Univerza v Ljubljani
Fakulteta za elektrotehniko

1. Laboratorijska vaja razpoznavanje vzorcev

Mentor: as. dr. Klemen Grm, mag. inž. el.

Avtor: Blaž Ardaljon Mataln Smehov

Datum: 8.12.2021

Naloga 1a: predobdelava slik, upragovljanje z maksimizacijo informacije

Pri prvi vaji smo izvedli upragovljanje slik z maksimizacijo informacije. Za izvajanje vaje smo že dobili predlogo kode, v kateri smo morali dopolniti določene funkcije.

Upragovljanje smo izvedli na dveh slikah zobnikov. Prva naloga je bila obdelava sivinske slike, v komentarju kode so že bile podane predlagane funkcije za obdelavo iz knjižnice cv2. Med predlaganimi funkcijami so bile cv2.equalizeHist, cv2.GaussianBlur in cv2.medianBlur

cv2.equalizeHist – ta funkcija služi za izravnavo histograma, kar izboljša kontrast naše slike, kot argument v funkcijo podamo sivinsko sliko, funkcija nam vrne sliko z izravnanim histogramom
cv2.GaussianBlur – ta funkcija nam služi za filtriranje slike, kot vhodne argumente moramo navesti dolžino in širino jedra, standardno deviacijo v smeri x in y. Ta funkcija je učinkovita pri odstranjevanju Gaussovega šuma s slike,
cv2. medianBlur – funkcija vzame mediano vseh pikslov pod območjem jedra in na mesto srednjega elementa zapiše vednost mediane. Funkciji kot argument podamo sliko, ki jo želimo obdelati in velikost jedra.

Koda:

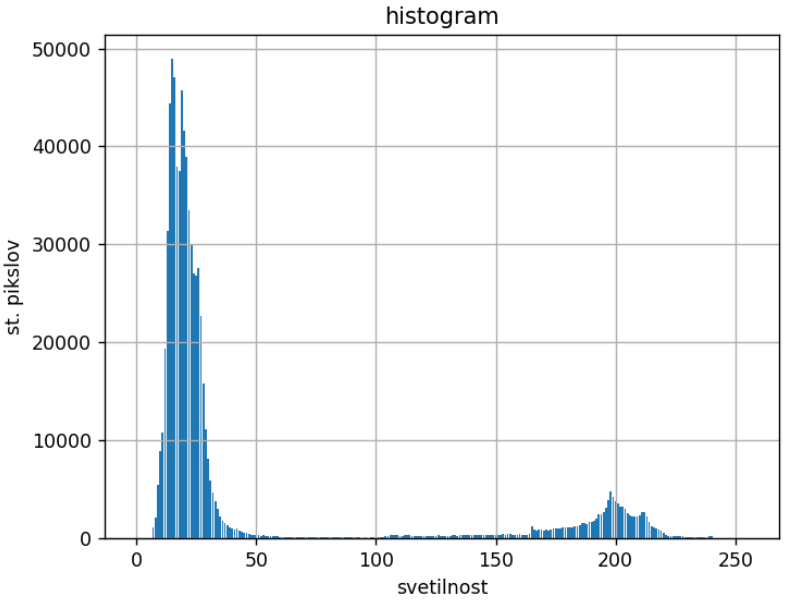
```
    slika = cv2.imread(filename)
    slika = cv2.cvtColor(slika, cv2.COLOR_BGR2RGB)
    sivaSlika = cv2.cvtColor(slika, cv2.COLOR_RGB2GRAY)

    # TODO: Obdelava sivinske slike, npr. s postopki
    # izravnave histograma, filtrom mediane in Gaussovim glajenjem
    # gl. funkcije: cv2.equalizeHist, cv2.GaussianBlur, cv2.medianBlur

    obdelanaSlika = cv2.medianBlur(sivaSlika,9)

    # Izračun in prikaz histograma ter slik

    histogram = izracunajHistogram(obdelanaSlika)
    narisiHistogram(histogram)
```

V kodi od zgoraj so zapisani klici funkcij, prva funkcija cv2.imread prebere sliko in jo zapiše kot matriko, v kateri ima vsak piksel svojo sivinsko vrednost, druga funkcija cv2.cvtColor sliko spremeni iz BGR v RGB format, z enako funkcijo potem RGB sliko spremenimo v sivinsko sliko.

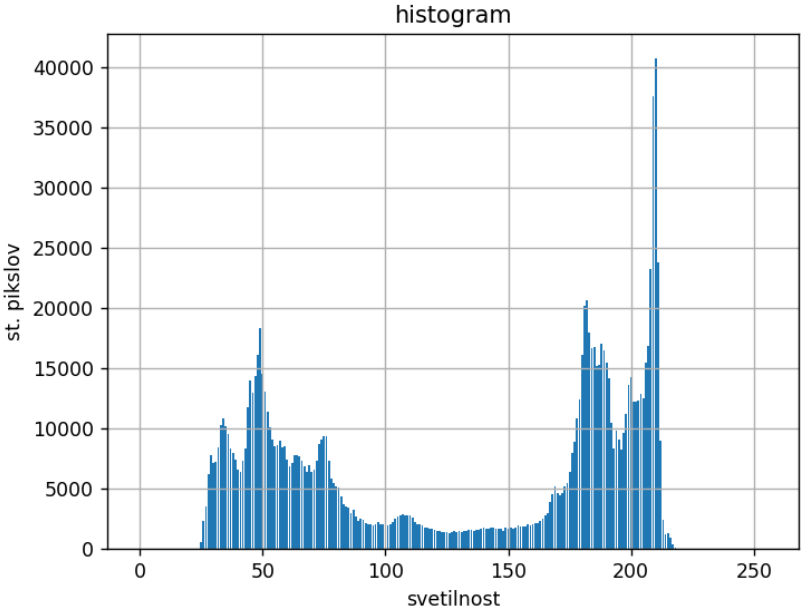
obdelanaSlika = cv2.medianBlur(sivaSlika,9), to je del kode, ki sem ga zapisal sam. Ta funkcija izvede filtriranje slike s pomočjo mediane, kot je opisano zgoraj

V predlogi kode sta že bili zapisani dve funkciji za izračun in risanje histograma.

Histogram slike zobnik1: na histogramu je na x osi zapisana svetilnost, oziroma sivinski nivo slike, na y osi je prikazano koliko pikslov ima takšno vrednost svetilnosti.



Histogram slike zobnik2: na histogramu je na x osi zapisana svetilnost, oziroma sivinski nivo slike, na y osi je prikazano koliko pikslov ima takšno vrednost svetilnosti.



Poleg obdelave slike smo morali pri prvem delu vaje zapisati tudi funkcijo, ki izračuna takšen prag, ki nam da maksimalno informacijo. Enačba po kateri izračunamo vrednost optimalnega praga je bila podana v navodilih vaje:

$$H_0(t) = -\sum_{i=0}^{t} \frac{P_i}{P^*(t)} \log\left(\frac{P_i}{P^*(t)}\right)$$

$$H_1(t) = -\sum_{i=t+1}^{L-1} \frac{P_i}{1-P^*(t)} \log\left(\frac{P_i}{1-P^*(t)}\right).$$

$$t^* = \arg \max_{t=0,\cdots,L-2} \{H_0(t) + H_1(t)\}$$

Koda:

```
    imenovalec = 0
    H0 = np.zeros_like(histogram)
    H1 = np.zeros_like(histogram)

    for t in range(1,255):

        imenovalec += P[t]

        for i in range(1, 255):
            if (i <= t and imenovalec>0 and P[i]>0):
                H0[t] -= ((P[i]/imenovalec)*np.log(P[i]/imenovalec))

            if (i > t and imenovalec<1 and P[i]>0):
                H1[t] -= ((P[i]/(1-imenovalec))*np.log(P[i]/(1-imenovalec)))

        informacija[t]= H0[t]+H1[t]


    #TODO: izračunaj informacijo pri vsaki možni vrednosti praga
    #TODO: določi vrednost praga, ki maksimizira informacijo

    # izračun informacije pri vsakem možnem pragu:
    # iskanje vrednosti praga, kjer je informacija maksimalna

    t = np.argmax(informacija)
```

V prvem koraku sem ustvaril dva seznama ničel, ki sta velikosti seznama histogram. To sem naredil s funkcijo np.zeros_like, ta funkcija je na voljo v knjižnici numpy.

V zanki od 1 do 255, torej taki ki zajema vse možne vrednosti svetilnosti sem računal imenovalec, ki se vsako iteracijo poveča za vrednost ulomka, ki je definiran kot kvocient med številom pikslov, ki pripadajo neki vrednosti svetilnosti in številom vseh pikslov. V nadaljevanju sem naredil še eno for zanko. To sem dodal, ker koda brez nje ni delovala pravilno, nujno je bilo potrebno dodati v if stavku pogoja i <= t in i > t. Brez tega pogoja se račun ni pravilno izvedel, saj je bil v nekaterih primerih if stavek preskočen.

Vrednosti H0 in H1 sem nato izračunal po enačbi in ju vsako iteracijo seštel. Vsota teh dveh vrednosti se je shranila v seznam informacija.

Na koncu se je izvedla funkcija np.argmax(informacija). S to funkcijo poiščemo indeks, oziroma mesto v seznamu, kjer se nahaja najvišja vrednost.

Koda za izris upragovljene slike:

```
    plt.figure()
    plt.title("upragovljenaSlika")
    plt.imshow(upragovljenaSlika, cmap="gray")
```

V knjižnici matplotlib so podane funkcije za izris slik, izriše jih na podlagi seznama v katerem so zapisane vrednosti svetilnosti.

S plt.title izberemo naslov slike, s plt.imshow izrišemo sliko, kot argument podamo seznam, ki ga želimo izrisati, s cmap = 'gray' jo prikažemo kot sivinsko.

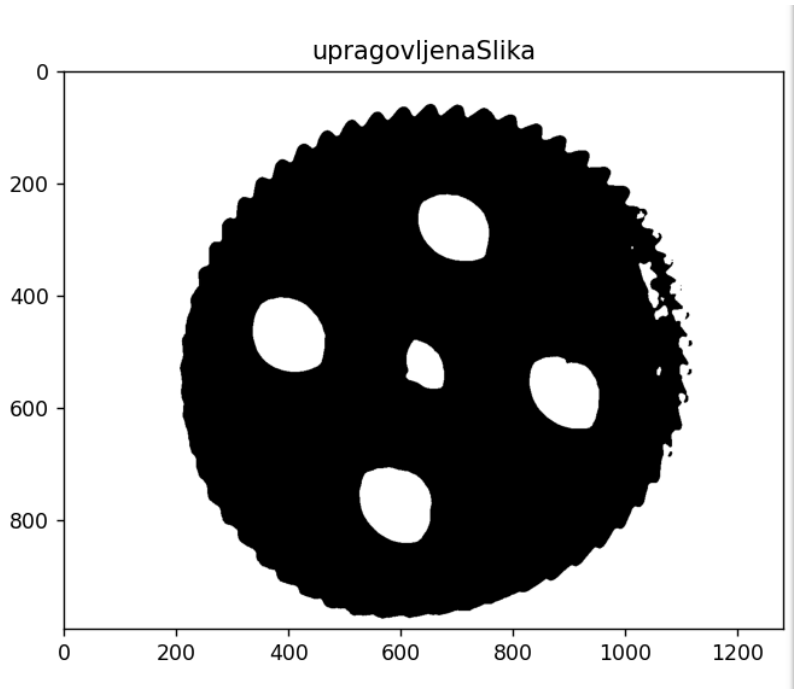
Rezultati:

Za zobnik1:

Vrednost praga: 37, vrednost praga je odvisna od tega s katerim postopkom smo obdelali sliko.

Zobnik2:

Vrednost praga: 124



Naloga 1b: iskanje obrisov področij

Pri drugi vaji v prvem sklopu smo morali dopolniti program za iskanje in izrisanje obrisa dveh zobnikov iz prejšnje naloge. Program je iskal obrisa iz upragovljenih slik.

V prvem koraku smo morali ponovno obdelati sliko z eno izmed podanih funkcij, ki sem jih že opisal pri prejšnji nalogi, ponovno sem izbral cv2.medianBlur:

```
obdelanaSlika = cv2.medianBlur(sivaSlika, 9)
```

Predloga kode je že vsebovala funkcijo, ki je izračunala začetne točke obrisov in obrise. Prva funkcija, ki smo jo morali sami napisati je bila ta, ki poišče kateri izmed obrisov je najdaljši

```
def podaj_najdaljsi_obris(self):
        """Vrne najdaljšega izmed najdenih obrisov, ki ponavadi pripada
        največjemu objektu na sliki."""
        # TODO: spiši funkcijo

        seznam_obrisov = self.obrisi
        max_index = 0
        max_dolzina = 0

        for i in range(len(seznam_obrisov)):

            if(len(seznam_obrisov[i]['kode']) > max_dolzina):

                max_dolzina = len(seznam_obrisov[i]['kode'])
                max_index = i

        return self.obrisi[max_index]
```

v prvem koraku sem ustvaril seznam obrisov, seznam_obrisov = self.obrisi. S self.obrisi dostopamo do seznama slovarjev obrisi iz razreda, da dostopamo do razreda je potrebno zapisani self., obrisi je lista slovarjev. En slovar nam podaja informacijo za en obris. V slovarju je zapisana začetna točka, koda oziroma pot po kateri se premikamo od začetne točke do končne, da dobimo obris in množica vseh točk, ki sestavljajo ta obris.

S for zanko se program pomika po vseh slovarjih v seznamu, for zanka se izvaja tolikokrat koliko je seznamov, število ponovitev sem zapisal z i `range(len(seznam_obrisov)):`, pri tem nam len() vrne dolžino seznama.

Velikost obrisa vemo preko tega kako dolga je koda, oziroma koliko točk sestavlja naš obris.

V mojem primeru sem primerjal dolžino kode z if stavkom. Dolžino kode sem dobil z zapisom `len(seznam_obrisov[i]['kode'])`, funkcija len() nam vrne dolžino segmenta, ki ga podamo. Do željene informacije v slovarju dostopamo tako da v oglatem oklepaju definiramo ime dela slovarja, ki nas zanima, torej v tem primeru ['kode']. Za primerjavo med obrisi, sem na začetku definiral maksimalno dolžino, v vsaki iteraciji v kateri je nova dolžina večja od prejšnje se dolžina te daljše informacije zapiše v maksimalno vrednost in se shrani vrednost indeksa, ki nam pove na katerem mestu v seznamu obrisov se nahaja slovar z najdaljšo kodo.

Naslednja funkcija, ki smo jo morali dopolniti je za izris obrisov. Vsak obris mora imeti različno sivinsko vrednost.

```
def narisi_vse_obrise(self):
        """Vrne sliko z vsemi obrisi vrisanimi. Vsak obris ima na končni sliki
        drugačen sivi nivo."""

        slika_obrisov = np.zeros_like(self.slika)
        # TODO: spiši funkcijo

        seznam_obrisov = self.obrisi
        sivi_nivo = 50

        for i in range(len(seznam_obrisov)):

            sivi_nivo = sivi_nivo + 10
            for tocka in seznam_obrisov[i]['tocke']:
                slika_obrisov[tocka[1],tocka[0]] = sivi_nivo

        return slika_obrisov
```

Najprej sem definiral seznam v katerem so se shranile sivinske vrednosti obrisov, ta seznam kasneje služi za izris slike. Seznam sem definiral z `slika_obrisov = np.zeros_like(self.slika)`, funkcija np.zeros_like nam da seznam ničel v velikosti seznama, ki ga podamo kot argument.

V naslednjem koraku sem kot prej seznam slovarjev z razreda shranil v nov seznam in definiral spremenljivko, s katero obrisom zapišemo sivinsko vrednost.

V for zanki, ki se izvaja tolikokrat koliko je slovarjev v seznamu vsako iteracijo povečamo sivinsko vrednost, ki se zapiše v obris za 10, s tem je zagotovljeno, da ima vsak obris drugačno vrednost svetilnosti.

V naslednji for zanki v spremenljivko tocka program shrani koordinate točk v slovarju, seveda za vsak obris posebej. Do informacije o točkah sem ponovno dostopil na enak način kot pri prejšnji nalogi z oglatim oklepajem ['tocke']. S koordinatami vsake točke lahko na

ustrezno mesto v seznamu slika_obrisov vnesemo vrednost svetilnosti. Po končani zanki je končni seznam podan kot koda s katero lahko izrišemo sliko z obrisi. Funkcija ob klicu vrne ta seznam.

Naslednja funkcija, ki smo jo morali podati je podobna prejšnji

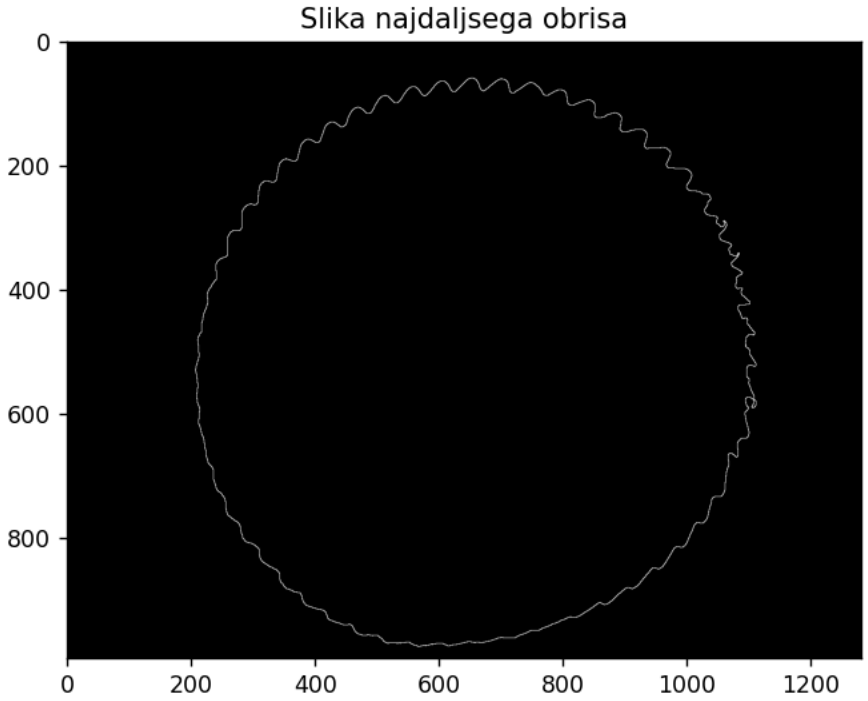
```
def narisi_obris(slika, obris):
    """Izriše podani obris na sliko iste resolucije, kot vhodna slika."""
    slika_obrisa = np.zeros_like(slika)
    # TODO: spiši funkcijo

    for tocka in obris['tocke']:
        slika_obrisa[tocka[1],tocka[0]] = (255,255,255)

    return slika_obrisa
```

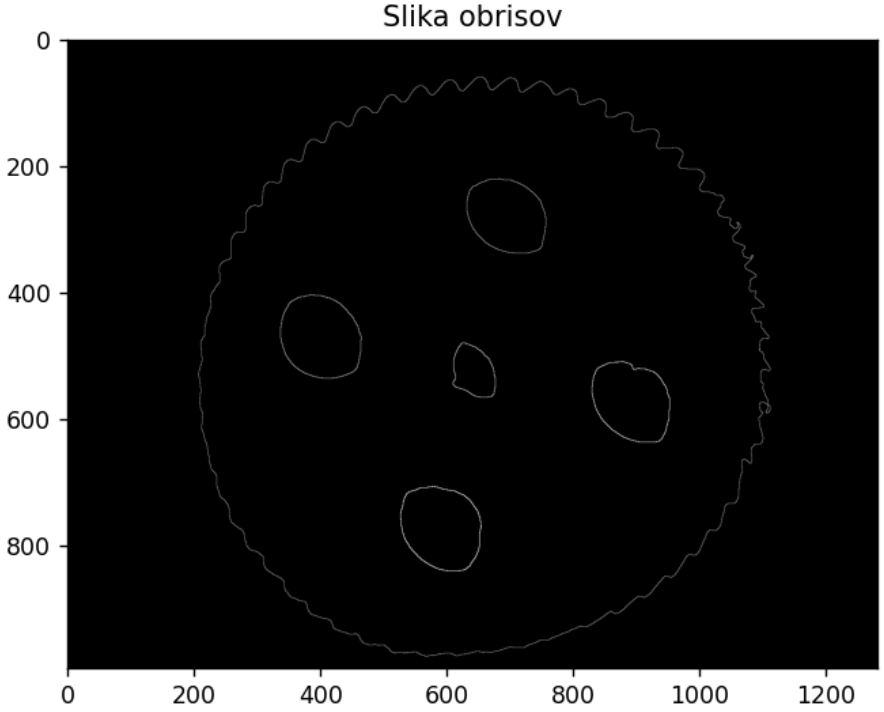
Ponovno sem ustvaril seznam ničel enake velikosti kot je seznam slika. V tem primeru ponovno priredimo sivinski nivo obrisom, vendar v tem primeru z RGB vrednostmi, ki imajo svetilnost 255.

```
    Objekt = Iskalnik(upragovljenaSlika)

    Objekt.isci_obrise()

    max_obris_obris = Objekt.podaj_najdaljsi_obris()

    vsi_obrisi = Objekt.narisi_vse_obrise()

    plt.figure()
    plt.title("Slika obrisov")
    plt.imshow(vsi_obrisi,cmap="gray")

    # Izris najdaljsega obrisa
    slika_max_obrisa = narisi_obris(slika,max_obris_obris)
    plt.figure()
    plt.title("Slika najdaljsega obrisa")
    plt.imshow(slika_max_obrisa)
```

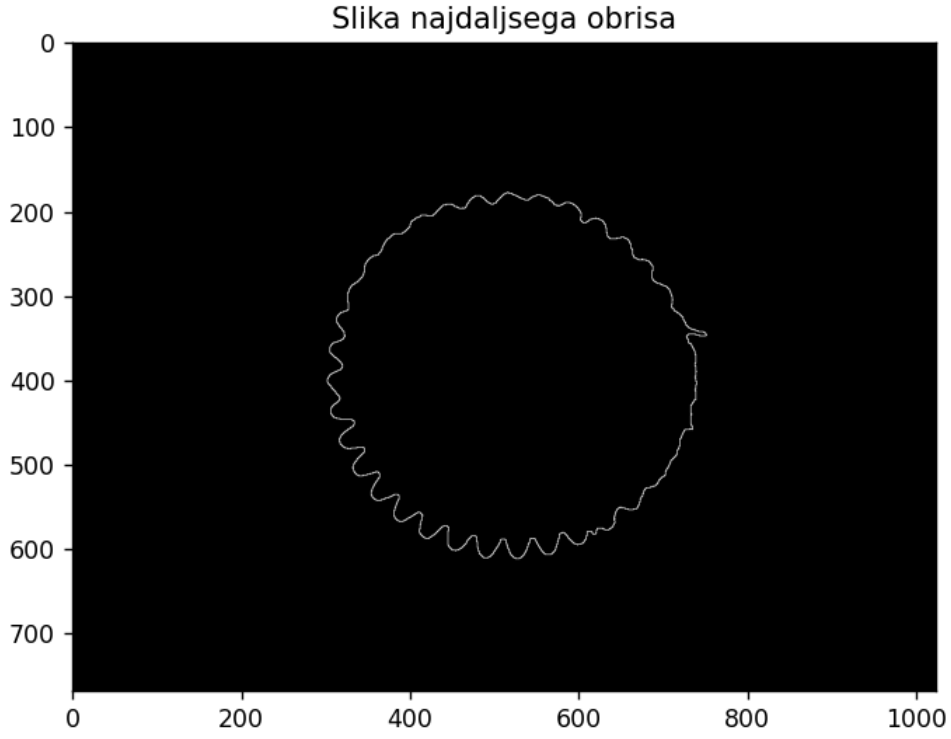
Rezultati:

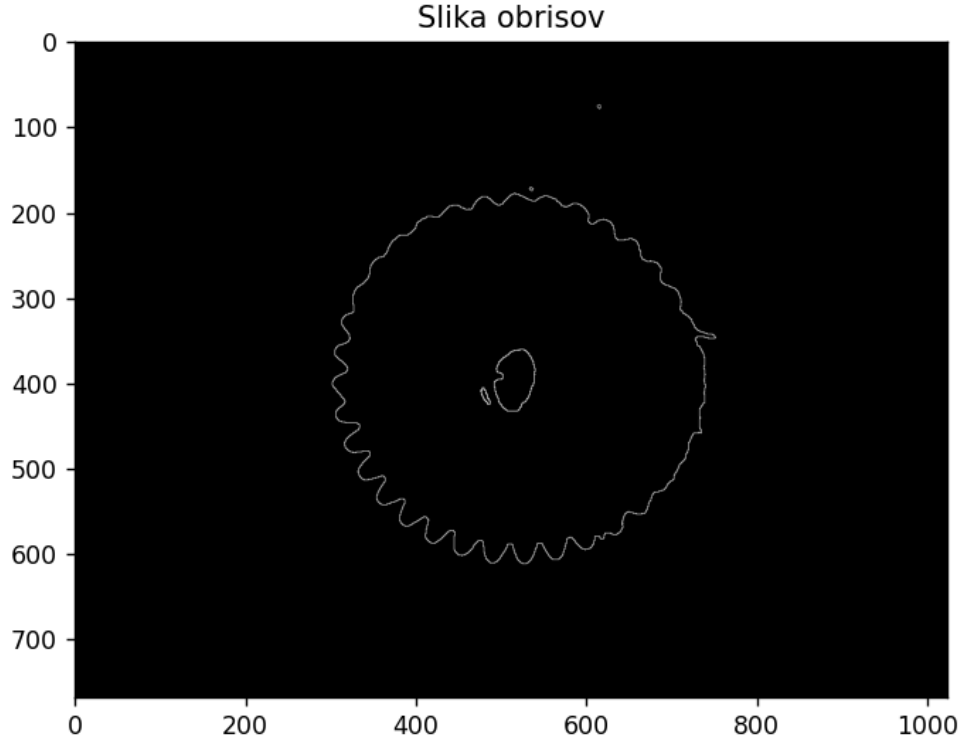
Za zobnik2:





Za zobnik1:


Slika najdaljsega obrisa
0
100
200
300
400
500
600
700
0
200
400
600
800
1000


Slika obrisov
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

Naloga 1c: iskanje obrisov področij

Pri zadnji nalogi smo morali opisati obrise s funkcijami Fourierovih koeficientov.

Najprej smo morali prebrati slike, ki so bile podane z nalogo.

```
    dn = "slike/"
    fns = "kladivo1 kladivo2 kladivo3 kljuc1 kljuc2 kljuc3".split(" ")
    fns = [dn + fn + ".png" for fn in fns]
    slike = []

    for k in range(len(fns)):
   #
        slika = cv2.imread(fns[k])
        slike.append(slika)
```

Del kode pred for zanko je bil že zapisan v predlogi. V for zanki, ki ima toliko iteracij kot je število elementov v seznamu fns, sem zapisal kodo s pomočjo funkcije iz knjižnice cv2. Vsako iteracijo se v spremenljivko slika shrani slika iz mape, ki je zapisana v seznamu fns. V naslednjem koraku se s funkcijo append element slika doda na zadnje mesto seznama slike.

Dopolniti smo morali tudi funkcijo obdelaj_sliko.

```
def obdelaj_sliko(slika):
    """Pomožna funkcija za predobdelavo slike."""
    # TODO: spiši funkcijo.
    # Predlagani koraki predobdelave:
    # 1. Prevzorčenje slike na velikost 256 x 256 pikslov
    siva_slika = cv2.cvtColor(slika, cv2.COLOR_RGB2GRAY)
    prevzorcena_slika = cv2.resize(siva_slika, dsize=(256, 256))
    # 2. Filtriranje prevzorčene slike z uporabo
    #    filtra mediane z okolico 11 pikslov
    obdelanaSlika = cv2.medianBlur(prevzorcena_slika, 11)
    # 3. Upragovljanje filtrirane slike z maksimizacijo informacije
    prag = dolociPrag(obdelanaSlika)
    _, upragovljena_slika = cv2.threshold(obdelanaSlika, prag, 255, 0)
    return upragovljena_slika
```

Za pravilno delovanje sem moral prvo sliko iz RGB formata spremeniti v sivi format, to sem storil z zapisom: siva_slika = cv2.cvtColor(slika, cv2.COLOR_RGB2GRAY). V naslednjem koraku je bilo potrebno slike obrezati v pravilne dimenzije za kar sem uporabil funkcijo cv2.resize. Sliko sem nato še obdelal s postopkom, ki sem ga že zapisal in izračunal prag za upragovljanje.

Pretvorba obrisa v signal:

```
def pretvori_obris_v_signal(obris):
    """Pretvorba obrisa,
    iz zapisa v obliki zaporedja točk v kompleksni signal."""
    tocke_obrisa = obris["tocke"]
    dolzina = len(tocke_obrisa)
    signal = np.zeros((dolzina,), dtype=np.complex128)
    # TODO: izvedi pretvorbo
    for i in range(dolzina):
       koordinata = tocke_obrisa[i]
       signal[i] = complex(koordinata[0],koordinata[1])
```

V danem segmentu je bilo potrebno spremeniti obris v signal zapisan s kompleksnimi vrednostmi. Predloga je že vsebovala del kode pred for zanko. V for zanki sem v spremenljivko koordinata shranil koordinate točk iz seznama slovarjev tocke_obrisa. Koordinate sem s funkcijo complex spremenil v kompleksne vrednosti in jih shranil v seznam signal.

V kodi smo morali dodati del v katerem izračunamo d_ij po formuli, ki je podana na prosojnicah z navodili:

```
    F_k = signal_fft[i + 1] **
    F_l = signal[-j + 1] ** i
    F1 = signal_fft[1] ** (i + j)
```

Koda za določitev vektorjev značilk:

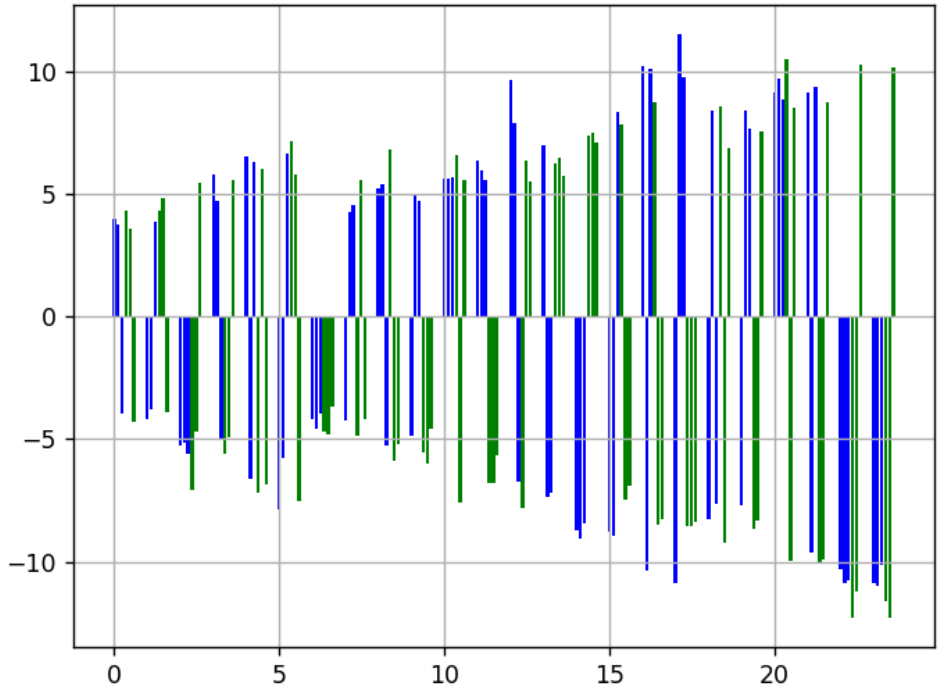
```
    vektorji = []
    for obdelana_slika in obdelane_slike:
        vektor = doloci_ffk(obdelana_slika, 4, 4)
        vektorji.append(vektor)
```

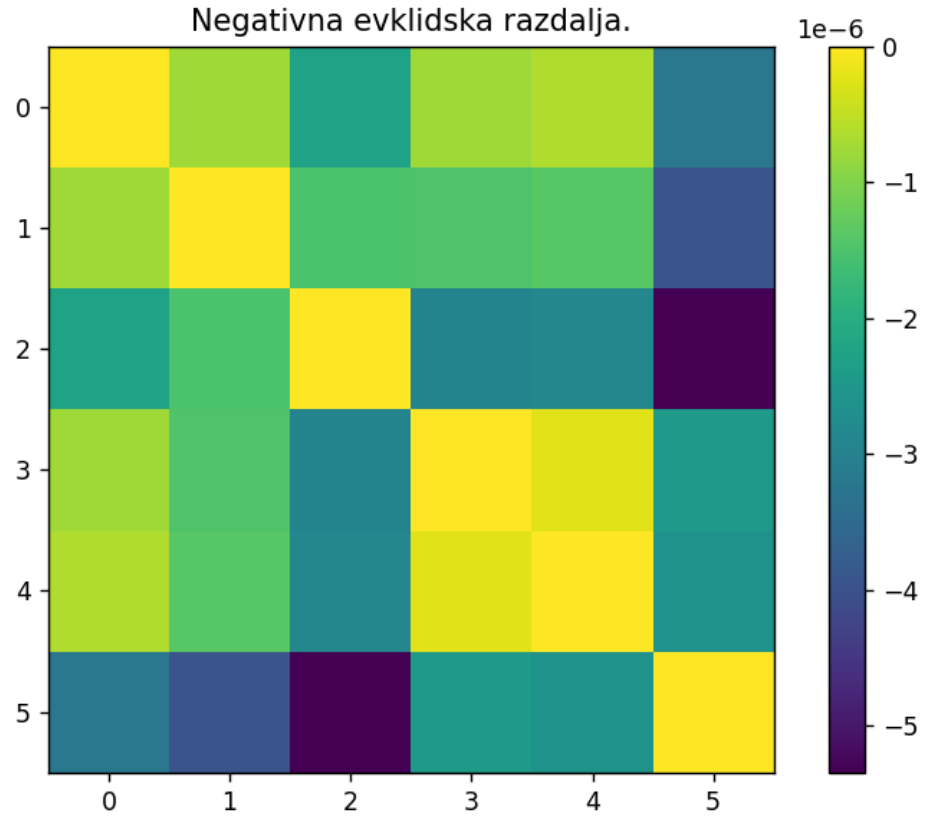
za boljši prikaz smo morali vektorje značilk spremeniti v logaritemsko merilo:

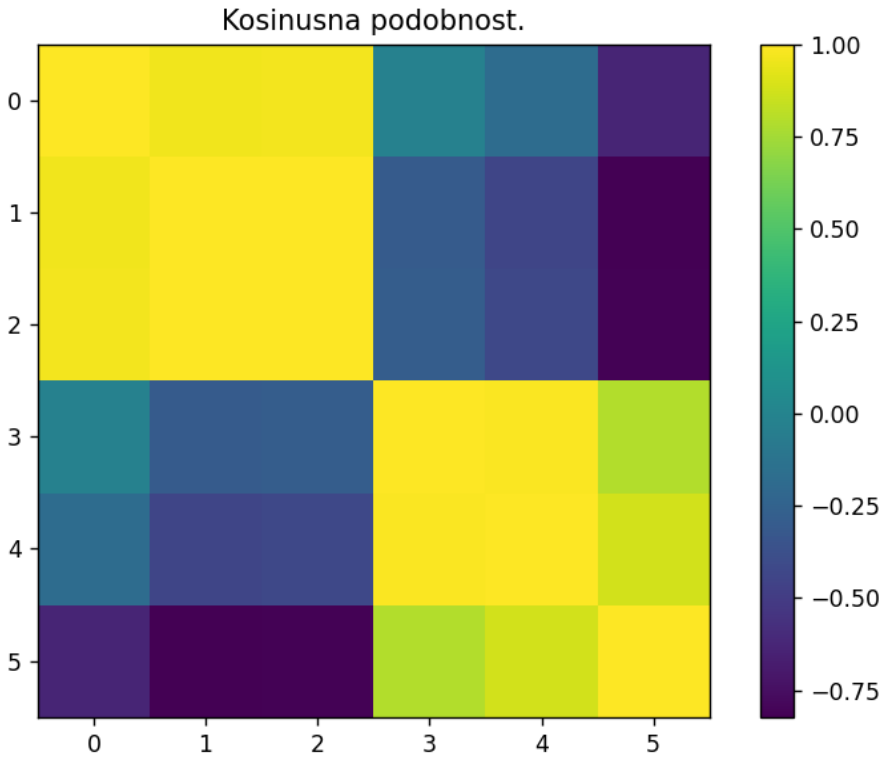
```
      log_prikaz = np.sign(vektor)*np.log10(np.abs(vektor))
      plt.bar(x, log_prikaz, width=0.1, color=barva)
```

enačba za izračun je bila že napisana v komentarju predloge

Rezultati:




Negativna evklidska razdalja.
1e−6
0
−1
−2
−3
−4
−5
0
1
2
3
4
5


Kosinusna podobnost.
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
1
2
3
4
5