

Univerzitet u Kragujevcu

Seminarski rad
iz predmeta Računarski vid

Tema:

Implementacija prebrojavanja entiteta na slici

mentor: prof. dr. Bogdan Milićević,

prof. dr. Igor Saljević

student: Željko Simić 3vi/2023

Kragujevac 2024.

1 Uvod

U ovom radu biće obrađena tema u vezi korišćenje implementacija koje su bile predviđene za prebrojavanje čelija kao i drugih objekata kroz predstavljena 3 programa. Uz uzete ulazne slike, biće primenjene obrade zarad efikasnijeg prepoznavanja ivica ili kontura, boja, kao i obrade tih kontura (spajanjima, podebljavanjima, konveksnih poleđina - hull).

Ovde će u obzir biti uzeto prebrojavanje elemenata na stampanoj ploči, kao i prepoznavanje kondenzatora na slici. Elementi koji mogu se naći vodova, kontakata, natpisa. Biće istaknuti konturama crvene boje na izlaznoj slici sa njihovom brojnošću.

2 Opis metoda i funkcija

Prvi program

U implementaciji 1.[1] uključene su biblioteke u rad programa: cv2, numpy, matplotlib-ove funkcionalnosti plotovanja.

```
1 import numpy as np
2 import cv2
3 import matplotlib.pyplot as plt
```

Implementacija 1: Uključivanje biblioteka

U implementaciji 2. u promenljivu `image` smešta se objekat učitane slike putem `cv2` funkcionalnosti `imread(...)` kojoj se prosleđuje relativna putanja do slike, gde je implicitno parametar `flag` postavljen na `IMREAD_COLOR` - koja ukazuje da se slika čita u memoriju sa sva 3 kanala za prezentaciju boja, samo što je u BGR (plava, zelena, crvena) redosledu boja za svaki bajt.

Vrši se takođe konverzija boja slike koja je uskladištena u promenljivu `image` putem `cv2`-ovog `cvtColor`-a i ukazuje se 2. parametrom kakva koverzija se obavlja. `COLOR_BGR2GRAY` označuje da se prelazi u režim grayscale, počevši od BGR režima. Skladišti se povratna vrednost u promenljivu `gray`.

Pritom, ističe se izlazni oblik rezultata u vidu plotovane slike uz `matplotlib`-ove `pyplot` funkcije `imshow` kojoj se prosleđuju argumenti promenljive slike `gray` i mapa boja koja će se koristiti parametrom `cmap` podešenim na `'gray'`.

```
1 image = cv2.imread('sp4.jpg')
2 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
3 plt.imshow(gray, cmap='gray')
```

Implementacija 2: Učitavanje slike, konvertovanje u sivu sliku, plotiranje

U implementaciji 3. vrši se zamagljivanje slike `GaussianBlur` u promenljivoj `gray` koja je prosleđena kao 1. argument, a torka od 2 elementa čine 2. argument koji podešavaju dimenziju veličine Gausovog kernel-a, a 3. argument je standardna devijacija. Namenjen za osposobljavanje prebrojavanja detekcijom ivica zarad uklanjanja šuma na slici. Takođe će biti prikazana isplotovana slika.

```
1 blur = cv2.GaussianBlur(gray, (11, 11), 0)
2 plt.imshow(blur, cmap='gray')
```

Implementacija 3: Učitavanje slike

U implementaciji 4. koristi se `Canny` algoritam zarad detektovanja ivica, 2. i 3. argumenti funkcije su pragovi kao vrednosti u domenu [30, 150] i uzimaju se kao ivice na slici, ilustrovano

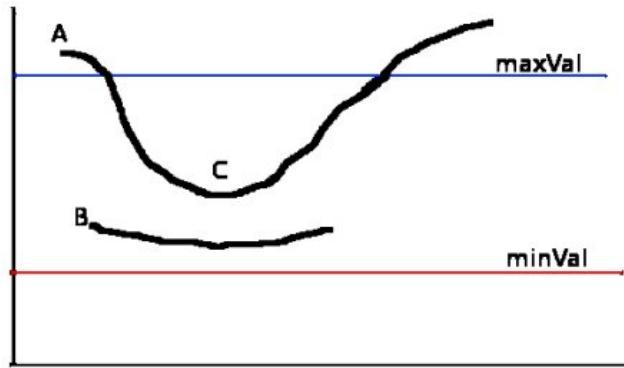
je na slici 1. 4. argument[2] se koristi za definisanje veličine Sobel kernela kroz kog se filtrira slika kojim se računaju po horizontalnom i vertikalnom pravcu, tj. G_x i G_y .

$$\text{Edge_Gradient } (G) = \sqrt{G_x^2 + G_y^2} \text{Angle } (\theta) = \tan^{-1} \left(\frac{G_y}{G_x} \right)$$

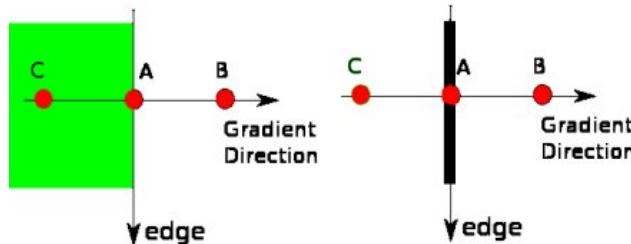
Pod pravim uglom je izračunat gradijent naspram ivica. Sve će se kategorisati u 4 uopštena uglova koji reprezentuju pravce horizontalni, vertikalni, 2 dijagonale. A zatim se uočavaju lokalni maksimumi u susetstvu po pravcu gradijenta kao na slici 2., gde tačkom A na ivici po vertikalnoj pravoj, tačkom B i C pokazuju pravac gradijenta naspram tačke A, pa tako je prihvaćena ako formira lokalni maksimum, inače je odbačena.

```
1 canny = cv2.Canny(blur, 30, 150, 3)
2 plt.imshow(canny, cmap='gray')
```

Implementacija 4: Istanje i plotovanje detektovanih ivica



Slika 1: Histerezis prag

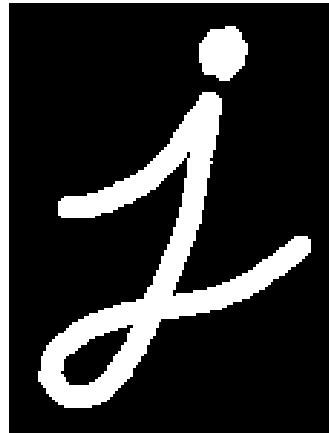


Slika 2: Nemaksimizujuće potiskivanje

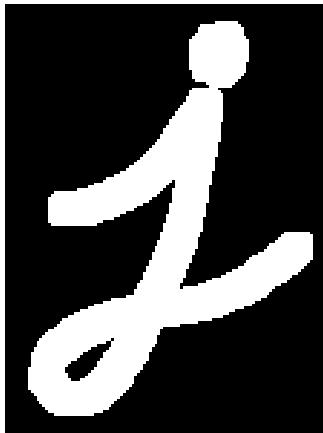
Po izostanku povezanosti ivica u implementaciji 5. se vrši povezivanje istih. Ivice će postati podebljani i vidljivije. Obrada koja će se obaviti je morfološka na osnovu oblika, uz primenu strukturnoških elemenata, tj. funkcija `dilation`[3]. Parametri su redom, ulazna slika, velicina kernela naspram kog je obavljena konvolucija, a dimenzije moraju biti neparne. Uvećava oblast beline na slici ili veličinu prvog plana slike, dat je primer na slici 3. i 4.

```
1 dilated = cv2.dilate(canny, (1, 1), iterations=0)
2 plt.imshow(dilated, cmap='gray')
```

Implementacija 5: Povezivanja i podebljivanje ivica



Slika 3: Početna slika

Slika 4: Izlazna slika
nakon
dilation-a

U implementaciji 6. nalaze se konture na slici cv2 funkcijom `findContours`.[4] Kontura važi za liniju koja spaja sve tačke kroz granice slike koje imaju isti intenzitet (tačaka). Najbolje je primenljiva nad slikama binarnih boja, tako da prethodno odrađeno detektovanje ivica je bilo delotvorno. Parametri koji su zastupljeni:

1. ulazna slika - gde je urađeno duboko kopiranje;
2. režim prihvatanja kontura - vrednost je `RETR_EXTERNAL` - što daje spoljašnje konture[5];
3. režim aproksimiranja kontura - kojom se čuvaju (x,y) koordinate granica oblika tačaka istih intenziteta. Ali ovime je ustanovljeno da li se baš sve tačke čuvaju ovim metodom. Vrednost je `CHAIN_APPROX_NONE`, kojom se naznačava da se sve tačke konture čuvaju.

A, višestruke povratne vrednosti koje su zastupljene:

1. izlazna slika (ali ovde je zanemarena);
2. konture;
3. hijerarhija.

Slika se iz BGR režima pretvara u RGB režim sačuvana u `rgb` promenljivu.

Iscrtavaju se konture sa funkcijom `drawContours` kojoj su prosleđene slika,[6] konture, vrednost `-1` koja označava da se iscrtavaju sve konture, torka elemenata domena $[0, 255]$ RGB režima boja za svaki kanal i vrednost `2` za debeljinu iscrtane konture.

```

1   (cnt, hierarchy) = cv2.findContours(
2       dilated.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
3   rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
4   cv2.drawContours(rgb, cnt, -1, (255, 0, 0), 2)
5
6   plt.imshow(rgb)

```

Implementacija 6: Spajanje kontura, isticanje kontura na slici, plotovanje novonastale slike

U implementaciji 7. ispisuje se brojnost kontura.

```
1   print("coins in the image : ", len(cnt))
```

Implementacija 7: Ispis broja kontura

Drugi program - prebrojavanje boja

U implementaciji 8. koristiće se numpy i cv2 biblioteke.[7]

```
1 # import the necessary packages
2 import numpy as np
3 import cv2
```

Implementacija 8: Uključivanje biblioteka

U implementaciji 9. priprema se rečnik counter koji će se koristiti za prebrojavanje elemenata.

```
1 # dict to count colonies
2 counter = {}
```

Implementacija 9: Iniciranje rečnikom

U implementaciji 10. vrši se učitavanje slike, a zatim se izvlače visina i širina slike.

```
1 # load the image
2 image_orig = cv2.imread("kondenzatori.jpg")
3 height_orig, width_orig = image_orig.shape[:2]
```

Implementacija 10: Učitavanje slike

U implementaciji 11. vrši se duboko kopiranje slike zarad kasnijeg generisanja slike sa konturnama.

```
1 # output image with contours
2 image_contours = image_orig.copy()
```

Implementacija 11: Duboko kopiranje slike

U implementaciji 12. definiše se lista elemenata vrednosti naziva boja koje će se detektovati, tj. plave i žute. Zatim se ostvaruje svaka iteracija pretlje po boji.

```
1 # DETECTING BLUE AND YELLOW CAPACITORS
2 colors = ['blue', 'yellow']
3 for color in colors:
```

Implementacija 12: Započeto iteriranje po bojama

U implementaciji 13. vrši se duboko kopiranje slike zarad neophodne međuobrade.

```
1 # copy of original image
2 image_to_process = image_orig.copy()
```

Implementacija 13: Duboko kopiranje

U implementaciji 14. inicira se vrednost za ključ boje rečnika brojača na 0.

```
1 # initializes counter
2 counter[color] = 0
```

Implementacija 14: Iniciranje odgovarajućeg ključa brojača

U implementaciji 15. na osnovu iteracije na osnovu boje definisu se BGR vektori boja za donju i gornju granicu boja koje se mogu uvažiti kao detektovana posmatrana boja.

```
1 # define NumPy arrays of color boundaries (GBR vectors)
2 if color == 'blue':
3     lower = np.array([ 60, 100, 20])
4     upper = np.array([170, 255, 150])
5 elif color == 'yellow':
6
7     lower = np.array([ 50, 100, 100])
8     upper = np.array([100, 255, 255])
```

Implementacija 15: Nameštanje granica na osnovu boja



Slika 5: Početna slika

Slika 6: Izlazna slika
nakon erode-a

U implementaciji 16. se proverava `inRange`-om da li elementi liste leže između elemenata druge 2 liste, a broj elemenata određen zavisno od dimenzije. Po tome se određuje da li će pikselu kanal biti skroz zanemaren ili postavljen na maksimalnu vrednost, pa se sve to smesti u `image_mask`, u vidu bitovske maske.[8] Potom se proverava da li maska 1 bajta je nenula vrednost, ako jeste onda se uvažava izračunata konjunkcija bajtova između 2 prosleđene slike.[9]

```

1 # find the colors within the specified boundaries
2 image_mask = cv2.inRange(image_to_process, lower, upper)
3 # apply the mask
4 image_res = cv2.bitwise_and(image_to_process, image_to_process, mask =
image_mask)
```

Implementacija 16: Bitovsko maskiranje i konjunkcije obavljene nad slikom

U implementaciji 17. se vrši konverzija iz BGR režima u grayscale `cvtColor`-om, pa se potom vrši zamagljivanje uz `GaussianBlur` gde je velicina Gausovog kernela (5,5), a standardna devijacija na obe ose postavljena je na 0.

```

1 ## load the image, convert it to grayscale, and blur it slightly
2 image_gray = cv2.cvtColor(image_res, cv2.COLOR_BGR2GRAY)
3 image_gray = cv2.GaussianBlur(image_gray, (5,5), 0)
```

Implementacija 17: Prosivljivanje i zamagljivanje slike

U implementaciji 18. vrši se prepoznavanje ivica uz `Canny` i njihovo poboljšavanje `dilate`-om u 1 iteraciji. Erozija je primenjena računanjem lokalnog minimuma (za razliku od `dilation`-a) nad oblašću zadatog kernel-a. Sada ivice bivaju tanje gde može se videti na slici 5 i 6. Uz ovu kombinaciju prvo se podebljanije vrši da bi se dobio efekat spajanja svih ivica, tako da nema prekida, a onda se istanjuju novodobijene ivice.

```

1 # perform edge detection, then perform a dilation + erosion to close
2 # gaps in between object edges
3 image_edged = cv2.Canny(image_gray, 50, 100)
4 image_edged = cv2.dilate(image_edged, None, iterations=1)
5 image_edged = cv2.erode(image_edged, None, iterations=1)
```

Implementacija 18: Detekcija ivica i njihova obrada

Pronalaze se konture uz `findContours` u implementaciji 19. samo što za razliku od prvog programa ovde se izvlače tačke po režimu metoda aproksimiranja samo one krajnje ako je u

pitanju prava linija, inače kao i uvek sve tačke se uzimaju u obzir. Zatim se uzima prvi element liste koja je dobijena ovom funkcijom.

```
1 # find contours in the edge map
2 cnts = cv2.findContours(image_edged.copy(), cv2.RETR_EXTERNAL, cv2.
CHAIN_APPROX_SIMPLE)[0]
```

Implementacija 19: Detekcija kontura

Ulazi se u ugnježdenu petlju u glavnoj petlji koja iterira kroz konture u implementaciji 20.

```
1 # loop over the contours individually
2 for c in cnts:
```

Implementacija 20: Započeto iteriranje

Proverava se da li ima dovoljno kontura neophodnih za uzimanje u obzir u implementaciji 21.

```
1 # if the contour is not sufficiently large, ignore it
2 if cv2.contourArea(c) < 5:
3     continue
```

Implementacija 21: Ispitivanje kriterijuma dovoljnosti brojnosti kontura

Određuje se konveksna poleđina konture, tj. zove se cv2 funkcija convexHull u implementaciji 22.

```
1 # compute the Convex Hull of the contour
2
3 hull = cv2.convexHull(c)
```

Implementacija 22: Računanje konveksnih poleđina

U implementaciji 23. će se iscrtati konture drawContours-om, prosleđena je slika, lista u kojoj je samo jedan element promenljiva hull, iscrtava se nulta kontura pošto je prosleđena 0 za identifikator. Za plavu boju se iscrtava crvena kontura, a za žutu zelena, debljina konture će biti 1. Ovde se ugnježđena petlja koja iterira kroz konture završava.

```
1         if color == 'blue':
2             # prints contours in red color
3             cv2.drawContours(image_contours,[hull],0,(0,0,255),1)
4         elif color == 'yellow':
5             # prints contours in green color
6             cv2.drawContours(image_contours,[hull],0,(0,255,0),1)
```

Implementacija 23: IsCRTavanje kontura na osnovu boja

U implementaciji 24. se za boju brojač inkrementira za 1. I tu se završava petlja koja iterira kroz boje.

```
1 counter[color] += 1
2 # Print the number of capacitors of each color
3 print("{} {} capacitors".format(counter[color],color))
```

Implementacija 24: Ispis broja kontura po boji

U implementaciji 25. se izlazna slika sa svim konturama iscrtanim ubacuje u datoteku out.png.

```
1 # Writes the output image
2 cv2.imwrite("out.png",image_contours)
```

Implementacija 25: Izbacivanje fajla rezultujuće slike

Treći program

U implementaciji 26. sve na početku se vrši kao i u prvom programu, navođenje biblioteka i uključivanje slike iz zadatog fajla, sa grayscale bojama.[10]

```
1 import numpy as np
2 import cv2
3 img = cv2.imread("sp4.jpg", 0)
```

Implementacija 26: Učitavanje modula, učitavanje fajla slike

U implementaciji 27. se `fastNlMeansDenoising`-om vrši odšumljivanje koristeći se algoritmom nelokalizujućih središta sa nekolicinom sračunljivih optimizacija. Šum se očekuje da bude Gausovog belog šuma. Očekuje se da se radi sa grayscale slikama.[11] Implicitno se postavljaju parametri:

- prozora pretrage `search_window` sa vrednošću 21 - gde se određuje veličina po pikselima prozora kojim će se računati prosečna težina piksela posebno, neparna je vrednost, linearno utiče na performanse.
- veličina bloka sa vrednošću 7 - veličina po pikselima šablonu zaskrpe kojim se računaju težine, neparna.

Metod je zasnovan na jednostavnom principu, zamena boja piksela sa prosečno zastupljrenom bojom sličnih piksela. Ali, od najuobičajenijih piksela do posmatranog piksela nemaju potrebe biti bliski. Dozvoljeno je skenirati poveći deo slike pri pretrazi naspram svih piksela koji se stvarno razaznaju naspram piksela koji će biti odšumljivan.[12]

```
1 # Denoising
2 denoisedImg = cv2.fastNlMeansDenoising(img);
```

Implementacija 27: Odšumljivanje

U implementaciji 28. zove se `cv2`-ova `threshold` funkcija.[13] Postavljaju se argumenti ranije obrađivana slika, prag, maksimalna vrednost koja će se kombinovati sa režimima definisanim `THRESH_BINARY` i `THRESH_BINARY_INV` (obrnut uslov anuliranja i postavljanja maksimalne vrednosti piksela), i flegovi `THRESH_BINARY_INV` i `THRESH_OTSU` popaljeni bitskom disjunkcijom.

Za svaki piksel primenjuje se isti prag. Ako vrednost piksela je manja od praga, ona se anulira. Inače, postavlja se na korisnički postavljenu maksimalnu vrednost. Slika koja se prosleđuje u funkciju mora biti grayscale.

Po globalnom usklađivanju pragom, koristi se arbitrarno izabrana vrednost kao prag. Nasuprot tome, **Otsu-ov metod** koji zaobilazi biranje vrednosti i određuje ga automatski. Uzima se u obzir slika sa samo 2 vrednosti piksela slike (binarna boja), gde histogram biva sastojan od samo 2 amplitude. Prag je određen tako da minimizuje težine disperzije unutar klasa.

$$\sigma_w^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t)$$

$$q_1(t) = \sum_{i=1}^t P(i) \quad \& \quad q_2(t) = \sum_{i=t+1}^I P(i)$$

$$\mu_1(t) = \sum_{i=1}^t \frac{iP(i)}{q_1(t)} \quad \& \quad \mu_2(t) = \sum_{i=t+1}^I \frac{iP(i)}{q_2(t)}$$

$$\sigma_1^2(t) = \sum_{i=1}^t [i - \mu_1(t)]^2 \frac{P(i)}{q_1(t)} \quad \& \quad \sigma_2^2(t) = \sum_{i=t+1}^I [i - \mu_2(t)]^2 \frac{P(i)}{q_2(t)}$$

Sve je to odrađeno tako da vrednost t koja leži na oblasti među dveju amplituda tako da disperzija obe klase je minimalna.

Dobar prag može biti sredina dveju vrednosti. I ovim metodom određen je optimalan globalni prag kao vrednost uz pomoć histograma.

```
1 th, threshedImg = cv2.threshold(denoisedImg, 200, 255, cv2.THRESH_BINARY_INV |
    cv2.THRESH_OTSU) # src, thresh, maxval, type
```

Implementacija 28: Nameštanje praga

U implementaciji 29. funkcija `getStructuringElement` sa sobom definiše struktuirajući element, tj. kernel kojim se ističu režim (morphološki oblik MORPH_ELLIPSE - elipsa upisana u pravougaonik `Rect(0, 0, esize.width, 0, esize.height)`, pošto izričito radi sa kružno oblikovanim kernelima) i dimenzija.[14][15]

`morphologyEx`-om obavljene su morfološke transformacije korišćene `erosion` i `dilation` kao osnovne operacije.[16] Bilo koja operacija može biti obavljena u mestu. U slučaju više kanalnih slika, svaki kanal je procesuiran posebno nezavisno. Parametri koji su uzeti u obzir su:

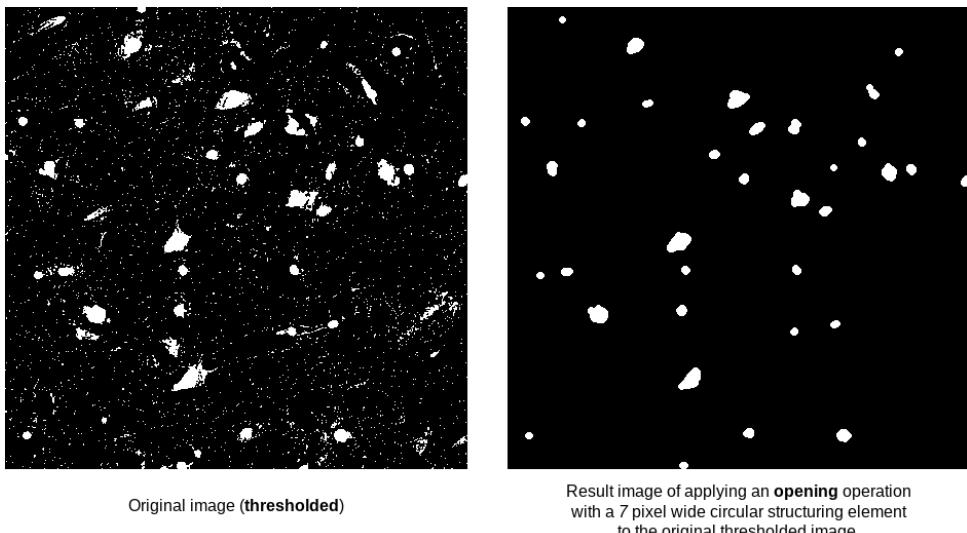
1. src - ulazna slika, broj kanala je arbitraran. Dubina može biti jedan od tipova, `unsigned` tipova kojima se ističe koliko memorije će zauzeti piksel, `cv` modula;
2. morfološka operacija tip, postavljena na `MORPH_OPEN`, pa npr. po njoj će se operacije obavljati redosledom: `erode` → `erode` → `dilate` → `dilate`, ako ima 2 iteracije. Korisno je za uklanjanje šumova, kao na slici 6;
3. kernel - preuzet od povratne vrednosti `getStructuringElement`-a;
4. anchor - usidrena pozicija sa kernelom, negativnom vrednošću se naznačava da usidrenje je kernelov centar, implicitna vrednost koja se daje je $(-1, -1)$;
5. iterations - broj iteracija ponavljanja primene erosion i dilation operacija, implicitno je vrednost 1;
6. borderType - vrednost granice u slučaju konstantne granice. Implicitno je `BORDER_CONSTANT`, što naznačava kako se vrši zagrančavanje po šablonu `iiiiii|abcdefgh|iiiiii`, gde naznačavaju se granice `|`, i je korisnički definisano. `BORDER_WRAP` nije podržan;
7. borderValue - gore pomenuto `i` - implicitno se radi na specijalan "magičan" način.

```
1 kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5,5))
2 morphologyImg = cv2.morphologyEx(threshedImg, cv2.MORPH_OPEN, kernel)
```

Implementacija 29: morfološke operacije

Na slici 7. može se videti primer ishoda morfološke operacije.

U implementaciji 30. koristi se `findContours`, gde prihvata prethodno obrađenu sliku, drugi argument režim prihvatanja kontura `RETR_TREE` (kojom se računa puna hijerarhija kontura), a uz `CHAIN_APPROX_SIMPLE` izvlače tačke po režimu metoda aproksimiranja samo one krajnje ako je u pitanju prava linija, inače kao i uvek sve tačke se uzimaju u obzir. `cvtColor` vrši konverziju

Slika 7: Morfološka operacija *opening*

Slika 8: Morfološke operacije

GRAY2RGB iz grayscale-a u RGB režim boja. drawContours kojoj su prosleđene slika, konture, vrednost -1 koja označava da se iscrtavaju sve konture, torka elemenata domena [0, 255] RGB režima boja za svaki kanal (u ovom slučaju plava) i vrednost 3 za debljinu iscrtane konture. imwrite-om dostavlja se fajl slike _result.jpg na odgovarajuću putanju. Kasnije se ispisuje broj kontura u tekstualni fajl result.txt, kojim se obogrljeno rezerviše tok fajla i razrešava tok resursa.

```

1 # Find and draw contours
2 contours, hierarchy = cv2.findContours(morphImg, cv2.RETR_TREE, cv2.
3                                         CHAIN_APPROX_SIMPLE)
4 contoursImg = cv2.cvtColor(morphImg, cv2.COLOR_GRAY2RGB)
5 cv2.drawContours(contoursImg, contours, -1, (0,0,255), 3)
6 cv2.imwrite("_result.jpg", contoursImg)
7 textFile = open("results.txt","a")
8 textFile.write("_result.jpg" + " Dots number: {}".format(len(contours)) + "\n")
9 print("Dots number: {}".format(len(contours)))
10 textFile.close()

```

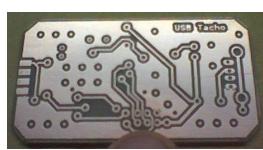
Implementacija 30: Nalaze se konture, konverzija boja, crtanje kontura, dostavljanje izlaznog fajla slike, ispis broja kontura

3 Rezultati koda

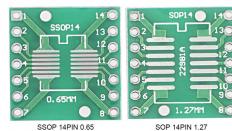
Biće izloženo obradi 4 slike 9., 10., 11., 12. štampanih ploča i slika kondenzatora 13., pa potom će se obaviti iscrtavanje kontura i prebrojavanje istih.



Slika 9: 1.
originalna
slika
štampane
ploče



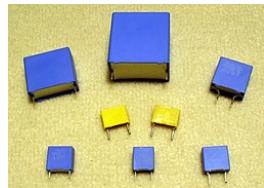
Slika 10: 2. origi-
nalna
slika
štampane
ploče



Slika 11: 3. origi-
nalna
slika
štampane
ploče



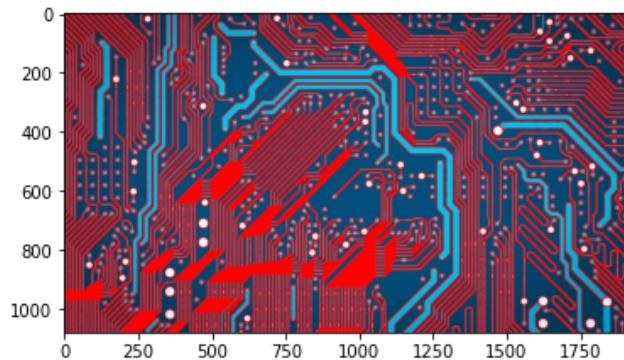
Slika 12: 4. origi-
nalna
slika
štampane
ploče



Slika 13: Slika
konde-
zatora

Prvi program

Cilj je obuhvatiti konturama vodove i konektore. Slika 14. prikazuje crvene konture koje su nađene za 1. originalnu sliku štampane ploče. A na izlazu 1. se ističe da je nađeno 526 kontura.

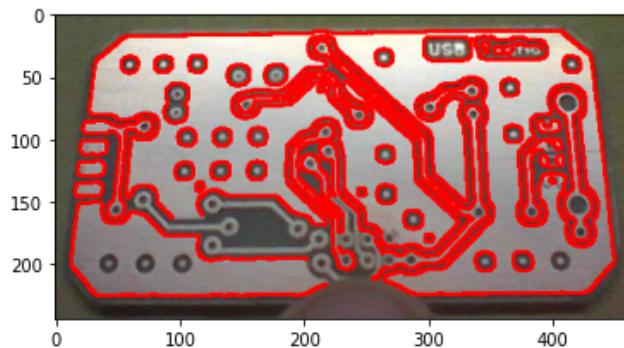


Slika 14: 1. program sa 1. slikom

```
1 coins in the image : 526
```

Izlaz 1: 1. program sa 1. slikom

Slika 15. prikazuje crvene konture koje su nađene za 2. originalnu sliku štampane ploče. A na izlazu 2. se ističe da je nađeno 48 kontura.



Slika 15: 1. program sa 2. slikom

```
1 coins in the image : 48
```

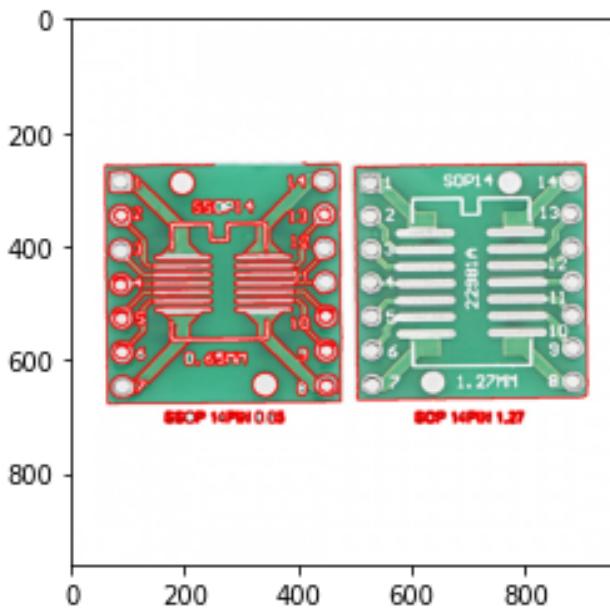
Izlaz 2: 1. program sa 2. slikom

Slika 16. prikazuje crvene konture koje su nađene za 3. originalnu sliku štampane ploče. A na izlazu 3. se ističe da je nađeno 111 kontura.

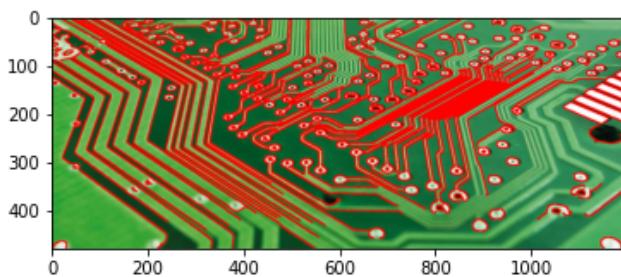
```
1 coins in the image : 111
```

Izlaz 3: 1. program sa 3. slikom

Slika 17. prikazuje crvene konture koje su nađene za 4. originalnu sliku štampane ploče. A na izlazu 4. se ističe da je nađeno 314 kontura.



Slika 16: 1. program sa 3. slikom



Slika 17: 1. program sa 4. slikom

```
1 coins in the image : 314
```

Izlaz 4: 1. program sa 4. slikom

Drugi program

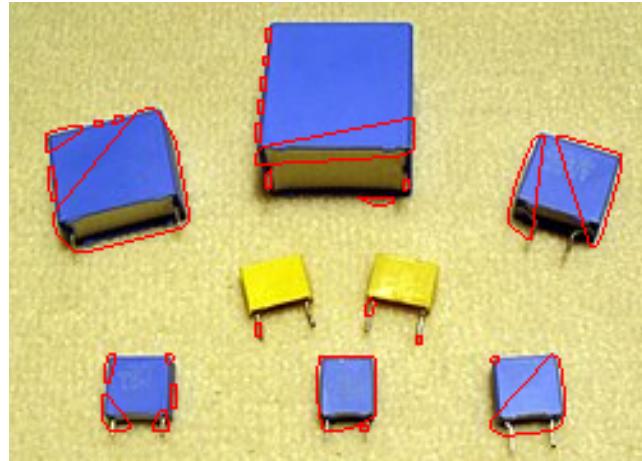
Cilj je obuhvatiti konturama kondenzatore po bojama i prebrojati za svaku boju. Slika 18. prikazuje crvenom nagoveštava plava, a zelenom žuta. Na izlazu 5. ističe se da je obuhvaćeno 28 plavih kondenzatora, 13 žutih kondenzatora.

```
1 28 blue capacitors
2 13 yellow capacitors
```

Izlaz 5: 2. program sa slikom plavih i žutih kondenzatora

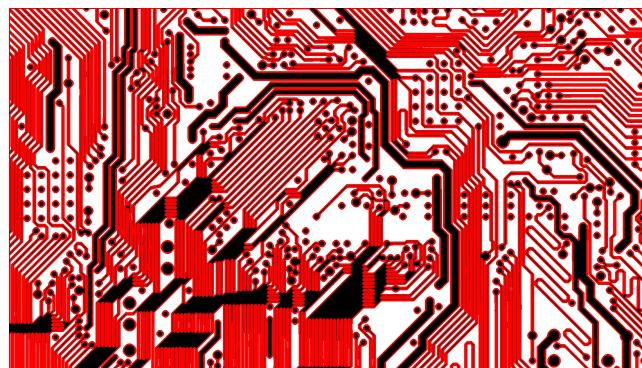
Treći program

Cilj je isti kao i u prvo programu samo pristup je drugačiji, može se primetiti da pozadina slike binarne boje. Slika 19. prikazuje crvene konture koje su nađene za 1. originalnu sliku



Slika 18: 2. program sa slikom plavih i žutih kondenzatora

štampane ploče. A na izlazu 6. se ističe da je nađeno 441 kontura.

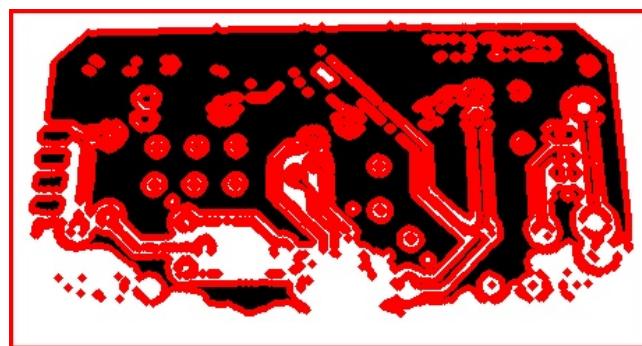


Slika 19: 3. program sa 1. slikom

```
1 Dots number: 441
```

Izlaz 6: 3. program sa 1. slikom

Slika 20. prikazuje crvene konture koje su nađene za 1. originalnu sliku štampane ploče. A na izlazu 6. se ističe da je nađeno 441 kontura.

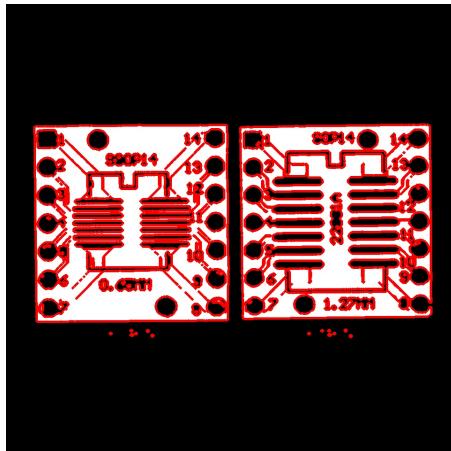


Slika 20: 3. program sa 2. slikom

`1 Dots number: 227`

Izlaz 7: 3. program sa 2. slikom

Slika 21. prikazuje crvene konture koje su nađene za 2. originalnu sliku štampane ploče. A na izlazu 7. se ističe da je nađeno 227 kontura.

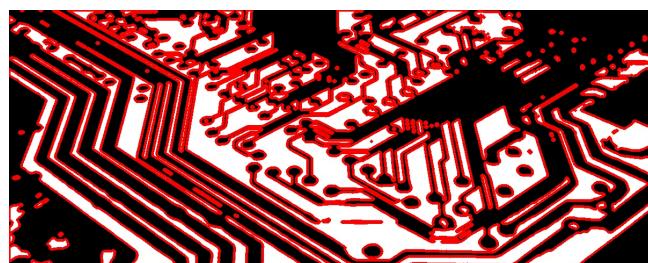


Slika 21: 3. program sa 3. slikom

`1 Dots number: 861`

Izlaz 8: 3. program sa 3. slikom

Slika 23. prikazuje crvene konture koje su nađene za 4. originalnu sliku štampane ploče. A na izlazu 9. se ističe da je nađeno 201 kontura.



Slika 22: 3. program sa 4. slikom

`1 Dots number: 201`

Izlaz 9: 3. program sa 4. slikom

4 Zaključak

Može se uočiti da se u sva tri programa koristile iste funkcije, ali poneki parametar za svaki program je bio zasebno služio kao jedinstven primer.

U prvom programu se davao značaj na posivljivanju, zamaglivanju, detektovanju ivica, pozivanju i podebljavanju ivica, nalaženju kontura naspram ivica svih tačaka, krajnjoj konverziji u RGB, pa iscrtavanju kontura.

Drugi program dao je značaj na detekciji boja uz definisanju gornjih i donjih ograničenja, radu sa maskama naspram tih granica, pa tek onda posivljivanju, **dilate** funkciji, a sada eroziji, nalaženju kontura, ispitivanju dovoljnosti kontura, iscrtavanju i prebrojavanju kontura po bojama.

U trećem zadatku se davao značaj na definisanju praga, korisnički definisanim maksimalnim vrednostima definisanje ili anuliranje piksela, otsu režima ustanavljanja pragova i detekcije ivica, rad sa **dilate** i **erosion** elipsoidom i njihovim redosledom izvšavanja zarad kranjeg rašumljavanja, pa se nalazile konture, iscrtavale i prebrojavale.

Rezultati za sva tri programa su bili približni i naziralo se da su bili delotvorni, ali ne i previše pouzdani.

Po prvom programu deluje da je obuhvaćeno obradom većina poželjnih entiteta na koje se ciljalo, ali ima izostanaka. Po slici i po brojnosti, ali je to opet upitno.

Po drugom programu se nazire neka doslednost da ide u pravcu nekog razaznavanja entiteta, ali to nije savršeno i moguća nepogodnost je ta što nije dovoljno velika rezolucija, a i pozadina zavarava prebrojavanje žutih kondenzatora. Brojnost je više od pet puta veća, ali i verovatno da konture nisu dosledne tačnosti.

Po trećem programu vizuelna uočljivost prepoznatih entiteta kontura na slici je većinski poklopljena sa ciljanim stanjem, ali brojnost je mnogostruko veća i nepouzdana.

Literatura

- [1] Count number of Object using Python-OpenCV,
<https://www.geeksforgeeks.org/count-number-of-object-using-python-opencv/>,
Datum posete: 24. jun 2024.
- [2] Canny Edge Detection,
https://docs.opencv.org/4.x/d22/tutorial_py_canny.html,
Datum posete: 24. jun 2024.
- [3] Erosion and Dilation of images using OpenCV in python,
<https://www.geeksforgeeks.org/erosion-dilation-images-using-opencv-python/>
Datum posete: 24. jun 2024.
- [4] Find and Draw Contours using OpenCV — Python,
<https://www.geeksforgeeks.org/find-and-draw-contours-using-opencv-python/>,
Datum posete: 24. jun 2024.
- [5] difference between CV_RETR_LIST,CV_RETR_TREE,CV_RETR_EXTERNAL?,
<https://stackoverflow.com/questions/8830619/difference-between-cv-retr-list-cv-retr-tree-cv-retr-external>,
Datum posete: 24. jun 2024.
- [6] Drawing Functions Image Processing,
https://docs.opencv.org/4.x/d6/d6e/group__imgproc__draw.html#ga746c0625f1781f1ffc9056259103edbc
Datum posete: 24. jun 2024.
- [7] Counting blue and white bacteria colonies with Python and OpenCV,
<http://www.sixthresearcher.com/counting-blue-and-white-bacteria-colonies-with-python-and-opencv/>,
Datum posete: 24. jun 2024.
- [8] Operations on arrays, Core functionality, inRange,
https://docs.opencv.org/3.4/d2/de8/group__core__array.html#ga48af0ab51e36436c5d04340e036ce981,
Datum posete: 24. jun 2024.
- [9] Operations on arrays, Core functionality, bitwise_and,
https://docs.opencv.org/3.4/d2/de8/group__core__array.html#ga60b4d04b251ba5eb1392c34425497e14,
Datum posete: 24. jun 2024.

[10] Count cells on image using python and OpenCV,

<https://stackoverflow.com/questions/60957257/count-cells-on-image-using-python-and-opencv>,

Datum posete: 24. jun 2024.

[11] fastNLMeansDenoising,

https://docs.opencv.org/3.4/d1/d79/group__photo__denoise.html,

Datum posete: 24. jun 2024.

[12] A. Buades, B. Coll, J.M. Morel, Non-Local Means Denoising (2011), IPOL Journal · Image Processing On Line,

http://www.ipol.im/pub/art/2011/bcm_nlm/,

Datum posete: 24. jun 2024.

[13] Image Thresholding,

https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

Datum posete: 24. jun 2024.

[14] Morphological Transformations,

https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html,

Datum posete: 24. jun 2024.

[15] MorphShapes,

https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html#gac2db39b56866583a95a5680313c314ad

Datum posete: 24. jun 2024.

[16] morphologyEx,

https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html#ga67493776e3ad1a3df63883829375201f,

Datum posete: 24. jun 2024.