Aplikacija za obradu slika korištenjem OpenCV funkcija

Barbarić, Marin

Sveučilište u Rijeci – Odjel za informatiku

Jednopredmetna informatika

Marin Barbarić

Aplikacija za obradu slika korištenjem OpenCV funkcija

Završni rad

Mentor: izv. prof. dr. sc. Marina Ivašić-Kos

Rijeka, 11.8.2020.

Sadržaj

Zadatak završnog rada

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 17. veljače 2020.

Zadatak za završni rad

Pristupnik: **Marin Barbarić**

Naziv završnog rada: **Aplikacija za obradu slika korištenjem OpenCV funkcija**

Naziv završnog rada na eng. jeziku: **Application for image processing using OpenCV functions**

Sadržaj zadatka: Objasniti što je računalni vid, njegovo područje primjene i interesa s naglaskom na funkcije za obradu slike. Proučiti OpenCV biblioteku i funkcije koje se koriste za rad sa slikama kao što je učitavanje slika, konverziju RGB slike u sliku sive razine ili crno bijelu sliku, detekcija rubova i kutova, segmentaciju i slično te opisati njihovo korištenje na različitim primjerima slika.

Testirati složenije aplikacije računalnog vida koje se temelje na vizualnim značajkama slike kao što je povezivanje više slijednih slika u panoramu, izdvajanje pozadine iz slike, usporedba slika i traženje istog uzorka ili detalja na različitim slikama. Dokumentirati eksperimente i objasniti rezultate koji se postižu na različitim primjerima slika.

Mentor

Izv. prof. dr. sc. Marina Ivašić-Kos

Voditelj za završne radove

Doc. dr. sc. Miran Pobar

Zadatak preuzet: 17.2.2020

M. Barbarić

(potpis pristupnika)

Sažetak i ključne riječi

OpenCV je biblioteka funkcija koje se primjenjuju u području računalnog vida. Funkcije za obradu slike koriste se za pronalaženje rubova, thresholding-a, detekciju značajki i slično. Neke jednostavne aplikacije koje se mogu izraditi pomoću OpenCV su aplikacija za izradu panorama ili odvajanje pozadine iz slika. U ovom radu korištene su funkcije za učitavanje slika, pronalaženje rubova i kutova, segmentaciju slike, thresholding. Na primjerima je prikazano kako svaka od tih funkcija radi. Neki od složenijih primjera uključuju detekciju značajki, prepoznavanje značajki jedne slike na drugoj, template matching te izradu panorame.

Ključne riječi: OpenCV, računalni vid, slika, threshold, rub, feature, detector, funkcija, matplotlib, piksel, gradijent, segmentacija, greyscale, matcher, algoritam, program, aplikacija.

Uvod

Ovaj završni rad bavit će se računalnim vidom odnosno pokazat će na koje sve načine se slike mogu obrađivat pomoću funkcija iz biblioteke OpenCV. Na konkretnim primjerima će biti pokazano kako pojedine funkcije rade te će se rezultat obrade različitih slika usporedit. Također će se opisati i objasniti što su zadaci računalnog vida, njegove primjene i važnost te primjere korištenja funkcije OpenCV i aplikacija koje koriste OpenCV funkcije. Rad se sastoji od uvodnog teorijskog dijela u kojem je opisano što je točno računalni vid i koje su njegove primjene, što je image processing te na koje načine se slika može obraditi. Dodatno je objašnjeno što je OpenCV i koje funkcije će se koristiti. Nakon teorijskog dijela slijedi prikaz odabranih funkcija, usporedba rezultata istih funkcija na više primjena te na kraju par složenijih primjena odabranih funkcija.

Računalni vid i obrada slike

Računalni vid je područje umjetne inteligencije koje se bavi računalnom obradom i analizom slika i video zapisa s ciljem da se zadaci koje ljudski vidni sustav radi razumiju i automatiziraju (Ballard & Brown, 1982). Računalni vid uključuje prikupljanje, obradu i analizu digitalnih slika. Image processing je proces u kojem se slika pomoću računala obrađuje koristeći algoritam (Gonzalez, 2018). Analiza slike je izvlačenje korisnih informacije iz slika koristeći se tehnikama obrade slike (eng. image processing-a) (Solomon & Breckon, 2010). Primijene računalnoga vida mogu biti jednostavne poput programa koji prepoznaje rubove na predmetima na nekoj slici ili mogu biti kompleksni poput sustava za autopilot koji prati prometne trake, prepoznaje znakove te stalno prati susjedne aute u prometu. Treba napomenut da primjer programa koji prepoznaje rubove predmeta zapravo sam po sebi nije jednostavan. Ono što ga čini jednostavnim je to što postoje već gotove funkcije koje nam to omogućavaju dok je ta funkcija sama po sebi vrlo kompleksna te se koriste razne matematičke i računalne metode da bi se dobio željeni rezultat. Kod računalnog vida svaka fotografija ili svaki video okvir se treba obradit. Kod obrade, ovisno o onome što želimo, primjenjuju se razni modeli iz geometrije, fizike, statistike itd. Neke od tehnika koje se koriste su prepoznavanje objekata na slici su segmentacija slike, detekcija pomaka… Kod prepoznavanja objekata oni mogu biti dvodimenzionalni ili trodimenzionalni te ovisno o tome postoje razne metode za prepoznavanje. Prepoznavanje obruba je jedna od tih metoda. Ona se bazira na matematičkim metodama koje uočavaju nagle promjene u oštrini, svjetlosti slike, karakteristikama materijala itd. Obrada slike se često bazira na pronalaženju značajki neke slike. Značajke(eng. feature) su zanimljivi dijelovi slike poput rubova, vrhova, uzoraka ili objekata. U području računalnoga vida puno algoritama počinje upravo obradom slike odnosno pronalaženjem značajki. Ovisno koje značajke se traže koriste se razni detektori. Za pronalaženje rubova poznati su Canny, Sobel, Prewitt detektori. Neki od detektora za vrhove su Harris, Hessina, Susan. Za pronalaženje regija i blobova koriste se LoG(Laplacian of Gaussian) i DoG(Difference of Gaussian) metode (Kaehler & Bradski, 2008). Primjenom nekog od ovih detektora računalo odredi točke interesa na slici. Da bi računalo moglo odrediti preklapaju li se neke od točaka interesa potrebno ih je opisati. Za to se koriste deskriptori. Neki od poznatih deskriptora su SIFT, SURF, FAST, ORB i ostali (Karami, Prasad, & Shetata, 2015). Područja obrade slika i računalnog vida se preklapaju u velikoj mjeri. Računalni vid koristi obradu slike te onda pomoću informacija koje se mogu izvući iz slike obavlja druge zadatke poput rekonstrukcije scena, navigacije, modeliranja 3d objekata i slično. Česte primjene računalnog vida su u real-time aplikacijama koje dobivaju slike ili video zapis pomoću nekog senzora te onda aplikacija u stvarnom vremenu obrađuje te vraća rezultat. Primjer toga bi bio sustav autopilota u automobilima.

Primjena OpenCV funkcija

OpenCV je biblioteka gotovih funkcija za različite zadatke računalnog vida koje su spremne za korištenje. Prvi put dostupan 2000. godine, OpenCV omogućuje razvoj aplikacija koristeći gotove funkcije. Te funkcije su dostupne u tri programska jezika koji su C++, Java i Python. U ovom radu zbog jednostavnosti i dostupnosti materijala koristit će se Python kao odabrani jezik za korištenje OpenCV-a. Osim metoda za obradu slike OpenCV sadrži biblioteku za strojno učenje. Sve funkcije se dostupne za Windows, Linux i macOS operacijske sustave te podržava razvoj za mobilne uređaje sa Android te iOS operacijskim sustavima.

Da bi se mogle koristiti funkcije iz OpenCV biblioteke potrebno ju je instalirati na računalo. S obzirom da je OpenCV ima BSD licencu nije ga potrebno kupiti već je dovoljno posjetiti njegovu web stranicu te tamo ga preuzeti. Ovisno o načinu na koji ga želimo koristiti postoje razne metode instalacije. U ovom završnom radu skripte će se pokretati kroz terminal na macOS operacijskom sustavu.

Učitavanje slike

Prva stvar koju je potrebno znati da bi mogli obraditi sliku je učitati ju u program. Postoji više načina za to. Za početak potrebno je importirati u program OpenCV biblioteku te numpy biblioteku. Naredni program izgleda ovako (Slika 1):

```
1  import numpy as np
2  import cv2
3
4  img = cv2.imread('koi.jpg',0)
5  cv2.imshow('image',img)
6
7  cv2.waitKey(0)
8  cv2.destroyAllWindows()
9
```

Slika 1. Primjer učitavanja slike

Kao što je napomenuto prvi korak je uključiti potrebne biblioteke (linija 1 i 2). Nakon toga poziva se funkcija *imread()* koja se nalazi u cv2 biblioteci te se njen rezultat sprema u varijablu *img*. Parametri te funkcije su putanja do slike koju želimo učitati te integer broj s kojim se označava da li će slika biti učitana u boji ili crno bijela. U ovom primjeru učitana slika će biti crno bijela. Sada je slika učitana u program. Da bi se slika prikazala u novom prozoru koristi se funkcija *imshow()* također iz cv2 biblioteke. U parametre se postavlja ime prozora u ovom slučaju *image* te ime varijable gdje je slika učitana. Za kraj koristi se funkcija *waitkey()* koja osigurava da prozor sa slikom bude vidljiv dok se ne pritisne neko dugme s tipkovnice te funkciju *destroyAllWindows()* koja zatvara sve prozore programa.

Kod pokretanja programa prozor koji će se otvorit će biti prevelik s obzirom da je slika velika. Kako bi se mogla kontrolirati veličina prozora moguće je koristiti funkciju *namedWindow()* koja stvara prozor u koji će se kasnije učitati slika. Na slici 2 je prikazana primjena namedWindow() funkcije.

```
5  cv2.namedWindow('image', cv2.WINDOW_NORMAL)
6  cv2.imshow('image',img)
```

Slika 2. Funkcija namedWindow()

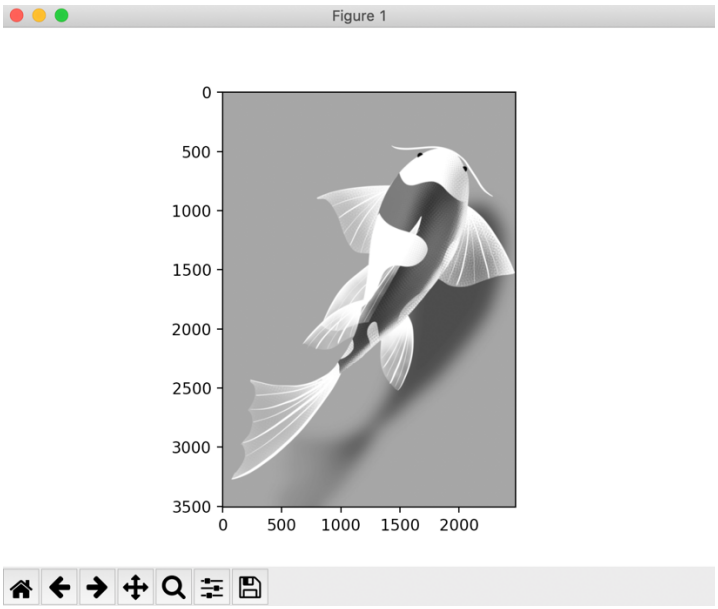
Drugi parametar ove funkcije omogućava izmjenu veličine prozora dok je prvi ime prozora.

Drugi način za učitat i prikazat sliku je koristeći Matplotlib biblioteku. Na taj način sliku je moguće spremiti, povećati itd. Tada program izgleda ovako (Slika 3):

```
1  import numpy as np
2  import cv2
3  from matplotlib import pyplot as plt
4
5  img = cv2.imread('koi.jpg',0)
6  plt.imshow(img, cmap = 'gray', interpolation = 'bicubic')
7  plt.show()
8
```

Slika 3. Kod za učitavanje slike koristeći matplotlib

Te onda dobiveni rezultat je ovakav: (Slika 4):



Slika 4. Izgled prozora koristeći matplotlib

Ovaj način učitavanja slika omogućava spremanje slike. Ako se ne koristi ta metoda već ona prije pokazana potrebno je dodati još par linija koda. Slika 5 opisuje spremanje slike pomoću imwrite() funkcije.

```
8  key = cv2.waitKey(0)
9  if key == 27:
10     cv2.destroyAllWindows()
11 elif key == ord('s'):
12     cv2.imwrite('koi2.png',img)
13     cv2.destroyAllWindows()
14
```

Slika 5. Spremanje slike

Ovaj put povratna vrijednost funkcije *waitKey()* sprema se u varijablu *key*. Nakon toga provjerava se ako je vrijednost varijable jednaka 27 odnosno Esc dugmetu te tada se prozor gasi, a ako je pritisnuto slovo **s** tada se slika sprema u radni direktorij s imenom koje je zadano kao parametar funkcije *imwrite()*. Nakon spremanja slike prozor se gasi.

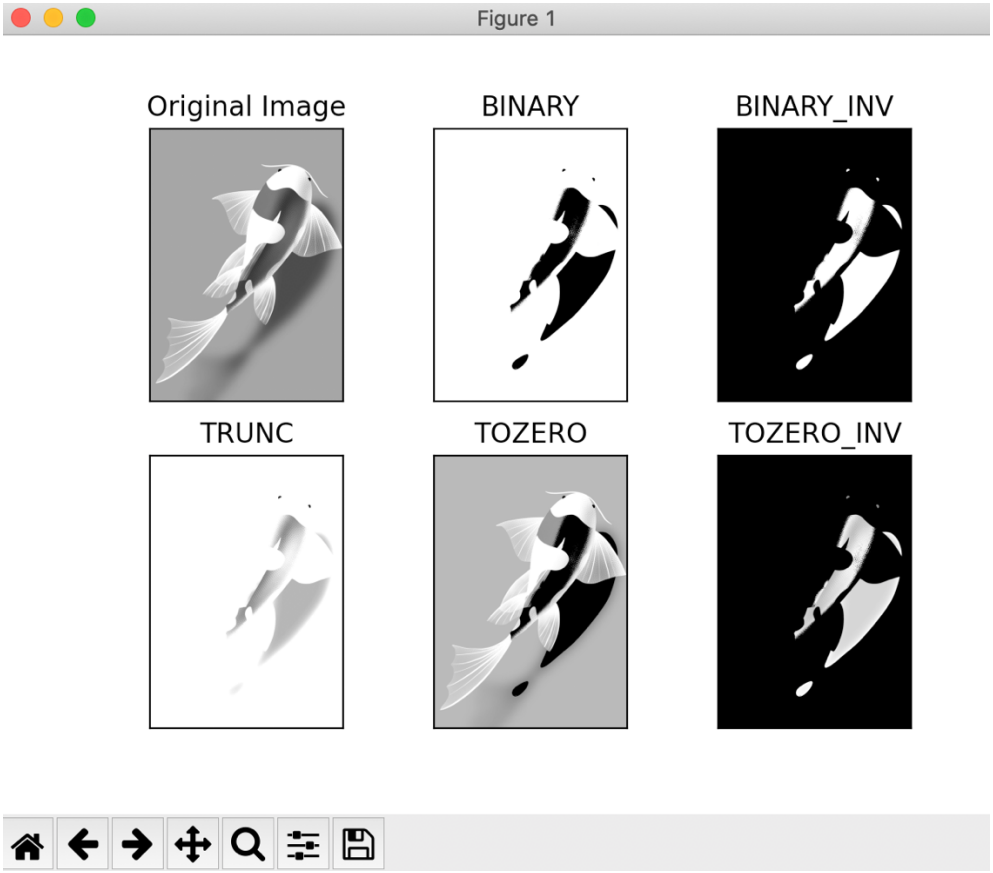
U nastavku će biti prikazani primjeri obrade slike počevši od jednostavnijih poput pretvorbe slike u gray-scale sliku, image thresholding-a, segmentacije i ostalo.

Image thresholding

Ovaj proces je jedan od najjednostavnijih procesa za segmentaciju slika. Segmentacija slika je postupak u kojem se slika dijeli na segmente tako da bi se mogla lakše analizirati i da bi se kroz jednostavniji prikaz slike lakše interpretiralo njeno značenje (Hapiro & Stockman, 2001). Uz threshold metodu ostale metode za segmentaciju slike su: edge based method, region based method, clustering based method, watershed based method, PDE based method, ANN based method (Hapiro & Stockman, 2001).

Image thresholding radi tako da se odabere neka vrijednost koja predstavlja intenzitet slike te onda svi pikseli koji imaju intenzitet manji od te vrijednosti pretvaraju u crne piksele dok oni čija je vrijednost veća ili jednaka postanu bijeli pikseli. Ova metoda funkcionira najbolje za greyscale slike no moguće ju je primijeniti i na slike u boji. Tada se za svaku RGB komponentu primjenjuje ova metode te se na kraju rezultati zbroje u jednu sliku.

U prvom primjeru koristit će se funkcija čiju vrijednost intenziteta piksela odnosno threshold sami određujemo. Slika 6 uspoređuje 5 različitih vrsta image thresholding-a.



Slika 6. Thresholding na primjeru

Na slici 6. prikazanu su rezultati 5 različitih vrsta thresholding-a koje nudi OpenCV na odabranu sliku. Binary je upravo onaj način koji je opisan iznad slike odnosno slika postaje binarna slika sa samo crnim i bijelim pikselima. Binary invert je obrnuto od binary: što je bilo bijelo je sada crno a što crno je sada bijelo. Treći način radi tako da provjerava vrijednosti

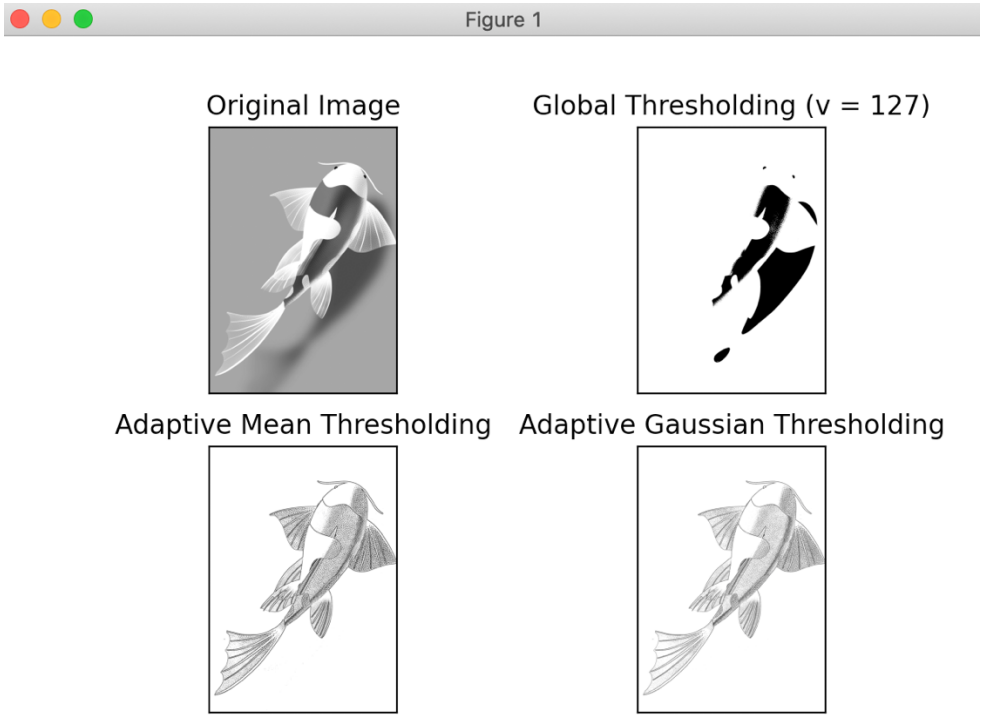
intenziteta piksela te ako je vrijednost veća od postavljene vrijednosti tada se vrijednost ujednačava sa postavljenom vrijednosti. To zero metoda postavlja sve piksele na nulu ako im je vrijednost ispod postavljene vrijednosti. Oni pikseli čija je vrijednost veća od postavljene ostaju isti. Suprotno tome, To zero inverted postavlja piksele čija je vrijednost veća od postavljene na nulu a ostali ostaju isti.

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

img = cv2.imread('koi2.png',0)
ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)
ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)
ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO)
ret,thresh5 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO_INV)

titles = ['Original Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]

for i in range(6):
    plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])

plt.show()

```

Slika 7. Thresholding kod

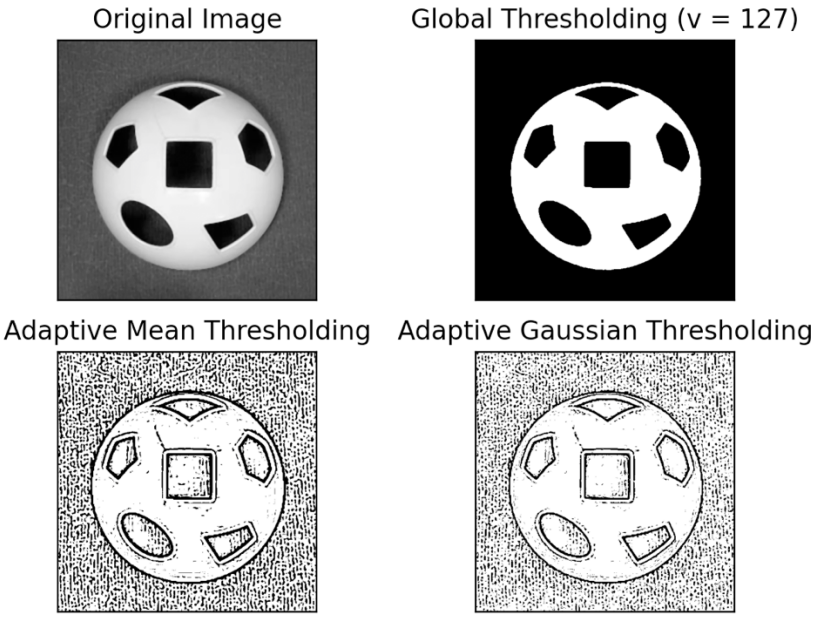
Na slici 7. prikazan je programski kod za thresholding slike. Za svaku metodu koristi se funkcija *cv2.thresholding()* čiji su parametri redom: varijabla u kojoj je slika spremljena, threshold odnosno postavljena vrijednost intenziteta piksela, maksimalna vrijednost na koju se pikseli postavljaju te na kraju odabrana metoda za obradu.

Iz primjera na slici 8. se vidi da postavljanje globalne vrijednosti thresholda nije najbolje rješenje. Ako slika ima različitu svjetlinu na nekim dijelovima ovaj postupak neće dati optimalan rezultat. Bolje rješenje je Adaptive Thresholding. Ova metoda radi tako da se slika dijeli na segmente i onda se thresholding algoritam primjenjuje na svaki dio slike da bi postigli bolje rezultate. Postoje dvije funkcije koje se razlikuju po načinu na koji se računa vrijednost threshold-a za svaki dio slike. Prva funkcija za vrijednost bira aritmetičku sredinu susjednih vrijednosti u tom dijelu slike, a druga radi težinsku sumu susjednih vrijednosti gdje su težine Gauss-ov prozor. Na slici 8 uspoređene su dvije adaptivne metode thresholding-a.



Slika 8. Primjer adaptive thresholding

Moguće je uočiti da adaptivni thresholding bolje radi na odabranoj slici nego globalni thresholding. Rubovi cijelog objekta su jasno detektirani te se sada slika može lakše dalje analizirati i obraditi. Međutim, to ne mora vrijediti za sve slike. Na slijedećem primjeru (Slika 9) još se jasnije vidi razlika između globalnog i adaptivnog thresholding-a te se može zaključiti da postoje slučajevi s jasno određenim granicama među svjetlijim i tamnijim područjima kod kojih se postižu boji rezultati s globalnim thresholdingom.



Slika 9. Usporedba thresholding metoda na drugom primjeru

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

img = cv2.imread('koi2.png',0)
img = cv2.medianBlur(img,5)

ret,th1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,\
            cv2.THRESH_BINARY,11,2)
th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,\
            cv2.THRESH_BINARY,11,2)

titles = ['Original Image', 'Global Thresholding (v = 127)',
            'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding']
images = [img, th1, th2, th3]

for i in range(4):
    plt.subplot(2,2,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()

```

Slika 10. Adaptive thresholding kod

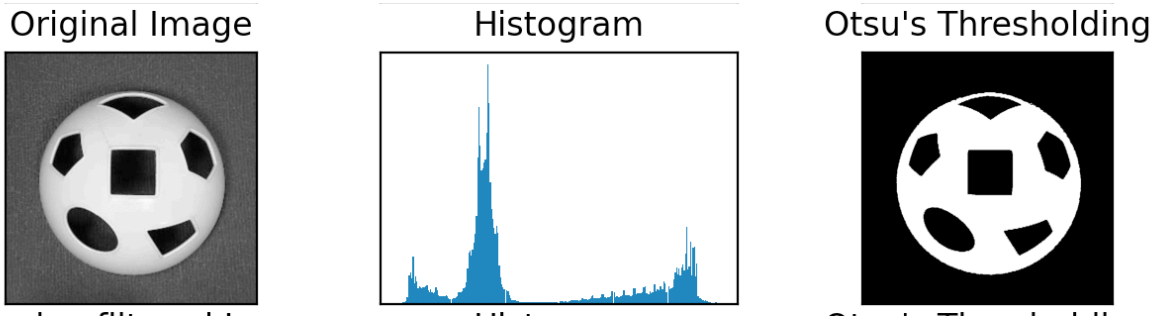
Program za adaptivni thresholding izgleda vrlo jednostavno (Slika 10). Na učitanu sliku **img** se primjeni funkcija s parametrom ADAPTIVE_THRESH_MEAN_C ili ADAPTIVE_THRESH_GASUSSIAN_C ovisno o odabranoj metodi, u ovom slučaju primijenjene su obje adaptivne funkcije te jednu funkciju s globalnom vrijednošću threshold-a. Koristi se matplotlib biblioteka za prikaz rezultata.

Postoji jedan poseban slučaj kada je odabrana slika bimodalna. Histogram te slike ima dva istaknuta vrha. Tada je moguće koristiti Otsu-ovu binarizaciju. Ova metoda za threshold bira srednju vrijednost između ta dva vrha u histogramu slike. Slika 11 prikazuje funkciju za Otsu-ovu binarizaciju.

```
ret,th = cv2.threshold(img,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
```

Slika 11. Otsu binarization funkcija

Funkcija je ista kao za binarni thresholding samo što za globalnu vrijednost threshold-a postavlja se 0 te se dodaje *+cv2.THRESH_OTSU* na kraj zadnjeg parametra. Slika 12 je prikaz Otsu-ove binarizacije na primjeru sa slike 9.



Slika 12. Otsu binarization primjer

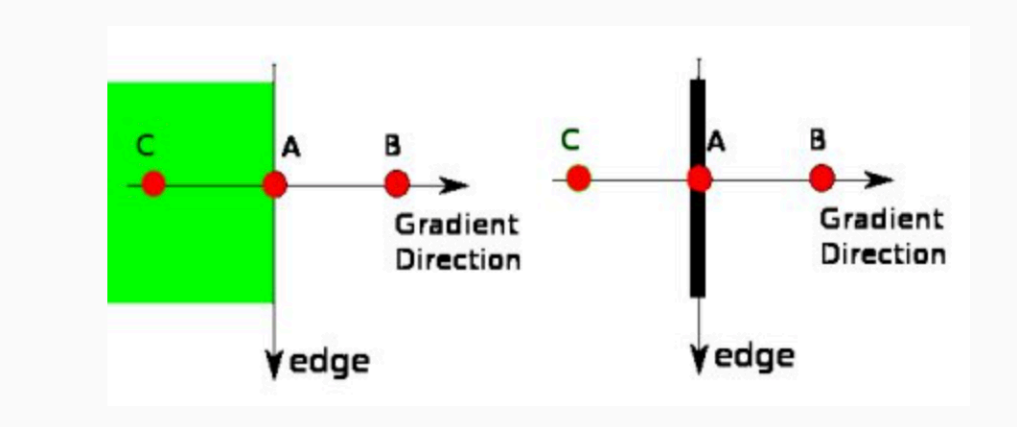
Još jedan primjer ove metode je prikazan na slici 13.

Slika 13. Otsu-ova binarizacija na primjeru sa slike 8

U nastavku biti će prikazane metode za detekciju rubova na originalnim i na već obrađenim slikama.

Detekcija rubova

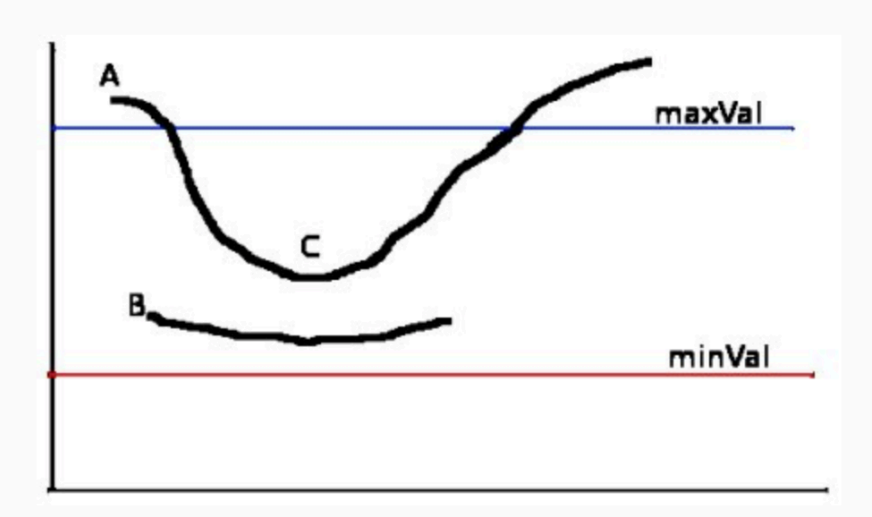
Detekcija rubova (engl. edge detection) je tehnika kojom se na slici pronalaze rubovi. Detektori su algoritmi koji služe upravo za to. Neki od poznatih detektora rubova su Canny, Sobel i Prewitt. Na prvom primjeru će biti prikazan Canny edge algoritam. Osmišljen 1986. godine od strane John-a F. Canny-a. Ovaj algoritam sastoji se od više faza. Prva faze je smanjivanje šuma(*eng. noise*) na slici. Šum na slici nastaje najčešće kao posljedica pogreške senzora, zbog sjene i nedostatka svjetla ili pogreške interpretacije rezultata senzora. Kako bi osigurali da algoritam ne detektira šum kao potencijalne rubove na slici potrebno je primjeni jedan od Noise Reduction filtera. Canny edge algoritam koristi Gaussian filter za to. Drugi korak je pronalaženje gradijenta intenziteta slike. Slika se provlači kroz Sobel-ov filter s kojim se detektiraju horizontalni, vertikalni i dijagonalni rubovi. Nakon što su svi smjerovi rubova zaokruženi na odgovarajuće kutove od 0, 45, 90 te 135 stupnjeva prelazi se na treću fazu. Treća faza zove se Non-maximum supression. To je jedna od metoda za stanjivanje rubova odnosno miču se neželjeni pikseli koji ne pripadaju rubovima. Cijela slika se skenira te se provjerava svaki piksel koji pripada nekom rubu te se uspoređuje sa njegovim susjedima koji se nalaze s njegove desne i lijeve strane odnosno iznad ili ispod njega u smjeru gradijenta. Ako je intenzitet piksela veći od ostala dva odnosno on je maksimum od njih tada se njegova vrijednost sprema. U suprotnom slučaju se njegova vrijednost postavlja na nulu. Taj proces postavljanja vrijednosti piksela na nulu zove se *supression.* Slika 14 prikazuje Non maximum supression. Točka A na slici predstavlja piksel na nekom rubu dok točke C i B su susjedni pikseli u smjeru gradijenta.



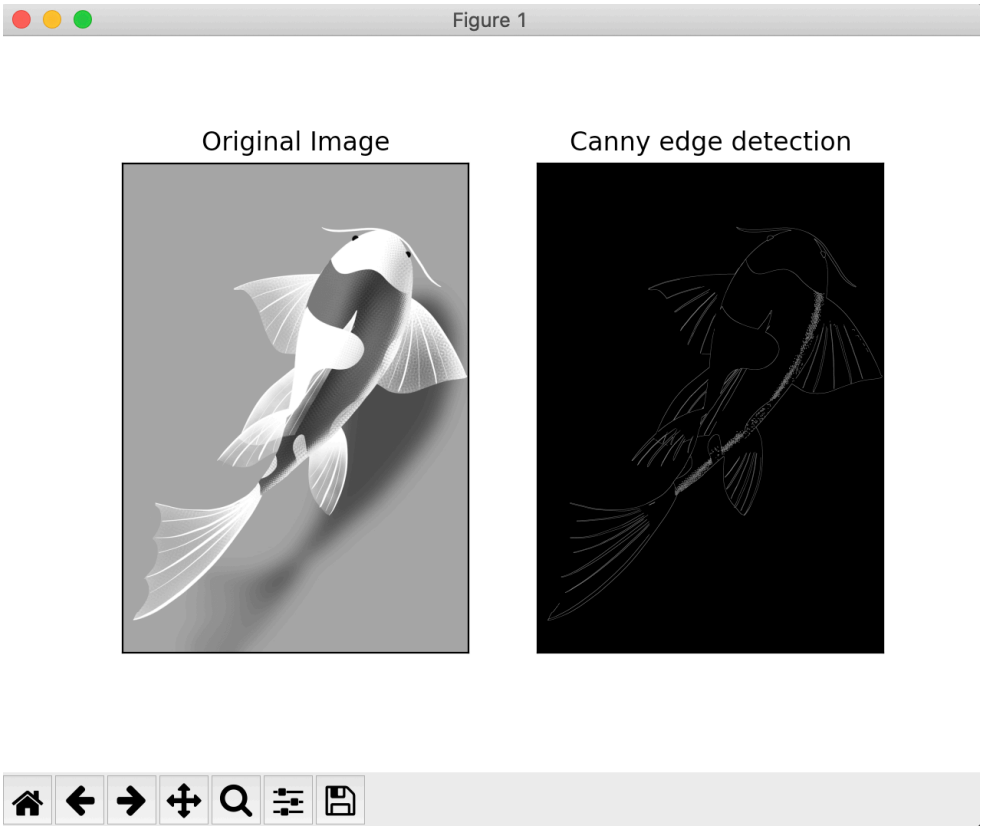
Slika 14. Non-maximum supression

U završnoj fazi odlučuje se koji su rubovi stvarno rubovi a koji to nisu, već su detektirani kao rezultat šuma ili varijance u boji. Biraju se dvije vrijednosti za provjeru: maksimalnu i minimalnu. Ako je vrijednost gradijenta intenziteta veća od maksimalne vrijednosti tada se taj rub klasificira kao siguran rub. Ako je vrijednost manja od minimalne tada to sigurno nije rub te se odbacuje. Ako se vrijednost nalazi između minimalne i maksimalne provjerava se da li je taj rub spojem s nekim sigurnim rubom, ako je spojen tada je on dio ruba te rub sam po sebi. U suprotnom slučaju, ako nije spojen sa sigurnim rubom ni sam nije rub te se odbacuje. Slika 15 opisuje završnu fazu algoritma. Rub A nalazi se iznad maksimalne vrijednosti te je sigurno rub, rub C se nalazi između dvije vrijednosti te je povezan sa sigurnim rubom A što znači da je i on

rub. Rub B se također nalazi između dvije vrijednosti no nije povezan sa sigurnim rubom stoga on se odbacuje.

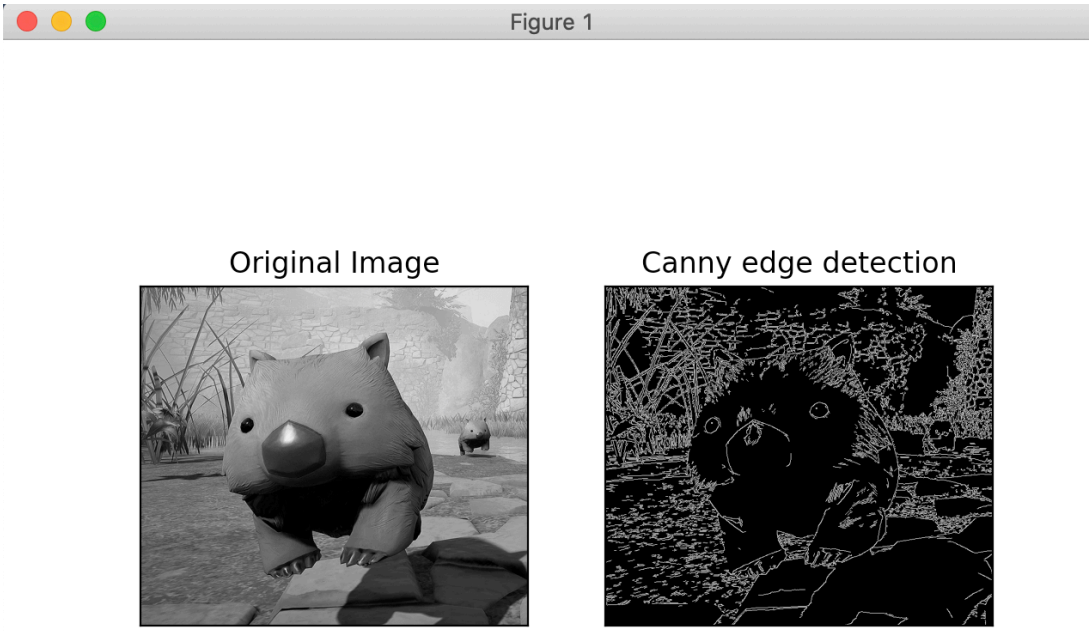


Slika 15. Zadnja faza



Slika 16. Canny edge detection primjer

Na slici 16 prikazan je rezultat algoritma. S obzirom da je slika iz primjera jednostavna rubovi su jasno određeni. Slika 17 prikazuje Canny edge algoritam na kompleksnijoj slici.



Slika 17. Canny edge detection primjer 2

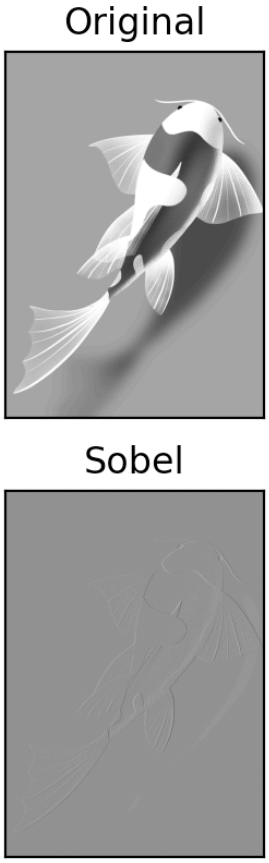
U ovom slučaju rubovi su i dalje jasno označeni no s obzirom da je slika sama po sebi kompleksnija na slici je detektirano mnogo rubova koji su dio teksture pozadine i nisu značajni za detekciju pojedinog objekta. Da bi dobili bolji rezultat moguće je odvojiti subjekt u fokusu od pozadine te onda primijeniti algoritam za detektiranje rubova.

```
1  import cv2
2  import numpy as np
3  from matplotlib import pyplot as plt
4
5  img = cv2.imread('koi2.png',0)
6  edges = cv2.Canny(img,100,200)
7
8  plt.subplot(121),plt.imshow(img,cmap = 'gray')
9  plt.title('Original Image'), plt.xticks([]), plt.yticks([])
10 plt.subplot(122),plt.imshow(edges,cmap = 'gray')
11 plt.title('Canny edge detection'), plt.xticks([]), plt.yticks([])
12
13 plt.show()
14
```

Slika 18. Canny edge detection kod

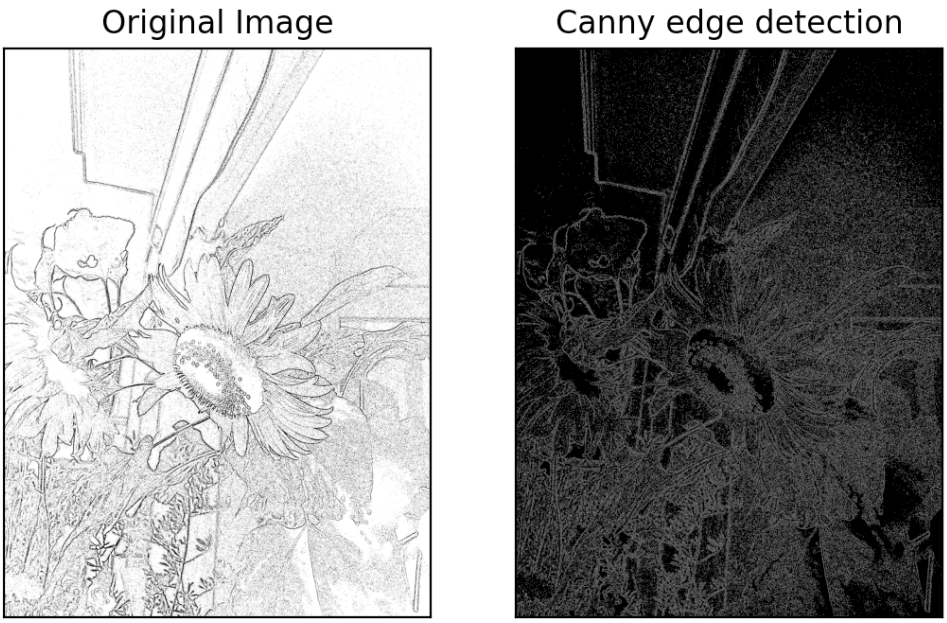
Slika 18 prikazuje programski kod za obradu slike iz primjera koristeći Canny edge algoritam. Funkcija koja se koristimo za obradit sliku je *cv2.Canny()* te matplotlib za prikaz rezultata. Prvi argument funkcije je upravo slika koja se obrađuje, a ostala dva su minimalna i maksimalna vrijednost o kojima se govorilo u prethodnom odlomku.

Drugi način za pronalaženje rubova je high-pass ili gradient filtere. Slika 19 prikazuje kako radi detekcija rubova koristeći Sobel-ovu metodu.



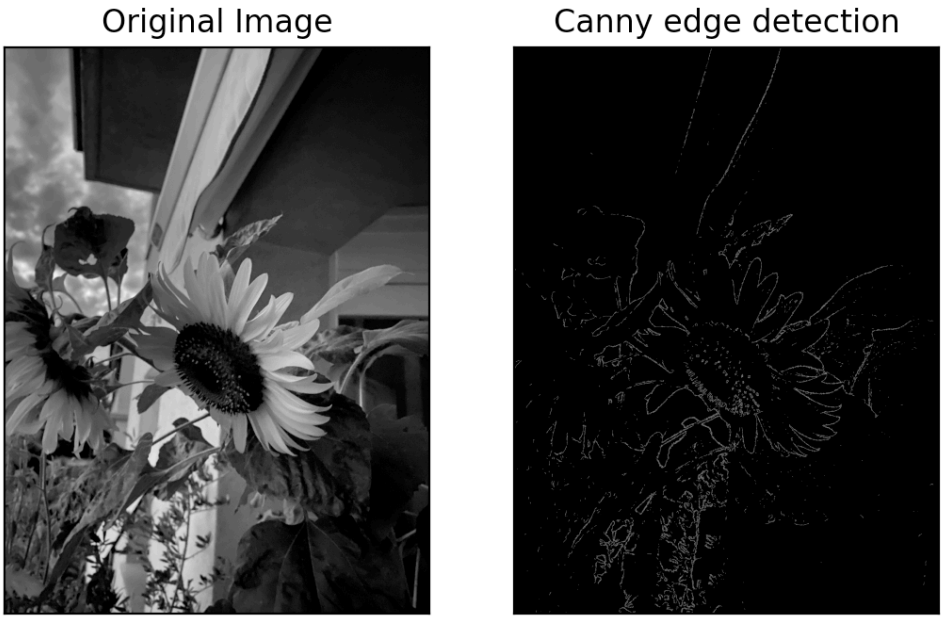
Slika 19 Sobel-ova metoda za detekciju rubova

Da bi se postigli najbolji rezultati slika se prvo treba obraditi metodom thresholding-a. Slika 20 prikazuje Canny edge algoritam na obrađenoj slici.



Slika 20. Canny edge detection na slici koja je obrađena

U usporedbi sa slikom 21, sada su rubovi puno jasnije označeni. Slika 21 predstavlja Canny edge detection na neobrađenoj slici.

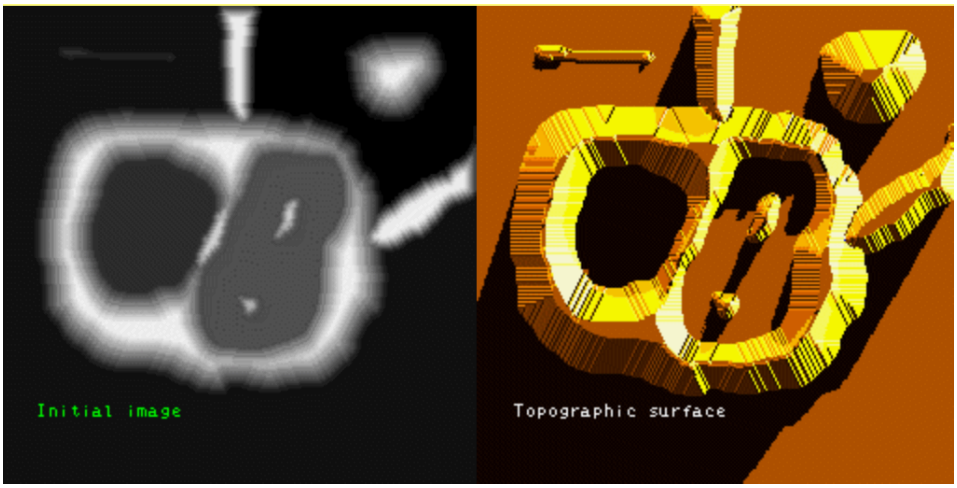


Slika 21. Canny edge detection na slici iz prethodnog primjera ali bez obrade

Segmentacija slike

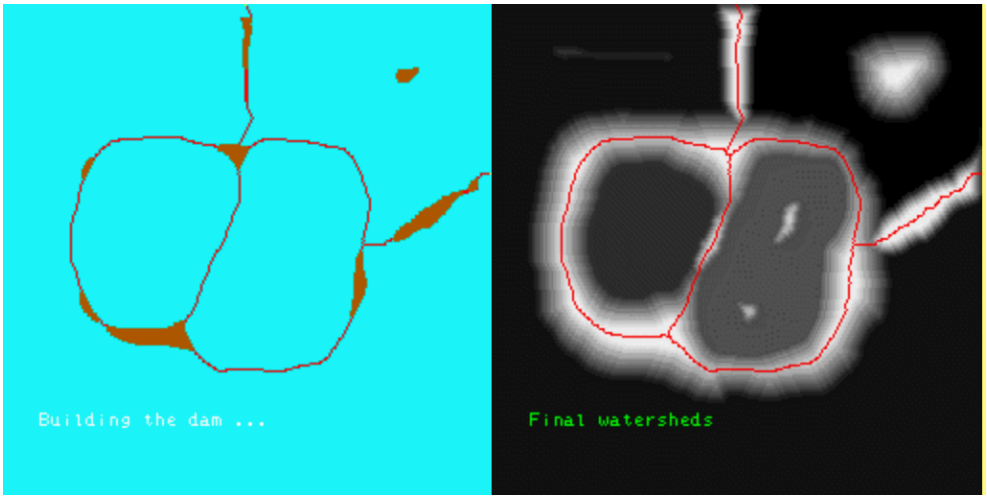
Segmentacija slika je postupak u kojem se slika dijeli na segmente tako da bi se mogla lakše analizirati i da bi se kroz jednostavniji prikaz slike lakše interpretiralo njeno značenje (Hapiro & Stockman, 2001). Pojednostavljenu sliku je lakše analizirati. Ovaj proces koristi se za pronalaženje objekata na slici i njihovih granica koje ih odvajaju. Kao rezultat dobiju se segmenti koji skupa čine cijelu sliku ili sliku sa obrubima. Segmenti slike razlikuju se po intenzitetu svjetlosti, boji te teksturi. Pikseli u jednom segmentu su slični i dijele iste karakteristike dok pikseli iz različitih segmenata su drastično različiti.

U nastavku biti će prikazan rad Watershed algoritma. Ideja ovog algoritma je da se slika gleda kao topografska mapa koja ima doline i vrhove. Doline su dijelovi slike s manjim intenzitetom dok vrhovi s većim (Slika 22).



Slika 22. Pretvorba slike u topografsku sliku

Sada kad je pronađena topografska mapa, slika se puni vodom krenuvši iz njenog minimuma. Kako voda raste povlače se granice tamo di se počinje miješati. Postupak se ponavlja dok svi vrhovi nisu potopljeni.



Slika 23. Potopljena slika

Slika 23 prikazuje rezultat prethodno opisanog postupka. No ovaj postupak nije najbolji zato što često šum ili nepravilnosti slike mogu uzrokovati probleme. Rješenje je koristiti Marker based watershed algoritam odnosno ručno odrediti koje dolini će se ujediniti a koje će ostati odvojene prije primjene Watershed algoritma. Proces je takav da se odredi na slici točno što je objekt ili *foreground,* točno što je pozadina slike te se oni dijelovi za koje nije sigurno gdje spadaju označe s 0.

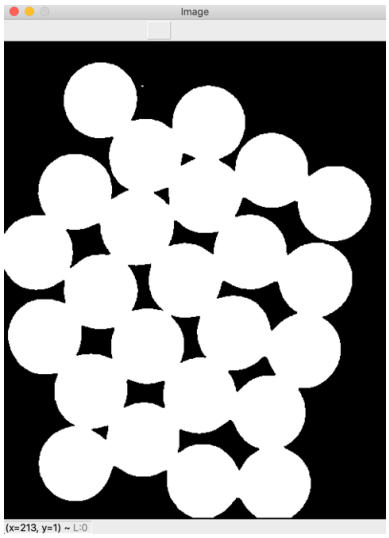
Rad algoritma će se prikazati na slici 24 koja prikazuje novčiće na podlozi koji se međusobno dodiruju.

Slika 24. Slika koja se koristi za primjer algoritma

Prvi korak je primjena image thresholding algoritma, a zatim se koristi Otsu-ovu binarizacija (Slika 25) da bi se dobila maska slike (Slika 26).

```
5  img = cv2.imread('coins.png')
6  gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
7  ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
8  cv2.imshow('Image',thresh)
```

Slika 25. Primjena Otsu-ove binarizacije



Slika 26. Rezultat Otsu-ove binarizacije

Kao rezultat je dobiven smo jedan povezani objekt iako se zapravo radi o većem broju istih okruglih objekata koji su poslagani jedan do drugoga. Slijedeći korak je maknut sav šum sa slike te nepravilnosti novčića. Koristimo morfološko zatvaranje te filter odnosno kernel veličine 3x3 (Slika 27).

```
11  kernel = np.ones((3,3),np.uint8)
12  opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel, iterations = 2)
```

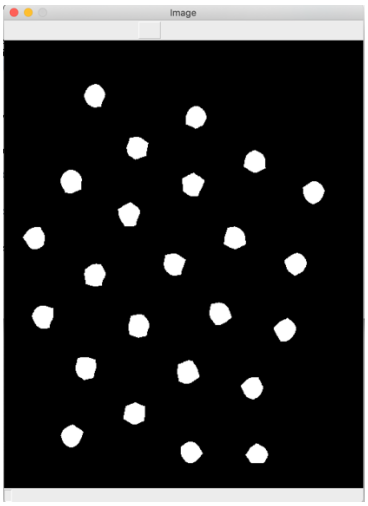
Slika 27. Uklanjanje šuma

U slučaju da se novčići ne dodiruju koristili bi eroziju. Taj postupak miče granice objekata te bi tada bili sigurni da ono što je ostalo unutar tih granica su sigurno novčići. No s obzirom da se u ovom slučaju novčići diraju koristit će se distance transform. Na taj način će se u maski pronaći novčići. Za pronaći što je pozadina slike primjenjuje se dilatacija (Slika 28).

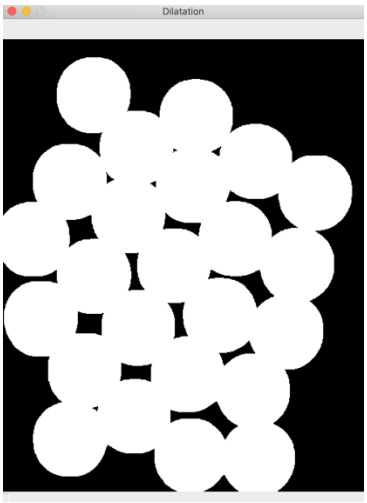
```
16  sure_bg = cv2.dilate(opening,kernel,iterations=3)
17  cv2.imshow('Dilatation',sure_bg)
18
19
20  dist_transform = cv2.distanceTransform(opening,cv2.DIST_L2,5)
21  ret, sure_fg = cv2.threshold(dist_transform,0.7*dist_transform.max(),255,0)
22  cv2.imshow('Image',sure_fg)
```

Slika 28. Prikaz koda pronlaženja pozadine i novčića

Rezultat prethodnog postupka je slijedeći (Slika 29 i Slika 30):



Slika 29. Bijelo označava novčiće



Slika 30. Crno označava pozadinu

U sljedećem koraku potrebno je pronaći granice novčića koje nisu obuhvaćene ni na jednoj od slika iznad. To se radi tako da slike oduzmemo jednu od druge (Slika 31).

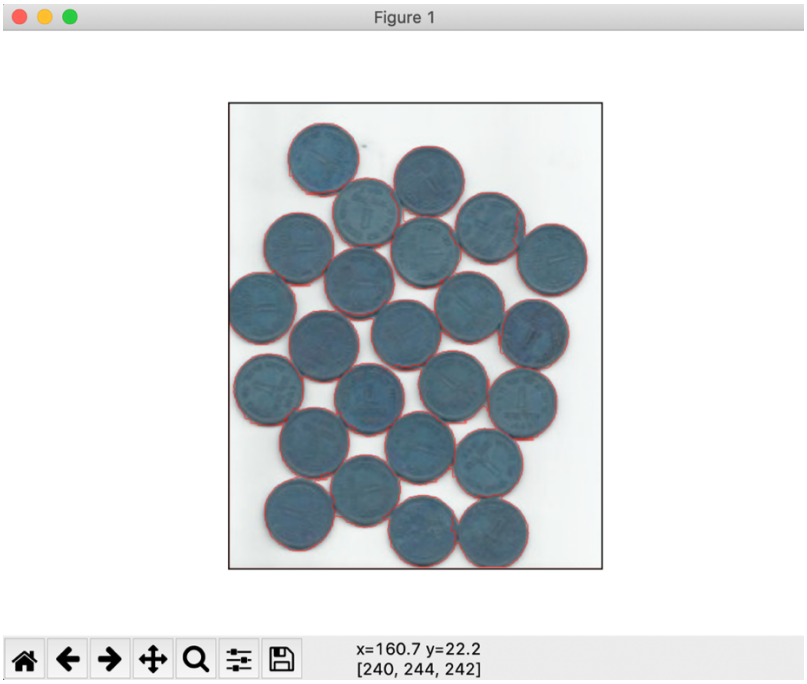
```
26  unknown = cv2.subtract(sure_bg,sure_fg)
```

Slika 31. Traženje nepoznatog dijela

Predzadnji korak je postavljanje markera na sliku. Marker će biti matrica veličine slike. Unutra matrice označavamo dijelove slike. S pozitivnim brojem označimo ono što nam je poznato dok s 0 označimo nepoznate dijelove. Za to se koristi funkcija *cv2.connectedComponents()*. Ona označava pozadinu slike s 0 a ostalo s pozitivnim brojevima. Također je potrebno marker povećati za 1 jer ako ostane 0 tada će algoritam pozadinu tretirat kao nepoznati dio slike. Mi ćemo nepoznati dio označit s 0. Na kraju se primjenjuje watershed algoritam (Slika 32).

```
28 # Označimo marker
29 ret, markers = cv2.connectedComponents(sure_fg)
30
31 # Marker povečavamo za 1
32 markers = markers+1
33
34 # Označavamo nepoznati dio slike
35 markers[unknown==255] = 0
36
37 markers = cv2.watershed(img,markers)
38 img[markers == -1] = [255,0,0]
39
40 #Prikaz slike
41 plt.imshow(img, cmap = 'gray', interpolation = 'bicubic')
42 plt.xticks([]), plt.yticks([])  # to hide tick values on X and Y axis
43 plt.show()
44
```

Slika 32. Označavanje markera i primjena algoritma



Slika 33. Rezultat Marker Based Watershed algoritma

Slika 33 prikazuje rezultat algoritma. Novčići su detektirani iako kod nekih novčića granica ne prati ispravno rub novčića.

Detekcija vrhova

U ovom poglavlju biti će prikazano kako se koristi detektor vrhova za pronalaženje vrhova na slici. Vrhovi su jedni od značajki slike te njihov pronalazak je ključan za uspoređivanje značajki dviju slika. Kako bi računalo moglo pronaći jedan dio slike na drugoj on opisuje te značajke

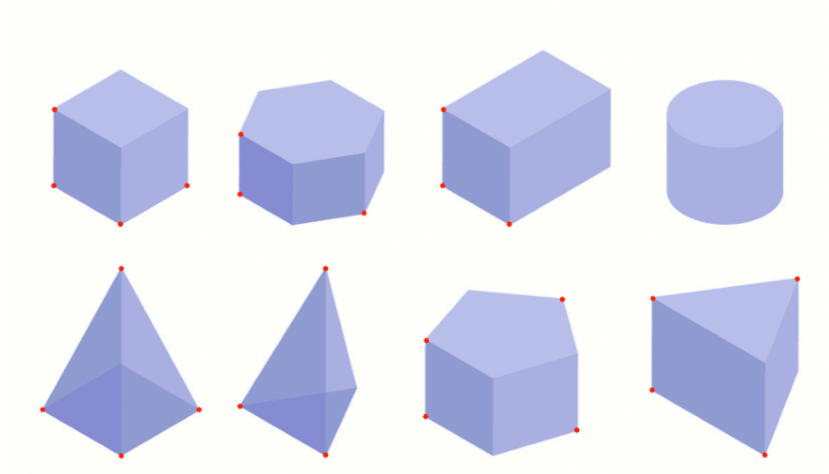
kao što ljudi rade kada traže neki predmet na slici. To se zove opisivanje značajki*(eng. feature description)*.

OpenCV nudi mnogo algoritama koji služe za detektiranje značajki na slici. Funkcija koja će se koristi u nastavku je *cv2.goodFeatureToTrack()*. Ona pronalazi određen broj najboljih vrhova na slici i koristi Shi-Tomasi detektor vrhova. Taj detektor razlikuje se od Harris-ova detektora po matematičkoj formuli koju koristi. Slika 34 prikazuje program za pronalaženje vrhova.

```
1  import numpy as np
2  import cv2
3  from matplotlib import pyplot as plt
4
5  img = cv2.imread('kutevi.png')
6  gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
7
8  corners = cv2.goodFeaturesToTrack(gray,25,0.01,10)
9  corners = np.int0(corners)
10
11 for i in corners:
12     x,y = i.ravel()
13     cv2.circle(img,(x,y),3,255,-1)
14
15 plt.imshow(img),plt.show()
```

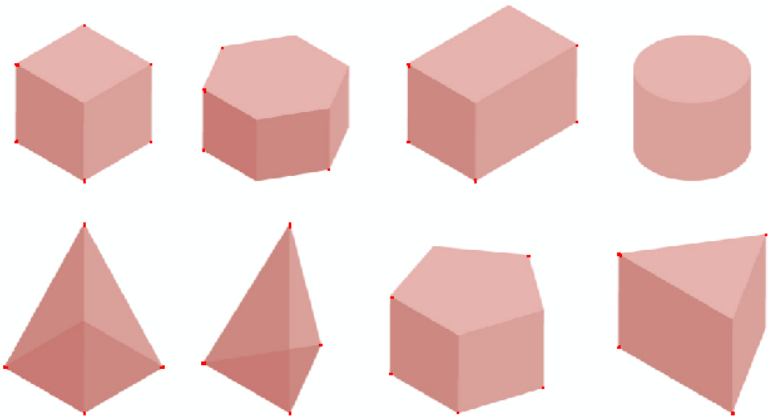
Slika 34. Pronalaženje vrhova

Učitanu sliku potrebno je pretvoriti u grayscale verziju. Koristi se prije opisana funkcija kojoj se određuje broj vrhova koji se želi naći, u ovom slučaju 25, te kvalitetu tog kuta i minimalnu udaljenost između dva kuta. Rezultat je prikazan na slici 35:



Slika 35. Pronađeni vrhovi na slici

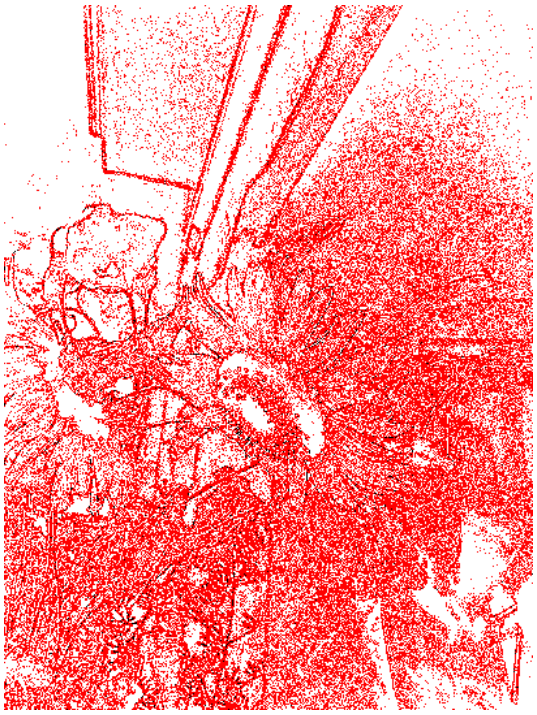
Drugi poznati corner detector je Harris-ov. Primjenom Harris detektora na primjeru sa slike 39. pronađeno više vrhova od nego u prethodnom slučaju kada je korišten Shi-Tomasi detektor ali svejedno nisu svi vrhovi pronađeni (Slika 36).



Slika 36. primjer Harris Corner detektora

Na konkretnoj slici suncokreta s kompleksnom pozadinom pore (Slika 37.) Harris-ov detektor pronalazi neke vrhove na cvijetu, međutim veliki broj vrhova i rubova je ostao nedetektiran. Nakon prethodne obrade slike metodom adaptivnog thresholdig-a dobivamo drastično veći broj vrhova na cvijetu i rubove kuće uz koju se nalazi (Slika 38).

Slika 37. Pronađeni vrhovi koristeći Harris Corner Detector



Slika 38. Pronađeni vrhovi na obrađenoj slici

U nastavku biti će prikazano kako usporedit značajke neke slike nakon što su pronađene.

Usporedba slike i usporedba značajki

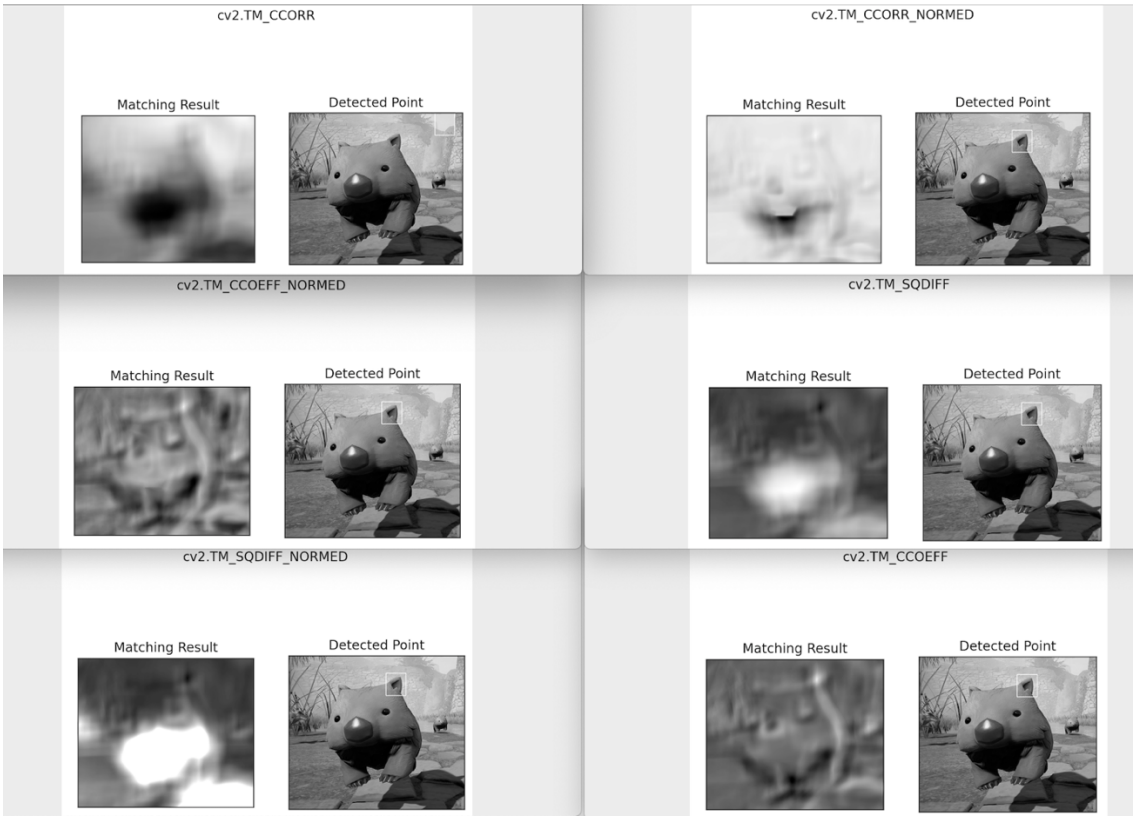
Kada se želi pronaći neki objekt ili dio nekog objekta u većoj slici ili kada se želi usporediti više slike koriste se metode za usporedbu objekata koje se temelje na izlučenim značajkama.

Template matching

Template matching je metoda za pronalaženje nekog objekta ili nekog dijela slike u većoj slici. Prvo je potrebno odrediti template odnosno koji dio slike se traži na slici koja se pretražuje. Algoritam radi tako da se template provlači preko druge slike te se uspoređuje s tom slikom dok se ne pronađe taj dio na slici. OpenCV nudi 6 metoda za uspoređivanje template-a sa originalnom slikom. Za primjer kao template koristit će se sljedeća slika (Slika 39):

Slika 39. Template

Slika 40 prikazuje usporedbu metoda: TM_SQDIFF, TM_SQDIFF_NORMED, TM_CCORR, TM_CCORR_NORMED, TM_CCOEFF, TM_CCOEFF_NORMED.



Slika 40. Template matching primjer

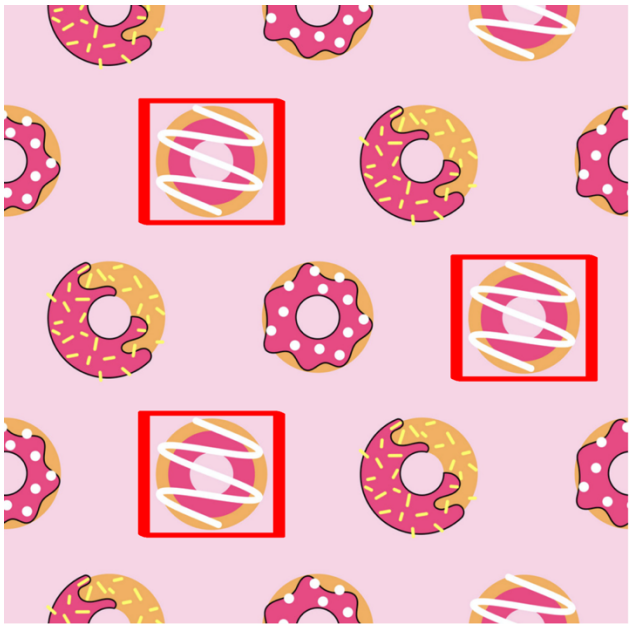
Rezultat prikazuje da 5 od 6 metoda su bile uspješne u pronalaženju template-a na slici. Ono što razlikuje ove metode međusobno je njihova matematička podloga te se može zaključiti da ovisno o templateu i slici koja se pretražuje potrebno je prvo istražiti koja će metoda najbolje odgovarati.

Ako se želi pronaći predmet koji se ponavlja više puta na slici program će izgledati ovako (Slika 41):

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

img_rgb = cv2.imread('donuts.png')
img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2GRAY)
template = cv2.imread('donut.png',0)
w, h = template.shape[::-1]

res = cv2.matchTemplate(img_gray,template,cv2.TM_CCOEFF_NORMED)
threshold = 0.8
loc = np.where( res >= threshold)
for pt in zip(*loc[::-1]):
    cv2.rectangle(img_rgb, pt, (pt[0] + w, pt[1] + h), (0,0,255), 2)

cv2.imwrite('res.png',img_rgb)
```

Slika 41. Program za pronalaženje predmeta s više pojavljivanja na slici

Slika 42 prikazuje da je template uspješno pronađen više puta na jednoj slici.



Slika 42. primjer multiple template matching

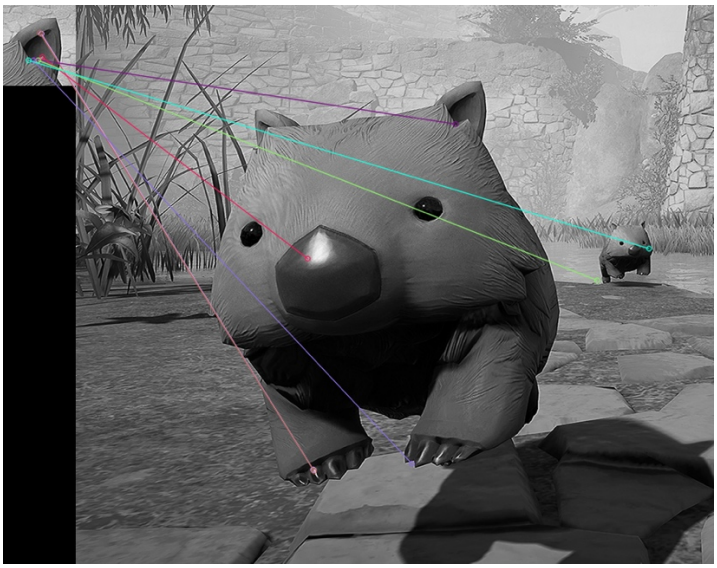
Brute Force i FLANN based Matcher

Brute Force Matcher metoda uspoređuje jedan opis značajke s prve slike te ga uspoređuje s opisima svih značajki druge slike i vraća najbliži. Može se koristiti sa SIFT, SURF, ORB i ostalim algoritmima. Slika 43 prikazuje primjer Brute Force Matchera s ORB deskriptorom.

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

img1 = cv2.imread('box.png',0)          # queryImage
img2 = cv2.imread('box_in_scene.png',0) # trainImage

# Initiate SIFT detector
orb = cv2.ORB()

# find the keypoints and descriptors with SIFT
kp1, des1 = orb.detectAndCompute(img1,None)
kp2, des2 = orb.detectAndCompute(img2,None)

# create BFMatcher object
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

# Match descriptors.
matches = bf.match(des1,des2)

# Sort them in the order of their distance.
matches = sorted(matches, key = lambda x:x.distance)

# Draw first 10 matches.
img3 = cv2.drawMatches(img1,kp1,img2,kp2,matches[:10], flags=2)

plt.imshow(img3),plt.show()
```

Slika 43. Brute Force Matcher uz ORB

Dobiveni rezultat je slijedeći (Slika 44):

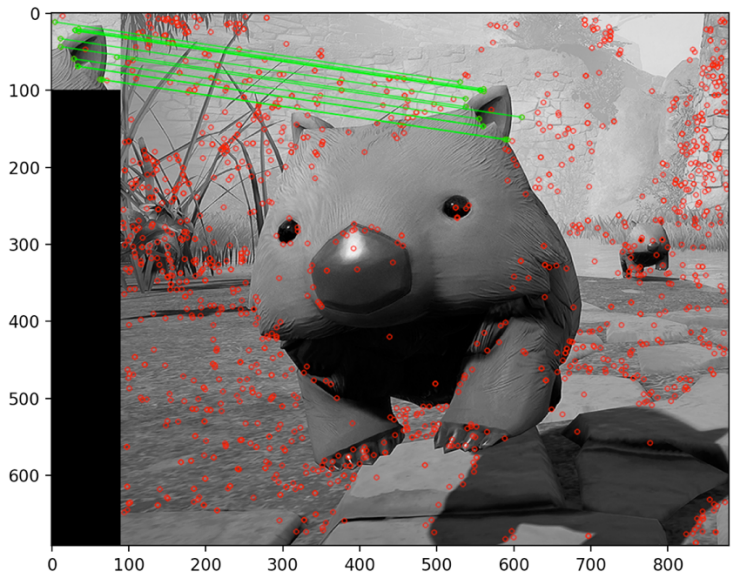


Slika 44. Rezultat BF matcher-a

Uočljivo je da je program pronašao samo neke karakteristike jedne slike na drugoj te je dosta točaka krivo prepoznano. Drugi primjer je FLANN based Matcher. On sadrži kolekciju algoritma za brzo pronalaženje najbližih susjeda te radi brže od BF Matcher-a za veliki set podataka. Program izgleda nešto drugačije kao i dobiveni rezultat. Na slici 45 prikazan je FLANN based Matcher.

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

img1 = cv2.imread('box.png',0)          # queryImage
img2 = cv2.imread('box_in_scene.png',0) # trainImage

# Initiate SIFT detector
sift = cv2.SIFT()

# find the keypoints and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)

# FLANN parameters
FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks=50)   # or pass empty dictionary

flann = cv2.FlannBasedMatcher(index_params,search_params)

matches = flann.knnMatch(des1,des2,k=2)

# Need to draw only good matches, so create a mask
matchesMask = [[0,0] for i in xrange(len(matches))]

# ratio test as per Lowe's paper
for i,(m,n) in enumerate(matches):
    if m.distance < 0.7*n.distance:
        matchesMask[i]=[1,0]

draw_params = dict(matchColor = (0,255,0),
                   singlePointColor = (255,0,0),
                   matchesMask = matchesMask,
                   flags = 0)

img3 = cv2.drawMatchesKnn(img1,kp1,img2,kp2,matches,None,**draw_params)

plt.imshow(img3,),plt.show()
```

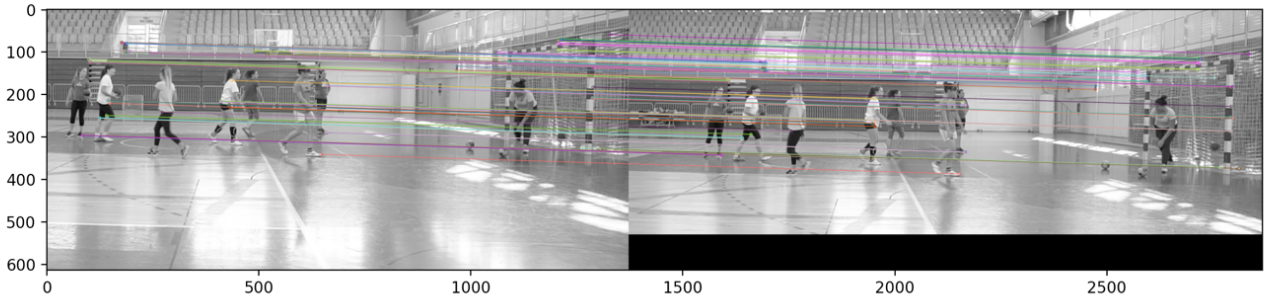
Slika 45. FLANN based Matcher



Slika 46. Rezultat FLANN Matcher-a

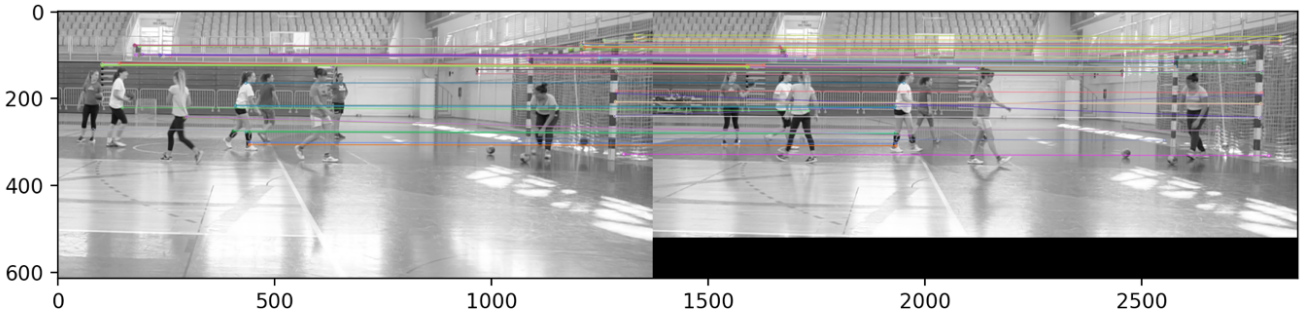
Iz slike se vidi da ovaj put puno više dijelova prve slike je pronađeno na drugoj te uz to su označene značajke obje slike (Slika 46).

U slijedećem primjeru biti će prikazano kako BF i FLANN matcher-i rade u dva slučaja: za dva preuzeta susjedna frame-a iz video zapisa i za prvi i peti frame iz video zapisa. Slika 47 prikazuje BF matcher za dva susjedna frame-a.



Slika 47. BF matcher na dva susjedna video frame-a

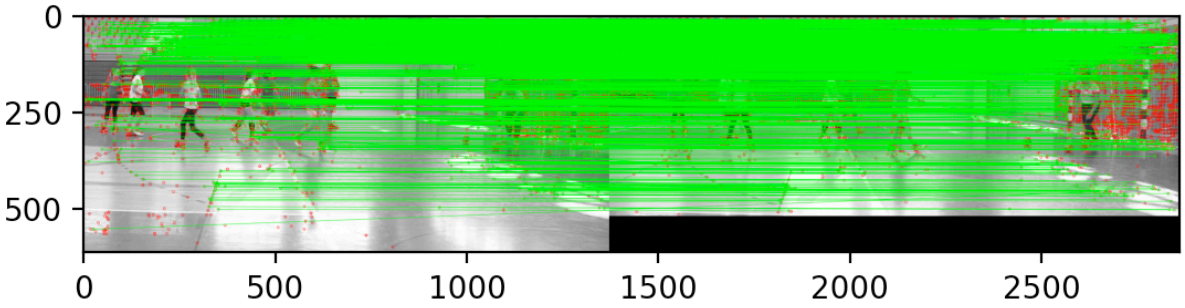
Slika 48 prikazuje BF matcher na prvi i peti video frame.



Slika 48. BF matcher na prvi i peti video frame

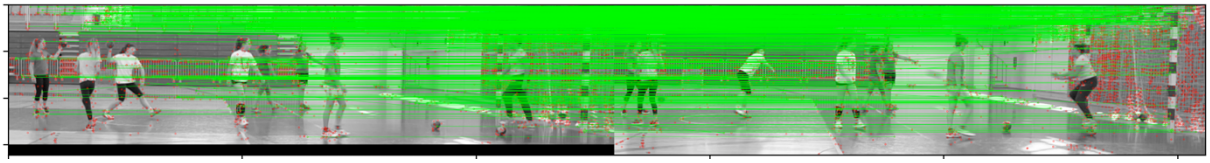
Iz prikazanog se može zaključiti da je algoritam pronašao određene značajke, pogotovo one koje su stacionarne poput stative gola te ako se tijela ljudi na slici nisu previše pomakla također su pronađene značajke sa njih.

FLANN based matcher je pronašao više značajki te rezultat izgleda ovako (Slika 49):



Slika 49. FLANN matcher na prvi i peti frame

U slučaju da se uzmu dva frame-a iz video zapisa koja su još više udaljena tada FLANN matcher postiže slijedeći rezultat (Slika 50).



Slika 50. FLANN matcher na dva udaljena frame-a

Može se uočiti da su statične točke još uvijek detektirane i pronađene na drugoj slici no jasno je da što su više dva frame-a udaljena vremenski jedan od drugog ima sve manje preklapanja na objektima koji su u pokretu.

Kreiranje panorame: spajanje više slika u jednu

Panorama je slika koja se dobije spajanjem barem dvije slike koje se preklapaju na jednom dijelu. Kreiranje panorame koristi prepoznavanje značajnih točki slike da bi se one mogle spojiti u jednu. Prvi korak u obradi je detekcija značajki na obje slike. Nakon toga slijedi faza koja se zove predstavljanje značajki (eng. feature description). Pomoću tih opisa računalo uspoređuje svaku značajnu točku te odlučuje koja dvije su jednake. Primjenom jednog od feature matcher-a značajne točke se pronalaze međusobno na slikama. U sljedećem koraku potrebno je maknuti pogrešna preklapanja. Vidjeli smo na primjeru kod FLANN based matcher-a da su se neke značajne točke krivo spojile zbog pomaka ili rotacije. Sada je bitno da ih se ukloni inače završna slika neće biti dobro spojena. Nakon toga koristi se homography matrica koja mapira gdje se neka točka s prve slike nalazi na drugoj, a nakon toga se slike spajaju u jednu. Ovisno kako rezultat ispadne slika se može izrezati i s nje ukloniti nabore i nepravilnosti.

Slika 51 prikazuje 3 slike koje će se koristiti za kreiranje panorame.

Slika 51. Slike korištene za panoramu

Slika 52 dobivena je spajanjem 3 slike u panoramu.

Slika 52. Dobivena panorama

Za bolji rezultat sliku je još potrebno izrezati da se makne postojeća distorzija.

U nastavku je prikazan rezultat panorame kada se tri susjedna frame-a iz video zapisa probaju spojit (Slika 54). Slike koje se spajaju su prikazane na slici 53.

Slika 53. Video frame-ovi za panoramu

Slika 54. Spajanje frame-ova iz video zapisa

Algoritam je iz 3 frame-a izabrao najbolje dijelove iz spojio ih u jednu sliku. Na slici nisu vidljive veće nepravilnosti.

Ako se odaberu 3 više udaljena frame-a iz video zapisa onda algoritam vraća zadnju učitanu sliku s obzirom da se slike ne podudaraju. Slika 55 prikazuje odabrane frame-ove.

Slika 55. Odabrani frame-ovi

Možemo uočiti da je dobiveni rezultat (Slika 56) jednak zadnjoj slici sa slike 55.

Slika 56. Rezultat spajanja udaljenih frame-ova

Jedina uočljiva razlika je na rubovima slike no to je rezultat toga što slike nisu bile jednako izrezane.

Zaključak

Računalni vid i obrada slika područja su umjetne inteligencije koja se posljednjih godina intenzivno razvijaju. S poboljšanjem rezultata povećavaju se i mogućnosti primjene koje se već sada moguća u različitim područjima od svakodnevnog života, do automobilističke industrije, sigurnosti i medicine.

U ovom završnom radu upoznao sam se sa osnovnim metodama obrade slike i predstavljeni su rezultati metode obrade slika koje su podržane u biblioteci OpenCV na različitim primjerima. Radi se o velikom i zanimljivom području tako da ću svakako nastaviti istraživati i učiti o tome s obzirom da sam dosta naučio i zainteresirao se tokom pisanja ovog završnog rada.

Popis slika

Popis literature i izvora

1. Pyimagesearch: https://www.pyimagesearch.com/2018/12/17/image-stitching-with-opencv-and-python/ (last visited 16.8.2020.)
2. Wikipedia Homography: https://en.wikipedia.org/wiki/Homography_(computer_vision) (last visited 16.9.2020.)
3. OpenCV Python Tutorials: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html (last visited 15.8.2020.)
4. Wikipedia feature Detection: https://en.wikipedia.org/wiki/Feature_detection_(computer_vision) (last visited 15.8.2020)
5. Wikipedia Shi-Tomasi Corner Detection: https://en.wikipedia.org/wiki/Corner_detection#The_Harris_&_Stephens_/_Shi–Tomasi_corner_detection_algorithms (last visited 15.8.2020.)
6. CMM Image Segmentation: http://www.cmm.mines-paristech.fr/~beucher/wtshed.html (last visited 14.8.2020)
7. Wikipedia Image segmentation: https://en.wikipedia.org/wiki/Image_segmentation (last visited 14.8.2020.)
8. OpenCV Tutorials Image segmentation: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_watershed/py_watershed.html (last visited (14.8.2020.)
9. Wikipedia Image gradient: https://en.wikipedia.org/wiki/Image_gradient (last visited 13.8.2020.)
10. OpenCV Documentation Object Detection: https://docs.opencv.org/2.4/modules/imgproc/doc/object_detection.html?highlight=matchtemplate#matchtemplate (last visited 13.8.2020)
11. Wikipedia Thresholding: https://en.wikipedia.org/wiki/Thresholding_(image_processing) (last visited 13.8.2020.)
12. Wikipedia OpenCV: https://en.wikipedia.org/wiki/OpenCV (last visited 10.8.2020)
13. Wikipedia Edge Detection: https://en.wikipedia.org/wiki/Edge_detection (last visited 12.8.2020)
14. Wikipedia Image Analysis: https://en.wikipedia.org/wiki/Image_analysis#Techniques (last visited 12.8.2020)
15. Wikipedia Computer Vision: https://hr.wikipedia.org/wiki/Računalni_vid (last visited 11.8.2020.)
16. OpenCV Tutorials Template Matching: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_template_matching/py_template_matching.html (last visited 13.8.2020.)

17. OpenCV Tutorials: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_gui/py_image_display/py_image_display.html#display-image (last visited 11.8.2020.)
18. OpenCV Tutorials Canny edge: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html#canny (last visited 14.8.2020.)
19. Khattab, Dina & Ebied, Hala & Hussein, Ashraf & Tolba, Mohamed. (2015). Automatic GrabCut for Bi-label Image Segmentation Using SOFM. Advances in Intelligent Systems and Computing. 323. 579-592. 10.1007/978-3-319-11310-4_50.
20. Ballard, Dana H. & Brown, Christopher M.: Computer Vision, Prentice Hall, 1982
21. Gonzalez, Rafael: Digital image processing, New York, Pearson, 2018
22. Hapiro, Linda G. & Stockamn, George: Computer vision, New Jeresy, Prentice Hall, 2001
23. Kaehler, Adrian & Bradski, Gary: Learning Computer Vision With The OpenCV Library, O'Reilly Media, 2008
24. Karami E. & Prasad S. & Shetata M.: Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images, 2015
25. Solomon, C.J. & Breckon, T.P.: Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab, Wiley-Blackwell, 2010