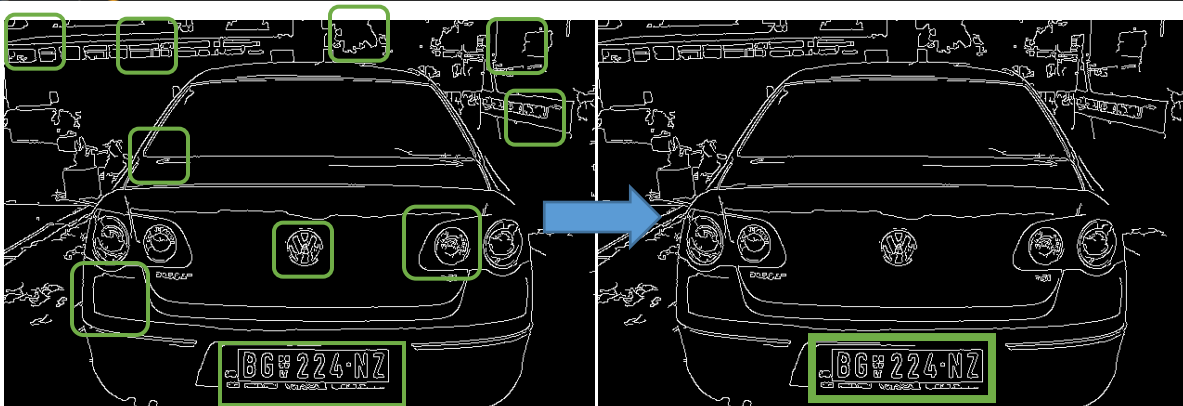
```
contours = imutils.grab_contours(keypoints)
```

После ова за да го олесниме процесот на избирање правоаголник ќе ги одбереме 100те најголеми контури. Ова го правиме со методот `sorted()`

```
contours = sorted(contours, key=cv2.contourArea, reverse=True)[:100]
```

Следно треба да ја најдеме најголемата форма која има 4 рабови. Тоа го правиме со следниот код:

```
27
28     location = None
29     for contour in contours:
30         peri = cv2.arcLength(contour, True)
31         approx = cv2.approxPolyDP(contour, 0.018 * peri, True)
32         if len(approx) == 4:
33             location = approx
34             break
```



7. Креирање на маска

Следно треба да направиме маска на најдениот четириаголник, односно на регистрацијата на автомобилот.

Најпрво иницијализираме црна маска со истите димензии како сликата. Функцијата `np.zeros()` од NumPy се користи за да се создаде низа исполнета со нули. Податочниот тип е специфициран како `np.uint8`, што значи дека секој пиксел во маската може да има вредности во опсегот `[0, 255]`. Почетно, сите пиксели во маската се поставени на 0, односно се црни.

```
35  
36 mask = np.zeros(gray.shape, np.uint8)  
37
```

После ова треба контурата дефинирана со променливата `location` (која претставува четириаголник) да се нацрта врз маската. Функцијата `cv2.drawContours()` го прави тоа со примање на неколку аргументи:

0: Индексот на контурата во листата. Во овој случај, има само една контура во листата, па нејзиниот индекс е 0.

255: Вредноста на бојата која сакате да се користи за цртање на контурата. Тука е поставена на 255, што одговара на бела боја.

-1: Овој параметар специфицира дека сакате да го исполните контурата со одредената боја. Практично, целата област затворена со контурата станува бела на маската.

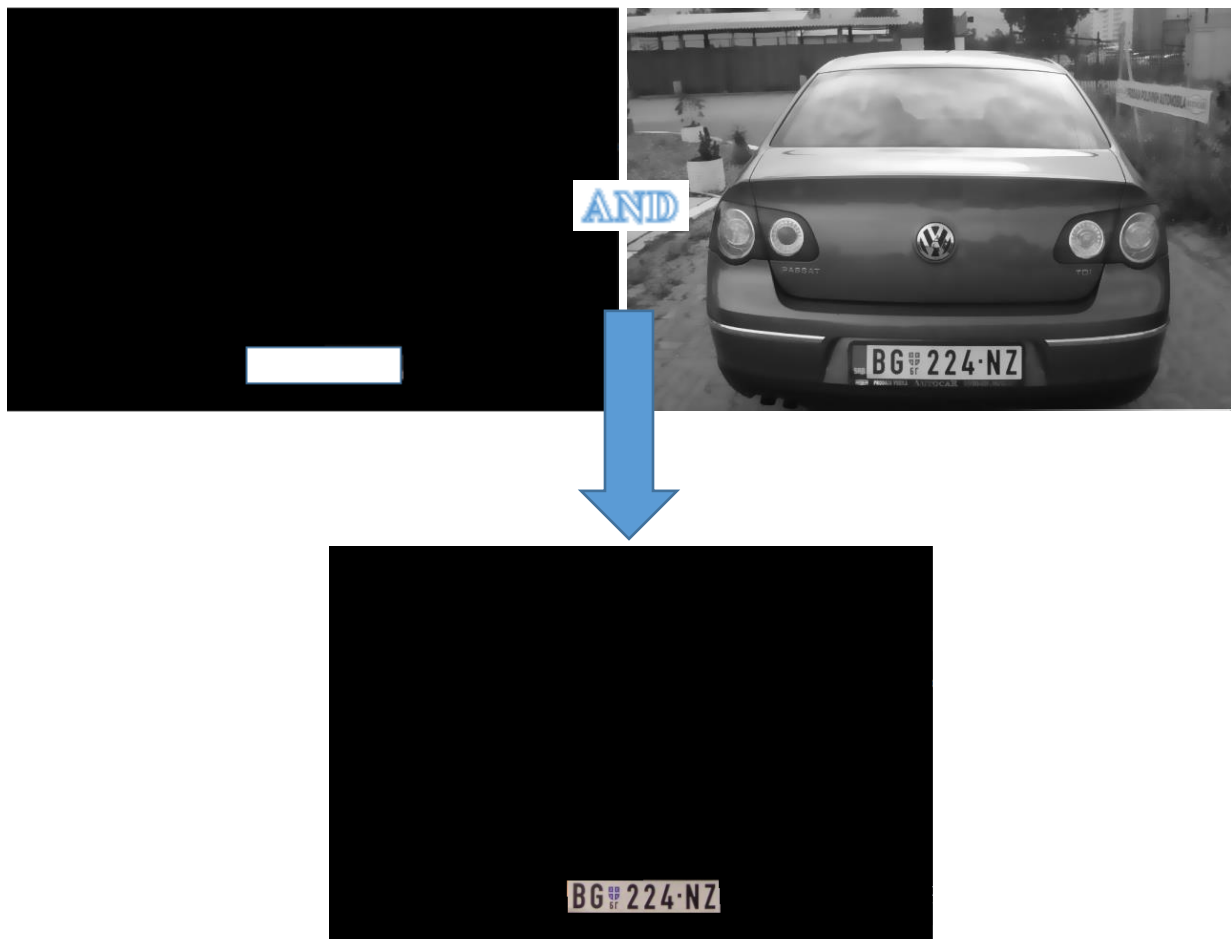
```
37  
38 new_image = cv2.drawContours(mask, [location], 0, 255, -1)  
39
```



Следно извршуваме операција на битово И (Bitwise AND) помеѓу оригиналната слика (img) и маската (mask) користејќи ја функцијата `cv2.bitwise_and()`.

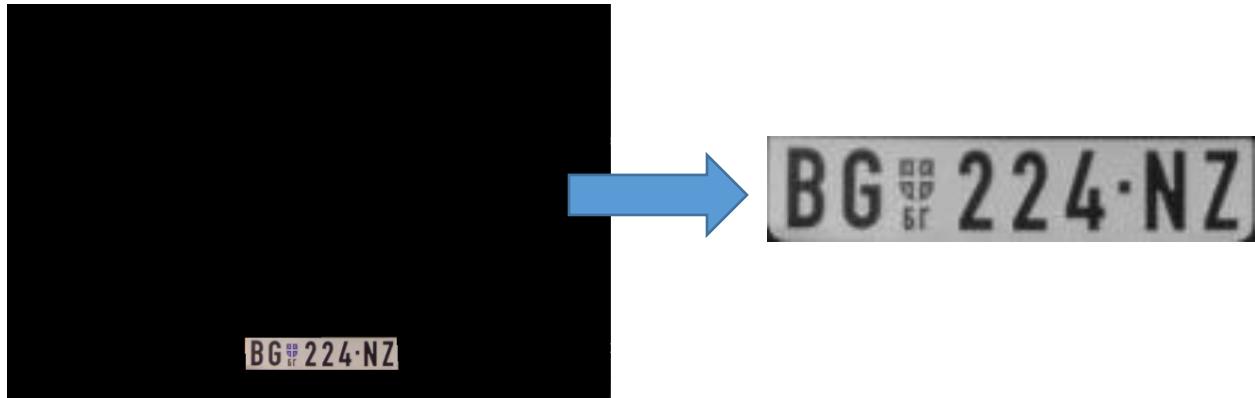
Резултатот од оваа операција е нова слика каде само регионот специфициран со маската е видлив, а останатата слика е црна. Практично, ја изолира областа од сликата која одговара на детектираната контура.

```
39  
40 new_image = cv2.bitwise_and(img, img, mask=mask)  
41
```



И за крај го отстрануваме црниот дел од сликата и ни останува само регистрацијата на посебна слика. Тоа го правиме со следниот код

```
43 (x, y) = np.where(mask == 255)
44 (x1, y1) = (np.min(x), np.min(y))
45 (x2, y2) = (np.max(x), np.max(y))
46 cropped_image = gray[x1:x2 + 1, y1:y2 + 1]
```



8. Optical Character Recognition (OCR)

OCR (Optical Character Recognition) е технологија која се користи за претварање на текст од различни видови документи, како скенирани хартиени документи, PDF датотеки или слики од дигитален фотоапарат, во компјутерски текст врз кој може да се прават измени и манипулации.

OCR софтверот ги лоцира регионите во сликата каде што се наоѓа текстот. Тој ги идентификува границите на поединечни знаци, зборови и пасуси и ги анализира облиците обидувајќи се да ги препознае. Тоа се прави со база на познати знаци. Напредни OCR системи користат алгоритми за машинско учење, како невронски мрежи, за да го подобрат квалитетот на распознавањето.

Некои од најпознатите OCR софтвери се:

- Adobe Acrobat OCR,
- Google Cloud Vision OCR,
- ABBYY FineReader,
- Tesseract OCR...

Бидејќи во нашава ситуација препознавањето на знаците е прилично едноставно ќе го користиме EasyOCR која ни овозможува лесно и ефикасно распознавање на текст од слики. Оваа библиотека е дизајнирана да биде едноставна за користење и нуди висока точност во распознавањето на текстот.

Првично иницијализираме OCR читач користејќи го EasyOCR. Создаваме инстанца на класата Reader и наведувате дека треба да се распознае текст на англиски (['en']). Ова значи дека OCR машината ќе се обиде да распознае и извлече текст на англиски јазик од сликата.

```
5
6 reader = easyocr.Reader(['en'])
7
```

Следно сликата која што содржи текст, се предава на OCR читачот (reader). Се повикува методот readtext() на класата Reader за да се обработи сликата и да се извлече секој текст што може да се пронајде.

Резултатот се зачувува во променливата result. Овој резултат обично содржи листа на региони со текст заедно со соодветниот распознат текст и координати на рамки околу текстот.

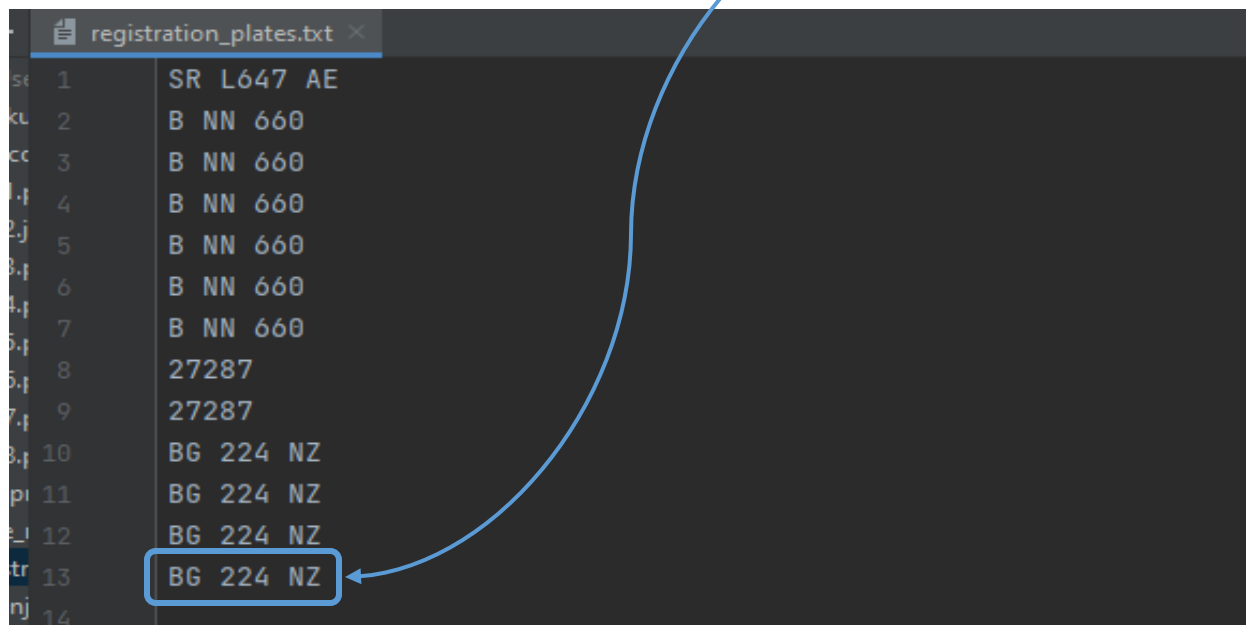
```
52
53 result = reader.readtext(cropped_image)
54
```

Понатаму овој резултат треба да го зачуваме во .txt документ односно во “registration_plates.txt”. Тоа го правиме со отворање на текстуална датотека наречена 'registration_plates.txt' во режим на 'допишување' ('append') и запишуваме текстуални податоци во неа, базирани на result (резултатот) што го добиеме од операцијата за оптичко распознавање на знаци (OCR).

```
54
55 with open('registration_plates.txt', 'a') as file:
56     for (_, text, _) in result:
57         file.write(f'{text} ')
58         file.write(f'\n')
59
```



BG 224 NZ



9. Можни грешки

9.1 Грешки во наоѓање на регистарските табlici

Вакви грешки настануваат во околу 10% од случаевите. Ова се случува кога во сликата ќе има некој објект поголем од регистарската таблица а со иста форма.



9.2 Грешки во OCR

Вакви грешки настануваат во околу 5% од случаевите. Ова се случува кога во таблицата ќе има некој објект кој личи на некоја буква или бројка е не означува ништо.



10. Целосен код на програмата

```
1 import cv2
2 import numpy as np
3 import imutils
4 import easyocr
5
6 reader = easyocr.Reader(['en'])
7
8 img = cv2.imread('img.png')
9 gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
10
11 ...
12
13
14 bfilter = cv2.bilateralFilter(gray, 11, 17, 17) # Noise reduction
15
16 ...
17
18 edged = cv2.Canny(bfilter, 30, 200)
19
20 ...
21
22
23
24 keypoints = cv2.findContours(edged.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
25 contours = imutils.grab_contours(keypoints)
26 contours = sorted(contours, key=cv2.contourArea, reverse=True)[:100]
27
28 location = None
29 for contour in contours:
30     peri = cv2.arcLength(contour, True)
31     approx = cv2.approxPolyDP(contour, 0.018 * peri, True)
32     if len(approx) == 4:
33         location = approx
34         break
35
36 mask = np.zeros(gray.shape, np.uint8)
37 new_image = cv2.drawContours(mask, [location], 0, 255, -1)
38 new_image = cv2.bitwise_and(img, img, mask=mask)
39
40 ...
41
42
43 cropped_image = gray[x1:x2 + 1, y1:y2 + 1]
44
45 ...
46
47
48 ...
49
50
51 result = reader.readtext(cropped_image)
52
53 with open('registration_plates.txt', 'a') as file:
54     for (_, text, _) in result:
55         # if 'e' in text:
56         #     text = text.replace('e', ' ')
57         file.write(f'{text} ')
58         # print(text, end="")
59     file.write(f'\n')
60
```

11. Користена литература

<https://courses.finki.ukim.mk/course/view.php?id=2261>

<https://www.fit.vut.cz/research/product-file/30/anpr.pdf?fbclid=IwAR1afB5UTEBRB8OPlwImd-ijTwGUKVwoeE33-rFM5-85LtdBAI1oyw2IdTo>

<https://academica-e.unavarra.es/xmlui/bitstream/handle/2454/7126/578015.pdf?sequence=1&isAllowed=y&fbclid=IwAR3MV6tj6EpdAJ8bTsAW4vwksmh3ENiWu7dGjny9tUKonJwRVzBBNYFZKjk>