Univerzitet u Kragujevcu

Master 4.0

Predmet: Obrada slike

Domaci zadatak 1

OpenCV with Python for Image and Video Analysis

Student: Kristina Satarić

20/2019

Profesor: red. prof. dr Nenad Filipović

Asistent: Tijana Šušteršič

Kragujevac, april 2020.

Sadržaj

Uvod i učitavanje slike – OpenCV u Python-u 3
Video 1. Intro and loading Images – OpenCV with Python for Image and Video Analysis 3
Učitavanje video izvora, uključivanje kamere računara i snimanje 4
Video 2. Loading Video Source – OpenCV with Python for Image and Video Analysis 4
Crtanje i pisanje na slici 6
Video 3. Drawing and Writing on Image – OpenCV with Python for Image and Video Analysis 6
Operacije sa slikom 8
Video 4. Image Operations – OpenCV with Python for Image and Video Analysis 8
Aritmetika i logika 9
Video 5. Image arithmetics and Logic – OpenCV with Python for Image and Video Analysis 9
Segmentacija pragom – Thresholding 15
Video 6. Thresholding – OpenCV with Python for Image and Video Analysis 15
Filtriranje boje 20
Video 7. Color Filtering – OpenCV with Python for Image and Video Analysis 20
Zamagljivanje i izoštravanje 22
Video 8. Blurring and Smoothing – OpenCV with Python for Image and Video Analysis 22
Morfološke transformacije 23
Video 9. Morphological Transformations – OpenCV with Python for Image and Video Analysis 23
Detekcija ivica i gradijenti 26
Video 10. Edge Detection and Gradients – OpenCV with Python for Image and Video Analysis 26
Podudaranje šablona 27
Video 11. Template Matching – OpenCV with Python for Image and Video Analysis 27
GrabCut ekstrakcija pozadine 29
Video 12. GrabCut Foreground Extraction – OpenCV with Python for Image and Video Analysis 29
Detekcija ugla 30
Video 13. Corner Detection – OpenCV with Python for Image and Video Analysis 30
Podudaranje odlika (Homografija), iscrpljujuća pretraga 31
Video 14. Feature Matching (Homography) Brute Force – OpenCV with Python for Image and Video Analysis 31
MOG redukcija pozadine 33
Video 15. MOG Background Reduction – OpenCV with Python for Image and Video Analysis 33
Haar kaskadno otkrivanje lica i očiju 34
Video 16. Haar Cascade Object Detection Face & Eye – OpenCV with Python for Image and Video Analysis 34
Izrada sopstvene Haar kaskade – Uvod 36
Video 17. Making your own Haar Cascade Intro – OpenCV with Python for Image and Video Analysis 36
Skupljanje slika za Haar kaskadu 36
Video 18. Gathering Images for Haar Cascade – OpenCV with Python for Image and Video Analysis 36
Čišćenje slika i kreiranje opisnih fajlova 37
Video 19. Cleaning images and creating description files – OpenCV with Python for Image and Video Analysis 37
Treniranje Haar kaskadnog objekta detekcije 38
Video 20. Training Haar cascade object detection – OpenCV with Python for Image and Video Analysis 38
Haar kaskada za klasifikaciju slika i video objekata 39
Video 21. Haar Cascade for image & video object classification – OpenCV w/ Python for Image Video Analysis 39
Reference 41

Uvod i učitavanje slike – OpenCV u Python-u

OpenCV (Open Source Computer Vision) je biblioteka programskih funkcija, uglavnom u realnom vremenu, prvobitno razvijena u Intelovom istraživačkom centrom u Rusiji, ranije podržavana od strane Willow Garrage, a sada podržavana od strane Itseez.
Biblioteka je međuplatformska, omogućeno korišćenje biblioteke različitim softverima, besplatna za korišćenje pod open-source BSD licencom [1].
Komanda za instalacija biblioteke OpenCV je pip install opencv-python.
Svi fajlovi se nalaze na linku: https://github.com/kristinasataric5552015/ImageProcessing

Video 1. Intro and loading Images – OpenCV with Python for Image and Video Analysis

Unutar fajla prikazanog na Slici 1. dodali smo smo biblioteke cv2, numpy i matplotlib.Učitali orginalnu sliku satic.jpg, koja je prikazana na Slici 2., zatim primenili cv2.IMREAD_GRAISCALE, koji učitava sliku u režimu sive boje. Alternativno, za ovaj mod može se pisati i celobrojna vrednost 0.
Rezultat primene koda se vidi na Slici 3.

```
8   import cv2
9   import numpy as np
10  import matplotlib.pyplot as plt
11
12  img = cv2.imread('satic.jpg', cv2.IMREAD_GRAYSCALE)
13  #druga opcija
14  #IMREAD_COLOR isto je ako stavimo samo 1
15  #za IMREAD_GRAYSCALE je dovoljno i samo 0
16  #IM_UNCHANGED isto je kao -1, tj. =-1
17
18
19  cv2.imshow('Prozorcic', img)
20  cv2.waitKey(0) #ceka da bilo koje dugme bude pritisnuto
21  cv2.destroyAllWindows() #cim bude pritisnuto unisti sve prozore
22
23  #DRUGI NACIN SA MATPLOTLIB
24  #plt.imshow (img, cmap= 'gray', interpolation='bicubic')
25  #plt.plot([50,100,], [80,100], 'c', linewidth=5)
26  #plt.show()
27  cv2.imwrite('satSIVI.jpg', img)
```

Slika 1. Učitavanje slike i Gray filte

Slika 2. 'satic.jpg'

Slika 3. 'satSIVI.jpg'

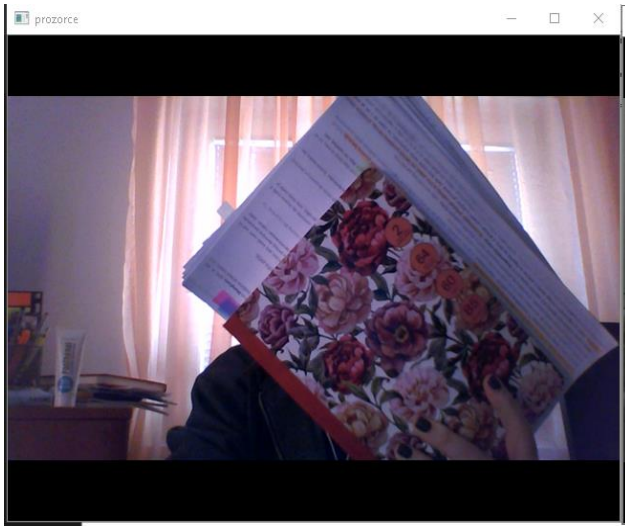
Učitavanje video izvora, uključivanje kamere računara i snimanje

Video 2. Loading Video Source – OpenCV with Python for Image and Video Analysis

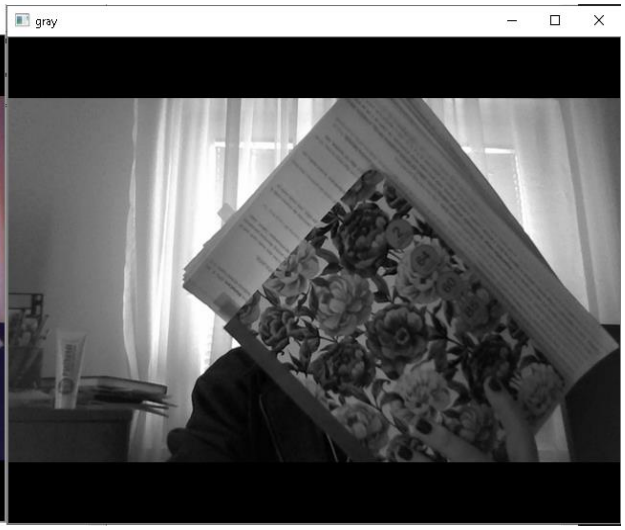
Unutar koda prikazanog na *Slici 4.* aktivira se kamera na računaru, vrši se filtriranje, tj prebacivanje u sivo. Rezultat koda je prikazan na *Slikama 5.* i *6.* .

```
 8   import cv2
 9   import numpy as np
10
11
12   cap = cv2.VideoCapture(0)
13   #0 znaci da ce biti prva video kamera u sistem
14
15   while True: #beskonacna petlja
16       ret, frame = cap.read()
17
18       #2.deo, prebacujemo u sivo
19       gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
20
21       cv2.imshow('prozorce', frame)
22       cv2.imshow('gray', gray)
23
24
25       #kako bi izasli iz beskonacne petlje
26       if cv2.waitKey(1) & 0xFF == ord ('q'):
27           break
28
29   cap.release()
30   cv2.destroyAllWindows()
```

Slika 4. Uključivanje kamere, bez promena i sivo



Slika 5. Izgled prozora 1



Slika 6. Izgled prozora 2

Pomoću funkcije VideoWriter, unutar biblioteke cv2 postoji mogućnost čuvanja videa na računar. Kod realizacije prikazan je na *Slici 7.*.

```
import cv2
import numpy as np


cap = cv2.VideoCapture(0)
#0 znaci da ce biti prva video kamera u sistem

fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter ('output.avi', fourcc, 20.0,(640,480))

while True: #beskonacna petlja
    ret, frame = cap.read()

    #2.deo, prebacujemo u sivo
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    #3.deo za cuvanje snimka
    out.write(frame)

    cv2.imshow('prozorce', frame)
    cv2.imshow('gray', gray)


    #kako bi izasli iz beskonacne petlje
    if cv2.waitKey(1) & 0xFF == ord ('q'):
        break

cap.release()
out.release()#za cuvanje snimka
cv2.destroyAllWindows()
```


Slika 7. Dodavanje dela za čuvanje snimka

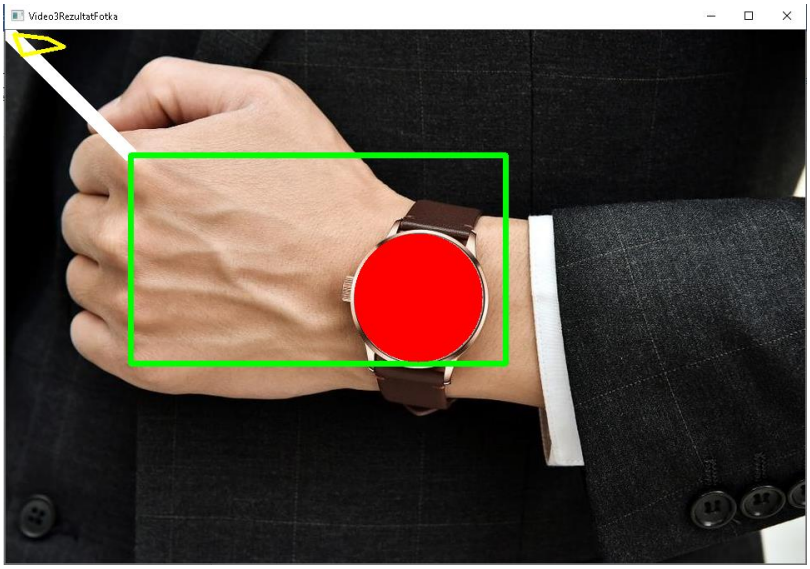
Crtanje i pisanje na slici

Video 3. Drawing and Writing on Image – OpenCV with Python for Image and Video Analysis

Unutar biblioteke cv2 postoje iscrtavanje linija, trouglova, krugova, poligona. Funkcije su prikazane na *Sllici 8.*, gde je svaka funkcija objašnjena, tj. njeni parametri. Rezultat koda je prikazan na *Slici 9.*.

```
import numpy as np
import cv2
img = cv2.imread('satic.jpg', cv2.IMREAD_COLOR)

#za iscrtavanje linije preko slike
#prvi argument je fotka,
#drugi parametar je gde pocinje linija,
#treci argument gde se zavrsava
#cetvrti je boja bgr, 255 255 255 je bela
#peti je opcionalan, debljina linije
cv2.line(img, (0,0),(150, 150), (255, 255,255), 15)

##crtanje trougla
##1. argument na cemu crtamao
##2. argument je gde pocinje, pocetna ta;ka
##3. argument je zavrsna tacka
##4. boja bgr
##5. debljina linije
cv2.rectangle(img, (150, 150),(600,400), (0,255,0), 5)

#krug
##1. argument na cemu crtamo
##2. je centar kruga
##3. radius tj poluprecnik
##4. je boja
##5. debljina linije ili ako stavimo -1 obojice ceo krug
cv2.circle(img, (495,321), 77, (0,0,255),-1 )

#points tacke
pts = np.array([[10,5],[20,30],[70,20],[50,10]],np.int32)
##pts = pts.reshape((-1,1,2))

#poligon
##1. argument na cemu crtamo
##2. drugi argument niz tacaka
##3. argument True znaci da hocemo da povezemo prvu tacku sa poslednjom
##4. boja bgr
##5. argument debljina linie
cv2.polylines(img, [pts], True, (0,255,255),3)

cv2.imshow('Video3RezultatFotka',  img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Slika 8. Crtanje i pisanje na slici



Slika 9. Rezultat iscrtavanja

Može se unositi teskt, pomoću funkcije putText. Na *Slici 10.* je prikan kod i opisani argumenti funkcije. Rezultat je prikazan na *Slici 11.*.

```
font= cv2.FONT_HERSHEY_SIMPLEX
#pisanje teksta
##1. argument na cemu pisemo
##2. argument tekst
##3. argument je tacka gde se pocinje
##4. font
##5. velicina
##6. boja
##7. Debljina slova
cv2.putText(img, 'OpenCV la la', (0,130), font, 1,(255,0,255), 5 , cv2.LINE_AA)
```

Slika 10. Ispisivanje teksta

Slika 11. Rezultat ispisivanje

Operacije sa slikom

Video 4. Image Operations – OpenCV with Python for Image and Video Analysis

Unutar koda, prikazanog na *Slici 12.*, urađena je izmena piksela. Rezultat je prikazan na *Slici 13.*.

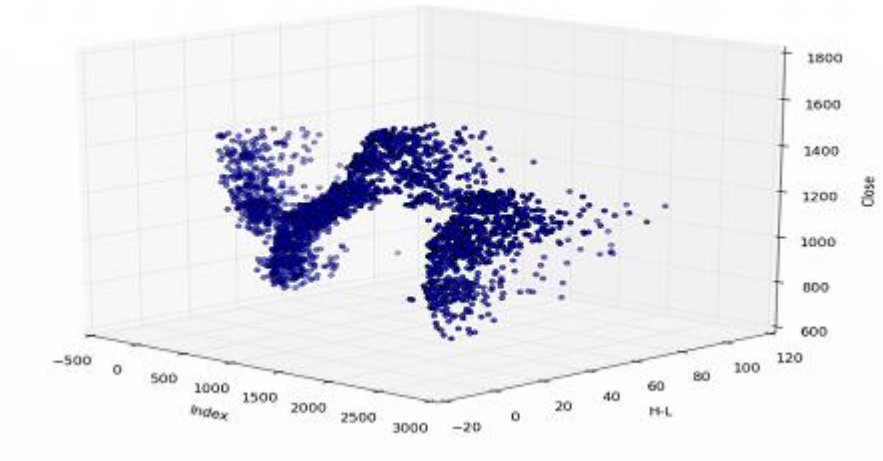
```
import numpy as np
import cv2

img = cv2.imread ('satic.jpg', cv2.IMREAD_COLOR)

#odredjeni pisel na fotki
px = img [55,55]
#print(px)

#izmena piksela
img[55,55] = [255,255,255]
print(px)

#Region Of Image
##roi = img[100:155, 100:150]
##print(roi)

#deo slike pretvaramo u belo
img[100:155, 100:150] = [255,255,255]

#oblast slike, roi
watch_face = img[37:111, 107:194]
#111 - 37 = 74 , 194 - 107 = 87

#uzeo je piksele od watch_face
img[0:74, 0:87] = watch_face


cv2.imshow ('Fotka', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Slika 12. Izmena piksela

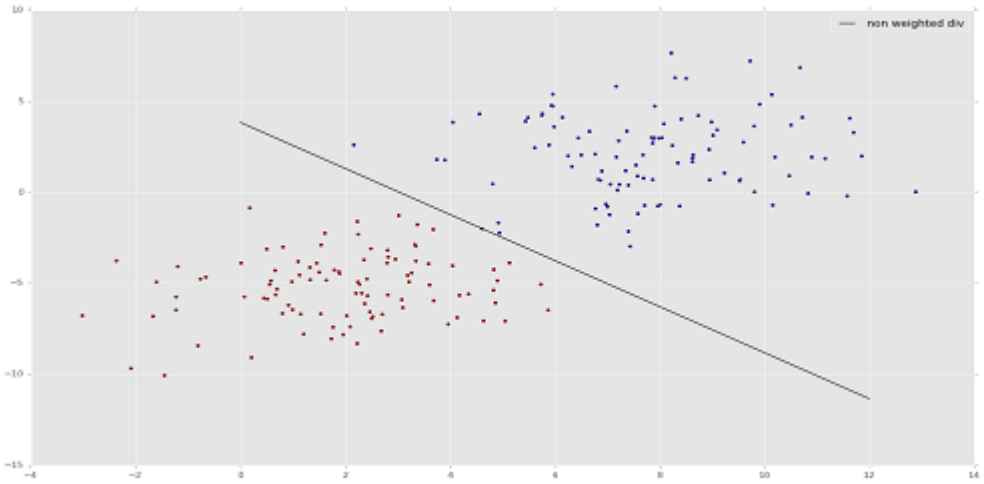
Slika 13. Rezultat izmene piksela

Aritmetika i logika

Video 5. Image arithmetics and Logic – OpenCV with Python for Image and Video Analysis



Slika 14. '3D-Matplotlib.png'

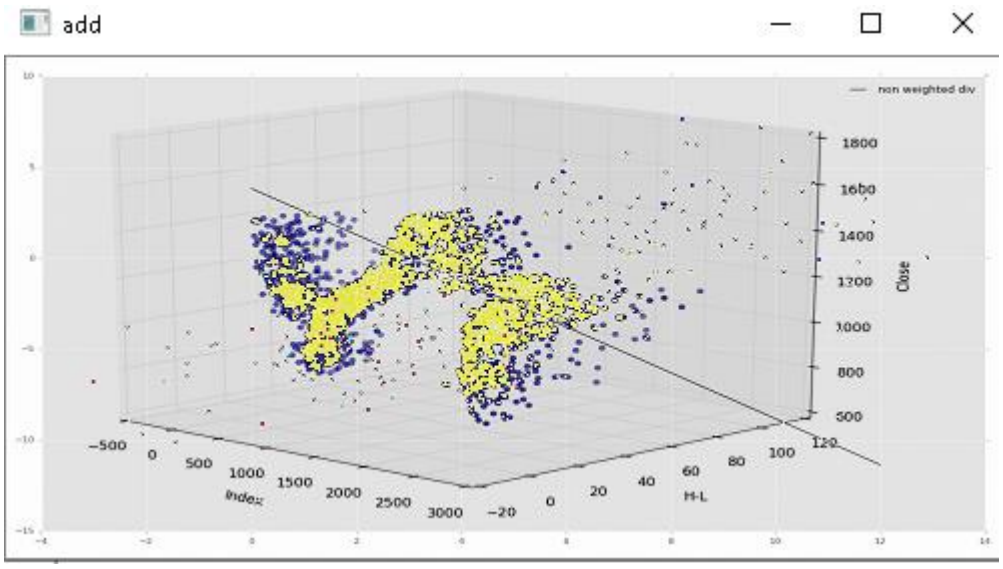


Slika 15. 'mainsvmimage.png'

Unutar koda, prikazanog na *Slici 16.*, učitali smo dve slike (*Sliku 14.* i *Sliku 15.*), zatim ih spojili u jednu. Rezultat je prikazan na *Slici 17.*.

```
import numpy as np
import cv2

# 500 x 250, iste su velicine
img1 = cv2.imread('3D-Matplotlib.png')
img2 = cv2.imread('mainsvmimage.png')

add = img1 + img2

cv2.imshow('add',add)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Slika 16. Spajanje slika

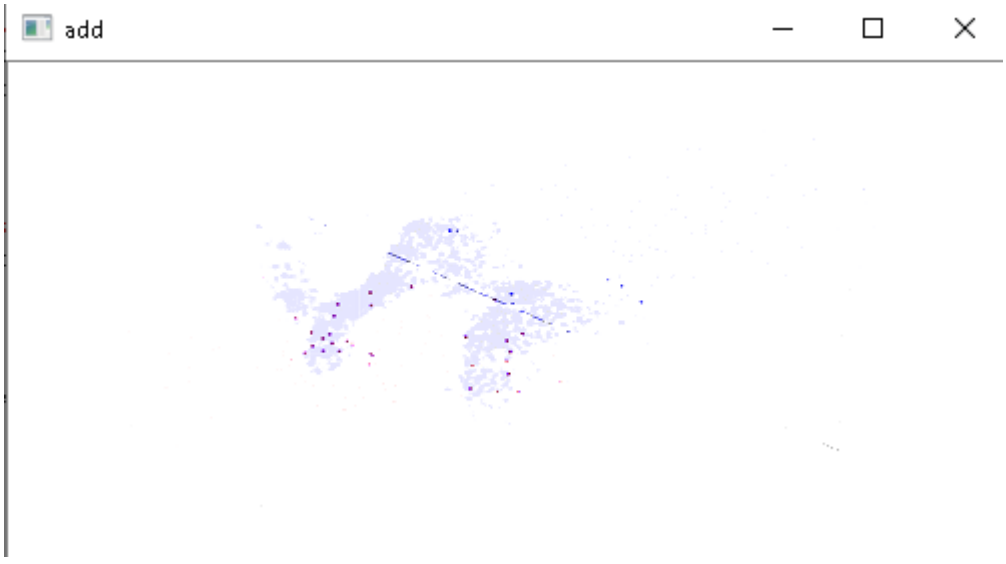


Slika 17. Rezultat spajanja

Postoje razni načini spajanja slike, unutar bibliteke cv2, postoji metod add, prikazan i opisan na *Slici 18.*, ali se retko koristi u praksi. Rezultat tog koda prikazan je na *Slici 19.*.

```
import numpy as np
import cv2

# 500 x 250, iste su velicine
img1 = cv2.imread('3D-Matplotlib.png')
img2 = cv2.imread('mainsvmimage.png')

#add = img1 + img2

#dodati su svi pikseli sa obe slike
#(155,211,79) + (50, 170, 200) = 205, 381, 279...prevedeni u
#(205, 255,255)
add = cv2.add(img1,img2)

cv2.imshow('add',add)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Slika 18. add

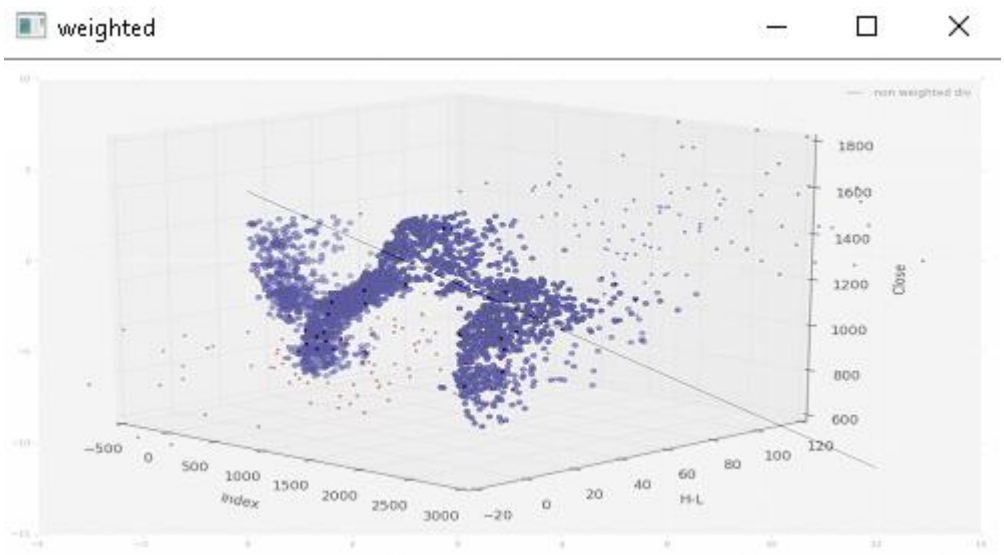


Slika 19. Rezultat add

Još jedan od načina je funkcija addWeighted opisana i prikazana na *Slici. 20.*, dok je rezultat koda prikazan na *Slici 21.*.

```
#1. parametar je slika
#2. parametar je tezina
#3. parametar je druga slika
#4. parametar je ta tezina
#5. parametar je gamma, sto je mera svetlosti,
#ostavljamo je na 0 za sada
weighted = cv2.addWeighted(img1,0.6, img2, 0.4, 0)

#cv2.imshow('add',add)
cv2.imshow('weighted',weighted)
```

Slika 20. addWeighted



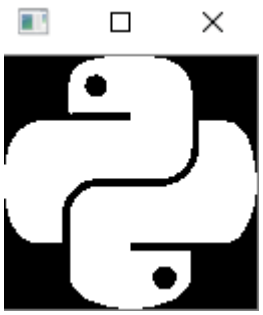
Slika 21. Rezultat

U kodu na *Slici 22.* se ubacuje logo u gornji levi ugao slike sa grafikom. Kreira se maska i threshold, opisani slici takođe na *Slici 22.*.

```
import numpy as np
import cv2

#ucitavanje slika
img1 = cv2.imread('3D-Matplotlib.png')
img2 = cv2.imread('mainlogo.png')

#stavljamo logo u gornji levi ugao, kreiramo ROI
rows,cols,channels = img2.shape
roi = img1[0:rows, 0:cols ]

# kreiranje maske loga
#kreiranje sive verzije loga
img2gray = cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)

#dodavanje threshold, praga
##1.parametar gde dodaje treshold
##2.parametar je vrednost tresshold
##3.je maksimalna vrednost
##4.ako je iznad 220 pretvorice je u 255, ako je ispod 220 pretvorice u crno
ret, mask = cv2.threshold(img2gray, 220, 255, cv2.THRESH_BINARY_INV)

cv2.imshow('mask',mask)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Slika 22.

Slika 23. ‘’mainlogo.png

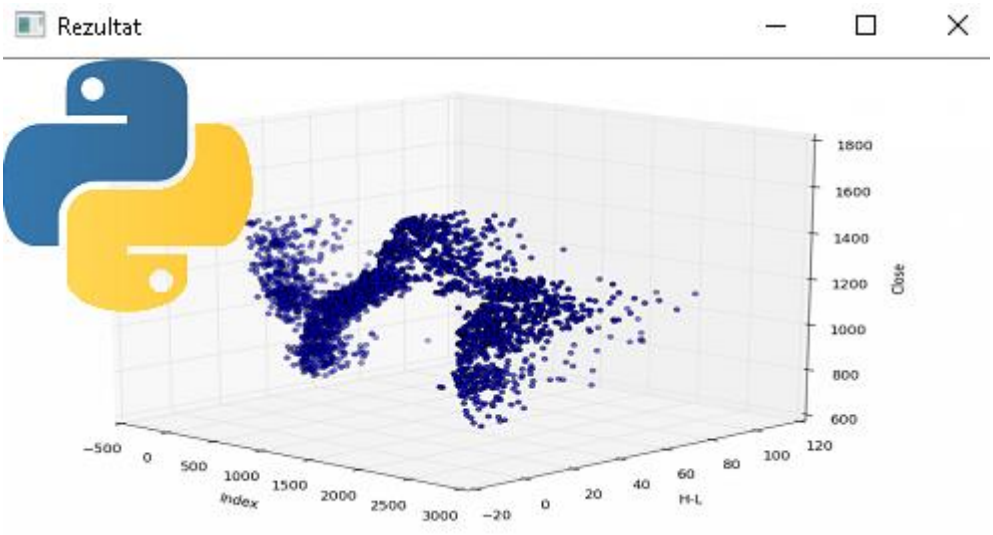


Slika 24. ’mask’

Unutar crvenog pravougaonika na *Slici 25.* je prikazam i opisan postupak ubacivanja loga. Rezultat koda je prikazan na *Slici 26.*.

```
import numpy as np
import cv2

#ucitavanje slika
img1 = cv2.imread('3D-Matplotlib.png')
img2 = cv2.imread('mainlogo.png')

#stavljamo logo u gornji levi ugao, kreiramo ROI
rows,cols,channels = img2.shape
roi = img1[0:rows, 0:cols ]

# kreiranje maske loga
#kreiranje sive verzije loga
img2gray = cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)

#dodavanje threshold, praga
##1.parametar gde dodaje treshold
##2.parametar je vrednost tresshold
##3.je maksimalna vrednost
##4.ako je iznad 220 pretvorice je u 255, ako je ispod 220 pretvorice u crno
ret, mask = cv2.threshold(img2gray, 220, 255, cv2.THRESH_BINARY_INV)

#invertuje svaki bit u nizu
mask_inv = cv2.bitwise_not(mask)

#zatamljujemo oblast loga u ROI
img1_bg = cv2.bitwise_and(roi,roi,mask = mask_inv)

#uzimamo samo regiju loga sa slike loga
img2_fg = cv2.bitwise_and(img2,img2,mask = mask)
dst = cv2.add(img1_bg,img2_fg)
img1[0:rows, 0:cols ] = dst

cv2.imshow('Rezultat',img1)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Slika 25. Ubacivanje loga na grafik



Slika 26. Rezultat ubacivanja

Segmentacija pragom – Thresholding

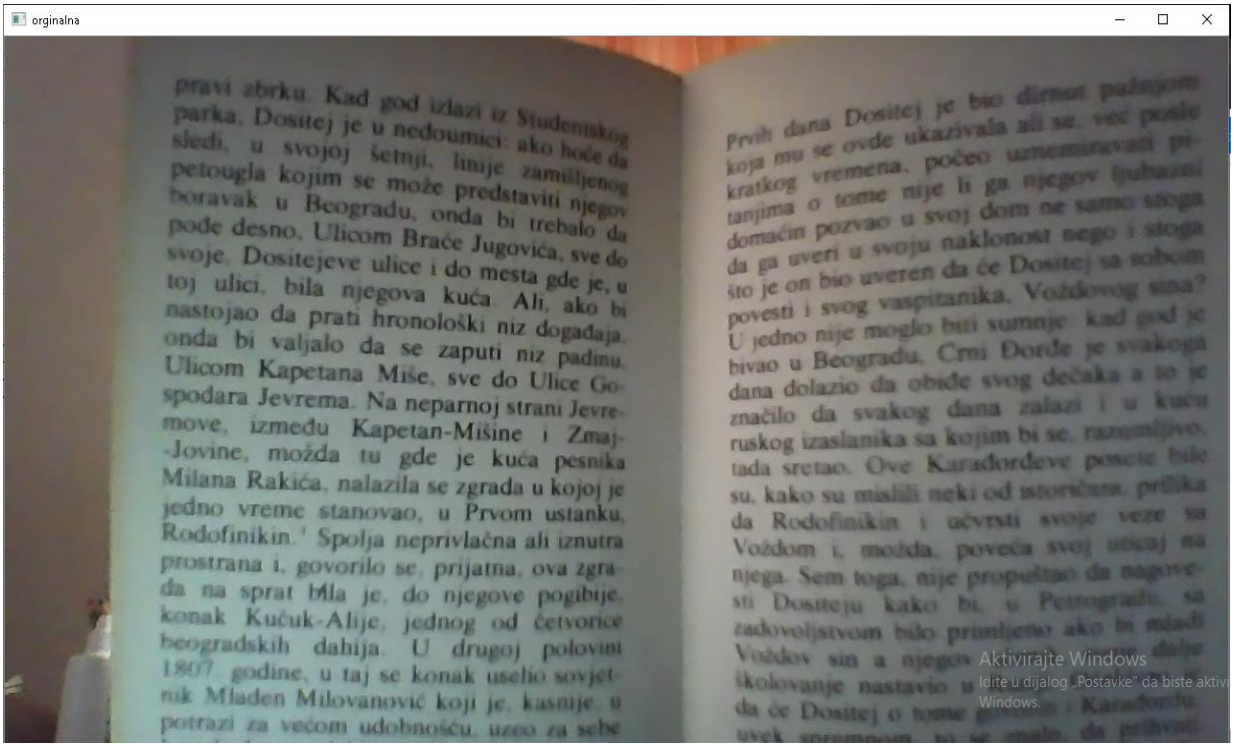
Segmentacija pomoću praga predstavlja grupu metoda zasnovanih na poređenju osvetljenosti piksela sa jednim ili više pragova, pa u tom slučaju imamo segmentaciju sa jednim i segmentaciju sa više pragova. Ukoliko imamo slike u boji tada se vrši poređenje boje između piksela. Segmentacija sa jednim pragom predstavlja najjednostavniji vid segmentacije i njegova osnovna primena je za odvajanje objekta od pozadine, ukoliko pozadina ima uniformnu osvjetljenost (boju) koja se razlikuje od objekta. Ovakav način segmentacije se može primjeniti, npr., kod izdvajanja pisanog ili štampanog teksta, analize nekih biomedicinskih slika, prepoznavanja tipa aviona koji leti, itd. Segmentacija sa više pragova je metoda koju je pogodno koristiti u slučaju kada imamo scene sa više različitih objekata [2].

Video 6. Thresholding – OpenCV with Python for Image and Video Analysis

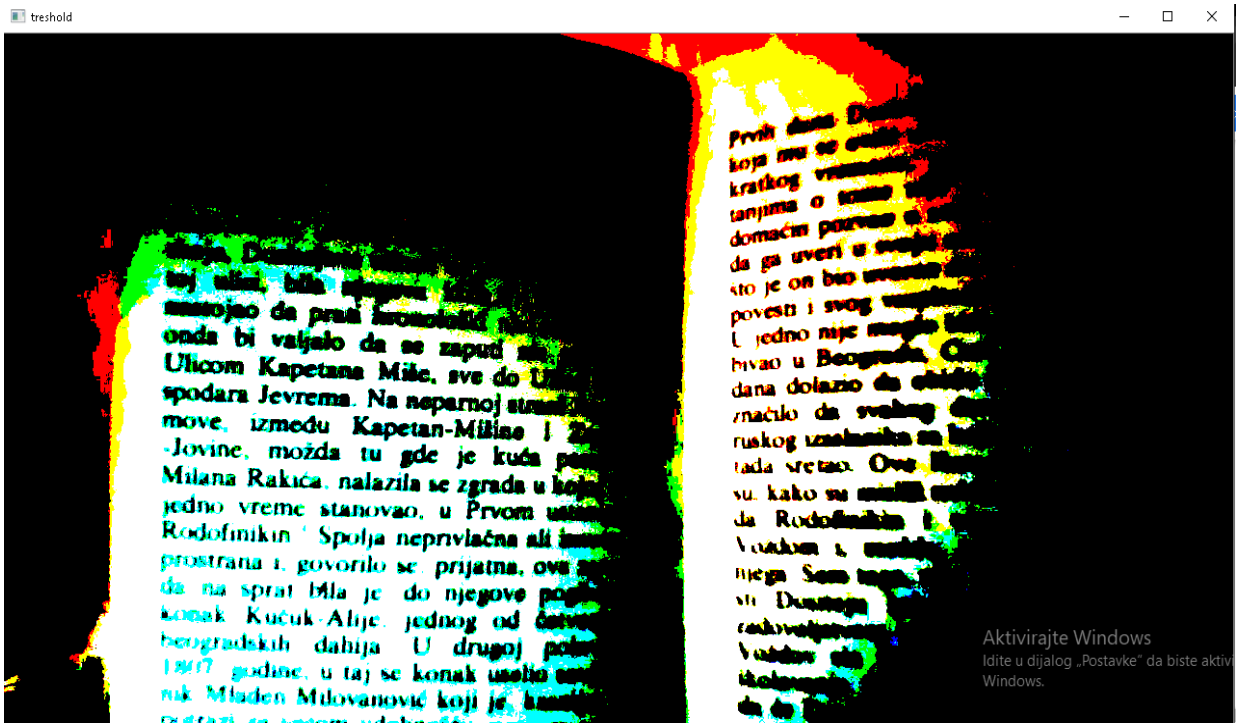
Na *Slici 27.* prikazan a i objašnjena implementacija funkcija treshold. Rezultat koda je prikazan na *Slici 29.*.

```
import numpy as np
import cv2

img = cv2.imread('book.jpg')


#cv.threshold koristi se za primenu praga.
##1. argument je izvorna slika, koja bi trebala biti slika u sivim
##tonovima
##2. argument je vrednost praga koja se koristi za klasifikaciju
##vrednosti piksela.
##3. argument je maksimalna vrednost koja se dodeljuje
##vrednostima piksela većim od praga.
##4. argument tip praga
retval, threshold = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY)

cv2.imshow('orginalna',img)
cv2.imshow('treshold', threshold)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Slika 27. Treshold



Slika 28. Orginalna slika

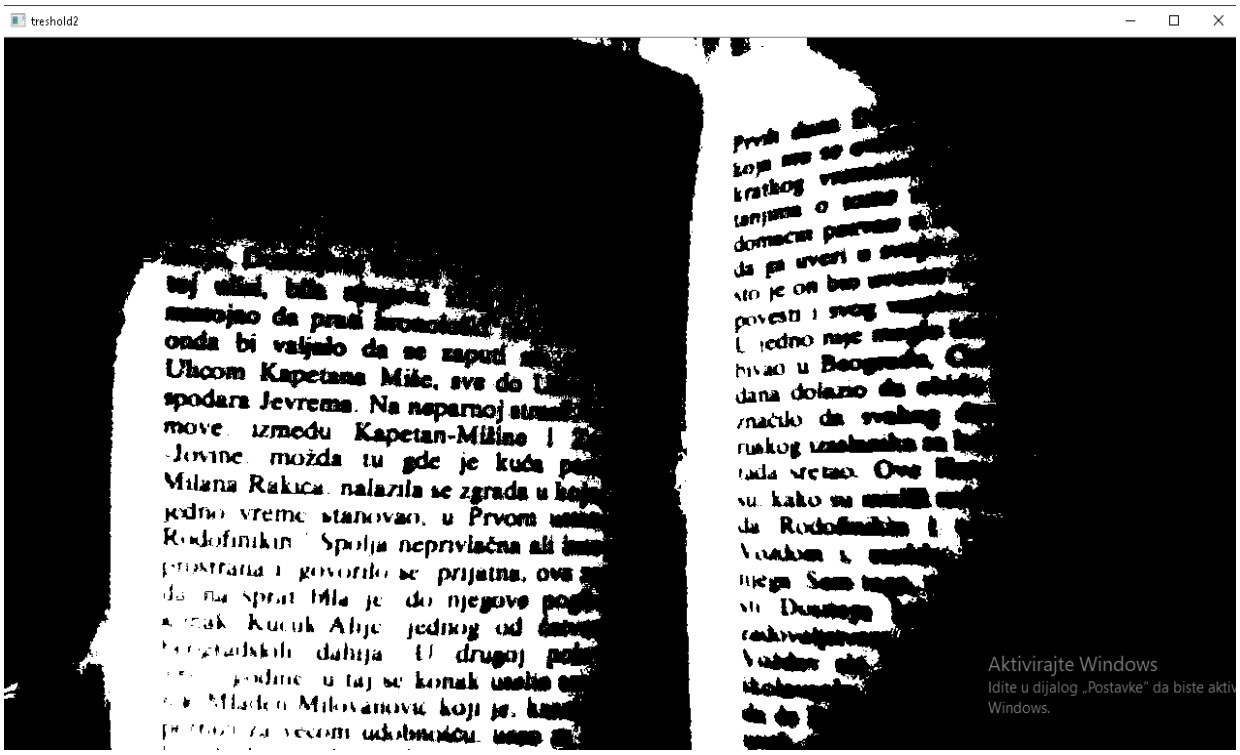


Slika 29. Rezultat treshold

Unutar crvenog pravougaonika na *Slici 30.* prikazane su funkcije za konvertovanje boje, koje su objašnjene u jednom od prethdnih poglavlja dokumentacije.

```
import numpy as np
import cv2

img = cv2.imread('book.jpg')


#cv.threshold koristi se za primenu praga.
##1. argument je izvorna slika, koja bi trebala biti slika u sivim
##tonovima
##2. argument je vrednost praga koja se koristi za klasifikaciju
##vrednosti piksela.
##3. argument je maksimalna vrednost koja se dodeljuje
##vrednostima piksela većim od praga.
##4. argument tip praga
retval, threshold = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY)

grayscaled = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
retval2, threshold2 = cv2.threshold(grayscaled, 120, 255, cv2.THRESH_BINARY)

cv2.imshow('orginalna',img)
cv2.imshow('treshold', threshold)
cv2.imshow('treshold2', threshold2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Slika 30. Konvertovanje boje

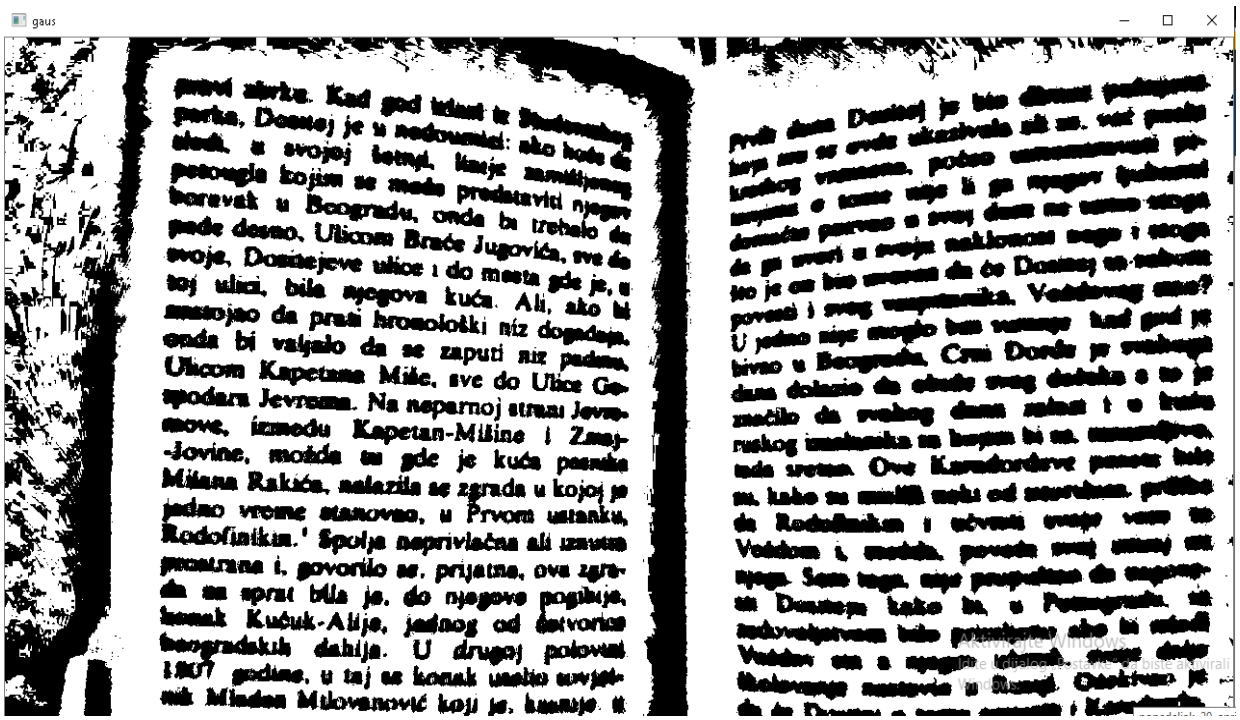


Slika 31. Rezultat konvertovanja

AdaptiveTreshold je metoda gde se vrednost praga izračunava za manje regione i zato će biti različitih vrednosti praga za različite regione. Funkcija je prikazana na *Slici 32.*. cv2.ADAPTIVE_THRESH_GAUSSIAN_C : threshold vrednost je procenjena weighted suma vrednosti suseda, čije težine (weights) su Gausov prozor (a gaussian window). Rezultat primene metoda je prikazan na *Slici 33.*.

```
import numpy as np
import cv2

img = cv2.imread('book.jpg')


#cv.threshold koristi se za primenu praga.
##1. argument je izvorna slika, koja bi trebala biti slika u sivim
##tonovima
##2. argument je vrednost praga koja se koristi za klasifikaciju
##vrednosti piksela.
##3. argument je maksimalna vrednost koja se dodeljuje
##vrednostima piksela većim od praga.
##4. argument tip praga
retval, threshold = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY)

grayscaled = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
retval2, threshold2 = cv2.threshold(grayscaled, 12, 255, cv2.THRESH_BINARY)

gaus = cv2.adaptiveThreshold(grayscaled, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 115, 1)
cv2.imshow('orginalna',img)
cv2.imshow('treshold', threshold)
cv2.imshow('treshold2', threshold2)
cv2.imshow('gaus', gaus)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Slika 32.adaptiveTreshold

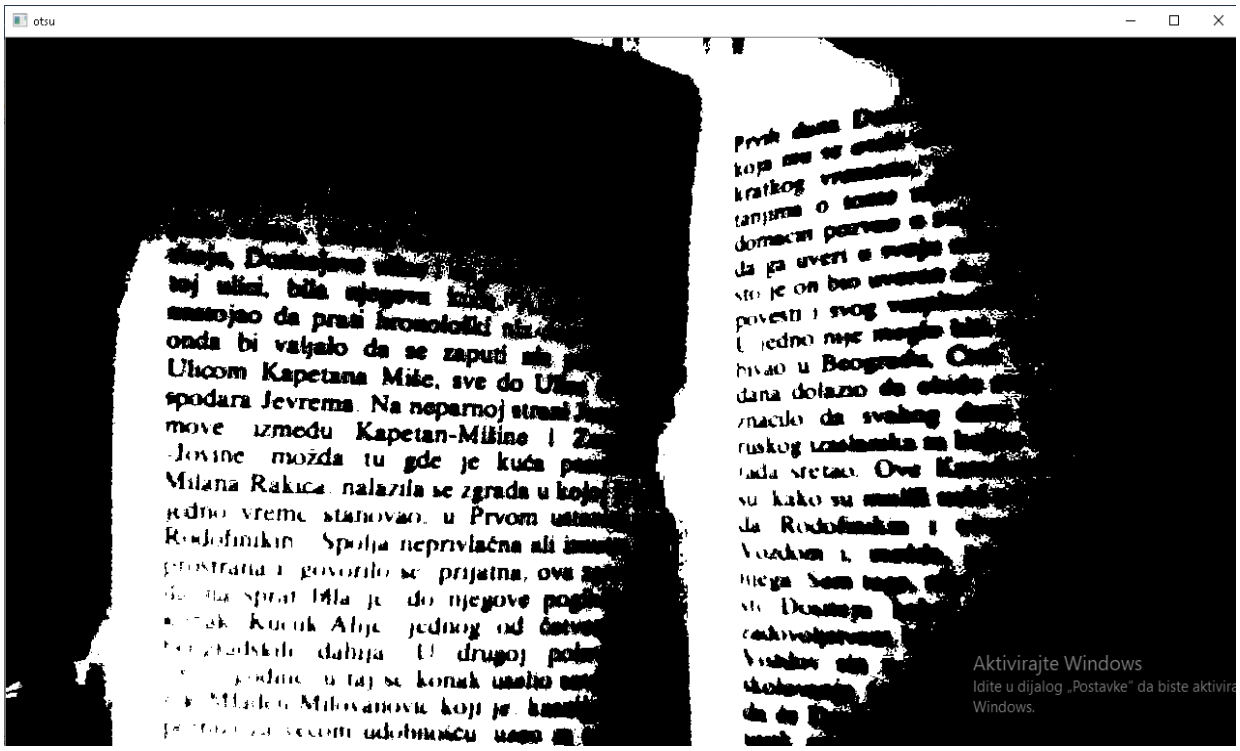


Slika 33. Rezultat adaptiveTreshold

Na isti način , *na Slici 34.*, primenjena je funkcija treshold, koja je objašnjena, ali kao četvrti argument, tip praga je OTSU treshold.
U opštem tresholdingu smo koristili proizvoljnu vrednost za vrednost praga,. Pa, kako možemo znati da je vrednost koju smo odabrali dobra ili ne? Odgovor je metoda pokušaja i greške. Ali razmotrimo bimdalnu sliku (bimodalna slika je npr. slika čiji histogram ima dva vrha). Za tu sliku možemo približno uzeti vrednost usred tih vrhova kao vrednost praga. To radi Otsu binarnost. Jednostavnim rečima, automatski izračunava vrednost praga iz histograma slike za bimodalnu sliku. (Za slike koje nisu bimodalne, binarnost neće biti tačna.)Za to se koristi funkcija cv2.threshold (), ali prosljeđuje dodatnu zastavu, cv2.THRESH_OTSU. Za vrednost praga, jednostavno prenesite nulu. Tada algoritam pronalazi optimalnu vrednost praga i vraća vas kao drugi izlaz, retVal. Ako se Otsu prag ne koristi, retVal je isti kao vrednost praga koju ste koristili.

```
import numpy as np
import cv2

img = cv2.imread('book.jpg')


#cv.threshold koristi se za primenu praga.
##1. argument je izvorna slika, koja bi trebala biti slika u sivim
##tonovima
##2. argument je vrednost praga koja se koristi za klasifikaciju
##vrednosti piksela.
##3. argument je maksimalna vrednost koja se dodeljuje
##vrednostima piksela većim od praga.
###4. argument tip praga
retval, threshold = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY)

grayscaled = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
retval2, threshold2 = cv2.threshold(grayscaled, 12, 255, cv2.THRESH_BINARY)

gaus = cv2.adaptiveThreshold(grayscaled, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 115, 1)
retval2,otsu = cv2.threshold(grayscaled,125,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)

cv2.imshow('orginalna',img)
cv2.imshow('treshold', threshold)
cv2.imshow('treshold2', threshold2)
cv2.imshow('gaus', gaus)
cv2.imshow('otsu', otsu)

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

Slika 34. otsu



Slika 35. Rezultat otsu

Filtriranje boje

Video 7. Color Filtering – OpenCV with Python for Image and Video Analysis

Na *Slici 36.* je prikazano filtriranje boje. cvtColor metoda koristi se za pretvaranje slike iz jednog prostora boja u drugi. U OpenCV-u postoji više od 150 metoda pretvaranja prostora-boja.
Sintaksa funkcije: *cv2.cvtColor(src, code[, dst[, dstCn]])*
Parametri:

- *src*: slika čiji se prostor boja menja
- *kod*: kod za konverziju prostora u boji.
- *dst*: izlazna slika iste veličine i dubine kao src slika. To je opcionalan parametar.
- *dstCn*: broj kanala u odredišnoj slici. Ako je parametar 0, tada se broj kanala automatski dobiva iz src-a i koda. To je opcionalan parametar.

Sintaksa funkcije: *cv2.inRange(src, lowerb, upperb[, dst]) → dst*
Parametri:

- src – prvi ulazni niz.
- lowerb – uključujući donji granični niz ili skalarni niz.
- upperb – uključujući gornj granični niz ili skalarni niz
- dst – izlazni niz iste veličine kao src i CV_8U.

Sintaksa funkcije: *cv2.bitwise_and(src1, src2[, dst[, mask]]) → dst*

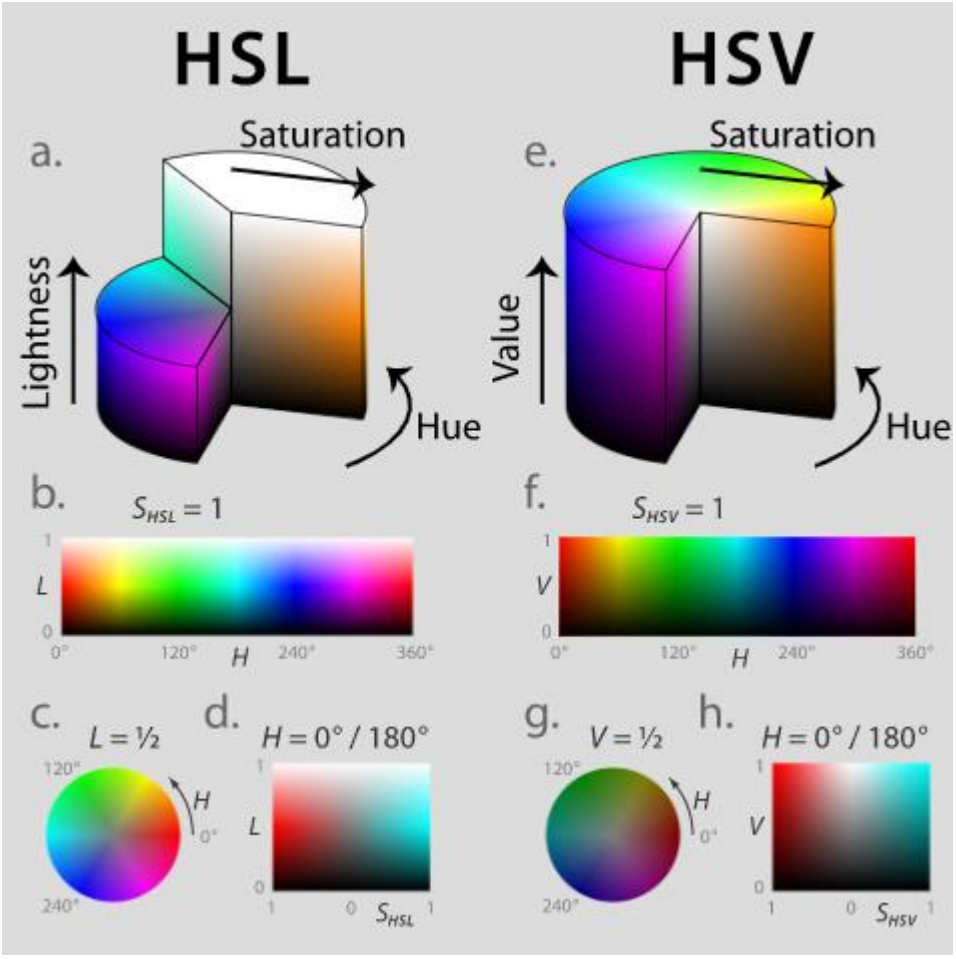
Parematri:

- src1 – prvi ulazni niz ili skalar
- src2 – drugi ulazni niz ili skalar
- dst – izlazni niz koji je iste veličine i tipa kao ulazni nizovi
- mask – opciona operacija mask, osmobitni niz, koji određuje elemente izlaznog niza koji treba promeniti.

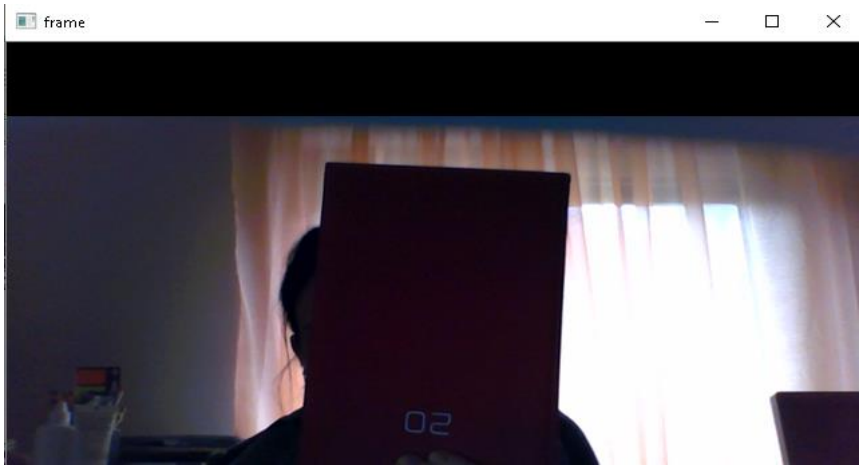
```
import numpy as np
import cv2


cap = cv2.VideoCapture(0)

while True:
    _,frame = cap.read()
#HSV, Hue Saturation Value
#Hue range is [0,179],
#Saturation range is [0,255]
#Value range is [0,255].
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    lower_red = np.array([100,100,40])
    upper_red = np.array ([200,255,100])

    mask = cv2.inRange(hsv, lower_red, upper_red)
    res = cv2.bitwise_and(frame, frame, mask = mask)

    cv2.imshow('frame', frame)
    cv2.imshow('mask', mask)
    cv2.imshow('res', res)

    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break

cv2.destroyAllWindows()
cap.release()
```

Slika 36. Color filtering

HSV je srkaćenica za Hue (nijansa) Saturion(saturacija) Value(vrednost). Na *Slici 37.* je prikazan HSV.

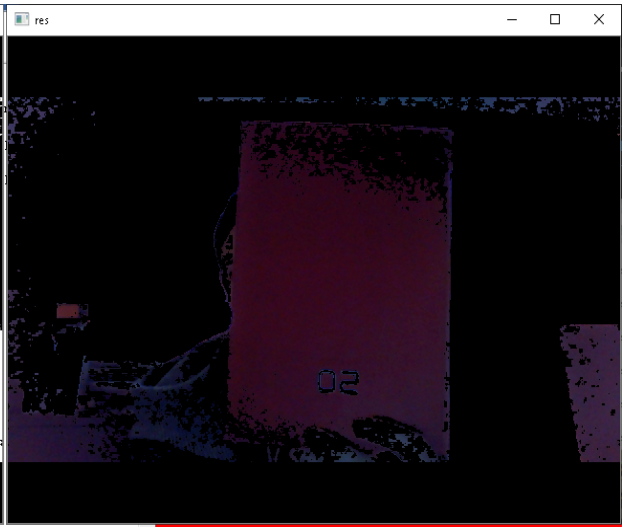


Slika 37. HUE



Slika 38. Orginalna slika

Slika 39. mask



Slika 40. res

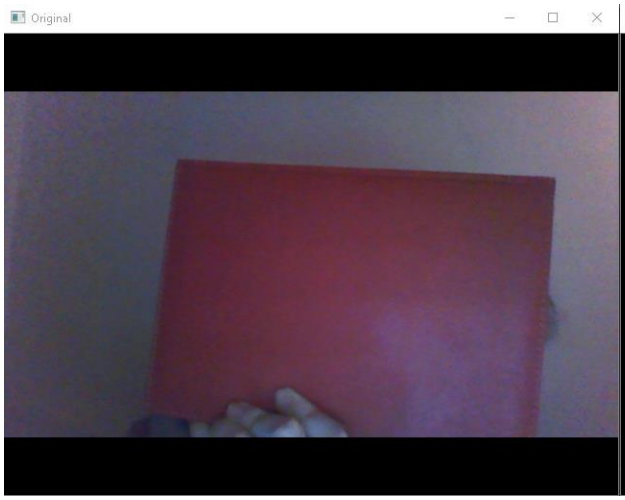
Zamagljivanje i izoštravanje

Video 8. Blurring and Smoothing – OpenCV with Python for Image and Video Analysis

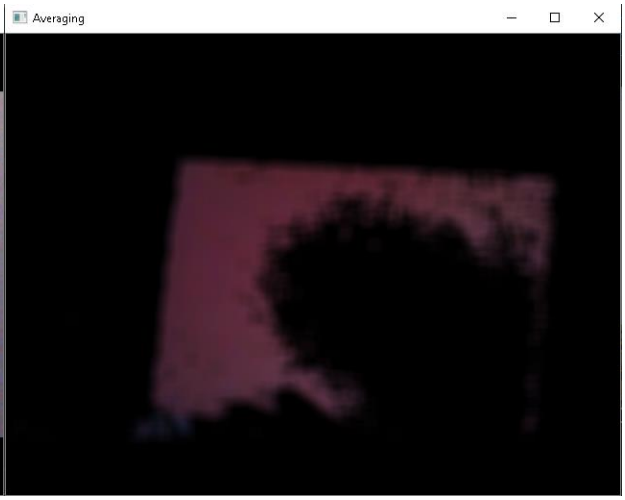
Sada primenimo jednostavno zamaglivanje, gde radimo neku vrstu sortiranja srednje vrednosti za blok piksela. U našem slučaju napravimo kvadrat veličine 15 x 15, što znači da imamo ukupno 225 piksela. Pozovamo četri funkcije za zamaglivanje. Kod je prikaz na Slici 41., dok su rezultati prikazani na *Slikama 42.-46.*.

```
import numpy as np
import cv2

cap = cv2.VideoCapture(0)

while(1):

    _, frame = cap.read()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    #hsv hue sat vrednost
    lower_red = np.array([100,100,40])
    upper_red = np.array ([200,255,100])

    mask = cv2.inRange(hsv, lower_red, upper_red)
    res = cv2.bitwise_and(frame,frame, mask = mask)

    #averaging
    #225 je jer je 15 x 15 = 225
    #15 po 15 piksela
    kernel = np.ones((15,15),np.float32)/225

    smoothed = cv2.filter2D(res,-1,kernel)
    cv2.imshow('Averaging',smoothed)

    blur = cv2.GaussianBlur(res,(15,15),0)
    cv2.imshow('Gaussian Blurring',blur)

    median = cv2.medianBlur(res,15)
    cv2.imshow('Median Blur',median)

    bilateral = cv2.bilateralFilter(res,15,75,75)
    cv2.imshow('bilateral Blur',bilateral)

    cv2.imshow('Original',frame)

    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break

cv2.destroyAllWindows()
cap.release()
```

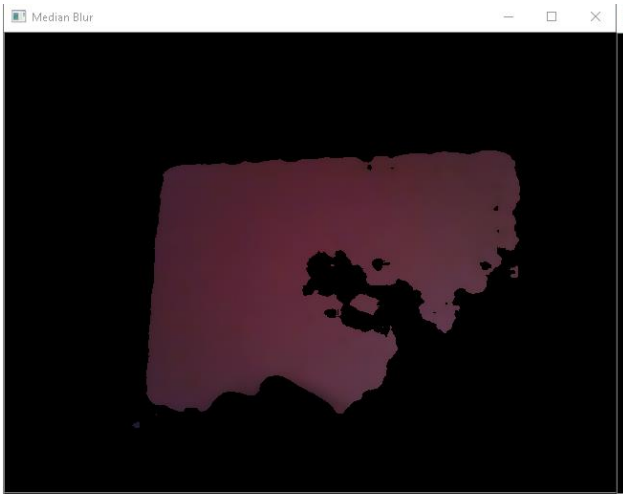
Slika 41. Funkcije Blur



Slika 42. Orginal



Slika 43. Averaging



Slika 44. Medium Blur

Slika 45. Gaussian Blurring

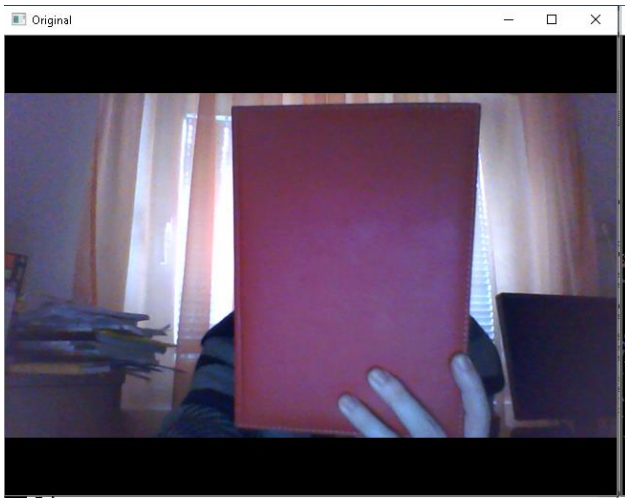
Morfološke transformacije

Video 9. Morphological Transformations – OpenCV with Python for Image and Video Analysis

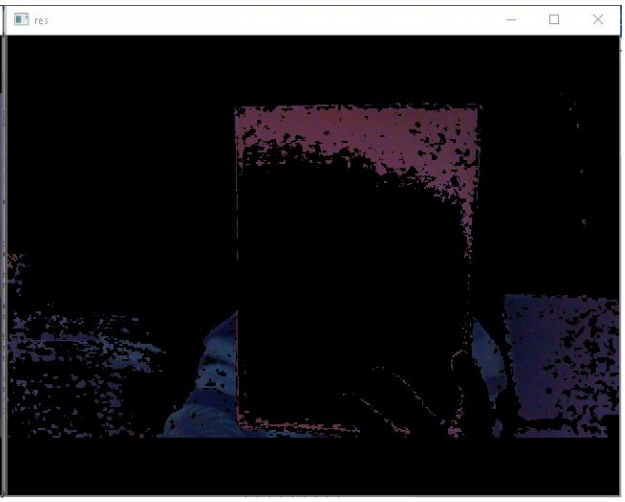
Morfološke transformacije će se koristit da se uklone smetnje, noise.
Oni obično dolaze u parovima. Prvi par o kome ćemo obraditi je Erozija i Dilatacija.
Erozija je ta koja ćemo "izbrisati" ivice. Način na koji ovo radimo sarađujemo sa sliderom (kernel). Klizaču dajemo veličinu, recimo 5 x 5 piksela. Ono što se dešava je da “pomeramo“ ovaj klizač okolo, a ako su svi pikseli beli, tada dobijamo beli, inače crni. Ovo može pomoći da se eliminiše neki beli šum. Druga verzija ovog postupka je Dilation, koja u osnovi čini suprotno: Klizi se unaokolo, ako celokupno područje nije crno, tada se pretvara u belo.
Kod je prikazan na Slici 46., dok je rezultat koda na Slika 47.-49.

```
import numpy as np
import cv2

cap = cv2.VideoCapture(0)

while(1):

    _, frame = cap.read()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    lower_red = np.array([100,100,40])
    upper_red = np.array ([200,255,100])

    mask = cv2.inRange(hsv, lower_red, upper_red)
    res = cv2.bitwise_and(frame,frame, mask = mask)

    kernel = np.ones((5,5),np.uint8)
    erosion = cv2.erode(mask,kernel,iterations = 1)
    dilation = cv2.dilate(mask,kernel,iterations = 1)

    cv2.imshow('Original',frame)
    cv2.imshow('res',res)
    cv2.imshow('Erosion',erosion)
    cv2.imshow('Dilation',dilation)



    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break

cv2.destroyAllWindows()
cap.release()
```

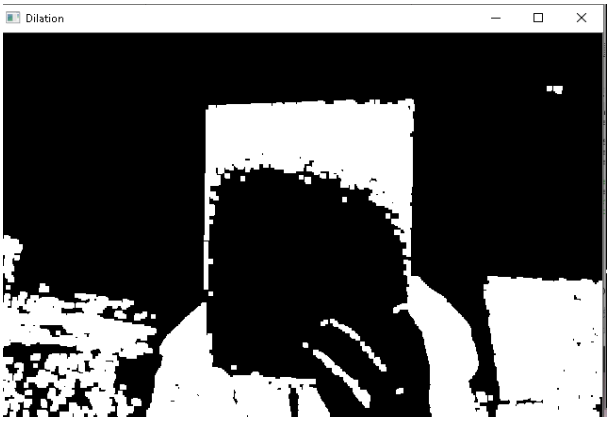
Slika 46. Kod Erozija i Dilatacija



Slika 47. Orginal



Slika 48. res



Slika 49. Dilatacija

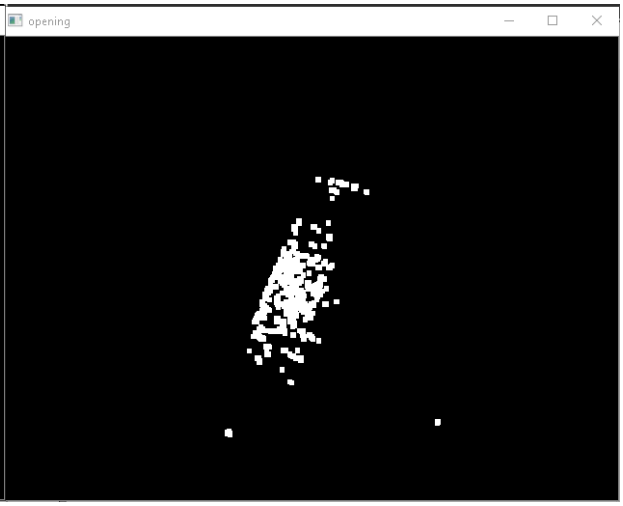
Slika 50. Erozija

Sledeći par je „otvaranje“ i „zatvaranje“. Cilj otvaranja je uklanjanje "lažnih pozitivnih rezultata". Ponekad, u pozadini, ovde i tamo dobijete neke piksele „buke“. Ideja "zatvaranja" je uklanjanje lažnih negativa. U osnovi ovde imate svoj otkriveni oblik, poput našeg šešira, a ipak imate neke crne piksele unutar objekta. Zatvaranje će pokušati da to raščisti. Implementacija je prikazan na *Slici.51.*, rezultat na *Slikama 52.* i *53.*.

```
import numpy as np
import cv2

cap = cv2.VideoCapture(0)

while(1):

    _, frame = cap.read()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    lower_red = np.array([30,150,50])
    upper_red = np.array([255,255,180])

    mask = cv2.inRange(hsv, lower_red, upper_red)
    res = cv2.bitwise_and(frame,frame, mask = mask)

    kernel = np.ones((5,5),np.uint8)
    erosion = cv2.erode(mask,kernel,iterations = 1)
    dilation = cv2.dilate(mask,kernel,iterations = 1)

    opening = cv2.morphologyEx(mask, cv2.MORPH_OPEN,kernel)
    closing = cv2.morphologyEx(mask, cv2.MORPH_CLOSE,kernel)

    cv2.imshow('Original',frame)
##  cv2.imshow('res',res)
##  cv2.imshow('Erosion',erosion)
##  cv2.imshow('Dilation',dilation)
    cv2.imshow('opening',opening)
    cv2.imshow('closing',closing)

    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break

cv2.destroyAllWindows()
cap.release()
```

Slika 51. Opening, closing

Slika 52. Closing



Slika 53. Opening

Dve druge opcije, a to su „tophat“ i „blackhat”.

```
#To je razlika između ulazne input slike i Otvaranja slike
cv2.imshow('Tophat',tophat)

#To je razlika između zatvaranja ulazne slike i ulazne slike.
cv2.imshow('Blackhat',blackhat)
```

Slika 54. Black i top hat

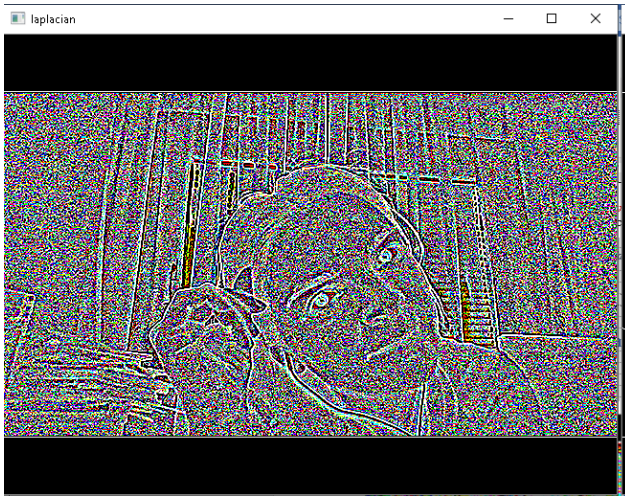
Detekcija ivica i gradijenti

Video 10. Edge Detection and Gradients – OpenCV with Python for Image and Video Analysis

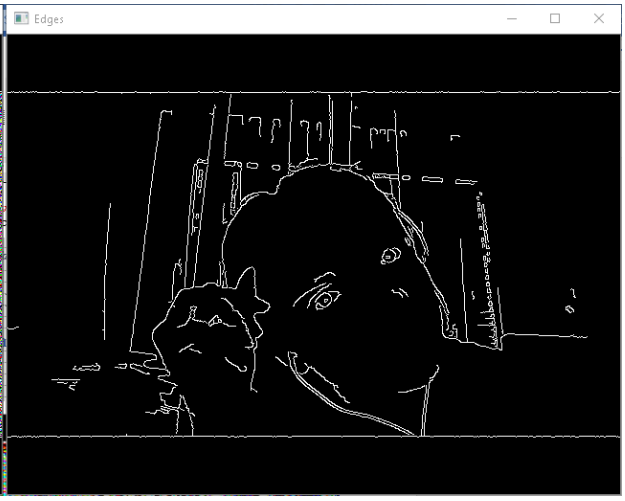
Na *Slici 55.* je prikazana implementacija gradijenata, Laplasov i Sobel. Detektor ivica, pozivajuću funkciju Canny.

```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)

while(1):

    #uzmi svaki okvir
    _, frame = cap.read()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    #Laplasov gradijent
    laplacian = cv2.Laplacian(frame,cv2.CV_64F)

    #Sobel
    #k size = kernel size
    #horizontalni gradijent
    sobelx = cv2.Sobel(frame,cv2.CV_64F,1,0,ksize=5)
    #vertikalni gradijent
    sobely = cv2.Sobel(frame,cv2.CV_64F,0,1,ksize=5)

    #detektor ivica
    edges = cv2.Canny(frame,100,200)

    cv2.imshow('Edges',edges)
    cv2.imshow('laplacian',laplacian)
    cv2.imshow('sobelx',sobelx)
    cv2.imshow('sobely',sobely)

    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break

cv2.destroyAllWindows()
cap.release()
```

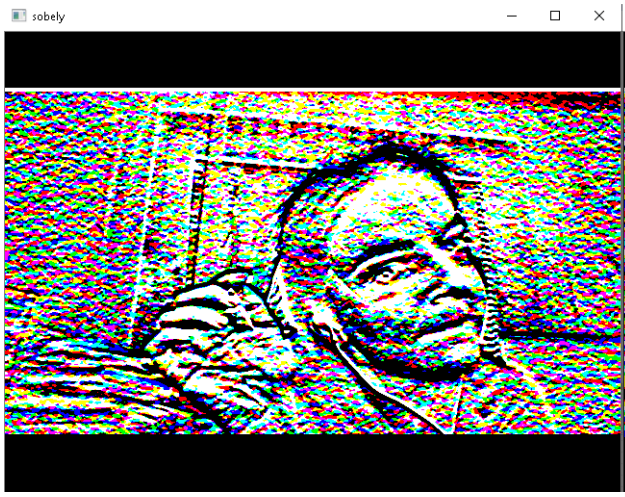
Slika 55. Gradijenti i detekcija ivica



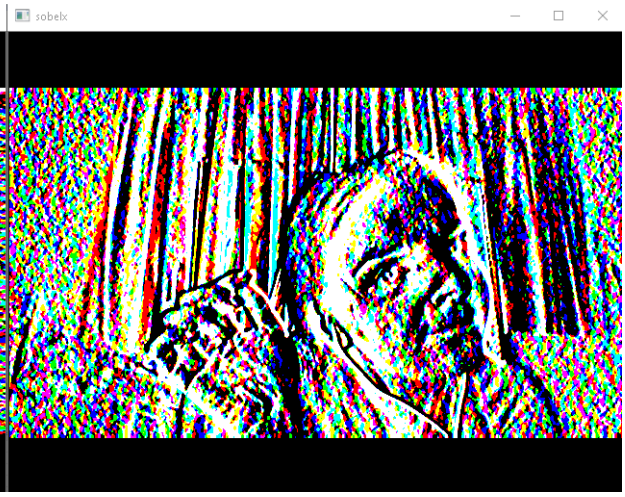
Slika 56. Laplasov



Slika 57. Detekcija ivica



Slika 58. Sobel y



Slika 59. Sobel x

Podudaranje šablona

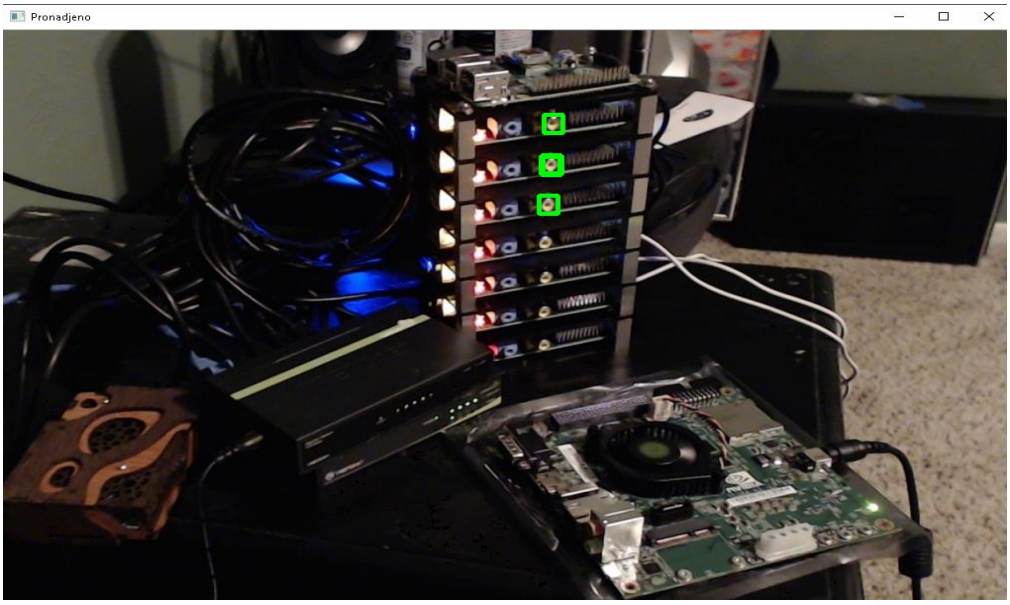
Video 11. Template Matching – OpenCV with Python for Image and Video Analysis

Ideja je pronaći identične oblasti slike koje se podudaraju sa obrascem koji dajemo i koji daje određeni prag. Za tačne podudarnosti predmeta, uz tačno osvetljenje / skalu / ugao, to može biti sjajno. Primer gde se ovi uslovi obično ispunjavaju je bilo koji GUI na računaru. Tasteri i slično su uvek isti, tako da možete koristiti zemplate matching. Uparite template maching sa nekim kontrolama miša i dobili ste a web-based bot.
Za početak će vam trebati glavna(main)slika i template. Treba uzeti svoj template upravo iz „stvari“ koju tražite na slici.
Za sada se učitavamo u obe slike, pretvaramo se u sivo. Zadržavamo originalnu RGB sliku i kreiramo verziju sive boje. Razlog zašto to radimo je taj što radimo svu obradu u verziji sive boje, a zatim koristimo iste koordinate za oznake na slici u boji.
Uz glavnu sliku, imamo samo verziju u boji i verziju sive boje. Učitavamo template i beležimo dimenzije.
Unutar koda zovemo matchTemplate između img_gray (naša glavna slika), template, a zatim metod podudaranja koji ćemo koristiti. Navodimo prag, ovde 0,8 za 80%. Zatim pronalazimo lokacije s logičnom izjavom, gde je

rezolucija veća ili jednaka 80%..Na kraju obeležavamo sve podudarnosti na originalnoj slici koristeći koordinate koje smo pronašli na sivoj slici. Kod je prikaz na *Slici 60.*, a *na Slici 61.* je prikazan rezultat.

```
import cv2
import numpy as np

#velika slika
img_rgb = cv2.imread('opencv-template-matching-python-tutorial.jpg')
img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2GRAY)

#mala
template = cv2.imread('opencv-template-for-matching.jpg',0)

#način da uzmemo shape
w, h = template.shape[::-1]

#rezultati, ccoeff_normed nam daje poklapanja
res = cv2.matchTemplate(img_gray,template,cv2.TM_CCOEFF_NORMED)

#80%
#ukoliko smanjimo prag, bice vise detekcija, nece biti toliko precizan
threshold = 0.9

#lokacija je gde je rezultat podudaranja >= od praga
loc = np.where( res >= threshold)

#oznaka na delove gde imamo poklapanja
for pt in zip(*loc[::-1]):
    cv2.rectangle(img_rgb, pt, (pt[0] + w, pt[1] + h), (0,255,0), 3)

cv2.imshow('Pronadjeno',img_rgb)
```

Slika 60. TemplateMatching



Slika 61. Rezultat – Pronađeno

GrabCut ekstrakcija pozadine

Video 12. GrabCut Foreground Extraction – OpenCV with Python for Image and Video Analysis

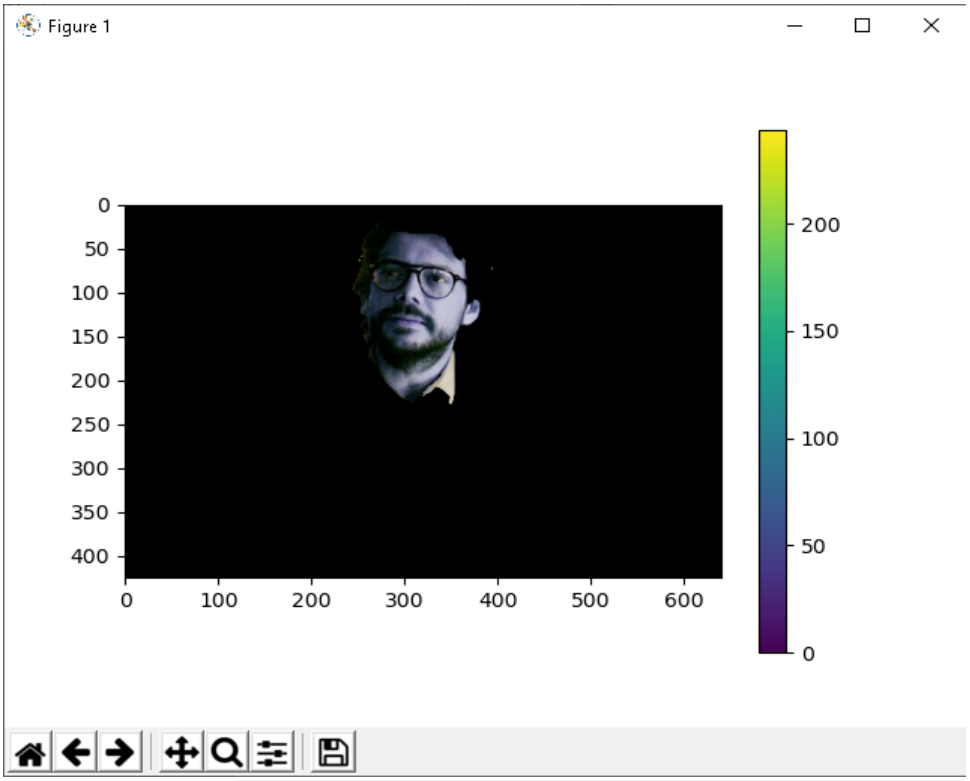
Ideja je pronaći prvi plan i ukloniti pozadinu. Slika koju ćemo koristiti je prikazana na *Slici 62.*, dok je kod prikazan na *Slici 63.*.

Slika 62. Orginal

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('profesor.jpg')
mask = np.zeros(img.shape[:2], np.uint8)

bgdModel = np.zeros((1,65), np.float64)
fgdModel = np.zeros((1,65), np.float64)

#rect = (start_x, start_y, width, height)
rect = (215, 3, 200, 230)

cv2.grabCut(img, mask, rect, bgdModel, fgdModel,5, cv2.GC_INIT_WITH_RECT)
mask2 = np.where((mask==2)|(mask==0), 0, 1).astype('uint8')
img = img*mask2[:,:,np.newaxis]

plt.imshow(img)
plt.colorbar()
plt.show()
```

Slika 63. Kod GrabCut

Učitavamo sliku, kreiramo masku, specificiramo model pozadine (background) i prednjeg plana (foreground), koji algoritam interno koristi. Važan deo je rect koji definišemo, rect = (start_x, start_y, širina, visina). Ovo je pravougaonik koji obuhvata naš glavni predmet.
Koristili smo cv2.grabCut. Prvi argument je ulazna slika, zatim maska, zatim pravougaonik za naš glavni objekt, pozadinski model, model prednjeg plana, količina iteracija koje treba pokrenuti i koji režim koristite. Maska mask2 menja tako da se svi 0 i 2 pikseli pretvaraju u pozadinu, gde su 1 i 3 pikseli sada u prvom planu. Množims s ulaznom slikom i dobijamo konačni rezultat. Rezultat je prikazan na *Slici 64.*.



Slika 64. Rezultat grabCut

Detekcija ugla

Video 13. Corner Detection – OpenCV with Python for Image and Video Analysis

Svrha otkrivanja uglova je praćenje stvari poput pokreta, 3D modeliranje i prepoznavanje objekata, oblika i likova. Slika za detekciju prikazana je na *Slici 65.*.

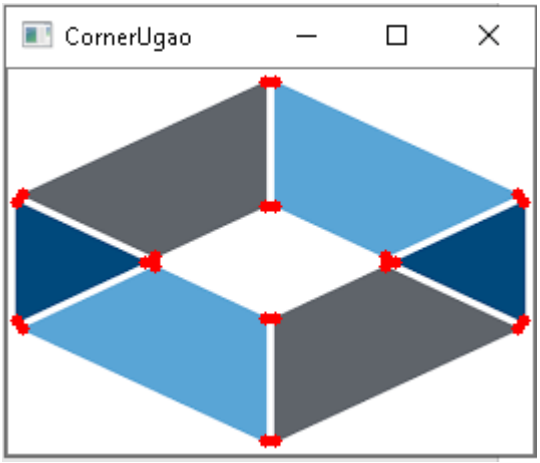
Slika 65. Orginalna slika

Cilj je pronaći sve uglove na slici. Napomenuću da ovde imamo nekih problema s izvlačenjem (nazubljeno-nastim u kosim linijama), tako da ćemo, ako to dozvolimo, naći dosta uglova i to sa pravom. Kao i obično sa OpenCV-om, teški deo je već završen za nas, a sve što treba da uradimo je da unesemo neke parametre. Krenićemo od učitavanja slike i podešavanja nekih parametara. Kod je prikazan na *Slici 66.*.

```
import numpy as np
import cv2

img = cv2.imread('cornerR.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray = np.float32(gray)

corners = cv2.goodFeaturesToTrack(gray, 800, 0.01, 0.5)
corners = np.int0(corners)

for corner in corners:
    x,y = corner.ravel()
    cv2.circle(img, (x,y), 3, (0, 0, 255), -1)

cv2.imshow('CornerUgao', img)
```

Slika 66. Corner Detection

Učitavamo sliku, pretvaramo se u sivu, a zatim u float32. Potomotkrivamo uglove pomoću goodFeaturesToTrack funkcije, "dobre funkcije traženja" . Parametri su slika, maksimalni uglovi za otkrivanje, kvalitet i minimalna udaljenost između uglova. Sada ponavljam za svaki ugao, praveći krug u svakoj tački za koju mislimo da je ugao. Rezultat koda je prikazan na *Slici 67.*.



Slika 67.Rezultat CornerDetection

Podudaranje odlika (Homografija), iscrpljujuća pretraga

Video 14. Feature Matching (Homography) Brute Force – OpenCV with Python for Image and Video Analysis

Započinjemo sa slikom za koju se nadamo da ćemo je pronaći, a zatim možemo da pretražimo tu sliku unutar druge slike. Lepota je u tome što slika ne mora da bude isto osvetljenje, ugao, rotacija ... itd. Karakteristike se jednostavno moraju podudarati.

Za početak, „template" ili slika koju ćemo pokušati uskladiti prikazana je na *Slici 68.*.

Slika 68. template

Zatim naša slika za pretraživanje ovog templejta je prikazana na *Slici 69.*.

Slika 69. Slika za pretraživanje

Ovde je slika našeg templata nešto manja u nego u slici koju ćemo tražiti. Takođe je različita rotacija i ima neke različite senke.
Koristićemo oblik podudaranja "brute force". Naći ćemo sve funkcije u obe slike. Zatim odgovaramo ovim karakteristikama. Tada možemo izvući onoliko koliko želimo. Ali pažljivo. Ako izvučete recimo 500 podudaranja, imaćete puno lažnih pozitivnih rezultata. Nacrtajte samo prvih nekoliko.
Na *Slici 70.* smo prikazali kod. Uvezli smo module koje ćemo koristiti i definisali naše dve slike, template (img1) i sliku u kojoj ćemo tražiti template (img2). Rezultat koda je prikazan na *Slici 71.*.

```
import numpy as np
import cv2
import matplotlib.pyplot as plt

img1 = cv2.imread('opencv-feature-matching-template.jpg',0)
img2 = cv2.imread('opencv-feature-matching-image.jpg',0)

#ovo je detektor koji ćemo koristiti
orb = cv2.ORB_create()

#pronalazimo ključne tačke i njihove deskriptore pomoću detektora orb
kp1, des1 = orb.detectAndCompute(img1,None)
kp2, des2 = orb.detectAndCompute(img2,None)

#ovo je BFMatcher objekat
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

#Ovde kreiramo podudarnosti deskriptora,
#a zatim ih sortiramo na osnovu njihove udaljenosti.
matches = bf.match(des1,des2)
matches = sorted(matches, key = lambda x:x.distance)

# iscrtvamo prvih 10 mečeva
img3 = cv2.drawMatches(img1,kp1,img2,kp2,matches[:10],None, flags=2)
plt.imshow(img3)
plt.show()
```

Slika70. Kod

Slika71. Rezultat Feature Matching

MOG redukcija pozadine

Video 15. MOG Background Reduction – OpenCV with Python for Image and Video Analysis

U ovom OpenCV tutorijalu objašnjeno je kako redukovati pozadinu slika, otkrivajući kretanje. Ovo zahteva da upotrebu video zapisa ili da imamo dve slike, jedna sa izostankom ljudi /objekata koje želite da pratite, a druga sa objektima /ljudima. Koristili smo video sa tutorijala, ljudi koji šetaju, ogledni video.
Ideja ovde je izvući pokretno deo (foreground) iz statičke pozadine (background). Može se koristite ovo poređenje dve slične slike i odmah da se izvuku razlike među njima. U našem slučaju, možemo videti da smo definitivno otkrili neke ljude, ali imamo neku "buku". Buka je zapravo lišće drveća koje se kreće malo pod okolnim vetrom. Kod je prikazan na *Slici 72.* dok je orginalni video prikazan na *Slici 73.*, a sam rezultat na *Slici 74.*.

```
import numpy as np
import cv2

cap = cv2.VideoCapture('people-walking.mp4')
fgbg = cv2.createBackgroundSubtractorMOG2()

while True:
    ret, frame = cap.read()

    fgmask = fgbg.apply(frame)

    cv2.imshow('orignal',frame)
    cv2.imshow('fg',fgmask)


    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

Slika 72. Kod MOG

Slika73. Orginal

Slika 74. Rezultat MOG

Haar kaskadno otkrivanje lica i očiju

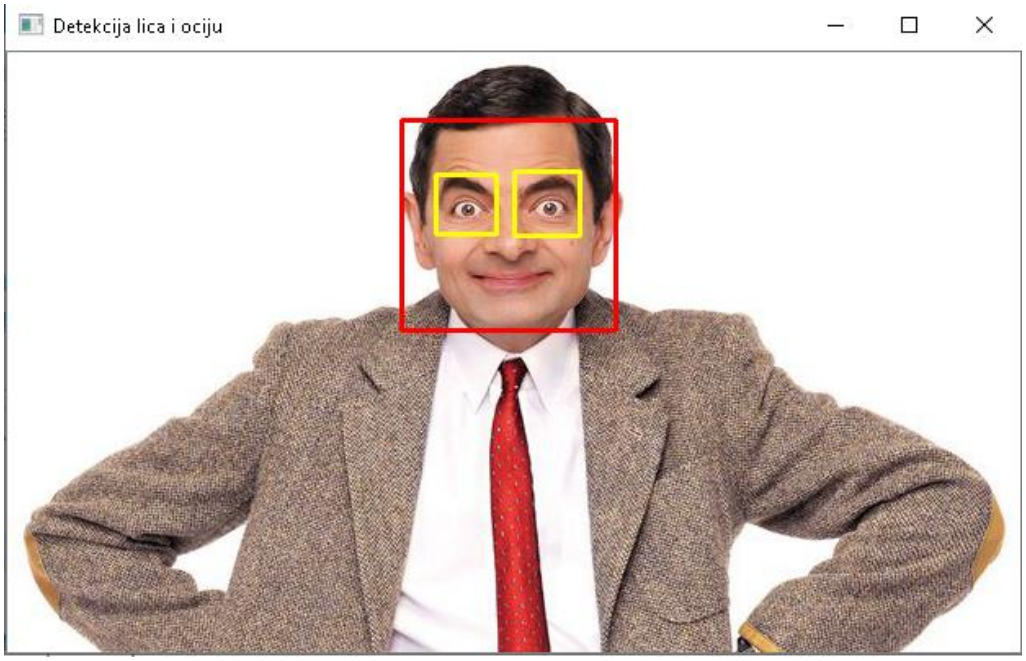
Video 16. Haar Cascade Object Detection Face & Eye – OpenCV with Python for Image and Video Analysis

U ovom tutorijualu radi se o otkrivanju objekata sa Haar kaskadama. Za početak ćemo uraditi otkrivanje lica i očiju. Da biste prepoznali / otkrili objekte sa kaskadnim datotekama, prvo su vam potrebne kaskadne datoteke. Za izuzetno popularne zadatke ovi već postoje. Otkrivanje stvari poput lica, automobila, osmeha, očiju i registarskih tablica, na

primer, prilično je rasprostranjeno. Koristićemo kaskadu lica i kaskadu očiju. Još nekoliko njih možete pronaći u korijenskom direktoriju Haar kaskada. Postoji licenca za korišćenjau ovih Haar kaskada, koji su u navednom linku. Neophodno je da preuzmemo i haarcascade_eie.kml i haarcascade_frontalface_default.kml sa veza u kodu i da imamo ove datoteke u direktoriju projekta.
Unutar koda, počinjemo sa uvozom cv2 i numpy, a zatim se učitavamo kaskade ya lice i oči. Jedina nova stvar ovde je stvaranje lica. U osnovi, to pronalazi lica. Sledeći korak je da prvo slomimo lica pre nego što dođemo do očiju. Pronalazimo lica, njihove veličine, crtanje pravougaonika i primećujemo ROI. Ako oči pronađemo, napravit ćemo , još pravokutnika. Opisani kod je prikzan na *Slici 75.*, dok na *Slici 76.* je prikazan rezultat detekcije.

```
import numpy as np
import cv2
#razlicite cascade se nalaze se na linku:
#https://github.com/Itseez/opencv/tree/master/data/haarcascades

#https://github.com/Itseez/opencv/blob/master/data/haarcascades/haarcascade_frontalface_default.xml
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

#https://github.com/Itseez/opencv/blob/master/data/haarcascades/haarcascade_eye.xml
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')

#mozemo kameru da aktiviramo sa 0, cv2.VideoCapture(0)
cap = cv2.VideoCapture('mr_bean.jpg')

#funkciju detectMultiScale pronalazi lice
while 1:
    ret, img = cap.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)

#pronalazimo lica, njihove veličine, crtamo pravougaonika i  ROI.
    for (x,y,w,h) in faces:
        #rectangle(izvor-slika), pocetne koordinate, krajnje, boja, debljina)
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,0,255),2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]

#ako oci pronađemo, napravit ćemo još pravougaonika.
        eyes = eye_cascade.detectMultiScale(roi_gray)
        for (ex,ey,ew,eh) in eyes:
            cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,255),2)

    cv2.imshow('Detekcija lica i ociju',img)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

Slika 75. Detekcija Kod



Slika 76. Detekcija Rezultat

Izrada sopstvene Haar kaskade – Uvod

Video 17. Making your own Haar Cascade Intro – OpenCV with Python for Image and Video Analysis

U ovom tutorijalu je pokazano kako da napravite svoje sopstvene Haar kaskade, tako da možete da pratite bilo koji predmet koji želite. Za izradu je potreban Linux, Linux server. U ovom tutorijalu je korišćen **Linux** VPS, kao i u svim narednim. Može se koristite besplatni nivo od Amazon Veb Services, mada će on biti bolno prespor i trebaće više RAM-a. Takođe, možete da dotebijete VPS od Digital Ocean za 5 USD mesečno.
Pa, kako zapravo idemo na ovaj proces? Dakle, kada želite da napravite Haar kaskadu, potrebne su vam „pozitivne“ slike i „negativne“ slike. “Pozitivne” slike su slike koje sadrže predmet koji želite da pronađete. To mogu biti slike koje uglavnom samo predmet sadrže ili to mogu biti slike koje sadrže predmet, a vi odredite ROI (region od interesa) gde se objekt nalazi. Pomoću ovih pozitivnih vrednosti mi gradimo vektorsku datoteku koja je u osnovi svih ovih pozitivnih vrednosti zajedno. Jedna dobra stvar kod pozitiva je ta što zapravo možete da imate samo jednu sliku objekta koji želite da otkrijete, a zatim imate nekoliko hiljada negativnih slika. Da, nekoliko hiljada. Negativne slike mogu biti bilo šta, osim što ne mogu sadržavati vaš objekt.
Odavde, sa jedinstvenom pozitivnom slikom, možete koristiti naredbu *opencv_createsamples* da ustvari stvorite gomilu pozitivnih primera, koristeći svoje negativne slike. Vaša pozitivna slika će biti naslonjena na ove negativnosti i biće pod uglom i svakakvim stvarima. Zapravo može delovati prilično dobro, pogotovo ako stvarno tražite samo jedan određeni objekt. Ako želite da identifikujete sve odvijače (screwdrivers), htećete da imate hiljade jedinstvenih slika odvijača, umesto da koristite *opencv_createsamples* za generisanje uzoraka za vas. Jednostavno ćemo koristiti i koristiti jednu pozitivnu sliku, a zatim ćemo stvoriti gomilu uzoraka s našim negativima.

Slika 77. Pozitiv slika

Postoji prilično koristan sajt, zasnovan na konceptu VordNet-a, pod nazivom ImageNet. Odavde možete pronaći slike gotovo svega. U našem slučaju želimo satove, pa tražite satove, a vi ćete pronaći tona kategorija satova. Idemo sa analognim satovima. Koristiti ćemo samo onaj pozitivni, tako da ćemo samo otkriti jedan sat. Ako želite da otkrijete “sve” satove, pripremite se da dobijete više od 50.000 slika satova i najmanje 25.000 “negativnih”slika. Nakon toga, pripremite se da imate prilično server, osim ako ne želite da vam Haar Cascade trening traje nedelju dana. Pa kako možemo dobiti negativ? Cela poanta ImageNet-a je u treningu slika, tako da su njihove slike prilično specifične. Dakle, ako tražimo ljude, automobile, čamce, avione ... šta god, šanse su, neće biti satova. Možda ćete videti neke satove kod ljudi ili nešto slično, ali imate ideju. Pošto ćete verovatno naći satove oko ili na ljudima, zapravo mislim da ćete možda dobiti slike ljudi.

Dakle, prvo ono što ćemo ovde uraditi je pisanje brze skripte koja će posetiti ove spiskove URL-ova, preuzeti veze, posećivati veze, izvući slike, promeniti veličinu, sačuvati ih i ponoviti dok ne završimo. Kada su naše mape pune slika, potrebna nam je i vrsta datoteke opisa koja opisuje slike. Za pozitivne napade, ova datoteka predstavlja veliku masu za ručno kreiranje, jer morate da odredite tačnu oblast interesa za vaš objekat, po slici. Srećom, metoda *create_samples* postavlja sliku nasumično i radi sve što radi za nas. Potreban nam je samo jednostavan deskriptor negativa, ali to nije problem, to možemo učiniti dok povlačimo i manipuliramo slikama.

Skupljanje slika za Haar kaskadu

Video 18. Gathering Images for Haar Cascade – OpenCV with Python for Image and Video Analysis

Skripta za skupljanje slika sa url linka je prikazana na *Slici 78.*,

```
import urllib.request
import cv2
import numpy as np
import os

def store_raw_images():
    neg_images_link = '//image-net.org/api/text/imagenet.synset.geturls?wnid=n00523513'
    neg_image_urls = urllib.request.urlopen(neg_images_link).read().decode()
    pic_num = 1

    if not os.path.exists('neg'):
        os.makedirs('neg')

    for i in neg_image_urls.split('\n'):
        try:
            print(i)
            urllib.request.urlretrieve(i, "neg/"+str(pic_num)+".jpg")
            img = cv2.imread("neg/"+str(pic_num)+".jpg",cv2.IMREAD_GRAYSCALE)
            # should be larger than samples / pos pic (so we can place our image on it)
            resized_image = cv2.resize(img, (100, 100))
            cv2.imwrite("neg/"+str(pic_num)+".jpg",resized_image)
            pic_num += 1

        except Exception as e:
            print(str(e))
```

Slika 78. Skript za preuzimanja slika pomoću linka

Ovaj skript će posetiti veze, preuzeti URL-ove i nastaviti da ih posećuje. Hvatamo sliku, pretvaramo je u nijanse sive, menjamo veličinu i zatim je čuvamo. Za imenovanje slika koristimo jednostavan brojač. Samo napred i trči. Kao što verovatno vidite, ima puno slika koje nedostaju i slično. U redu je. Problematičnije su neke od ovih slika grešaka. U osnovi su bele s nekim tekstom koji kaže da više nisu dostupni. Sada imamo nekoliko izbora. To možemo jednostavno ignorisati ili popraviti. Možete ih ručno izbrisati ... ili možemo jednostavno koristiti naše novo znanje analize slike da otkrijemo slike i uklonimo ih.
U sledećem tutorijalu, 19. po redu, napravljen je novi direktorij, sa nazivom „uglies“. Unutar tog direktorija, povukao je sve verzije ružnih slika (samo jednu od svake). Postoji samo jedan glavni prestupnik koji je pronađen sa negativima. Napišimo skriptu da bismo pronašli sve instance ove slike i izbrisali je. To sve je urađeno u sledećem poglavlju.

Čišćenje slika i kreiranje opisnih fajlova

Video 19. Cleaning images and creating description files – OpenCV with Python for Image and Video Analysis

Za sada imamo samo negative. Pokrećemo program za uklanjanje slika još jednom, samo pomoću URL-a: //image-net.org/api/tekt/imagenet.sinset.geturls?vnid=n07942152. Poslednja slika je bila # 952, pa započnimo pic_num na 953 i promenimo URL, kod funkcije je na *Slici 80.* prikazan.

```
def find_uglies():
    match = False
    for file_type in ['neg']:
        for img in os.listdir(file_type):
            for ugly in os.listdir('uglies'):
                try:
                    current_image_path = str(file_type)+'/'+str(img)
                    ugly = cv2.imread('uglies/'+str(ugly))
                    question = cv2.imread(current_image_path)
                    if ugly.shape == question.shape and not(np.bitwise_xor(ugly,question).any()):
                        print('That is one ugly pic! Deleting!')
                        print(current_image_path)
                        os.remove(current_image_path)
                except Exception as e:
                    print(str(e))
```

Slika 79. Funkcija find_uglies

```
#treci deo
def store_raw_images():
    neg_images_link = '//image-net.org/api/text/imagenet.synset.geturls?wnid=n07942152'
    neg_image_urls = urllib.request.urlopen(neg_images_link).read().decode()
    pic_num = 953

    if not os.path.exists('neg'):
        os.makedirs('neg')

    for i in neg_image_urls.split('\n'):
        try:
            print(i)
            urllib.request.urlretrieve(i, "neg/"+str(pic_num)+".jpg")
            img = cv2.imread("neg/"+str(pic_num)+".jpg",cv2.IMREAD_GRAYSCALE)
            # should be larger than samples / pos pic (so we can place our image on it)
            resized_image = cv2.resize(img, (100, 100))
            cv2.imwrite("neg/"+str(pic_num)+".jpg",resized_image)
            pic_num += 1

        except Exception as e:
            print(str(e))
```

Slika 80. Funkcija store

Sada imamo preko 2000 slika. Poslednji korak je da moramo kreirati datoteku deskriptora za ove negative. Opet ćemo koristiti kod prikazan na *Slici 81.*.

```
#cetvrti deo
def create_pos_n_neg():
    for file_type in ['neg']:

        for img in os.listdir(file_type):

            if file_type == 'pos':
                line = file_type+'/'+img+' 1 0 0 50 50\n'
                with open('info.dat','a') as f:
                    f.write(line)
            elif file_type == 'neg':
                line = file_type+'/'+img+'\n'
                with open('bg.txt','a') as f:
                    f.write(line)
```

Slika 81. Funkcija kreiranja

Na tutorijalu je pokrenuo sve i dobio bg.tkt datoteku. Neophodno je napraviti *create_samples*. To znači da moramo preseliti *neg* direktorijum i *bg.tkt* datoteku na „naš server“.

Treniranje Haar kaskadnog objekta detekcije

Video 20. Training Haar cascade object detection – OpenCV with Python for Image and Video Analysis

Na Windows se ne može create_samples i slično. Pravi je čarobnjak, ko je uspeo.
Prvo imate ime datoteke, zatim imate koliko vaših objekata na slici, a zatim sve njihove lokacije. Imamo samo jedan, tako da je to x, y, širina i visina pravougaonika za objekt unutar slike. Na *Slici 82.* je prikazana jedna od slika.

Slika 82. Slika

Teško je to videti, ali sat je na ovoj slici. Spusti se levo od leve strane osobe na slici. Dakle, ovo je „pozitiv“ slika, stvorena iz inače „negativne“ slike, a ta negativna slika će se takođe koristiti u treningu. Sada kada imamo pozitivne slike, sada moramo da stvorimo vektorsku datoteku, koja je u osnovi tamo gde zajedno spajamo sve naše pozitivne slike. Za to ćemo ponovo koristiti *opencv_createsamples*! Vektor fajl: *opencv_createsamples -info info/info.lst -num 1950 -w 20 -h 20 -vec positives.vec*
Sada treba da se pokrene naredba treniranja: *opencv_traincascade -data data -vec positives.vec -bg bg.txt -numPos 1800 -numNeg 900 -numStages 10 -w 20 -h 20*
Ovde kažemo gde želimo da podaci odu, gde je vektorska datoteka, gde je pozadinska datoteka, koliko pozitivnih slika i negativnih slika da se koristi, koliko faza i širina i visina. Imajte na umu da koristimo znatno manje numPos-a nego što imamo. Ovime se želi stvoriti prostor za stages, koje će dodati.
Glavni su ovde pozitivni i negativni brojevi. Opšti mišljenje je da u većini praksi želite da imate odnos 2: 1 u odnosu na: neg slike. Neke se situacije mogu razlikovati, ali ovo je opšte pravilo, za koje se čini da ljudi slede. Prva faza je obično prilično brza, faza 5 mnogo sporija, a faza 50 zauvek! Dakle, za sada radimo 10 faza. Ovde je uredna stvar da možete trenirati 10 etapa, vratiti se kasnije, promeniti broj u 20 i pokupiti se tamo gde ste stali. Slično tome, možete ubaciti nešto poput 100 faza, otići u krevet, probuditi se ujutro, zaustaviti ga, videti koliko ste stigli, zatim „trenirati“ sa toliko faza i odmah će vam se predstaviti kaskadna datoteka. Rezultat ove naredbe je zaista velika i sveta kaskadna datoteka. Ako zaista želite da pokrenete naredbu, ali ne želite da ostavite terminal otvoren, možete iskoristiti nohup: *nohup opencv_traincascade -data data -vec positives.vec -bg bg.txt -numPos 1800 -numNeg 900 -numStages 10 -w 20 -h 20 &*
Ovo će omogućiti komandi da nastavi sa pokretanjem, čak i nakon što zatvorite terminal.

Haar kaskada za klasifikaciju slika i video objekata

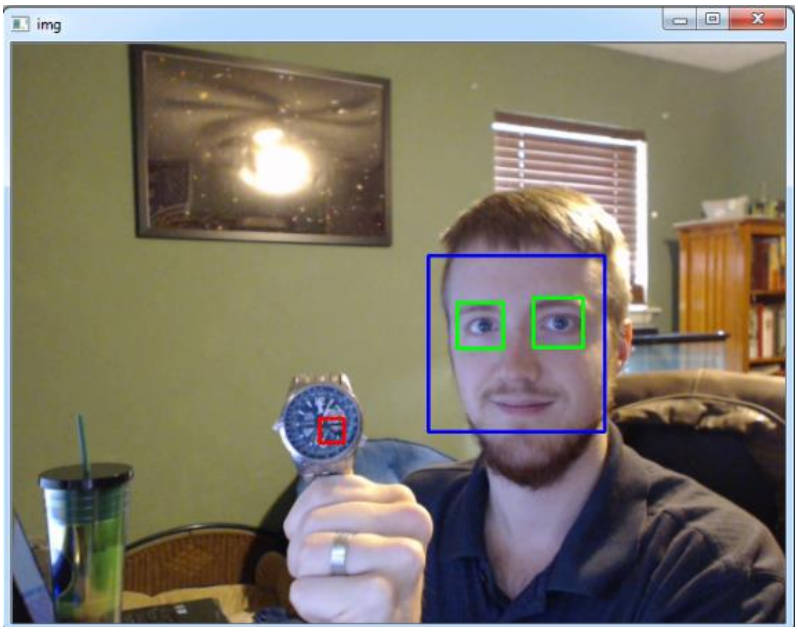
Video 21. Haar Cascade for image & video object classification – OpenCV w/ Python for Image Video Analysis

10 faza je trajalo nešto manje od 2 sata na tutorijalu, koristili su 2GB Digital Ocean serveru. Dakle, ili imate datoteku cascade.xml, ili ste zaustavili pokretanje skripte. Ako ste ga sprečili da radi, trebalo bi da imate gomilu stageKs.xml datoteka u vašem direktorijumu "data". Otvorite to, pogledajte koliko ste etapa napravili, a zatim možete ponovo pokrenuti *opencv_traincascade*, s tim brojem faza i odmah ćete dobiti datoteku *cascade.xml*.

```
import numpy as np
import cv2

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')

#kreirana kaskada
watch_cascade = cv2.CascadeClassifier('watchcascade10stage.xml')

cap = cv2.VideoCapture(0)

while 1:
    ret, img = cap.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)

    # add this
    # image, reject levels level weights.
    watches = watch_cascade.detectMultiScale(gray, 50, 50)

    # add this
    for (x,y,w,h) in watches:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,255,0),2)

    for (x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)


        roi_gray = gray[y:y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]
        eyes = eye_cascade.detectMultiScale(roi_gray)
        for (ex,ey,ew,eh) in eyes:
            cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)

    cv2.imshow('img',img)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

cap.release()
```

Slika 83. Kod sa kreiranom kaskadom

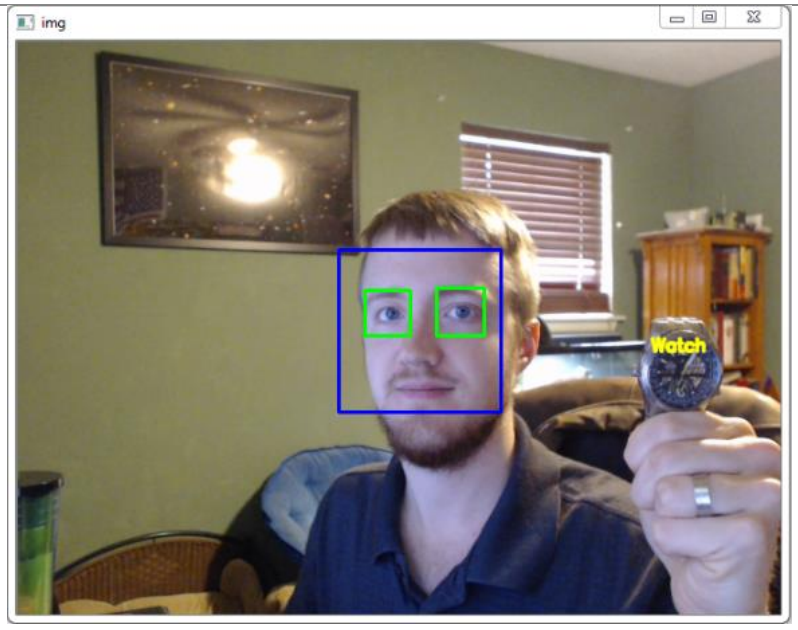
Nakon pokretanja koda, dobio je rezultat na *Slici 84.*



Slika 84. Rezultat

Umesto da se crta pravougaonik, zašto jednostavno ne napišemo tekst preko sata ili tako nešto? To je relativno jednostavno. Umesto da u satovima radimo *cv2.rectangle(img,(x,y),(x+w,y+h),(0,0,255),2).* Rezulata je prikazan na *Slici 85.*

```
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(img,'Watch',(x-w,y-h), font, 0.5, (11,255,255), 2, cv2.LINE_AA)
```



Slika 85. Rezultat Tekst

Reference

[1] Wikipedia the free encyclopedia, OpenCV (2020) , link: https://en.wikipedia.org/wiki/OpenCV

[2] Dragan Lazarević, Dr Milan Mišić , Dr Bogdan Ćirković, POSTOJEĆE TEHNIKE ZA SEGMENTACIJU SLIKE, Pristupljeno: 21. 04.2020. godine, link: http://www.cqm.rs/2014/cd2/pdf/papers/focus_1/10.pdf